Neural Networks - Exam topics

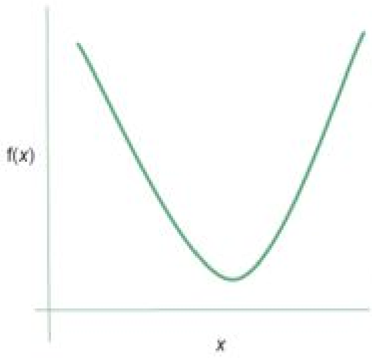
Ekart Csaba, 2019

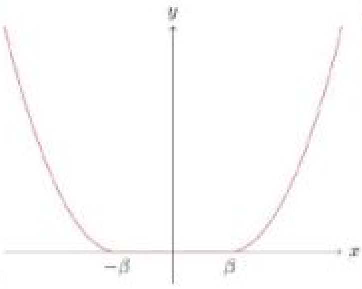
Értelemszerűen az előadás diákból van főleg, ahol az homályosan fogalmazott, vagy csak egyszerűen túl nagy káosz volt, hogy értelmes dolgokat lehessen belőle leszűrni (vagy all in all nem volt szó benne a témáról) ott kipótoltam netről. Nyilván nem vállalok felelősséget, meg ne alapozd erre az életed stb stb
Szupcsi videó tutorial sorozat, ami nagyon-nagy átfedésben van az anyaggal:
https://www.youtube.com/playlist?list=PL3FW7Lu3i5JvHM8ljYj-zLfQRF3EO8sYv
Ahol úgy éreztem, hogy van valami csudajó link, ami segíthet a dolog megértésében azt belinkeltem.

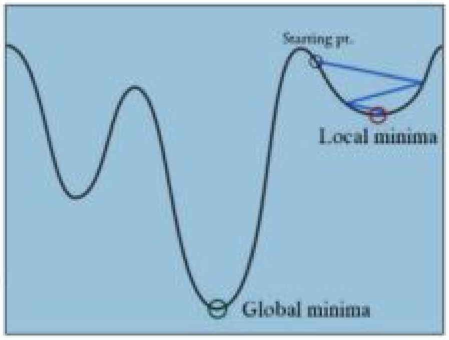
1. topic

Local optimization in non-convex cases (reason for non-convexity?)

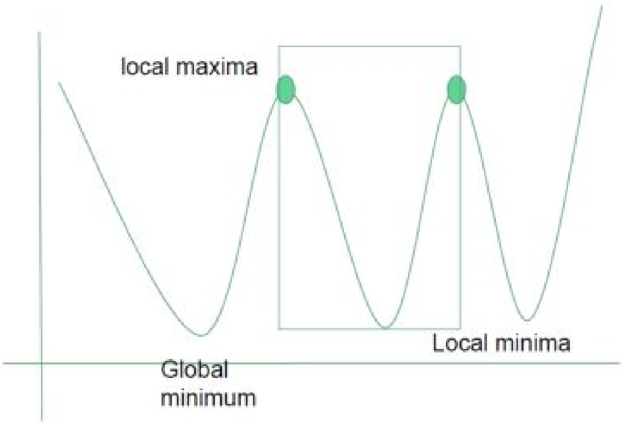
- In general a function based on convexity can be:
    - **Strongly convex function**: 1 local minimum
    - **Non-Strongly convex function**: infinity local touching minima, with the same values
    - **Non-convex function**: multiple non-touching local minima with different values







- Optimization is **done locally in a certain domain**, where the **function is assumed to be convex**.
- **Multiple local optimization is used** to find global minimum

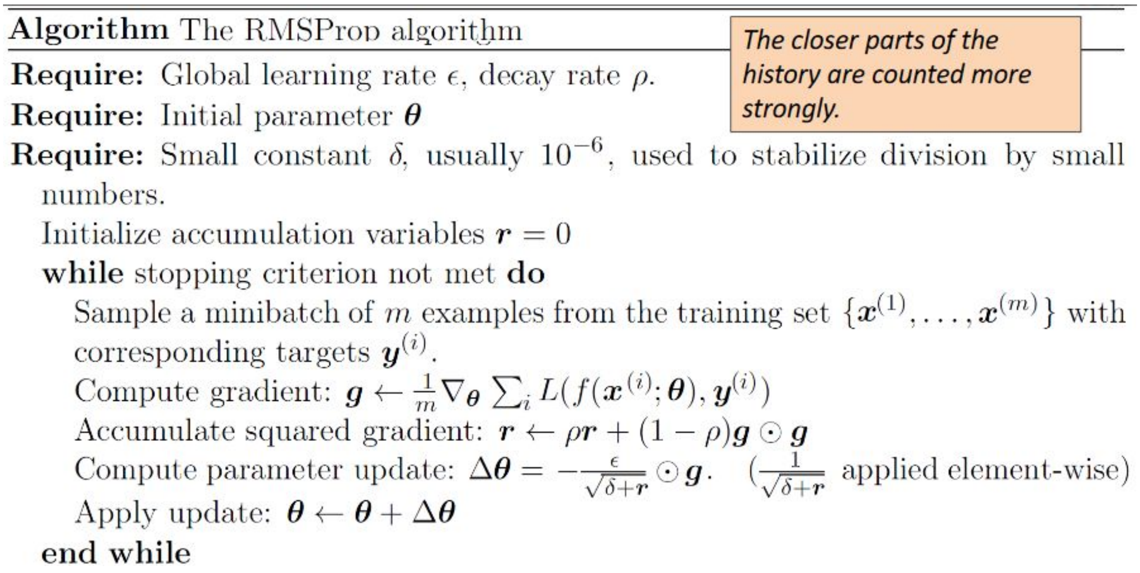


- The non-convexity is due to the use of a non-linear activation function in one of the layers.

https://www.quora.com/Why-is-a-neural-network-and-in-general-a-deep-network-non-convex

RMSP optimizer

- **Modified AdaGrad optimizer** to **perform better in the non-convex setting** by **changing the gradient accumulation** into an **exponentially weighted moving average**.
- In each step AdaGrad **reduces the learning rate, therefore after a while it stops entirely.**
- AdaGrad **shrinks the learning rate according to the entire history of the squared gradien**t and my have made the **learning rate too small before arriving at such a convex structure**
- RMSProp uses an exponentially decaying average to discard history from the extreme past, so that it can converge rapidly after finding a convex bowl, as if it were an instance of the AdaGrad algorithm initialized within that bowl.

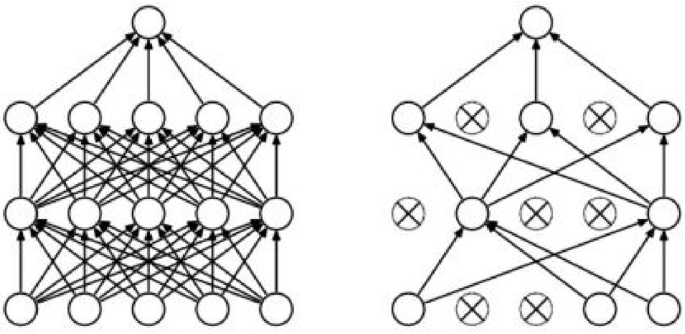


```
Algorithm The RMSProp algorithm
Require: Global learning rate $\epsilon$, decay rate $\rho$.
Require: Initial parameter $\boldsymbol{\theta}$
Require: Small constant $\delta$, usually $10^{-6}$, used to stabilize division by small numbers.
  Initialize accumulation variables $\boldsymbol{r} = 0$
  while stopping criterion not met do
    Sample a minibatch of $m$ examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Accumulate squared gradient: $\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1-\rho)\boldsymbol{g} \odot \boldsymbol{g}$
    Compute parameter update: $\Delta\boldsymbol{\theta} = -\frac{\epsilon}{\sqrt{\delta+\boldsymbol{r}}} \odot \boldsymbol{g}$.   ($\frac{1}{\sqrt{\delta+\boldsymbol{r}}}$ applied element-wise)
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
  end while
```

The closer parts of the history are counted more strongly.

Dropout

- Use **mini-batch training approach**
- For each minibatch a **random set of neurons** from hidden layer(s) (called **dropout layers**) is temporarily deactivated.
- Selection and deactivation probability is p
- In **testing phase** use all the neurons, but **multiply all the outputs** with p to account for the missing activation during training.
- **More training steps** but each is **simpler**, due to reduced number of neurons.
- Goal: **reduce overfitting by forcing the network to use different configurations / neural paths**

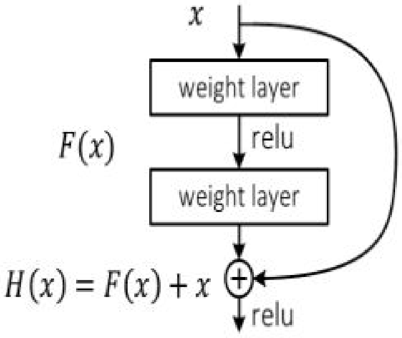


(a) Standard Neural Net (b) After applying dropout.

ResNet

- **Residual Network** makes it possible to train up to hundred or even thousands of layers and still achieves **compelling performance.**
- It is a **very deep** neural networks using residual connections.
- Why it is exist, and what the problem that it solves?
  - A **deeper network** always have the potential to **perform better but training can become difficult.**
  - **How** could we ensure that additional layers will **not decrease accuracy** (might even increase it)?
  - The trick is to use **residual connection** and as a starting point F(x) could be zero, and H(x) becames identity mapping.
  - So **H(x) won't change the performance, gradient will remain because the addition of x.**
  - Our accuracy won't be decreased, and might even be increased if we find a **proper F(x)**

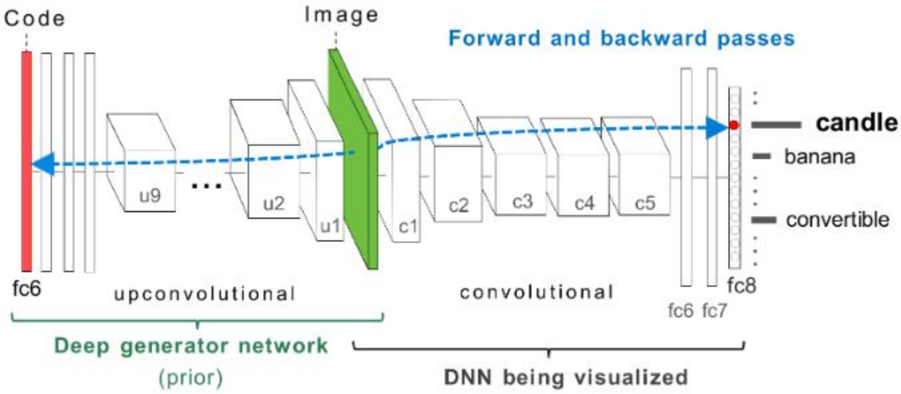
- ResNets had the **lowest error rate** as most competitions since 2015. (So it’s very good)



Gradient ascent

- Itt a lényeg, hogy: Találhatunk így képeket, amik nagyon jók egy adott klasszhoz. De mi van ha legeneráltatnánk vele, hogy számára milyen egy tökéletesen ideális kép.
- We could search in our database and find typical samples.
- It helps, but usually the network is good on this set (train accuracy). **We are curious about those images which the network has not seen.**
- Could we generate an **ideal image** for classes?
- **Normal training:**
    - input image - given
    - network parameters - given
    - expected label - given
- **Gradient ascent**
    - input image - variable
    - network parameters - given
    - expected label - given
- Hogyan működik ez?
    - **Generate synthetic image that maximizes the response of a neuron.**
    - This image has to be “**natural**”. The response should not **depend on pixels and cant have arbitrary values.**
        - **Gaussian blur** on the image
        - **Clipping image values**
        - **Clipping small gradients to 0.**

$$I^* = \arg\max_I \underbrace{\boxed{f(I)}}_{\text{Neuron value}} + \underbrace{\boxed{R(I)}}_{\text{Natural image regularizer}}$$

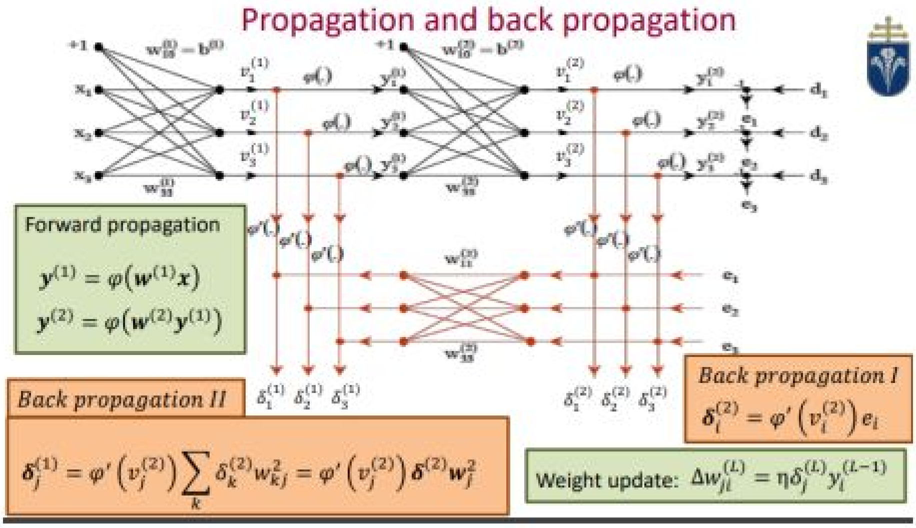


- **prior**: prior knowledge. untrained network: prior means we encode some previous knowledge / distribution to the network
- **Maximizing patterns for each kernel**
  - Making sense of these activations is **hard because** we usually **work with them as abstract vectors**
  - With **feature visualisation**, we can **transform** this **abstract vectors to more meaningful semantic dictionaries.**
- **Usage:** Style transfer

2. topic

Weight update strategies

- Apply all the input vectors in one after the others, selecting them randomly
- **Instance update**:
    - Update the weights **after each input**
- **Batch update**:
    - Updates are **calculated for each vector** and **averaged**
    - Update is done with the averaged values, after the entire batch is calculated
- **Mini batch**:
    - If the number of inputs are very high (100.000-1.000.000), batch would be ineffective
    - **Select random m input vectors** (m is a few hundred)
    - Updates are calculated for **each vector and averaged**
    - Update is done with the averaged values, after the mini batch is calculated
    - **Works efficiently when far away from minimum, but inaccurate close to minimum**
    - Requires reducing learning rate
- Valahogy ezt az updatesdit a **back propagationnál** is használtuk.



Propagation and back propagation

Forward propagation

$$y^{(1)} = \varphi\left(w^{(1)}x\right)$$

$$y^{(2)} = \varphi\left(w^{(2)}y^{(1)}\right)$$

Back propagation I

$$\delta_i^{(2)} = \varphi'\left(v_i^{(2)}\right)e_i$$

Back propagation II

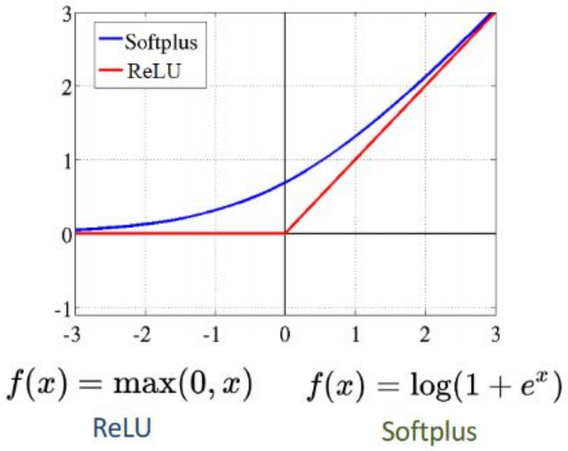
$$\delta_j^{(1)} = \varphi'\left(v_j^{(2)}\right)\sum_k \delta_k^{(2)} w_{kj}^2 = \varphi'\left(v_j^{(2)}\right)\delta^{(2)} w_j^2$$

Weight update: $\Delta w_{ji}^{(L)} = \eta \delta_j^{(L)} y_i^{(L-1)}$

ReLU and the dying ReLU problem

ReLU definition

- **Rectified Linear Unit**
- Widely used activation function in **hidden layers**.
- Very **easy to calculate**
- Also **easy to derivate**
- Smooth analytic approximation is the **soft plus function**, which asymptotically reaches ReLU



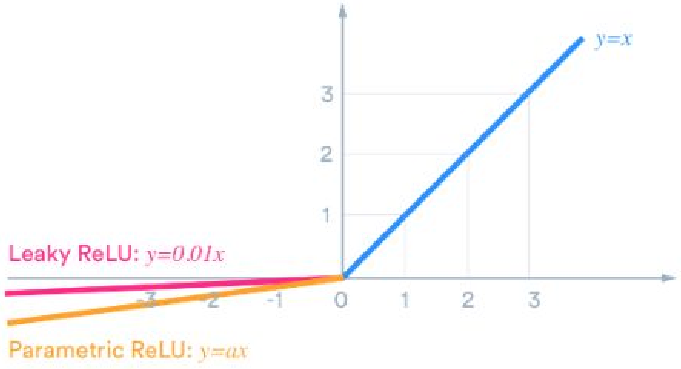
Dying ReLU problem

- During training it happens that the weight composition of a neuron **got a certain combination in a high gradient situation** (large jump happens during optimization), which leads to **generate zero output from that point on**.
  - Happens typically with **large learning rate**
  - E.g. very **large negative value appears in the bias**
- That neuron will output zero for each input vector from that point.
  - Irreversible
- Solution: Leaky ReLU, ELU, SELU etc.

Leaky ReLU

- No constant zero output, so neurons won't die
- Leaky ReLUs are not necessarily superior than normal ones.

$$f(x) = max(x, ax)$$

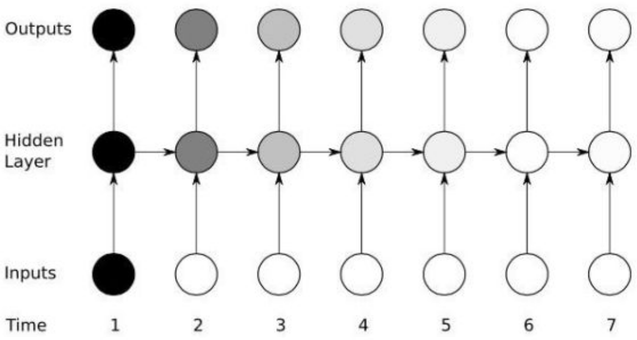


LSTM cell

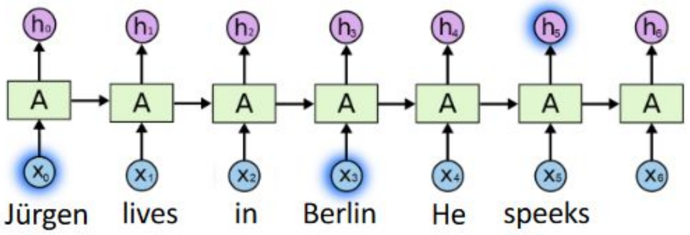
(igen ez nagyon hosszú, valószínűleg nem kell minden, de szerintem simán belekérdezhet, ezért a teljesség kedvéért itt vannak, itt meg lehet nézni alaposan, nagyon szépen érthetően és olvasmányosan ki van fejtve: https://colah.github.io/posts/2015-08-Understanding-LSTMs/)

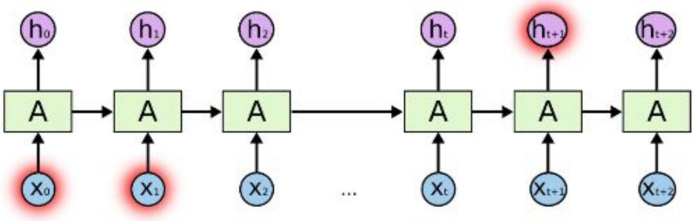
- **Miért?**
  - Eleinte voltak a sima rekurrens hálók és ezek nekünk teljesen jók voltak. Ezekkel fel tudtunk ismerni beszédet, nyelvet, videót stb. Ezt viszont tovább kellett fejleszteni mert volt velük egy-két probléma:
  - **Vanishing gradient**
    - In case of long input vector sequences the old vectors has a **strong fading effect in inference phase**

        - In the training phase the stacked gradient functions will be very small.

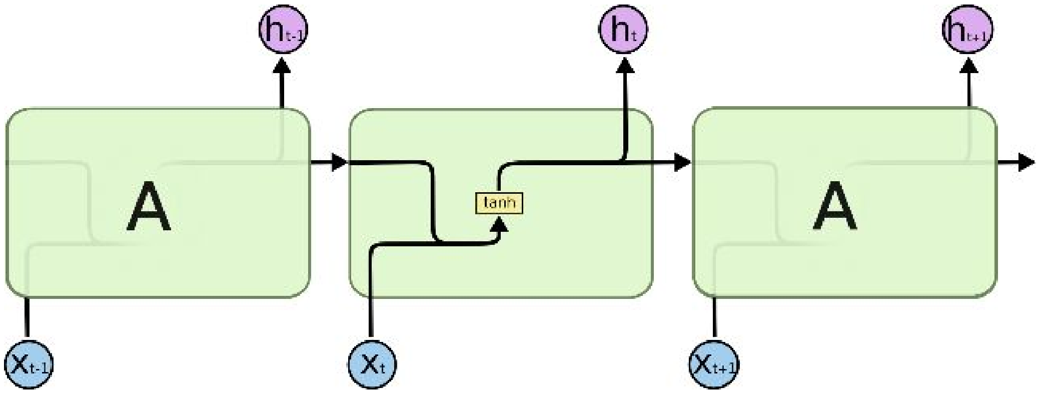


    - **Practical problem of long term dependencies**
        - Consider a network which predicts the next word in a text
            - If the **information** needed to predict is **close**, it can be **successfully trained**
            - If required **information** is **far**, the training will be **difficult**.

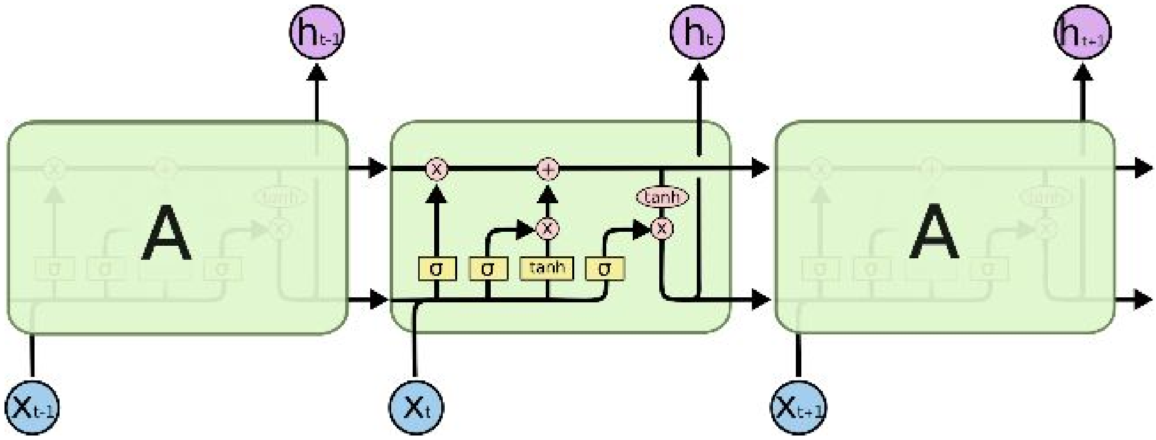




- Long Short Term Memory
- **Special type of RNN, capable for long term dependencies.**
- **Standard RNN:**

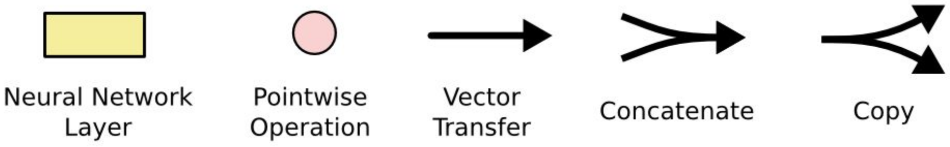


    - Simple architecture, chain structure, single tanh layer.
- **LSTM:**
    - LSTM has the same chain structure, but instead of having a single neural network layer, there are 4, interacting in a special way.

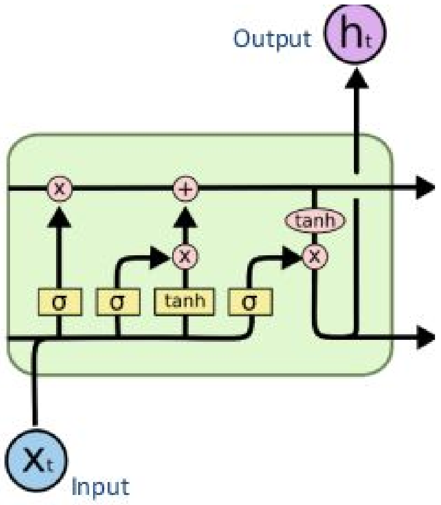


- **Idea:**
  - Able to **learn long term dependencies**
  - **Collects data** when the **input** is considered to be **relevant**
  - **Keeps it as long as** it considers to be **important**
- **Technique**:
  - Handle the **state as a memory** with **minor modification**
    - **no matrix multi.**
    - **no tanh.**
    - **apply memory handling kind signals** (data in, data out, write, enable) etc.

Components of LSTM



- **State of the LSTM (upper horizontal line in the architecture)**
  - **This is the actual memory**
  - It can pass the **previous values with or without update**
  - Indicated with **C_t**
- Old content can be removed value-by-value
- New content can be added
- Sigmoid layer: how much each component should be let through.

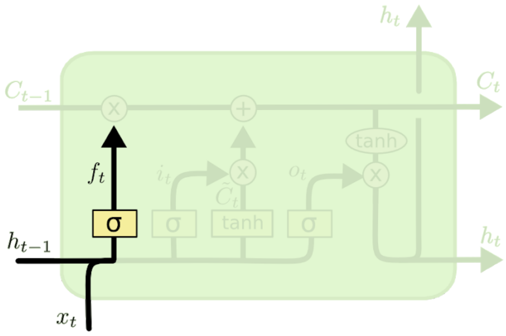


How LSTM works?

Step 1

- **Combines input and previous output (conc.)**

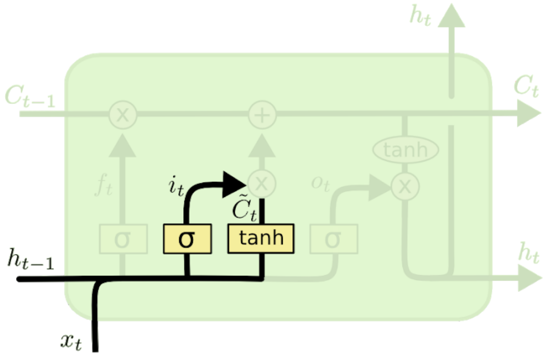
- Selects **which values to forget**
  - Done by: **Forget gate layer**
  - **NN with sigmoid output**



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

Step 2

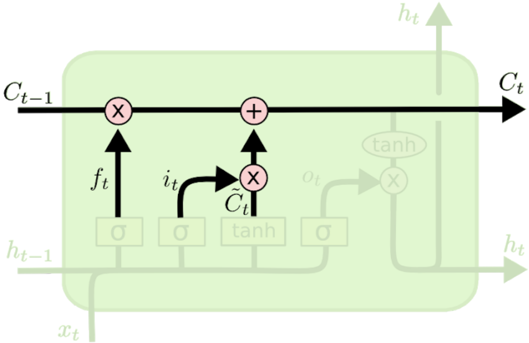
- What new information we're going to store in the cell state?
- **Selection of state values to be updates**
  - Which values will be updated
  - Done by: **Input gate layers**
  - **NN with sigmoid**
- **Calculation of the state update**
  - Done by: **Cell network**
  - Not yet the new, only the update value
  - **NN with tanh**



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

Step 3

- Update the old cell state
- **Calculation of the state update:**
  - **Add the old state and the state up**

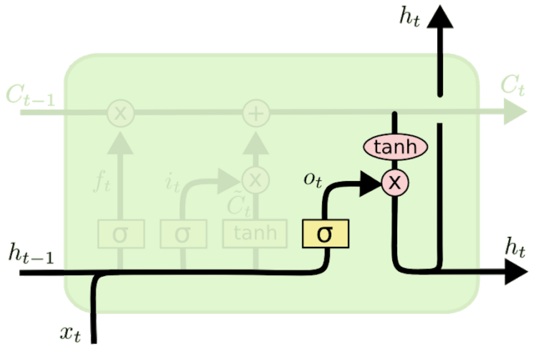


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Step 4

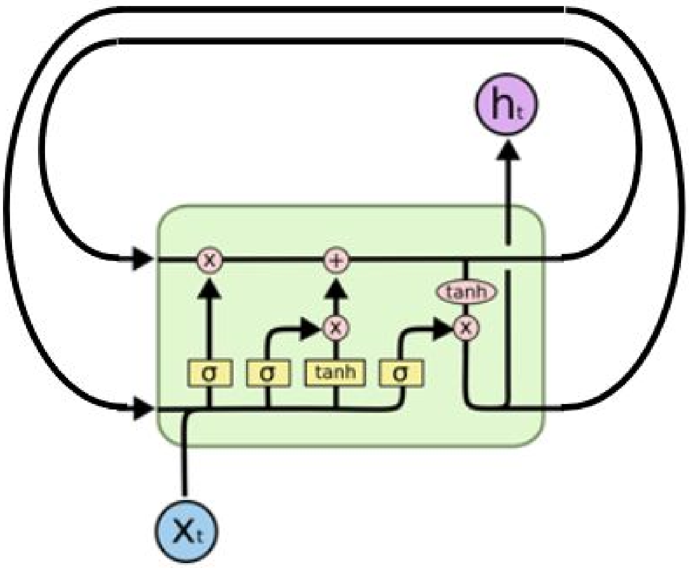
- Decide what we're going to output
- Selection of the new output values
  - **Sigmoid:** what parts of the cell state we are going to output
  - Done by: **Output gate**
  - Output gate decides **which values are relevant**

- **Apply activation function to the output**
  - squeeze values between -1 and 1
  - **tanh**

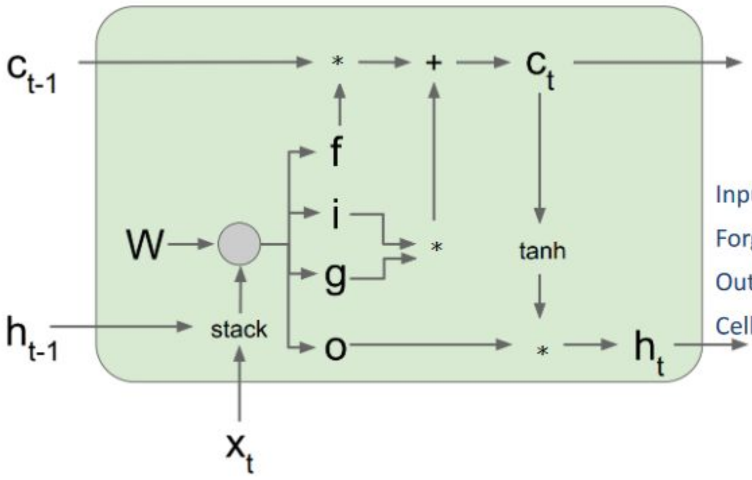


$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

General form of an LSTM network



Gradient calculation in LSTM



$$\begin{array}{l} \text{Input} \\ \text{Forget} \\ \text{Output} \\ \text{Cell Net} \end{array} \begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

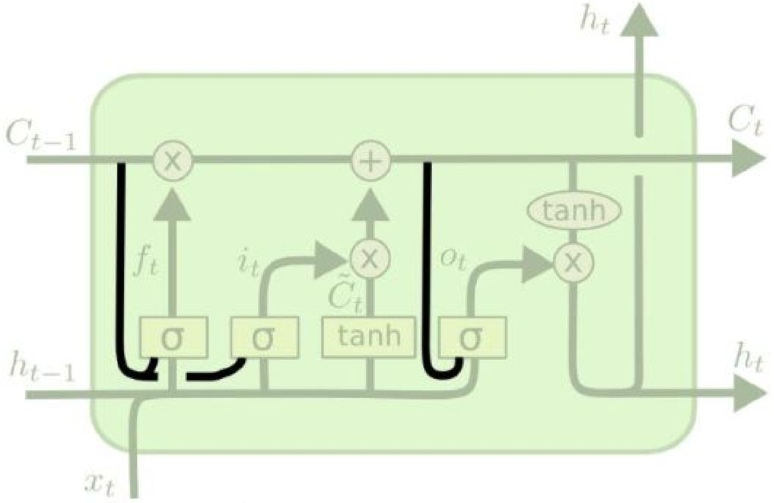
$$c_t = f\ * c_{t-1} + i\ * g$$
$$h_t = o\ * \tanh(c_t)$$

Achievements with LSTM networks

- Record results in natural language text compression
- Unsegmented connected handwriting recognition
- Natural speech recognition
- Smart voice assistants (Google Assistant, Alexa, Cortana, Siri etc.)

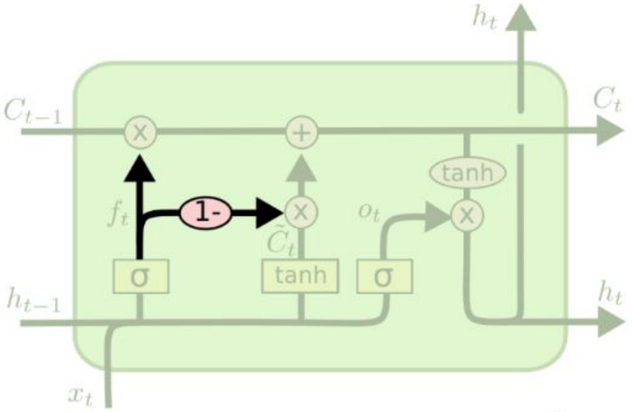
Variants of LSTM networks

- **Peephole connections**
  - **Let the gate layers look at the cell state.**
  - All the **three gates receives input from the previous state and the input**
  - **Output** can be **sparse** -> this version has **more information for gating**



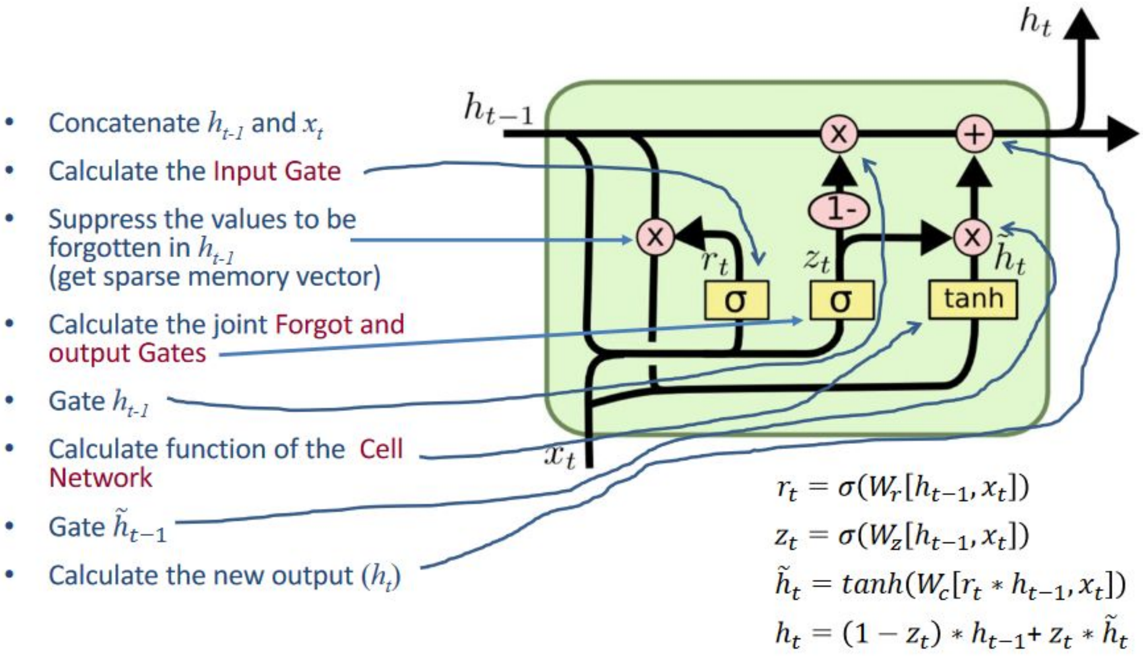
$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \ + \ b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \ + \ b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] \ + \ b_o\right)$$

- **Joined forget and input**
  - **Input & forget gates : same role**
  - Why not to **join them**?
  - Instead of separately deciding what to forget and what we should add new information to, we make those decisions together.
  - We only forget when we're going to input something in its place. We only input new values to the state when we forget something older.



$$C_t = f_t * C_{t-1} + \boldsymbol{(1 - f_t)} * \tilde{C}_t$$

- **Gated Recurrent Unit (GRU)**
  - **Combines** the **forget** and **input gates** into a single **"update gate."**
  - **Merges** the **cell state and hidden state**, and makes some other changes
  - Output won't be sparse
  - Learns faster on smaller dataset
  - How it works? (ábrán)



- Concatenate h_{t-1} and x_t
- Calculate the Input Gate
- Suppress the values to be forgotten in h_{t-1} (get sparse memory vector)
- Calculate the joint Forgot and output Gates
- Gate h_{t-1}
- Calculate function of the Cell Network
- Gate $\tilde{h}_{t-1}$
- Calculate the new output (h_t)

$$r_t = \sigma(W_r[h_{t-1}, x_t])$$
$$z_t = \sigma(W_z[h_{t-1}, x_t])$$
$$\tilde{h}_t = tanh(W_c[r_t * h_{t-1}, x_t])$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Convolution as a mathematical operation in continuous and discrete cases

- Convolution is a mathematical operation that
  - does the i**ntegral of the product of two functions** (signals)
  - with **one** of the signals **flipped, and shifted.**
- Mathematically:
  - **Continuous case**

$$\begin{aligned}(f * g)(t) &\overset{\text{def}}{=} \int_{-\infty}^{\infty} f(\tau) g(t-\tau)\, d\tau \\ &= \int_{-\infty}^{\infty} f(t-\tau) g(\tau)\, d\tau\end{aligned}$$

  - **Discrete case**

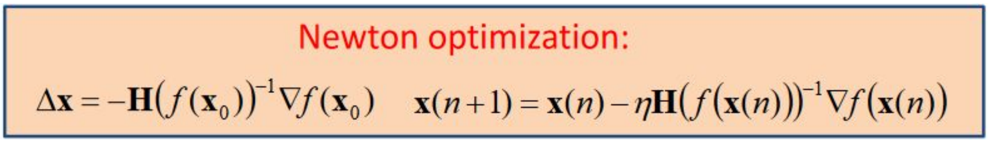
$$\begin{aligned}(f * g)[n] &= \sum_{m=-\infty}^{\infty} f[m] g[n-m] \\ &= \sum_{m=-\infty}^{\infty} f[n-m] g[m]\end{aligned}$$

- **Convolution most important properties**: commutativity, associativity, distributivity

3. topic

Newton optimization method

- Second order method.
- When f is a positive definite quadratic function, Newton's method jumps ins a single step to a minimum of the function directly.
- The method can reach the critical point much faster than 1st order gradient descent.



Newton optimization:

$$\Delta\mathbf{x} = -\mathbf{H}(f(\mathbf{x}_0))^{-1}\nabla f(\mathbf{x}_0) \qquad \mathbf{x}(n+1) = \mathbf{x}(n) - \eta\mathbf{H}(f(\mathbf{x}(n)))^{-1}\nabla f(\mathbf{x}(n))$$

Ensemble, bagging

- A regularization / optimization method
- Ensemble methods: Network duplications, bagging, dropout
- Idea of ensemble methods:
    - **Generate multiple copies of your net (same or slightly modified architectures)**
    - **Train them separately:**
        - Using different subsets of the training sets
        - Different objective functions
        - Different optimization functions
    - **Averaging the result will lead to smaller error**
- Requires **more computation and memory** both in training and inferencing (testing) phase

Dropout

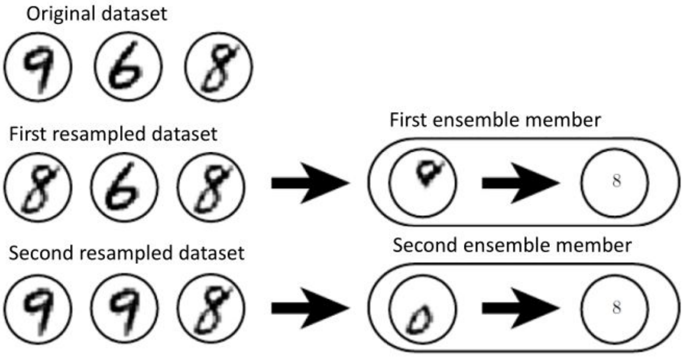
- Másik tételben ott van.

Network duplications

- **Train two architecturally identical copies of the network on two GPU-s.**
- Half of the neuron layers are on each GPU, and they can only communicate on certain layers.

Bagging

- Construct k different datasets
- Each with subset of the data, but with duplications
- Train with these
- Make result averaging



Comparison of loss functions

- Loss function determines the training process
  - **Tells the net the size of the error, and penalize according to it**
  - Eddig: difference of the output and the desired output

Quadratic loss function

- Mean squared differences between the desired and the actual outputs.

$$R_{emp}(\mathbf{w}) = \frac{1}{K}\sum_{k=1}^{K}\left(d_k - Net(\mathbf{x}_k, \mathbf{w})\right)^2$$

- Problem: can be very slow (with sigmoid even at large error) -> slow convergence

Conditional log-likelihood

- For classification
- Sum of negative logarithmic likelihood

$$C(\mathbf{w}) = -\frac{1}{K}\sum_{k=1}^{K}\left(-logP(\mathbf{y}_k|\mathbf{x}_k, \mathbf{w})\right)$$

Cross Entropy

- Better loss function (solve slowness of quadratic)

$$\boxed{C = -\frac{1}{n}\sum_x \left[y \ln a + (1-y)\ln(1-a)\right]}$$

$$C(\mathbf{w}) = -\frac{1}{K}\sum_{k=1}^{K}\left(d_k\, logP(\mathbf{y}_k|\mathbf{x}_k, \mathbf{w}) + (1-d_k)log\left(1 - P(\mathbf{y}_k|\mathbf{x}_k, \mathbf{w})\right)\right)$$

- Penalize heavily the confident, but wrong predictions.
- Better, because its partial derivative does not contains sigmoid derivative. The gradient is proportional with the value of the sigmoid and not with its derivative.

$$\frac{\partial C}{\partial w_j} = \frac{1}{K}\sum_{k=1}^{K} x_j(\sigma(\mathbf{wx} + \mathbf{b}) - d)$$

Negative log-likelihood (nem feltétlenül fontos, de ide tartozik)

$$L(\mathbf{y}) = \sum_{k=1}^{K} -\log(y)$$

- Loss function for softmax
- Sum of negative logarithms of the probability of the correct decision classes.
- Small, if the confidence of a good decision was high, large when the confidence is low.
- Partial derivative of a softmax layer with negative log-likelihood:

$$\frac{\partial C}{\partial v_j} = y_j - 1$$

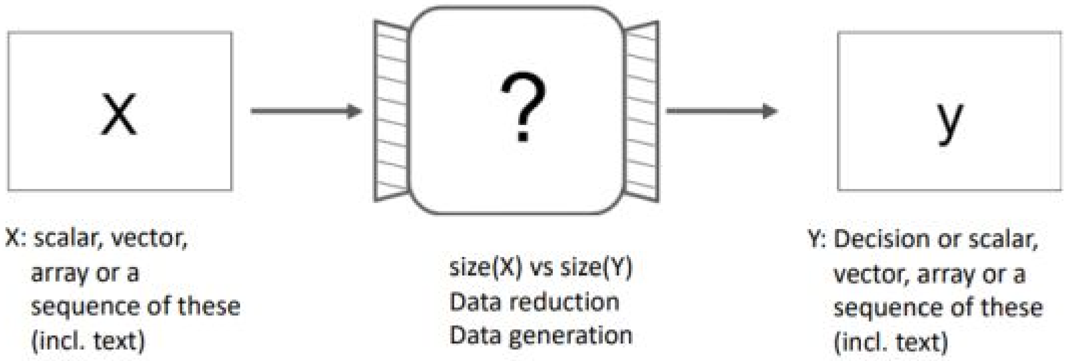
https://medium.com/deep-learning-demystified/loss-functions-explained-3098e8ff2b27

Machine learning vs traditional programming

Traditional programming approach	**Machine learning approach**

● **Trivial, or at least analytically solvable tasks** ○ Well established mathematical solution exist or at least can be derived ● Example: ○ Finding well defined data constellations in a database ○ Formal verification of the operation easy	● **Complex underspecified tasks** ○ No exact mathematical solution exists, the function to be implemented is not known ● Example: ○ Searching for “strange” data constellations in a database ○ Verification of the operation is difficult ● It’s very hard to know if you’re program works correctly, have to do massive amount of testing

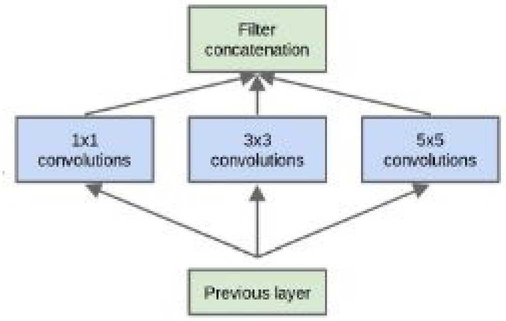
- **In machine learning each task is an input-output problem.**


X
?
y
X: scalar, vector, array or a sequence of these (incl. text)
size(X) vs size(Y)
Data reduction
Data generation
Y: Decision or scalar, vector, array or a sequence of these (incl. text)

- The reason why its only became popular in the recent years, is the appearance of new frameworks and methods, giant amount of data and very powerful hardwares.
- The main three types of learning: **Supervised learning** (on labeled examples), **Unsupervised learning** (unlabeled examples), **Reinforcement learning** (trial and feedback)

Inception

- Network architecture developed at Google
- **Main idea**
  - **Not** to introduce **different kernels in different layers,** but introduce **1x1, 3x3, 5x5 in each layers**, and let the **NN figure out**, what representation is the **most useful, and use that.**
  - **Parallel multiscale approach.**



- “Pooling of features” because we are reducing the depth of the volume, similar to how we reduce the dimensions of height and width with normal maxpooling layers.

Rethinking Inception (Ezt nem teljesen értem, nem is biztos, hogy kell)

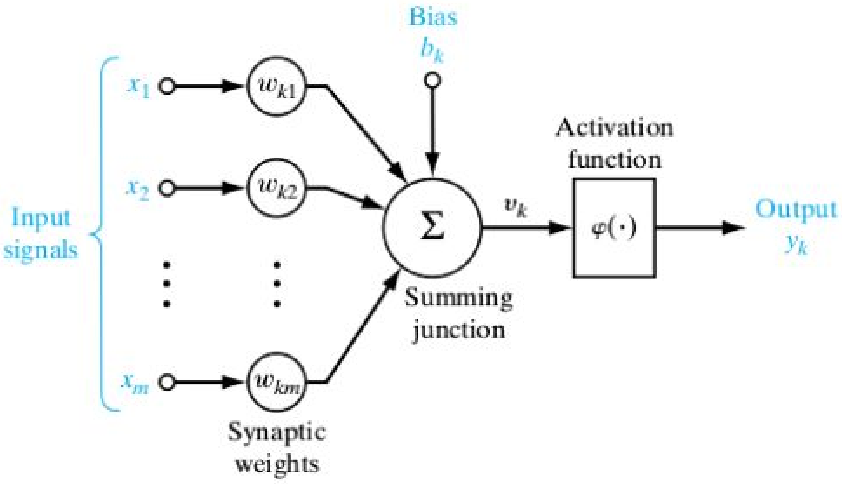
- Squeezing the number of channels for each kernel
- With concatenations the number of features increased in each layers, which introduced too many convolution

- To reduce these numbers they introduced the 1x1 layer. It can generate e.g. 16 feature maps from 64 feature maps
- Larger (5x5) convolutions were substituted by series of 3x3 convolutions
    - Reduction of number of parameters
    - Additional non-linearities (ReLU) can be introduced
- 2D convolutions were substituted by two 1D convolutions

4. topic

McCulloch-Pitts model

- **The artificial neuron is an information processing unit, that is basic constructing element of an artificial neural network.**



- x_i **input** vector
- w_{ki} : **weight** coefficient vector of neuron k
  - $w_i > 0$: excitatory input
  - $w_i < 0$: inhibitory input
- b : **bias** the sum to enable asymmetric behaviour
- **Activation function**: shapes the output signal
- **The output equation:**

$$y_k = \varphi\left(\sum_{i=1}^{m} w_{ki} x_i + b_k\right)$$

- With **included bias:**

w_0=b

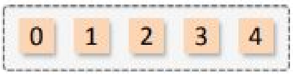
x_0=1

$$y_k = \varphi\left(\sum_{i=0}^{m} w_{ki} x_i\right) = \varphi(\mathbf{w}^T \mathbf{x})$$

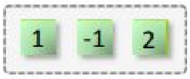
Parameters of convolution - filter, stride, padding, etc.

Egy-két szó a konvolúcióról in general

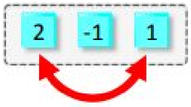
- Oké, korábban már volt a konvolúció képlete, azt azért felpingálnám a papíromra a vizsgán, hogy látszódjon, hogy tudom miről beszélek. Mivel szerintem ez a rész inkább arról szól, hogy érted e a mókát, ezért itt csak megpróbálom elmagyarázni. Nyilván vizsikén ennyit nem kell leírni, mert ott már érteni fogod ugyebár.
- Szóval ugye digjelen játszottunk már ilyen 1D-s konvolúciósat. Ennek a lényege, hogy:
  - Van egy függvényed, kinek a neve f és így néz ki:



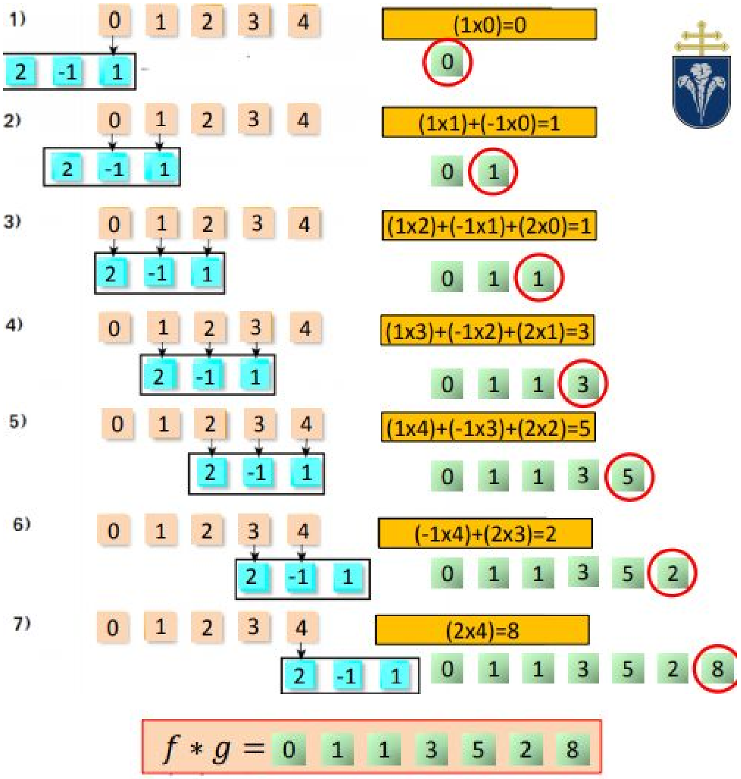
- Neki pedig ugye van egy barátja g, akivel majd ügyesen össze konvuláljuk:



- Na most ilyenkor az első dolgod, hogy megtükrözöd a drágát (g-t):



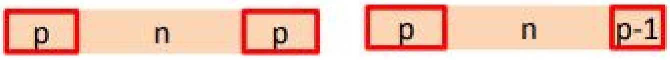
- Ezután pedig ezt a cuccost, az alábbi leírásnak megfelelően tologatod, és az egymás alatt lévő számokat összeszorzod és összeadod. A kapott eredményeket egymás mögé írod és tádá kész vagy:



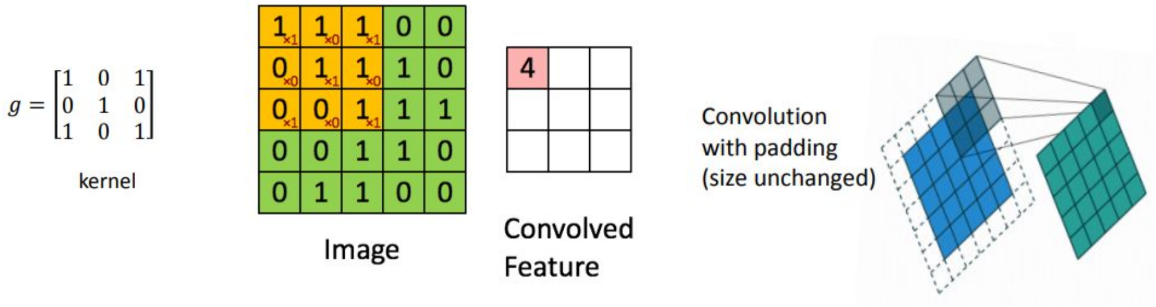
- Van itt két fontos definíció:
    - **Valid positions**: the flipped g is **completely inside** f (fully overlapping positions) (ábrán 3-4-5. lépés)
    - **Boundary positions**: **partially overlapping** positions (a többi lépés)
- In practice convolution is used as a **filter**, where **f is the measurement data** and **g is the filter function descriptor** (**kernel**).
- **Size of the result**
    - size(f) >> size(g)
    - size(f) = n, size(g) = k, n>=k
    - size(f*g) = n + k -1 (ha az összeset számolod)

Padding

- In CNN, we calculate the **valid values only**
- We do not want **size changes** on the data blocks.
- To **avoid** these size changes, we have to **pad the data block with zeros at boundaries** (if the kernel size is odd, the padding is symmetric, and if it’s even th padding is asymmetric)



- Persze van akkor ez 2D-ben is, ott egy táblázatot tologatsz egy nagyobb táblázaton.

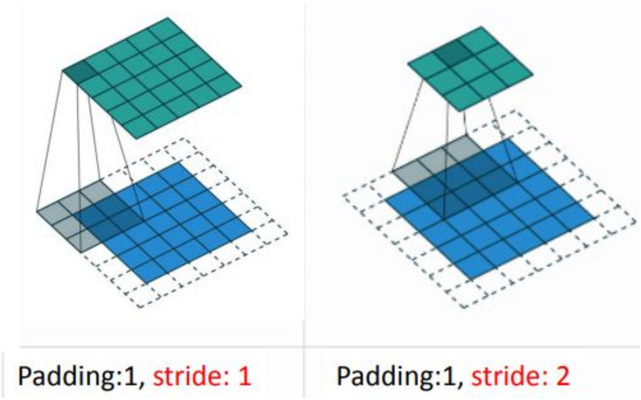

$g = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$
kernel
1 1 1 0 0
0 1 1 1 0
0 0 1 1 1
0 0 1 1 0
0 1 1 0 0
Image
4
Convolved Feature
Convolution with padding (size unchanged)

- **Why use padding?**
  - **Simplifies** the execution code
  - Do not have to deal with different calculation methods at the boundaries
  - **Same code runs in the entire array**

Strides

- Stride is **the number of pixel what we slide the kernel** (horizontal stride, vertical stride)
- Can use to down sample the image.
- size(f) = n, size(g) = k, p = padding, s = stride

$$\frac{n+2p-k}{s} + 1$$



Filter

- Convolution in the Fourier domain is a multiplication:

$$\mathcal{F}\{f * g\} = \mathcal{F}\{f\} \cdot \mathcal{F}\{g\}$$

$$\mathcal{F}\{f \cdot g\} = \mathcal{F}\{f\} * \mathcal{F}\{g\}$$

- Therefore:

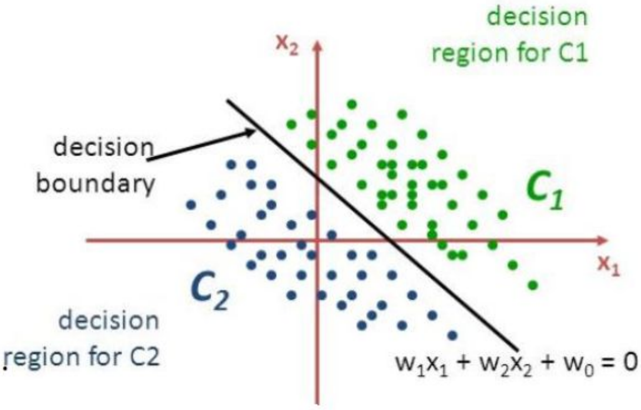
$$\boxed{f * g = \mathcal{F}^{-1}\{\, \mathcal{F}\{f\} \cdot \mathcal{F}\{g\}\}}$$

$$f \cdot g = \mathcal{F}^{-1}\{\, \mathcal{F}\{f\} * \mathcal{F}\{g\}\}$$

- We can use convolution to filter an image, with these we can find different features, properties on the images. Different kernels produces different results (edge-detection filter etc.)

Linear classifier, margin of the classifier

- In a **2D input space** the **hyperplane is a straight line.**
- **Above the line is classified 1**
- **Below the line is classified 0**



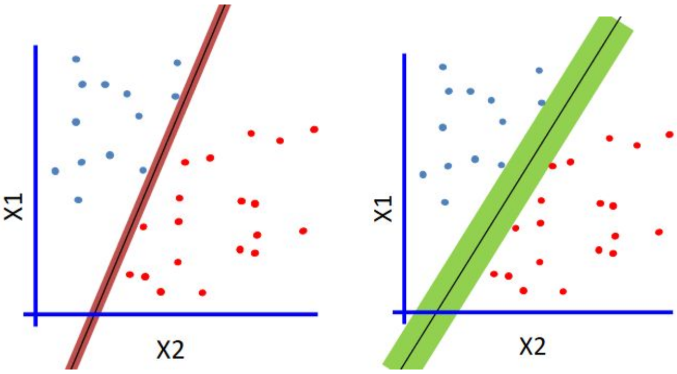
- The **decision boundary** is a **hyperplane** defined:

$$\mathbf{w}^T\mathbf{x} = 0$$

- **Why hyperplane?**
  - **Most logic functions has this complexity.**
  - **Common** in **mathematical** and computational tasks
  - Using **multiple hyperplanes** -> **more complex decision boundary.**
- Two sets are **linearly separable** if there exists **at least one hyperplane** in the space with **all of the blue points on one side of the line and all the red points on the other side.**

Margin of the classifier

- **Maximum margin**: Define the margin of a linear classifier as the **width that the boundary could be increased by before hitting a data point.**



https://towardsdatascience.com/linear-classifiers-an-overview-e121135bd3bb

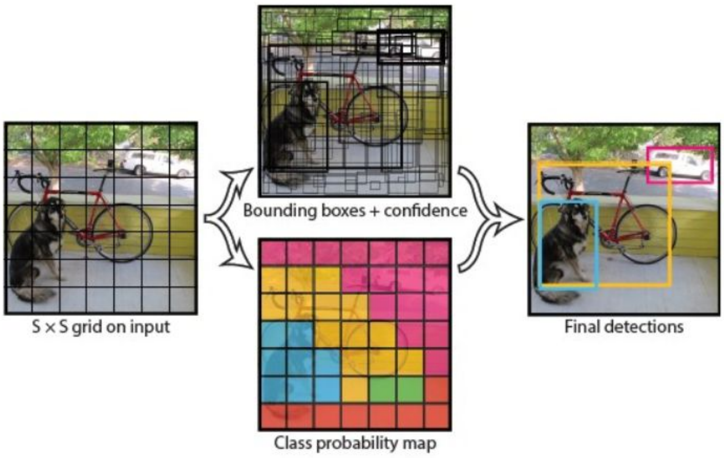
Data augmentation

- A **regularization / optimization** method (increase loss, reduce overfitting)
- **Idea**: Increase the generalization capability of the net by **enlarging the training set**.
- Increase the number of the training vector by introducing **fake (artificial) input-output pairs.**
- **Typical methods**: translating, slight rotation, scaling, add noise, flipping etc.
- **U-net** például használata szépen, mert ott a specifikus téma (orvosi akármi) kevés a training data.
- De az **AlexNet-ben** is van

YOLO

- **You Only Look Once**
- Special **network architecture,** Unified, **Real-Time Object Detection**
- **Model detection as a regression problem:**
    - **Divide the image into a grid** and **each cell can vote** for the **bounding box position of possible object**.
    - Boxes can have arbitrary sizes
    - Each cell can proposes a bounding box one category
    - Non-suppression on the boxes
- **No need for scale search**, the image is **processed once** and **objects in different scales can be detected**

Unified detection



- **Confidence scores**: reflect how confident is that the box contains an object + how accurate the box is.

$$\Pr(\text{Object}) * \text{IOU}_{\text{pred}}^{\text{truth}}$$

- **Conditional class probabilities**: conditioned on the grid cell containing an object

$$\Pr(\text{Class}_i|\text{Object})$$

$$\Pr(\text{Class}_i|\text{Object}) * \Pr(\text{Object}) * \text{IOU}_{\text{pred}}^{\text{truth}} = \Pr(\text{Class}_i) * \text{IOU}_{\text{pred}}^{\text{truth}}$$

- **At test time, multiply the conditional class probabilities and the individual box confidence predictions.**
- Giving **class-specific confidence scores for each box.**
- Showing both the probability of that class appearing in the box and **how well the predicted box fits the object**

https://medium.com/@jonathan_hui/real-time-object-detection-with-yolo-yolov2-28b1b93e2088

5. topic

Statistical learning theory

- **Empirical error**

$$R_{emp}(\mathbf{w}) = \frac{1}{K}\sum_{k=1}^{K}\left(d_k - Net(\mathbf{x}_k, \mathbf{w})\right)^2$$

- **Theoretical error**

$$\left\|\mathrm{F}(\mathbf{x}) - \mathrm{Net}(\mathbf{x}, \mathbf{w})\right\|^2 = \int \underset{X}{\cdots} \int \left(\mathrm{F}(\mathbf{x}) - \mathrm{Net}(\mathbf{x}, \mathbf{w})\right)^2 dx_1 \ldots dx_N$$

- The **theorem** says that

$$\underset{K\to\infty}{\mathrm{l.i.m.}}\, \mathbf{w}_{\mathrm{opt}} = \mathbf{w}^{(K)}_{\mathrm{opt}}$$

$$\lim_{K\to\infty} R_{emp}(\mathbf{w}) = R_{th}(\mathbf{w})$$

$$\lim_{K\to\infty} \frac{1}{K}\sum_{k=1}^{K}\left(d_k - Net(\mathbf{x}_k, \mathbf{w})\right)^2 = \int \underset{X}{\cdots} \int \left(\mathrm{F}(\mathbf{x}) - \mathrm{Net}(\mathbf{x}, \mathbf{w})\right)^2 dx_1 \ldots dx_N$$

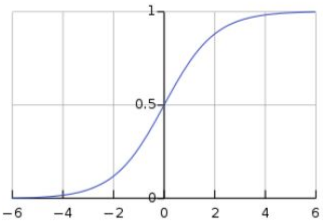
Various activation functions and their properties

- **Activation function**: shapes the output signal
  - **Non-linear function**
  - Typically **clamps the output**
  - **Monotonic increasing**
  - **Differentiable, or at least continuous**
- Originally it was the **step function, but later the needs went higher.**
- **Strong nonlinearities** to support approximation of wide range of functions
- To drive individual neurons in the hidden layers to a parameter zone, where they are silence for a set of vectors, and active for a different set.
- **Letting gradient go through them**
- **Work well with a loss function (select them synchrony)**

Sigmoid

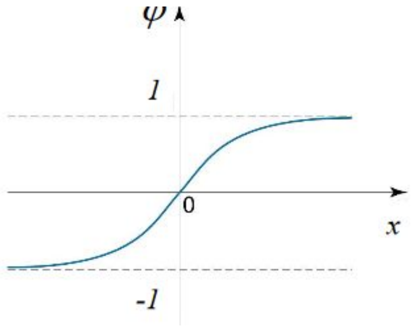
- Continuous and continuously differentiable
- Used in the **output layer of a fully connected (mostly in probability problems)**

$$S(x) = \frac{1}{1+e^{-x}}$$



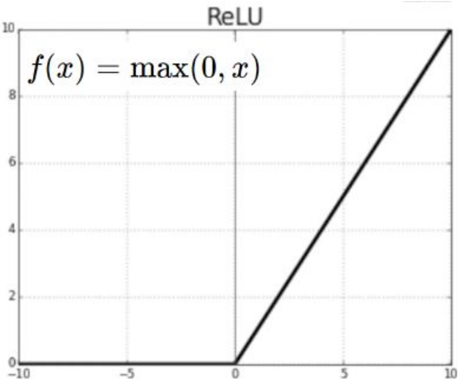
Tanh

- **Bipolar activation function** (useful, when **bipolar output** is expected)
- Continuous and continuously differentiable
- Used in the **output layer of a fully connected**



ReLU

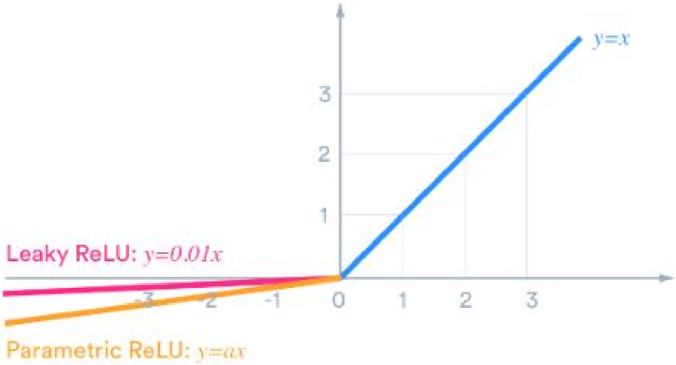
- **Rectified Linear Unit (ReLU)**
- Most commonly used nonlinearity **in hidden layers** of deep neural networks



Leaky ReLU

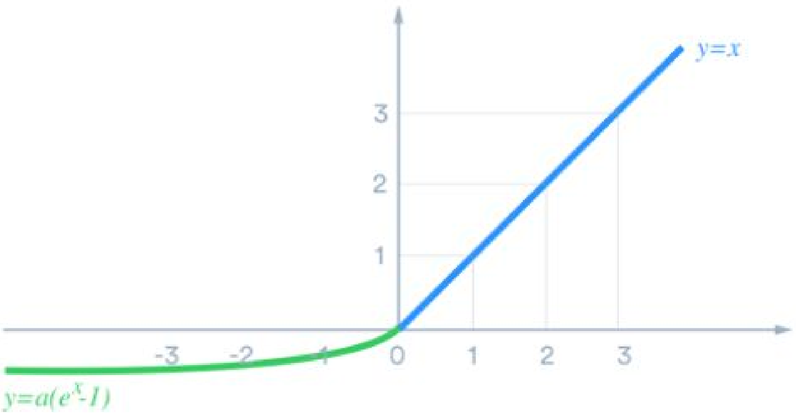
- **No constant zero output, so neurons won't die**
- Leaky ReLUs are **not necessarily superior than normal ones.**

$$f(x) = max(x, ax)$$



ELU

- **Exponential Linear Units**
- Variation of leaky ReLU
- **Better classification accuracy, but requires more computations**
- a is a **hyperparameter: tuned during training**

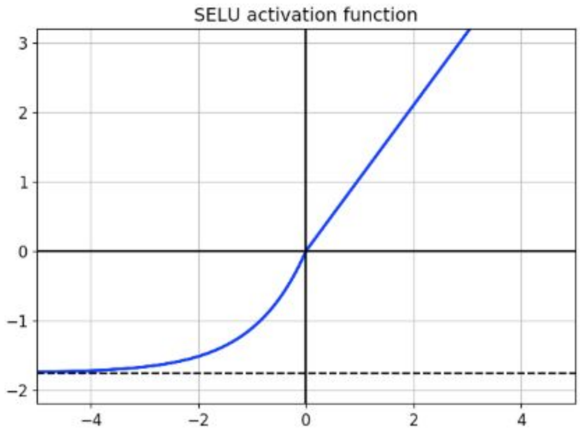


$$f(x) = \begin{cases} x & \text{if } x \geq 0 \\ a(e^x - 1) & \text{otherwise} \end{cases}$$

a is a hyper-parameter to be tuned and $a \geq 0$ is a constraint.

SELU

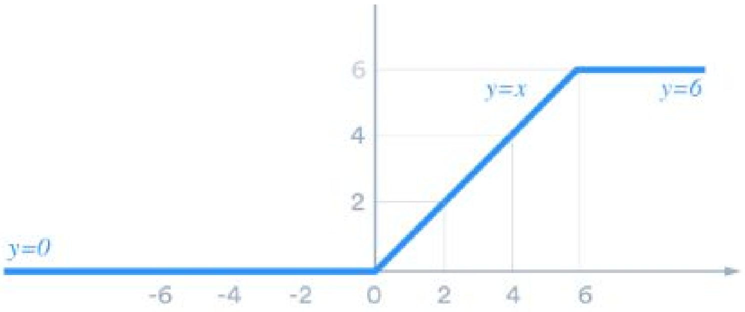
- **Scaled Exponential Linear Units**
- **Variation of leaky ELU**



$$\text{selu}(x) = \lambda \begin{cases} x & \text{if } x > 0 \\ \alpha e^x - \alpha & \text{if } x \leqslant 0 \end{cases}$$

ReLU6

- **Learn sparse features faster**

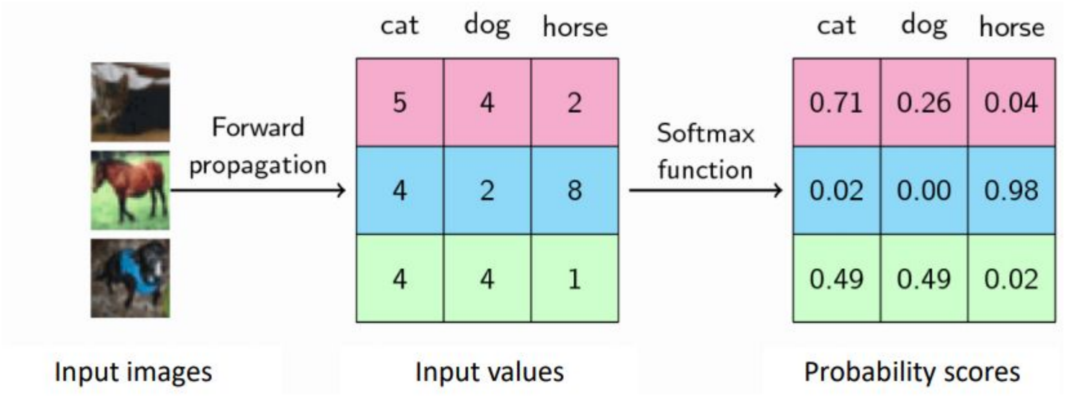


Softmax

- **Activation function**
- Normalized exponential functions of the output unit
- Softmax combines a layer of output neurons

- **Probability distribution of n class**
- Properties
    - **Squashes a vector of size n between 0 and 1**
    - Improves interpretability
    - **Generalization of sigmoid function for one-of-n class**
    - **Exponential function strongly penalize the non-winners**

$$y_i = softmax(v)_i = \frac{e^{v_i}}{\sum_{j=1}^{n} e^{v_j}}, \quad \text{where } v = w^T x$$

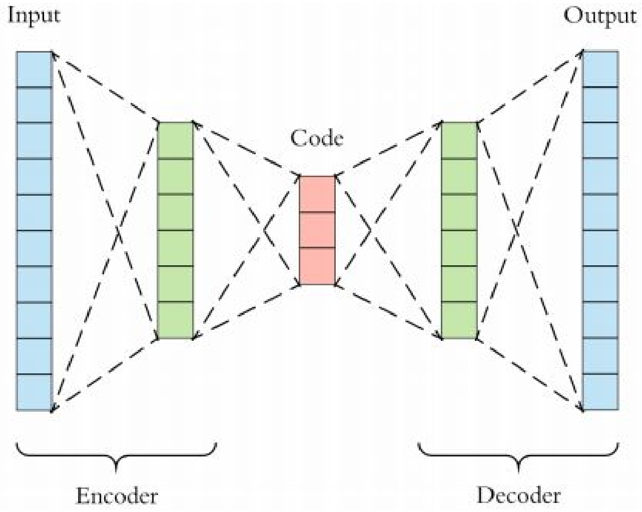


cat	dog	horse
5	4	2
4	2	8
4	4	1

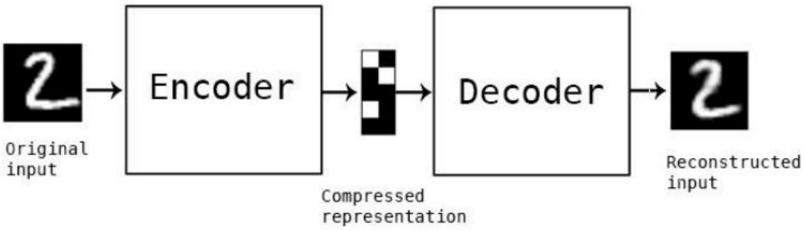
cat	dog	horse
0.71	0.26	0.04
0.02	0.00	0.98
0.49	0.49	0.02

Autoencoders

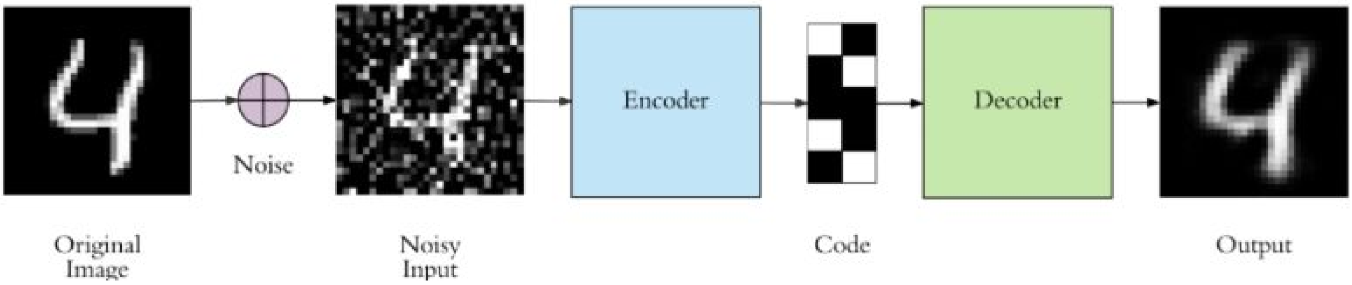
- **Neural network used for efficient data coding**
- Uses the same vector for the input and the output
    - **No labelled data set is needed**
    - **Unsupervised learning**
- Two parts:
    - **Encoder**: reduces data dimension
    - **Decoder**: reconstructs the data
    - **Middle layer**: code



- **The network is trained with the same input-output pairs.**
- **Loss function: MSE / Cross Entropy**
- **After the network is trained, remove decoder part**



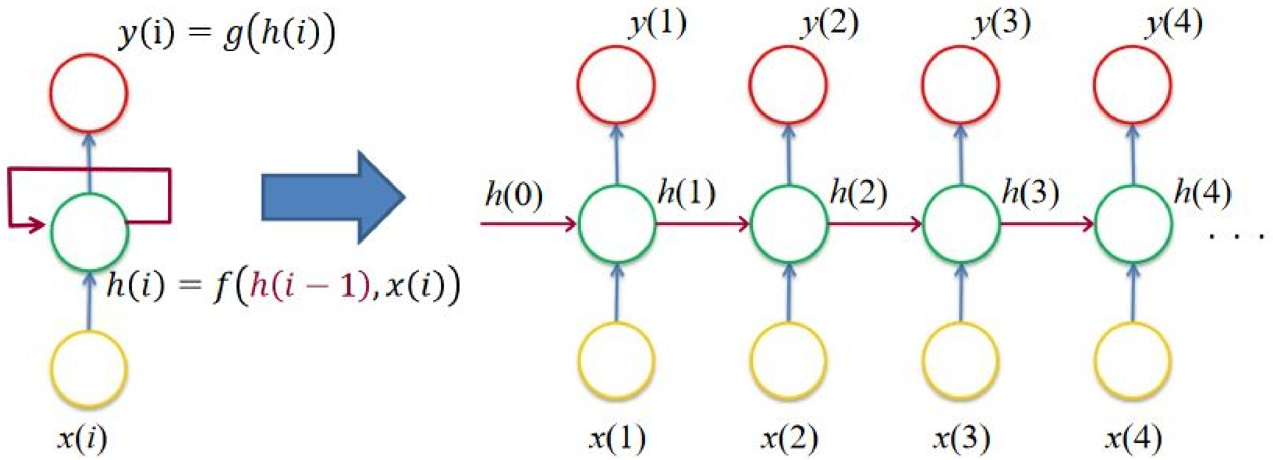
- We say that the autoencoder is **undercomplete** if the **width (dimension) of hidden layer is smaller than width of the input / output layer.**
- Can be used for **denoising** (add noise to the input, the output will be cleaner)



Graph unrolling and parameter sharing in recurrent neural networks

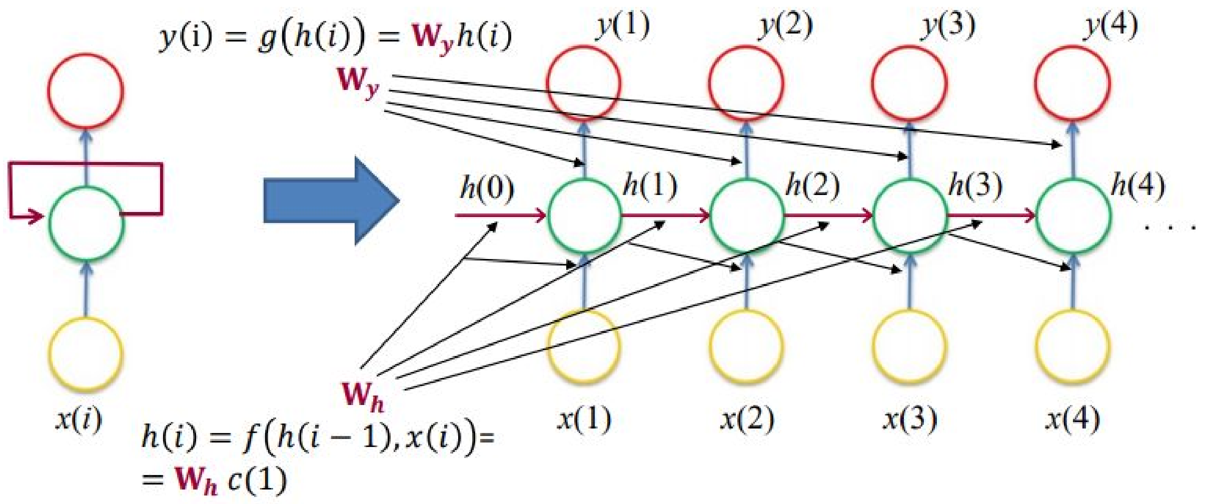
Graph unrolling

- **RNN-hez kapcsolódik**
- **Unrolling generates an acyclic directed graph from the original graph structure**
- It generates a **FIR** filter **from** the original **IIR** filter



Parameter sharing

- **RNN re-uses the same weight matrix in every unrolled steps.**
- We use it to **reduce the number of parameters that the model has to learn.**
- An example:
  - Compare **“Yesterday I ate an apple”** and **“I ate an apple yesterday”.** These two sentences mean the **same, but the “I ate an apple” part occurs on different time steps**. By **sharing parameters, you only have to learn what that part means once**. Otherwise you’d have to learn it for every time step, where it could occur in your model.



MobileNet

- **Special network architecture**, where **feature depths are squeezed before each operation**
- In a squeezed architecture we will use downscale 128 feature maps to 16, using linear combination (1x1 convolution)
- After the 3x3 convolutions we expanded back to 128 layers by 1x1 convolution again.
- From linear combination of these elements the new maps are created.
- It uses **depthwise separable convolutions which basically means it performs a single convolution on each colour channel rather than combining all three and flattening it**. This has the effect of filtering the input channels
- **reducing computation and model size**
- **It is also very low maintenance thus performing quite well with high speed.**

https://towardsdatascience.com/transfer-learning-using-mobilenet-and-keras-c75daf7ff299

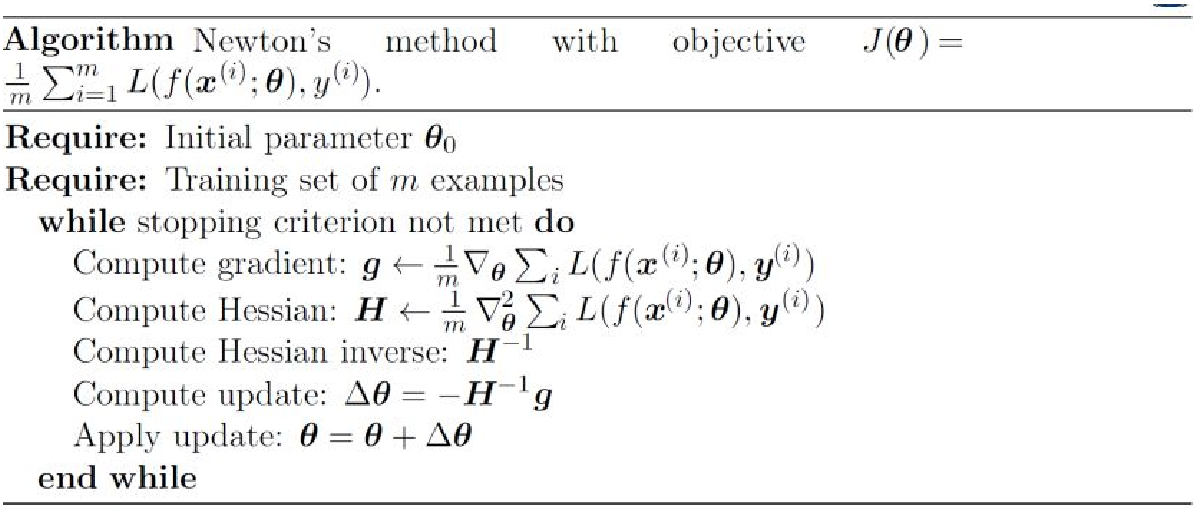
6. topic

Machine learning problem definition

- Itt én ugyanazt mondanám el, mint a Machine learning vs traditional programming-nál írtam a 3. topicban.

Newton optimizer

- Topic 3-nál a Newton optimization methodnál leírtam a lényeget. Még annyi, hogy, van egy csúnya algoritmus hozzá, aminek itt a kódja, de szerintem senki se tanulja meg mert nagyon félelmetes:



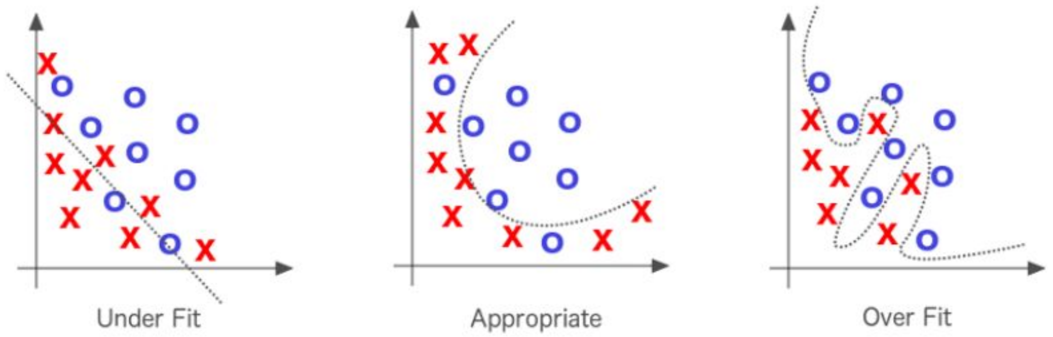
```
Algorithm Newton's method with objective $J(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m} L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}), y^{(i)})$.
Require: Initial parameter $\boldsymbol{\theta}_0$
Require: Training set of $m$ examples
  while stopping criterion not met do
    Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Compute Hessian: $\boldsymbol{H} \leftarrow \frac{1}{m}\nabla_{\boldsymbol{\theta}}^2 \sum_i L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Compute Hessian inverse: $\boldsymbol{H}^{-1}$
    Compute update: $\Delta\boldsymbol{\theta} = -\boldsymbol{H}^{-1}\boldsymbol{g}$
    Apply update: $\boldsymbol{\theta} = \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
  end while
```

- **Typically not used due to the computational complexity**
- **Parameter space much higher than first order (where it is already very high)**

Effects and relationship of model capacity and complexity - overfitting, underfitting

Overfitting and underfitting

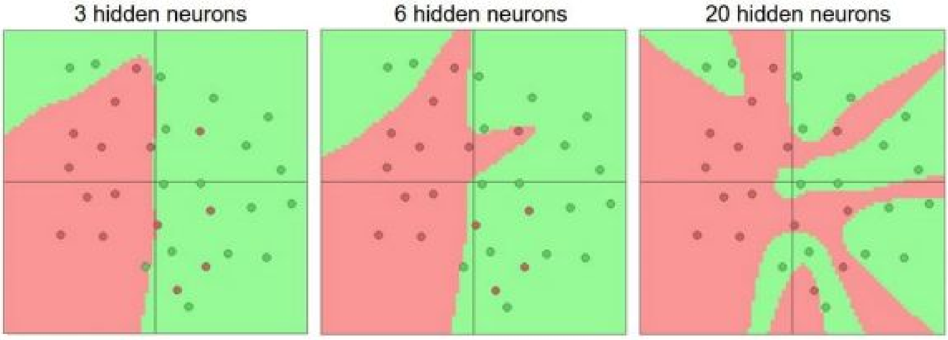
- **The network exactly learned the training vectors, but lost the generalization capabilities.**
- It's the **global minimum**
- Overfitting occurs, when a model with high capacity **fits the noise in the data instead of the (assumed) underlying relationship.**



- Underfitting occurs when a statistical model or machine learning algorithm cannot adequately capture the underlying structure of the data. It occurs when the model or algorithm does not fit the data enough. **It is often a result of an excessively simple model.**

Capacity and complexity

- **In general, the more layers we have, the more neurons there are, the larger the capacity.**
- There is no adequate method, to predict the required complexity.
- **Even if a network is capable to learn a task, it is not guaranteed, that it will.**



https://towardsdatascience.com/overfitting-vs-underfitting-ddc80c2fc00d

How to increase complexity in a smart way?

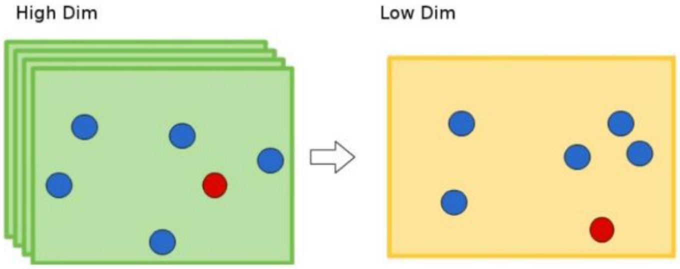
- **Increase the number of hidden layers?**
  - **Number of free parameters exploding**
  - **Numerical problems** arises after using too many layers
- **Solution**: hierarchical architecture with reusable components (**Residual networks**)

t-Distributed Stochastic Neighbor Embedding

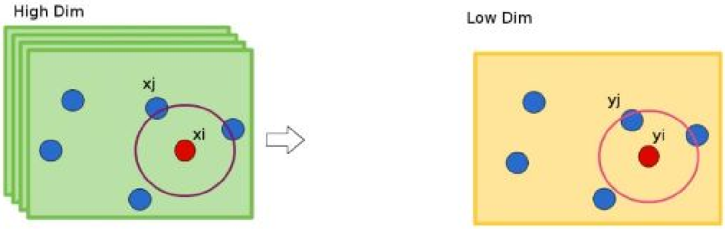
- **t-SNE**
- **Generates a low dimensional representation of the high dimensional data set iteratively**
- Aims to minimize the divergence between two distributions:
  - **Pairwise similarity of the points in the higher-dimensional space**
  - **Pairwise similarity of the points in the lower-dimensional space**
- Output: **original points mapped to a 2D or 3D space**
  - **similar objects are modeled by nearby points and**
  - **dissimilar objects are modeled by distant points with high probability**
- Unlike PCA it is **stochastic** (probabilistic)

t-SNE implementation

1. **Generate the points in the low dimensional data set (2D or 3D)**
   - Random initialization
   - First two or three components of PCA



2. **Calculate the pairwise similarities measures between data pairs (probability measure).**



The similarity of datapoint x_j to datapoint x_i means the conditional probability p_{ji} that x_i would pick x_j as its nearest neighbor.

$$p_{ij} = \frac{exp(-||x_i - x_j||^2/2\sigma^2)}{\sum_{k \neq l} exp(-||x_l - x_k||^2/2\sigma^2)} \qquad q_{ij} = \frac{(1 + ||y_i - y_j||^2)^{-1}}{\sum_{k \neq l}(1 + ||y_k - y_l||^2)^{-1}}$$

__Curse of dimensionality__: exponential normalization of the Euclidean distances are needed due to the high dimensionality.

3. **Define the cost function**
(Kullback-Leibler divergence of two distr.)
(előzőből: Similarity of data points in high dimension (p), similarity in low dimension (q))
Large p_ij and small q_ij -> large penalty
Large p_ij and large q_ij -> small penalty

$$C = KL(P||Q) = \sum_i \sum_j p_{ij} log \frac{p_{ij}}{q_{ij}}$$

4. **Minimize the cost function using gradient descent**
Optimization with momentum method
Form of gradient:

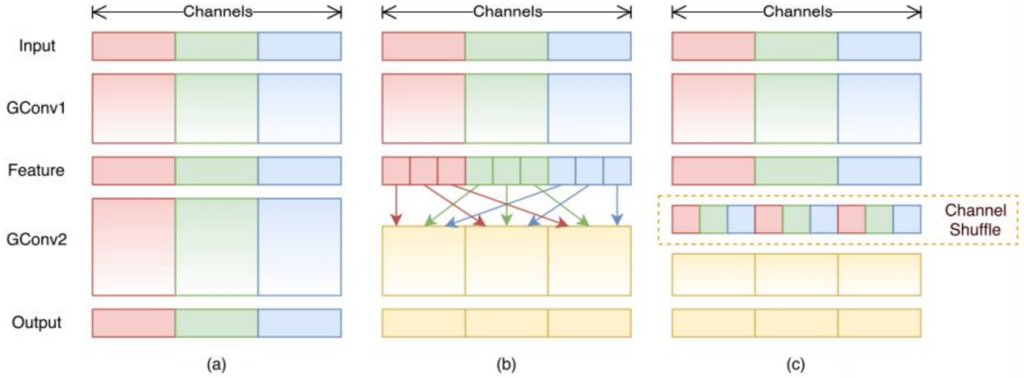
$$\frac{\partial C}{\partial y_i} = 4 \sum_{j \neq i} (p_{ij} - q_{ij})(1 + ||y_i - y_j||^2)^{-1}(y_i - y_j)$$

Physical analogy

- Map **points are all connected with springs in the low dimensional data map.**
- **Stiffness depends on p_i|j - q_i|j**
- **Let the system evolve according to the laws of physics**
  - If two map points are are apart while the data points are close, they are attracted together
  - If they are nearby while data points are dissimilar they are repelled

ShuffleNet

- **Extremely computation efficient CNN architecture**
- To overcome the side effects of group convolutions: **shuffle operation**
- Enables **more feature map channels, which is critical for performance**



7. topic

Credit approval problem

- Ezt nem teljesen tudom, de ez az valószínűleg:
- The last decade has seen an important rise of data gathering, especially in the financial sectors.
- Gathering and analyzing this data is a key feature for **decision making**, particularly **in banking sector**.
- One of the most important and frequent decision banks has to make, is **loan approval.**
- The challenge is to know **how to build a proactive, powerful, responsible and ethical exploitation of personal data, to make loan applicant proposals more relevant and personalized.**
- Machine learning is a **promising solution** to deal with this problem. Therefore, in the last years, many algorithms based on **machine learning** have been proposed to solve **loan approval issue**.

Objective functions in neural networks

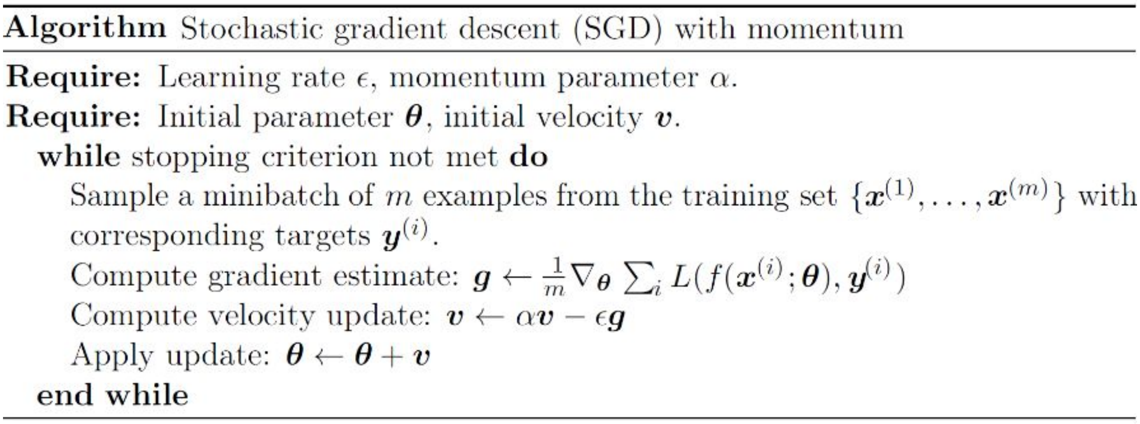
- **Objective function** or **error function, cost function, loss function, criterion** is a special function that **we have to minimize for a neural network by modifying its parameters**. (optimization)
- The loss is calculated from the **actual value and the predicted value by the network.**
- **Tells the net the size of the error, and penalize according to it.**
- 3-as topicban már beszéltünk róla milyen lossok vannak.
- **Optimization:**
  - Find the optimal weights:

$$\mathbf{w}_{opt} = min\, f(\mathbf{x}, \mathbf{d}, Net(\mathbf{x},\mathbf{w}))$$

  - Stochastic Gradient Descent method (there are more advanced ones)

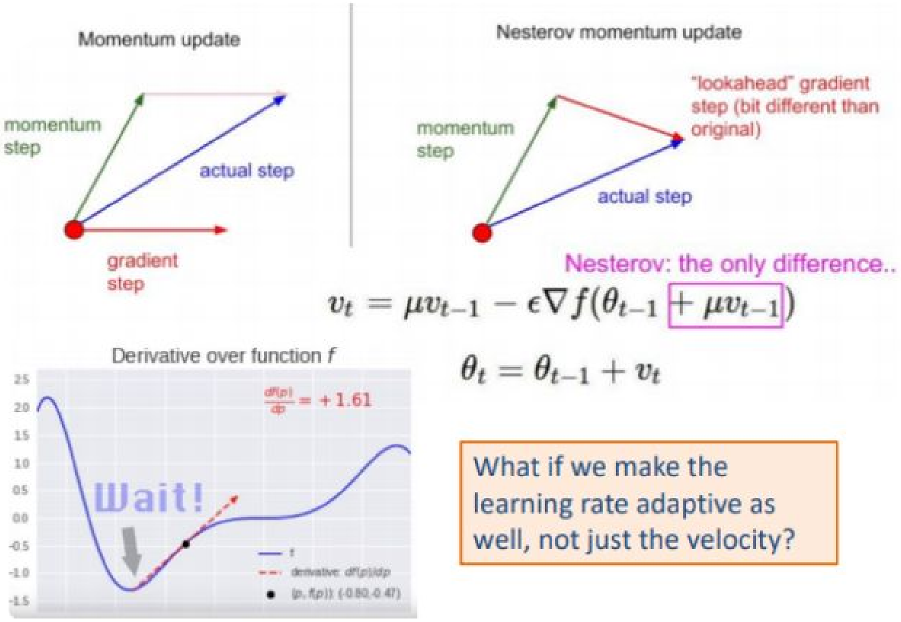
Nesterov momentum optimizer

- Stimulate a unity weight mass, having v velocity (follow Newton’s laws of dynamics) (Momentum)



```
Algorithm Stochastic gradient descent (SGD) with momentum
Require: Learning rate $\epsilon$, momentum parameter $\alpha$.
Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$.
  while stopping criterion not met do
    Sample a minibatch of $m$ examples from the training set $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ with
    corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient estimate: $\boldsymbol{g} \leftarrow \frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Compute velocity update: $\boldsymbol{v} \leftarrow \alpha\boldsymbol{v} - \epsilon\boldsymbol{g}$
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$
  end while
```

- **Nesterov momentum update**
  - **Calculates gradient not in the current point, but in the next**, and correct the velocity with the gradient over there (look ahead function)
  - It does not runs through a minimum, **because if there is a hill behind a minimum, than it starts decreasing the speed in time.**
  - **Learning rate is changing but not adaptive.**



Nesterov: the only difference..

$$v_t = \mu v_{t-1} - \epsilon \nabla f(\theta_{t-1} \boxed{+ \mu v_{t-1}})$$

$$\theta_t = \theta_{t-1} + v_t$$

What if we make the learning rate adaptive as well, not just the velocity?

Decomposition of large kernels

- **Convolution is associative**

$$f * (g * h) = (f * g) * h$$

- **Reduce the computational complexity**

$$\begin{bmatrix} 0.02 & 0.09 & 0.2 & 0.3 & 0.2 & 0.09 & 0.02 \\ 0.09 & 0.13 & 0.11 & 0.4 & 0.11 & 0.13 & 0.09 \\ 0.2 & 0.11 & -0.3 & -0.7 & -0.3 & 0.11 & 0.2 \\ 0.3 & 0.4 & -0.7 & -1.3 & -0.7 & 0.4 & 0.3 \\ 0.2 & 0.11 & -0.3 & -0.7 & -0.3 & 0.11 & 0.2 \\ 0.09 & 0.13 & 0.11 & 0.4 & 0.11 & 0.13 & 0.09 \\ 0.02 & 0.09 & 0.2 & 0.3 & 0.2 & 0.09 & 0.02 \end{bmatrix} = \begin{bmatrix} 0.2 & 0.5 & 0.2 \\ 0.5 & -3.1 & 0.5 \\ 0.2 & 0.5 & 0.2 \end{bmatrix} * \begin{bmatrix} 0 & 0.2 & 0.3 & 0.2 & 0 \\ 0.2 & 0.6 & 0.8 & 0.6 & 0.2 \\ 0.3 & 0.8 & 1.2 & 0.8 & 0.3 \\ 0.2 & 0.6 & 0.8 & 0.6 & 0.2 \\ 0 & 0.2 & 0.3 & 0.2 & 0 \end{bmatrix}$$

Laplacian of Gaussian kernel $(g * h)$ Laplacian (g) Gaussian kernel (h)

Number of operations: $49 * N_{pix}$ $9 * N_{pix} + 25 * N_{pix} = 34 * N_{pix}$

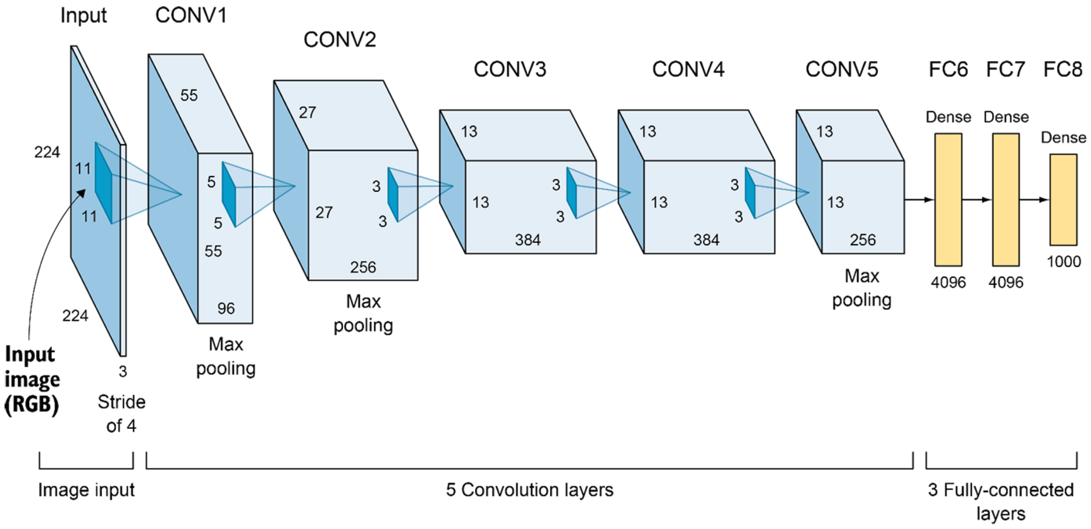
15% reduction of computational demand!!!

- **Not exact in most cases.** (approximate the kernels with a limited accuracy only)
- **Neural nets** however does **not sensitive for inaccurate decomposition.**
- Decomposition of larger kernels leads to **higher savings, so its widely used.**

Alexnet + ILSVRC

- **First fully trained deep (8 layers) convolutional neural network**
- **Motivation**: build **deeper network, that can learn more complex function**
- Built from **convolutional and max pooling layers, ReLUs, dropout layers, data augmentation.**
- **ILSVRC: ImageNet Large Scale Visual Recognition Challange**
  - **ImageNet: 15+ million labeled, high-resolution images in 22000 categories**
  - **ILSVRC uses a subset of imagenet**: **1000 category, ~1000 images per category**
  - **Each image should be classified.**

  - We see a **rapid decrease in classification errors since deep CNN-based designs became popular**
- Architecture: (ezt nyilván nem kell, csak hogy kb milyen layerek vannak benne, azért tettem ide)



8. topic

Delta learning rule

- If

$$\frac{\partial R_{emp}}{\partial w_{kj}} < 0$$

  than we have to **increase** w_{kj} , to get closer to the minimum

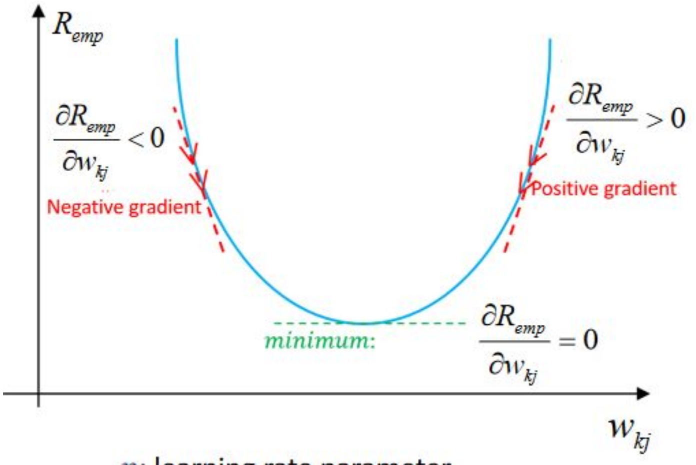
$$\Delta w_{kj} = -\eta \frac{\partial R_{emp}}{\partial w_{kj}}$$

- If

$$\frac{\partial R_{emp}}{\partial w_{kj}} > 0$$

  than we have to **decrease** w_{kj} , to get closer to the minimum

$$\Delta w_{kj} = -\eta \frac{\partial R_{emp}}{\partial w_{kj}}$$


R_{emp}
$\frac{\partial R_{emp}}{\partial w_{kj}} < 0$
Negative gradient
$\frac{\partial R_{emp}}{\partial w_{kj}} > 0$
Positive gradient
minimum:
$\frac{\partial R_{emp}}{\partial w_{kj}} = 0$
w_{kj}

η: learning rate parameter

Batch normalization

Batch normalization

- **Why?**
  - Distribution of the input vectors changes from layer to layer
    - First layer got normalized output, then the second layer somewhat shifts and twists on this norm etc.
    - **Data propagating through the layers will lose its normalized properties (covariance shift)**
    - This can shift the neuron out of its zero-centered position
  - Solution?: **Normalization on each layers**
  - **Noise to avoid local minima and overfitting**
- **Layer level**

- **Training**: done on **minibatch level**
- **Inferencing: do the normalization with the precalculated parameters of the entire training set.**
- Batch normalization is differentiable via chain rule: **back propagation can be be applied for batch normalized layers.**
- **Rewriting the normalization using probability terms:**

$$\hat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{Var[x^{(k)}]}} \qquad \begin{array}{l} \mathrm{E}\text{: } \textit{the expectation} \\ Var\text{: } \textit{the variance} \end{array}$$

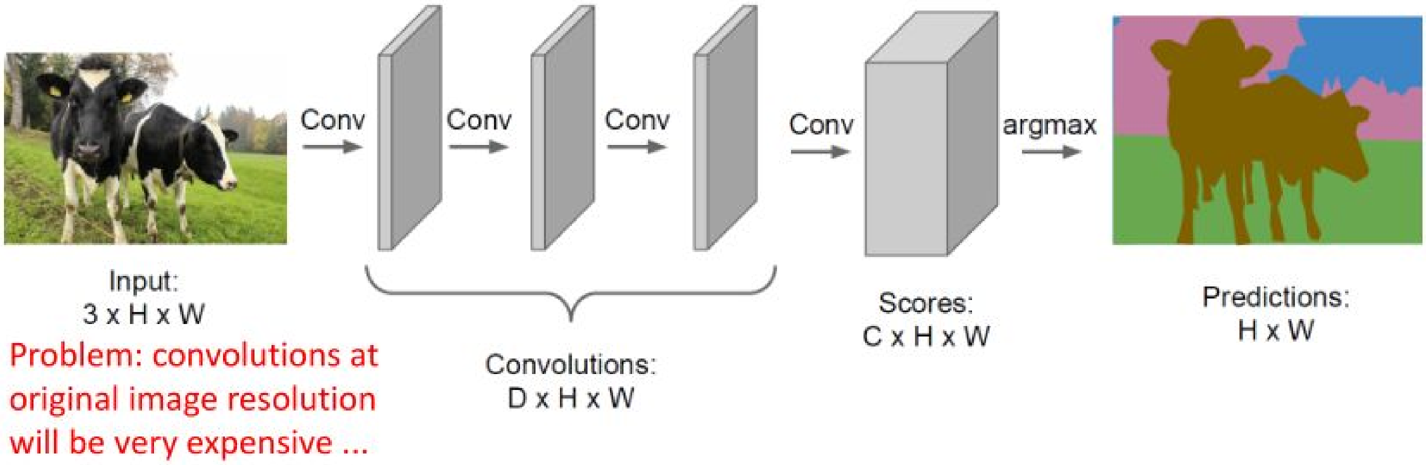
Local response normalization vs batch norm.

- **Both work within the convolutional layer**
- **Local response normalization**
  - Normalization either **through the feature maps or within one feature maps**
  - Normalization is done for **one input image**
- **Batch normalization**
  - Normalization done **for all pixels in all the feature maps within a layer**
  - Normalization is done for the **entire batch**

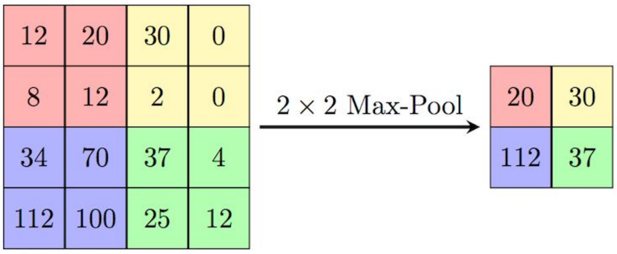
https://www.youtube.com/watch?v=dXB-KQYkzNU

Transposed convolution, atrous convolution

- Oké én itt bevezetésnek elmondanám, hogy itt eleve arról volt szó, hogyha te éppen képszegmentálást akarsz csinálni, akkor van ez a vágyad ugye, hogy az outputod akkora mint az inputod, csak a külön objektumok más színűek. Erre a legtriviálisabb megoldás ugye, ha csinálsz egy fully convolutional hálót az alábbiak szerint:



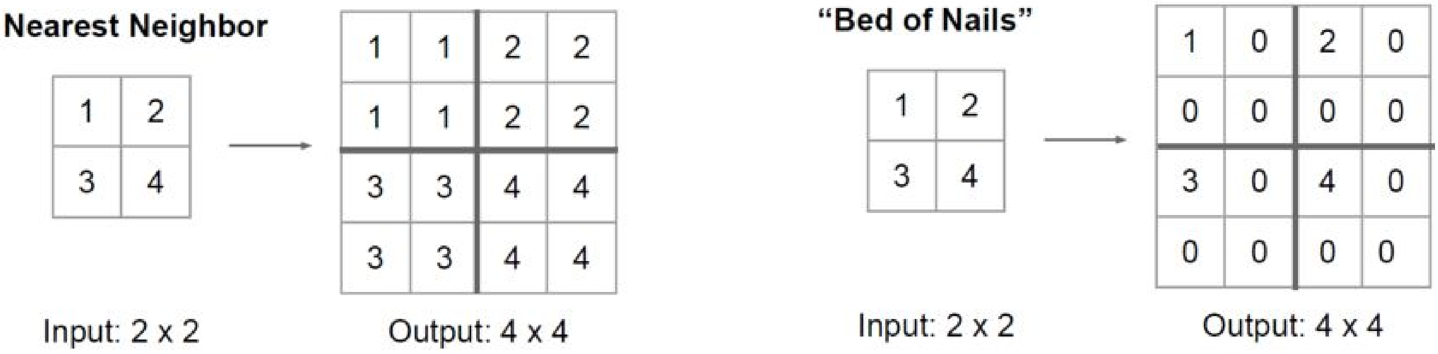
- De ahogy a mellékelt ábra mutatja ez elég kaksi, mert nagyon nagyon drága így végezni a műveleteket, ezért inkább csinálunk ilyen downsampling meg upsampling részt.
- És akkor azt tudjuk hogy downsamplingnél lehet ugye maxpoolingolni vagy average poolingolni.
- A maxpooling ugye ilyen a lényeget értjük: nagy mátrixból picit csinál, reménykedve benne hogy most megtartotta a fontos információt.

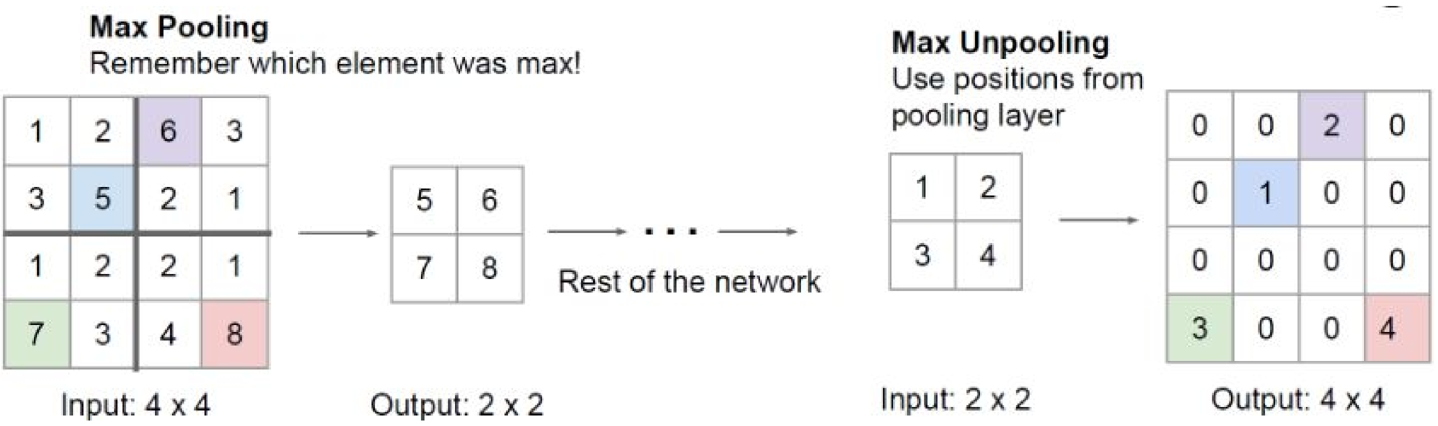


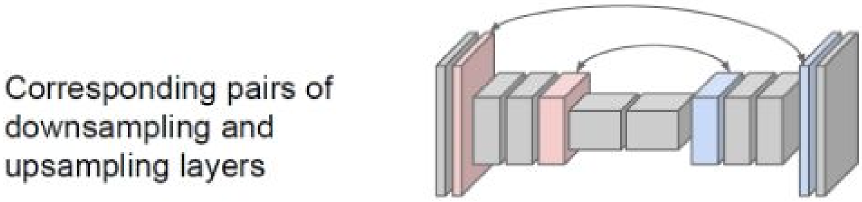
12	20	30	0
8	12	2	0
34	70	37	4
112	100	25	12

20	30
112	37

- Na de akkor ezt hogy lehetett felfelé csinálni? Hát úgy, hogy vannak ilyen unpooling dolgok:

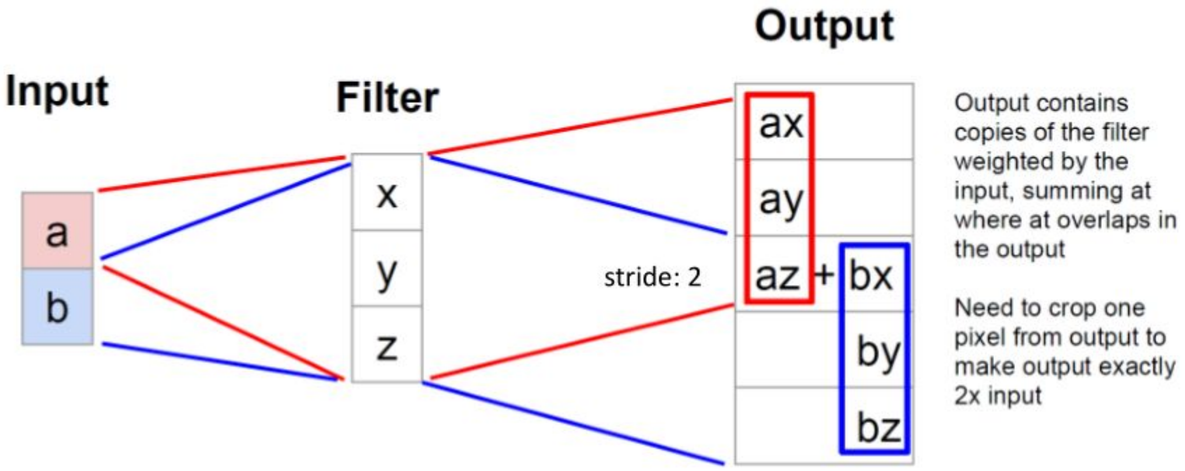




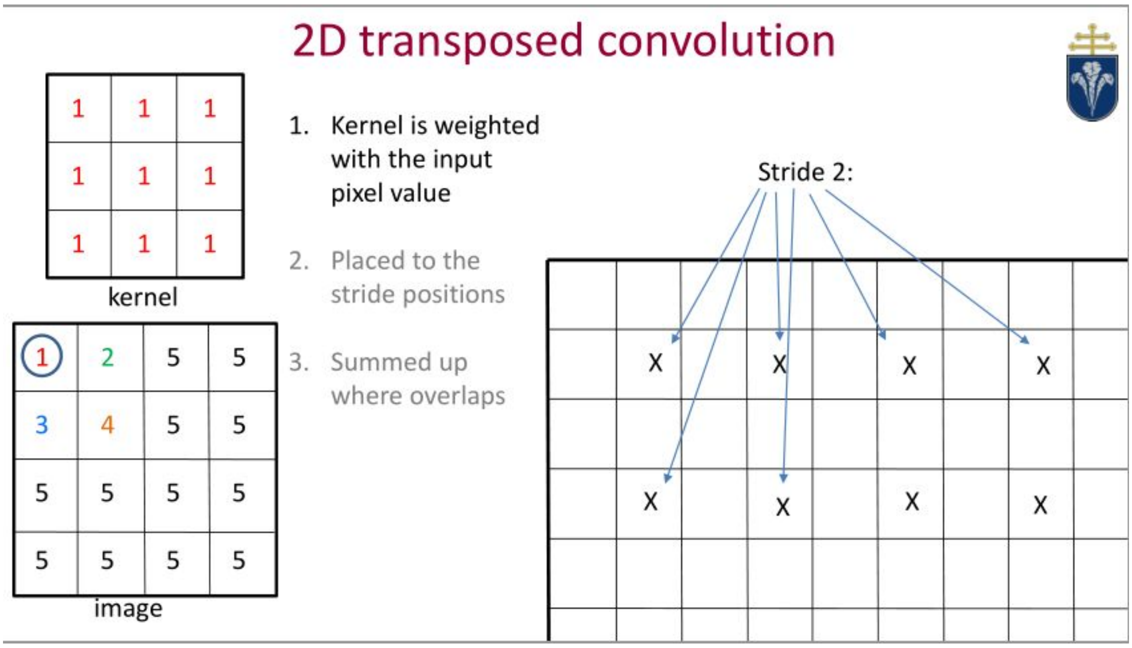


- De ugye tudjuk, hogy a lecsökkentős mágiára a convolution is képes, ha a strideokkal meg a paddinggal játszunk. Akkor tehát nem lehet egy olyan művelet, ami tulajdonképpen a konvolúció párjaként, hasonlóan vissza tudja tornázni a méreteket? De, és ez a transpose convolution vagy deconvolution. Mese vége.

1D example


Input
a
b
Filter
x
y
z
stride: 2
Output
ax
ay
az + bx
by
bz
Output contains copies of the filter weighted by the input, summing at where at overlaps in the output
Need to crop one pixel from output to make output exactly 2x input

2D example



2D transposed convolution

1	1	1
1	1	1
1	1	1

kernel

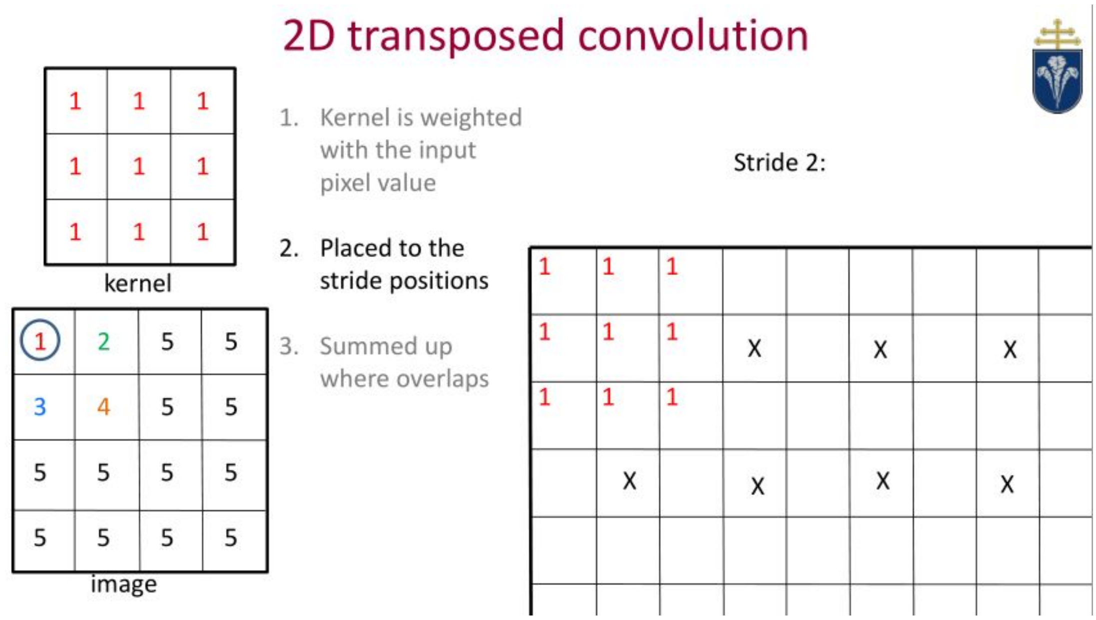
1	2	5	5
3	4	5	5
5	5	5	5
5	5	5	5

image

1. Kernel is weighted with the input pixel value
2. Placed to the stride positions
3. Summed up where overlaps

Stride 2:

	X		X		X		X	
	X		X		X		X	



2D transposed convolution

1	1	1
1	1	1
1	1	1

kernel

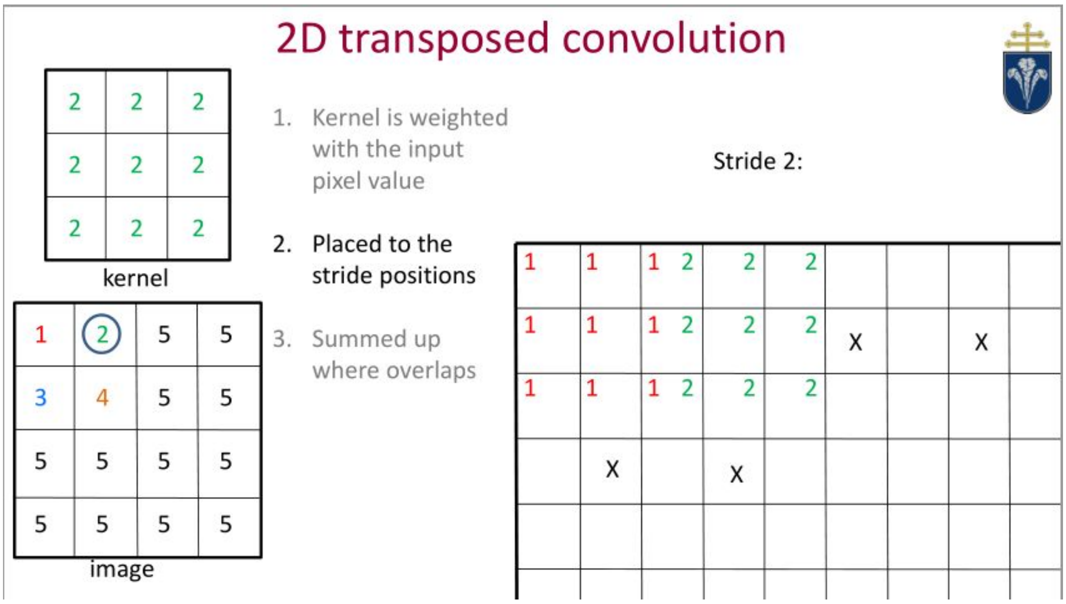
1	2	5	5
3	4	5	5
5	5	5	5
5	5	5	5

image

1. Kernel is weighted with the input pixel value
2. Placed to the stride positions
3. Summed up where overlaps

Stride 2:

1	1	1						
1	1	1	X		X		X	
1	1	1						
	X		X		X		X	



2D transposed convolution

2	2	2
2	2	2
2	2	2

kernel

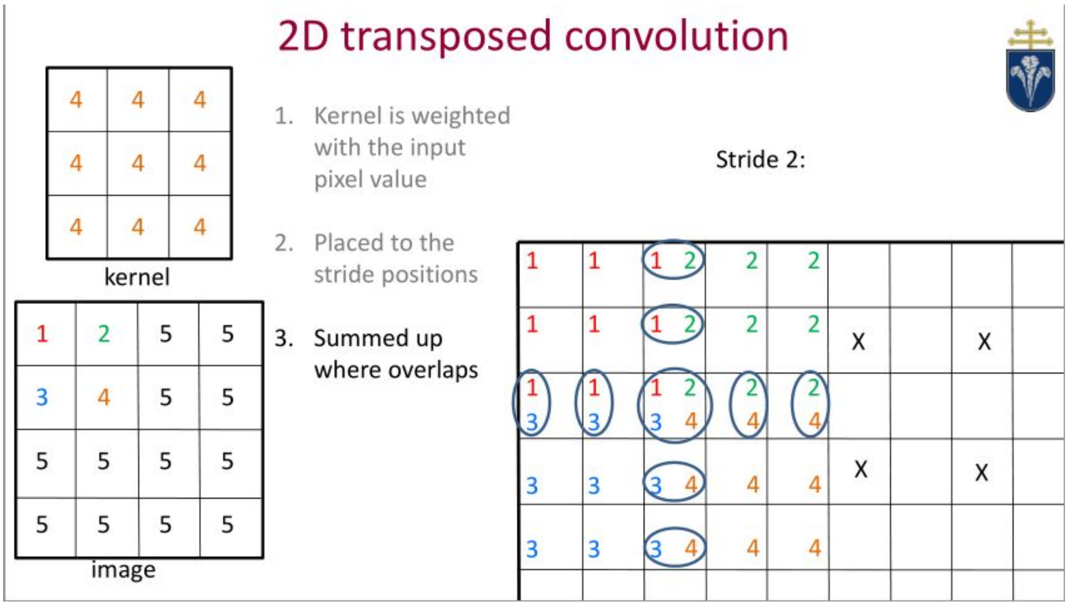
1	2	5	5
3	4	5	5
5	5	5	5
5	5	5	5

image

1. Kernel is weighted with the input pixel value
2. Placed to the stride positions
3. Summed up where overlaps

Stride 2:

1	1	1 2	2	2				
1	1	1 2	2	2	X		X	
1	1	1 2	2	2				
	X		X					



2D transposed convolution

kernel

4	4	4
4	4	4
4	4	4

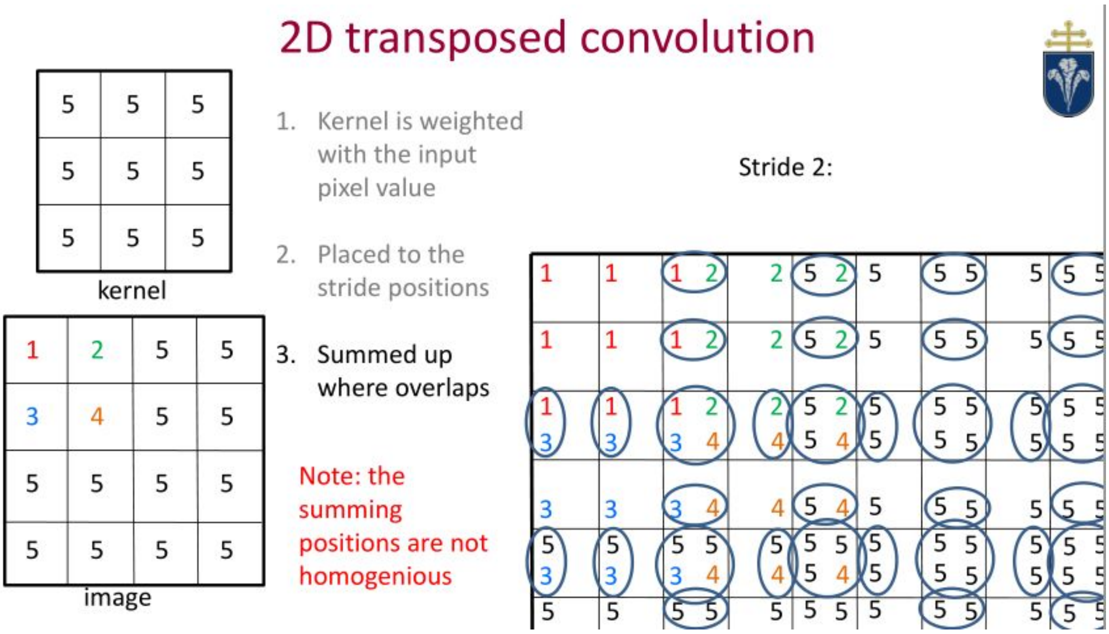
image

1	2	5	5
3	4	5	5
5	5	5	5
5	5	5	5

1. Kernel is weighted with the input pixel value
2. Placed to the stride positions
3. Summed up where overlaps

Stride 2:

1	1	1 2	2	2				
1	1	1 2	2	2	X		X	
1 3	1 3	1 2 3 4	2 4	2 4				
3	3	3 4	4	4	X		X	
3	3	3 4	4	4				



2D transposed convolution

kernel

5	5	5
5	5	5
5	5	5

image

1	2	5	5
3	4	5	5
5	5	5	5
5	5	5	5

1. Kernel is weighted with the input pixel value
2. Placed to the stride positions
3. Summed up where overlaps

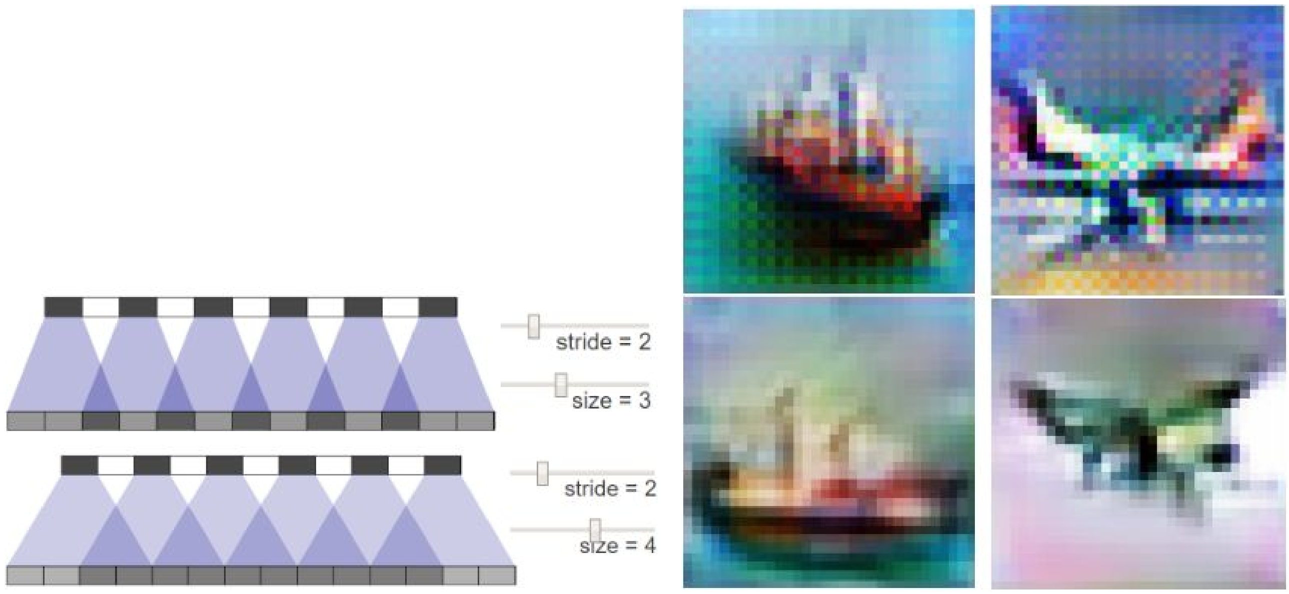
Note: the summing positions are not homogenious

Stride 2:

1	1	1 2	2	5 2	5	5 5	5	5 5
1	1	1 2	2	5 2	5	5 5	5	5 5
1 3	1 3	1 2 3 4	2 4	5 2 5 4	5 5	5 5 5 5	5 5	5 5 5 5
3	3	3 4	4	5 4	5	5 5	5	5 5
5 3	5 3	5 5 3 4	5 4	5 5 5 4	5 5	5 5 5 5	5 5	5 5 5 5
5	5	5 5	5	5 5	5	5 5	5	5 5

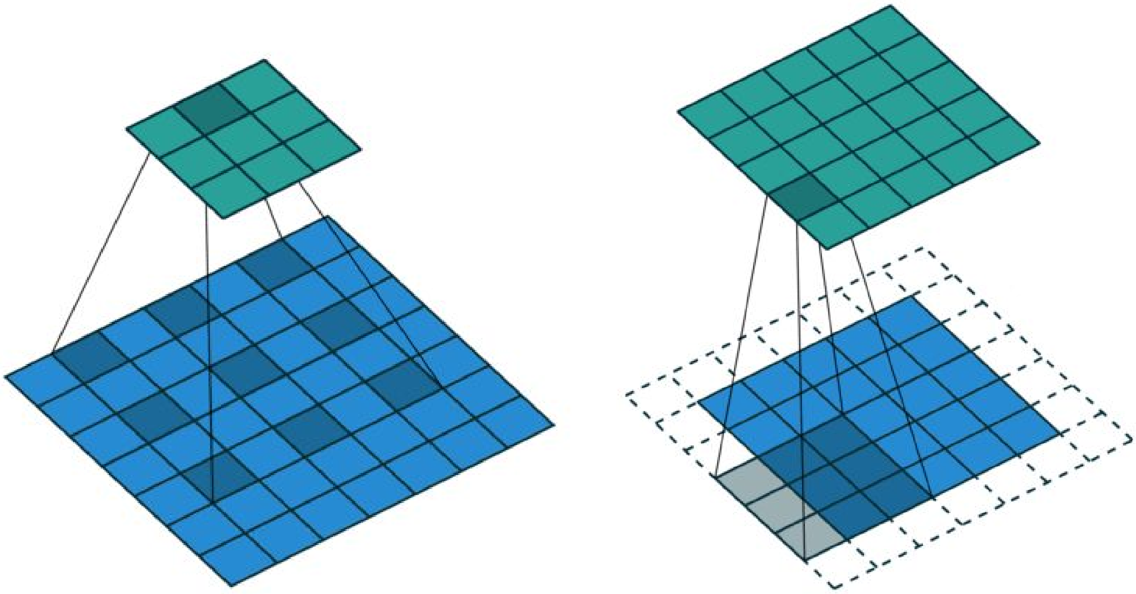
Avoiding checkerboard effect (Transpose convolution artefact)

- **Non-homogeneous transpose convolution causes checkerboard patterns.**
- Balanced stripe and kernel size can make it homogeneous.

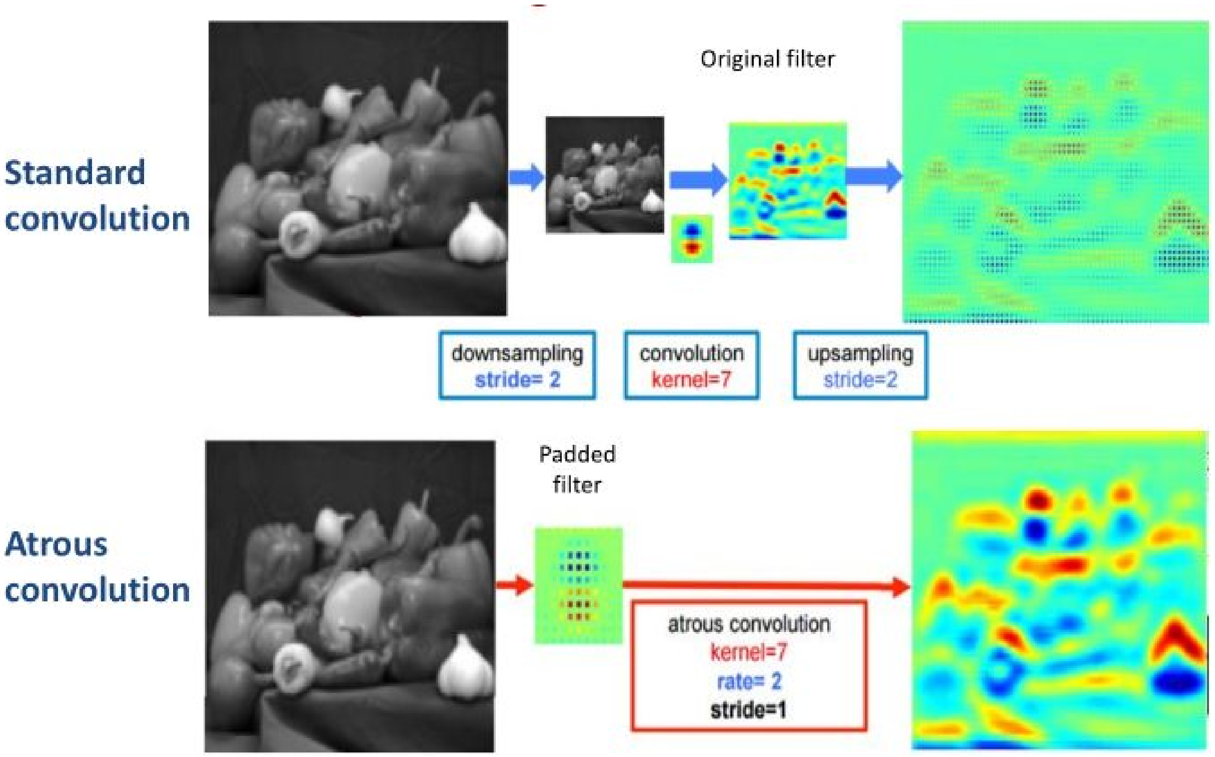


Atrous convolution

- How it works?
    - **Blows up the kernel**
    - **Filling up the holes with zeros**
        - **Atrous means very dark (like the holes between the values)**
- **Properties**
    - **Not** doing **downsampling**
    - **Not increasing computational load**
    - But reaches **larger neighborhood**
    - **Combines information from larger neighborhood**
- **Atrous convolution vs normal convolution:**



- **Normal convolution goes deeper with reducing resolution**
- **Atrous convolution goes deeper without further reducing resolution**
- **Visually:**



- This delivers a wider field of view at the same computational cost. **Dilated convolutions are particularly popular in the field of real-time segmentation. Use them if you need a wide field of view and cannot afford multiple convolutions or larger kernels.**

Filter size considerations

- **Small** field of view -> accurate **localization**
- **Large** field of view -> context **assimilation**
- Effective filter size increases

$$n_o\colon k \times k \quad \rightarrow \quad n_a\colon \big(k + (k-1)(r-1)\big) \times \big(k + (k-1)(r-1)\big)$$

n_o : original convolution kernel size

n_a : atrous convolution kernel size

r: rate

- However we take into account only the non-zero filter values:
    - Number of filter parameters is the same
    - Number of operations per position is the same

https://towardsdatascience.com/types-of-convolutions-in-deep-learning-717013397f4d

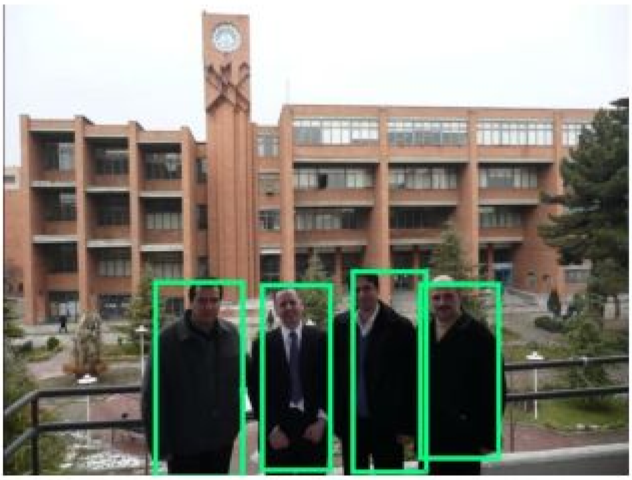
Object classification + localization vs. object detection vs. semantic segmentation vs. instance segmentation

https://medium.com/analytics-vidhya/image-classification-vs-object-detection-vs-image-segmentation-f36db85fe81

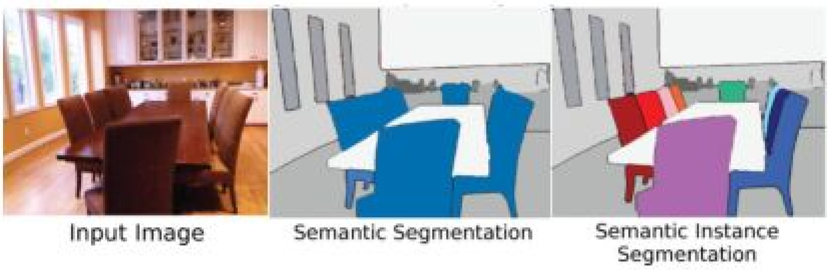
- **Object classification**: we make only one decision per image (what's on the picture)
    - eg. Alexnet

container ship

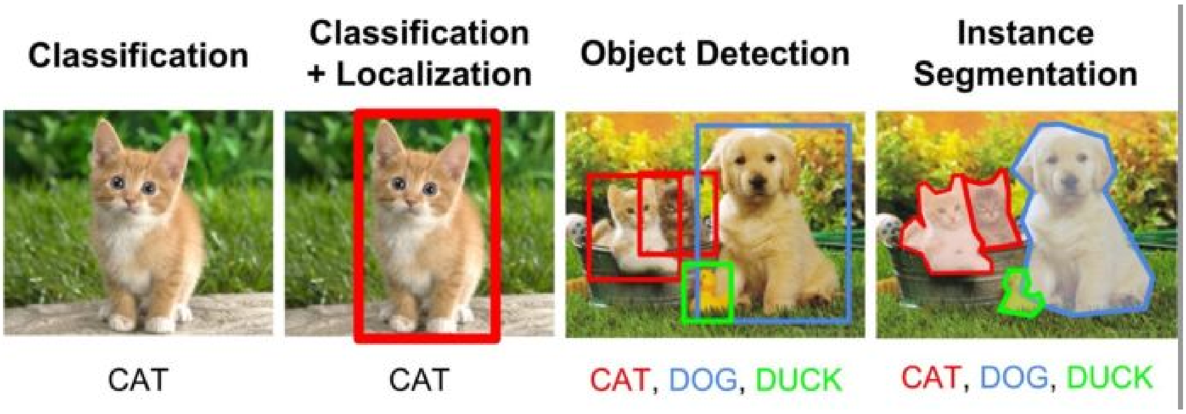
- **Detection and localization is more complex**: we make multiple decision per image (regressions for localization and classification for detection)
    - **PASCAL object recognition database and challenge**
        - Annotated image database
    - eg. R-CNN, Fast R-CNN, Faster R-CNN :D



- **Pixel level segmentation**: very high number of decisions (classification) per image
    - **Semantic segmentation**: label each pixel in the image with a category label
        - Use sliding window, fully convolutional aztán unpooling, transpose conv etc. (ebben a tételben volt róla szó feljebb)
    - **Semantic instance segmentation**: differentiate instances:
    - eg. U-net, DeConvNet, SegNet (atrous convolutiont használ)

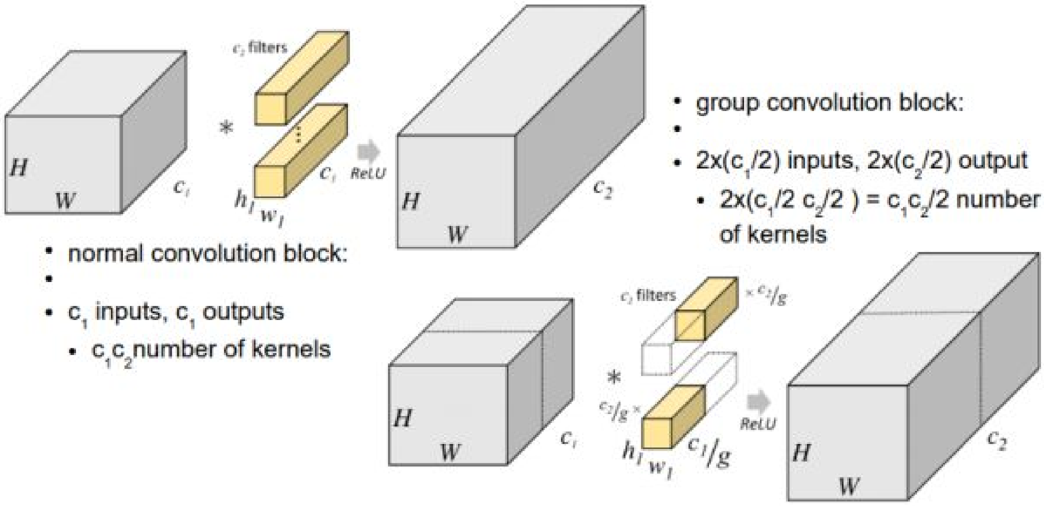


- **Chicken and egg problem:**
    - You need to **know that it is a bicycle before able to say** that both **a wheel part and a pipe segment belongs to the same object.**
    - You need to **know that the red box contains an object before you can recognize it. (cannot recognize a bicycle if you try it from separated parts)**
    - Our brain does it parallel
    - **How neural nets can solve it?**
        - **Detection by regression?**
            - Bounding boxes
            - Region proposals (find “blobby” image regions that are likely to contain objects
        - **Detection by classification?**



ResNext

- **ResNet kicsit pimpelve**
- The model name, ResNeXt, contains **Next**. **It means the next dimension, on top of the ResNet** . This next dimension is called the **“cardinality” dimension**. And ResNeXt becomes the 1st Runner Up of ILSVRC classification task.
- **Group convolution:**
  - Dividing feature maps into to groups and apply the convolutions to each groups separately
  - The number of convolutions will be halved



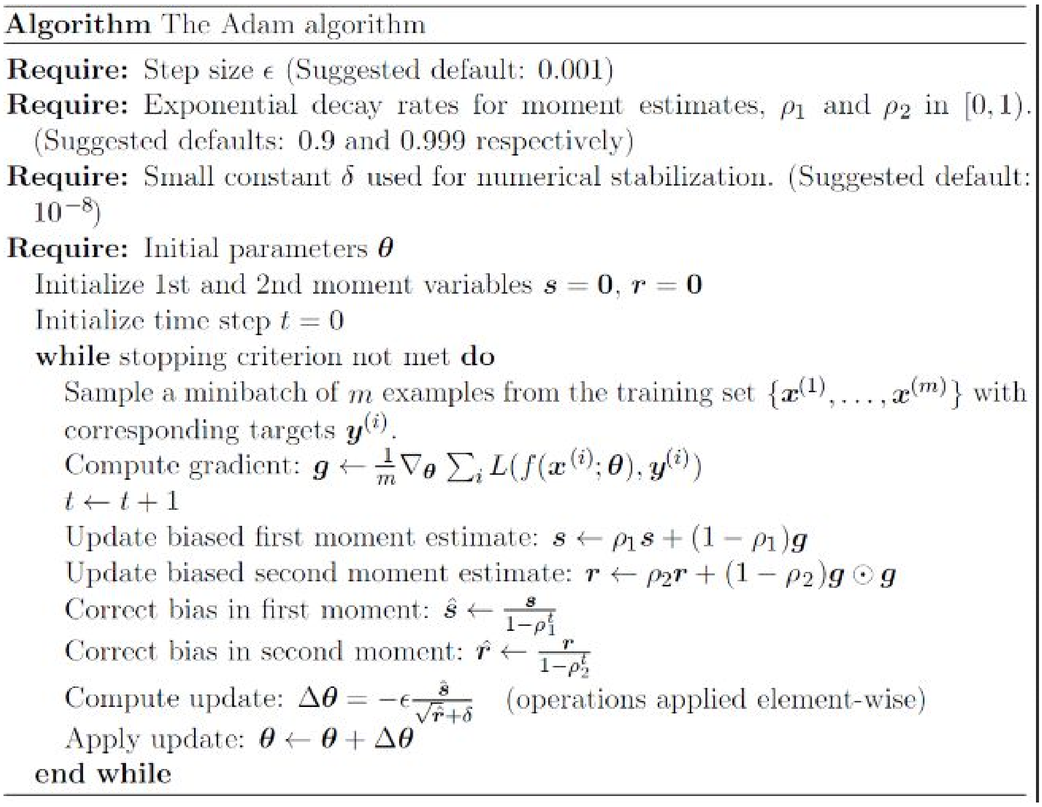
- group convolution block:
-
- 2x(c_1/2) inputs, 2x(c_2/2) output
  - 2x(c_1/2 c_2/2) = c_1c_2/2 number of kernels

- normal convolution block:
-
- c_1 inputs, c_1 outputs
  - $c_1c_2$number of kernels

9. topic

ADAM optimizer

- The name “**Adam**” derives from the phrase “**adaptive movements**”
- **Combination of RMSProp and momentum with few differences.**
- Momentum is incorporated directly as an estimate of the first order moment of the gradient.
- Includes **bias corrections** to the estimates of both the first-order moments (momentum term) and the (uncentered) second-order moments to account for their initialization at the origin.



```
Algorithm The Adam algorithm
Require: Step size $\epsilon$ (Suggested default: 0.001)
Require: Exponential decay rates for moment estimates, $\rho_1$ and $\rho_2$ in $[0, 1)$.
  (Suggested defaults: 0.9 and 0.999 respectively)
Require: Small constant $\delta$ used for numerical stabilization. (Suggested default:
  $10^{-8}$)
Require: Initial parameters $\boldsymbol{\theta}$
  Initialize 1st and 2nd moment variables $\boldsymbol{s} = \boldsymbol{0}$, $\boldsymbol{r} = \boldsymbol{0}$
  Initialize time step $t = 0$
  while stopping criterion not met do
    Sample a minibatch of $m$ examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with
    corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    $t \leftarrow t + 1$
    Update biased first moment estimate: $\boldsymbol{s} \leftarrow \rho_1 \boldsymbol{s} + (1 - \rho_1)\boldsymbol{g}$
    Update biased second moment estimate: $\boldsymbol{r} \leftarrow \rho_2 \boldsymbol{r} + (1 - \rho_2)\boldsymbol{g} \odot \boldsymbol{g}$
    Correct bias in first moment: $\hat{\boldsymbol{s}} \leftarrow \frac{\boldsymbol{s}}{1-\rho_1^t}$
    Correct bias in second moment: $\hat{\boldsymbol{r}} \leftarrow \frac{\boldsymbol{r}}{1-\rho_2^t}$
    Compute update: $\Delta\boldsymbol{\theta} = -\epsilon \frac{\hat{\boldsymbol{s}}}{\sqrt{\hat{\boldsymbol{r}}}+\delta}$   (operations applied element-wise)
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
  end while
```

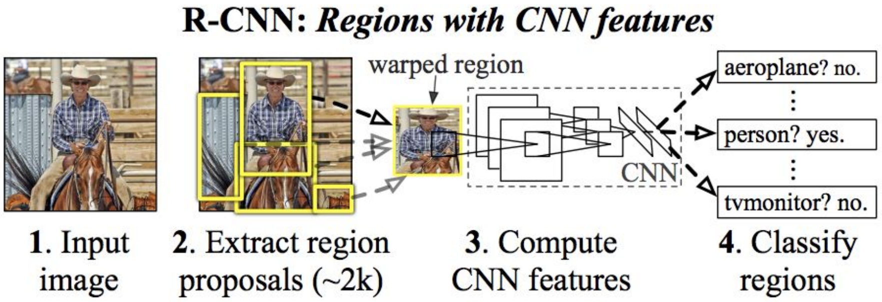
https://www.youtube.com/watch?v=nhqo0u1a6fw

The softmax function

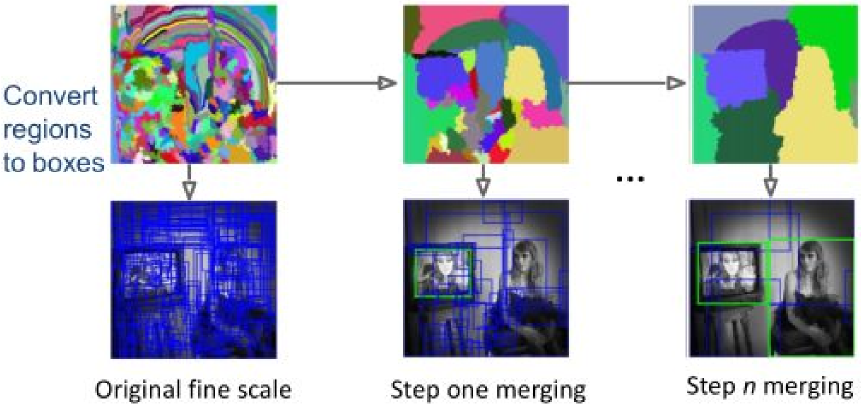
- 5. topicban már volt

R-CNN architectures - R-CNN, Fast R-CNN, Faster R-CNN

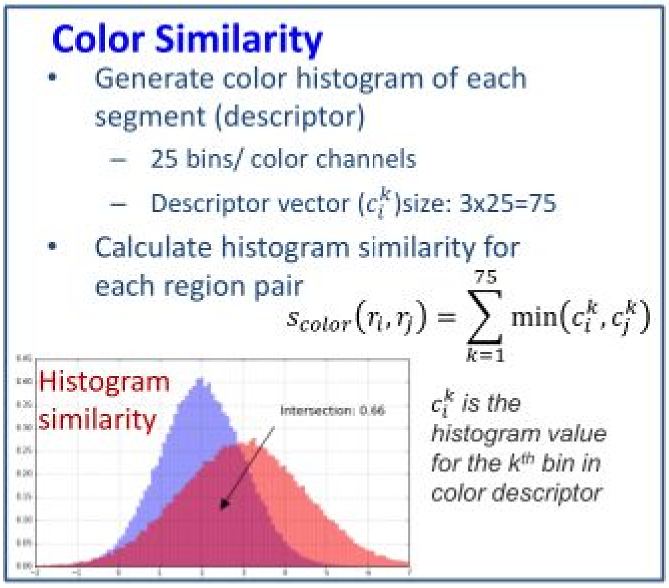
R-CNN



- **R-CNN in a glance:**
    1. **Input image**
    2. **Region proposals**
    3. **Compute CNN features with wrapped images**
    4. **Classification with Support Vector Machine (SVM)**
    5. **Ranking/selecting/merging -> detections**
    6. **Bounding box regression**
- **(2) Region Proposal**
    - Propose a large number (up to 2000) of regions (boxes) with different sizes
    - Still much better than exhausting search with multi-scale sliding window (brute force)
    - **Boxes should contain all the candidate objects with high probability**
    - **Region proposal methods:**
        - **Randomized prim**
        - **Selective search** (fastest and best)
        - etc.
    - **Selective search**
        - Graph based segmentation
        - Idea: **oversegment it and apply scaled similarity based merging:**



- Similarity measures:



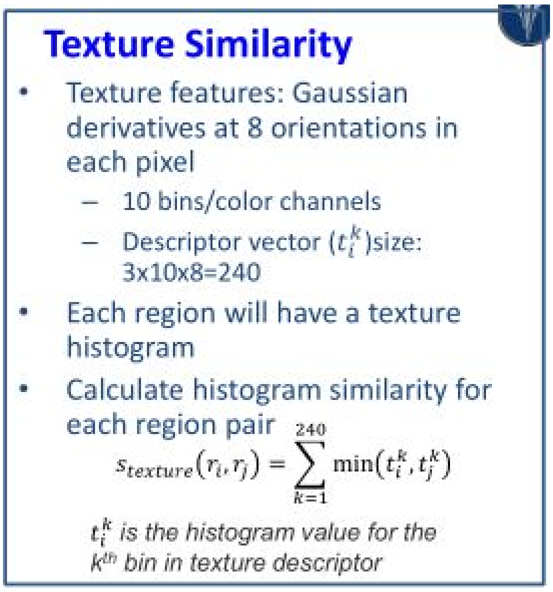
Color Similarity

- Generate color histogram of each segment (descriptor)
    - 25 bins/ color channels
    - Descriptor vector (c_i^k)size: 3x25=75
- Calculate histogram similarity for each region pair

$$s_{color}(r_i,r_j)=\sum_{k=1}^{75}\min(c_i^k,c_j^k)$$

c_i^k is the histogram value for the kth bin in color descriptor

Histogram similarity
Intersection: 0.66

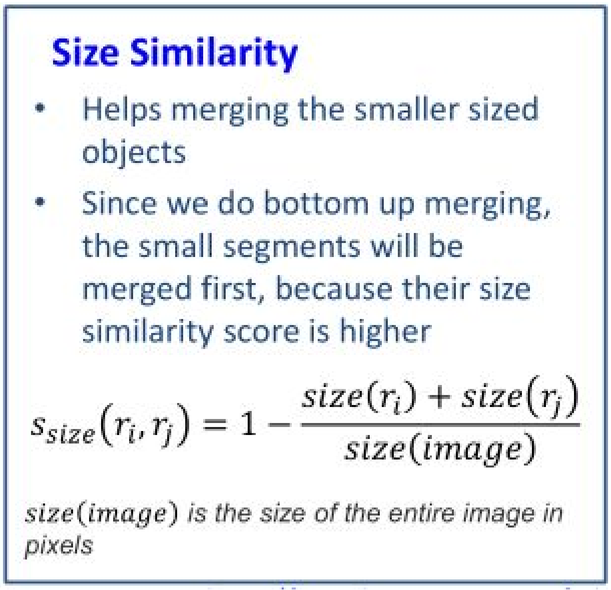


Texture Similarity

- Texture features: Gaussian derivatives at 8 orientations in each pixel
    - 10 bins/color channels
    - Descriptor vector (t_i^k)size: 3x10x8=240
- Each region will have a texture histogram
- Calculate histogram similarity for each region pair

$$s_{texture}(r_i,r_j)=\sum_{k=1}^{240}\min(t_i^k,t_j^k)$$

t_i^k is the histogram value for the kth bin in texture descriptor

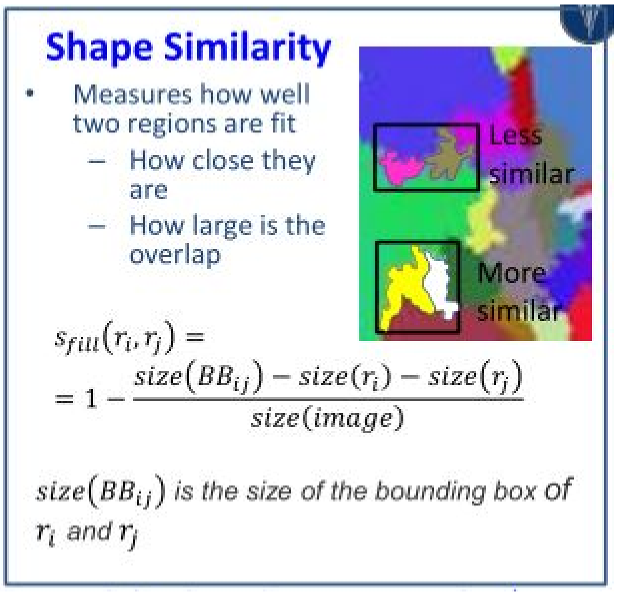


Size Similarity

- Helps merging the smaller sized objects
- Since we do bottom up merging, the small segments will be merged first, because their size similarity score is higher

$$s_{size}(r_i, r_j) = 1 - \frac{size(r_i) + size(r_j)}{size(image)}$$

$size(image)$ is the size of the entire image in pixels



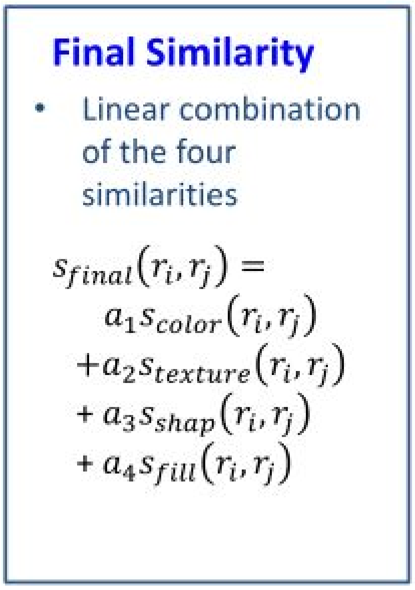
Shape Similarity

- Measures how well two regions are fit
  - How close they are
  - How large is the overlap

Less similar
More similar

$$s_{fill}(r_i, r_j) =$$
$$= 1 - \frac{size(BB_{ij}) - size(r_i) - size(r_j)}{size(image)}$$

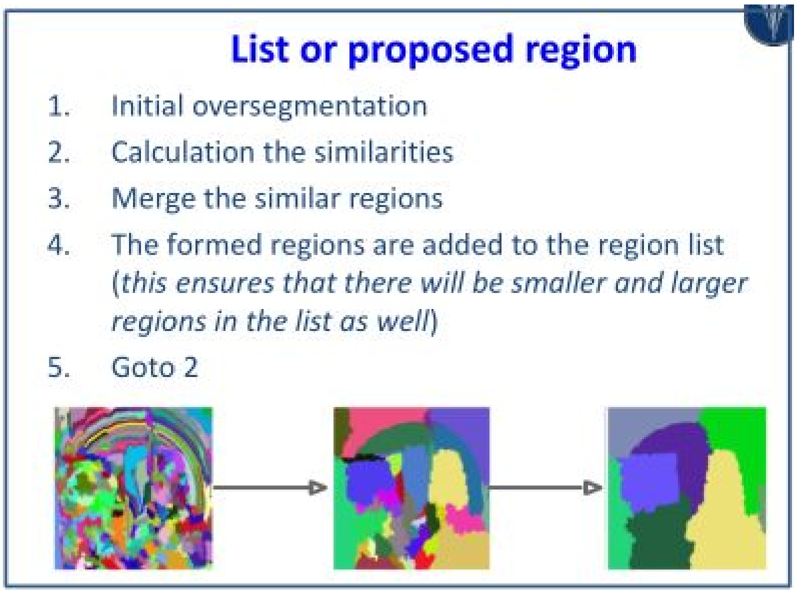
$size(BB_{ij})$ is the size of the bounding box of r_i and r_j



Final Similarity

- Linear combination of the four similarities

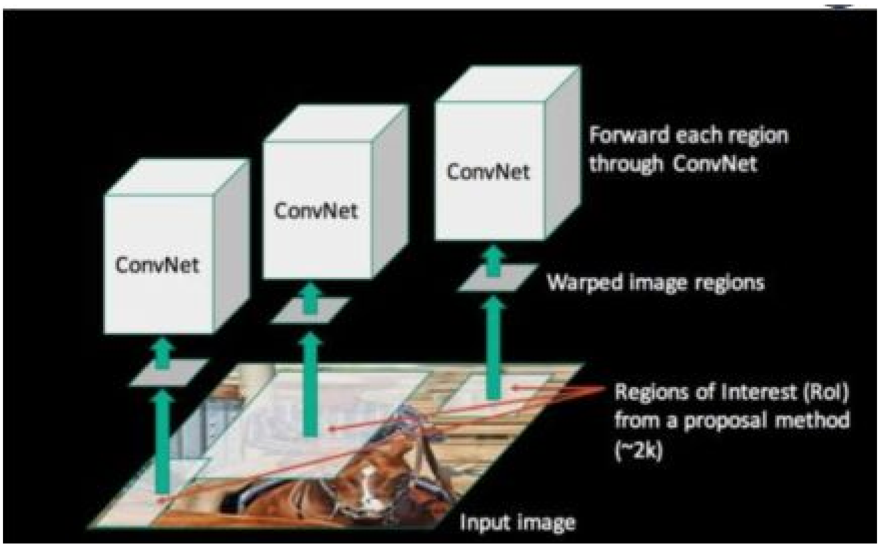
$$s_{final}(r_i, r_j) = a_1 s_{color}(r_i, r_j) + a_2 s_{texture}(r_i, r_j) + a_3 s_{shap}(r_i, r_j) + a_4 s_{fill}(r_i, r_j)$$



List or proposed region

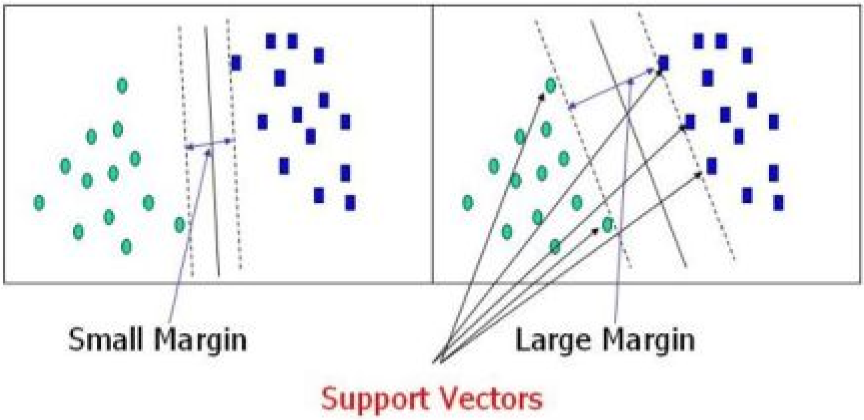
1. Initial oversegmentation
2. Calculation the similarities
3. Merge the similar regions
4. The formed regions are added to the region list (*this ensures that there will be smaller and larger regions in the list as well*)
5. Goto 2

- **(3) Computing the features of the regions**
  - Cut the regions one after the other
  - Resize (warp) the regions to the input size of the ConvNet
  - Calculate features of the individual regions

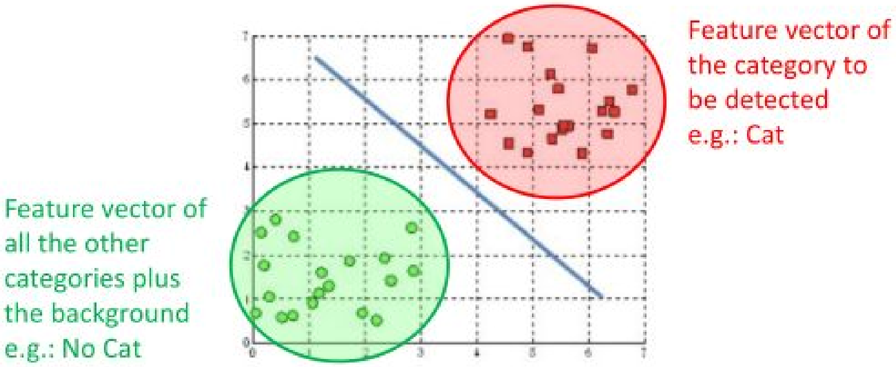

ConvNet
ConvNet
ConvNet
Forward each region through ConvNet
Warped image regions
Regions of Interest (RoI) from a proposal method (~2k)
Input image

  - **Convolutional network:**

        - Pre-trained AlexNet, later VGGNet
        - The decision maker SoftMax layer was cut
            - Outputs:
                - 4096 long feature vectors from each region
- **(4) Classification with Support Vector Machine (SVM)**
    - **Idea: Separate the data point in the data space with a boundary surface (hyperplane) with maximum margin**
    - **Vectors pointing to the data points touching the margins are the support vectors.**
    - The **parameters** of the **hyperplane** is calculated with **regression**
    - Similar to **single layer perceptron** but **optimized for maximum margin**
    - **Why SVM?**
        - Why not simple classification on the output of Alxnet?
        - During the training the Alexnet/VGGNet is not trained
        - **Only SVM is trained**
        - **Number of categories is much smaller: 20-200 categories rather than 1000**

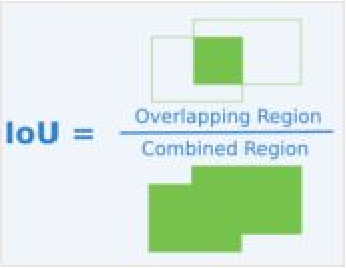


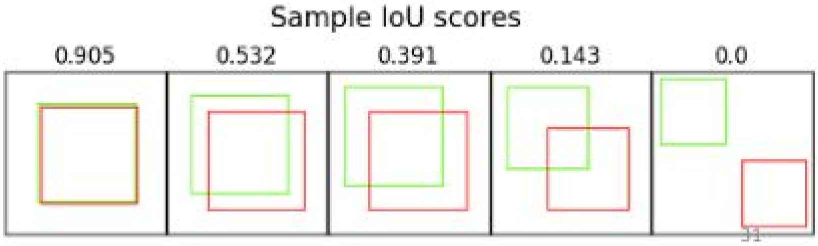
        - **Decision with SVM**
            - **As many separate SVM as many category we have**



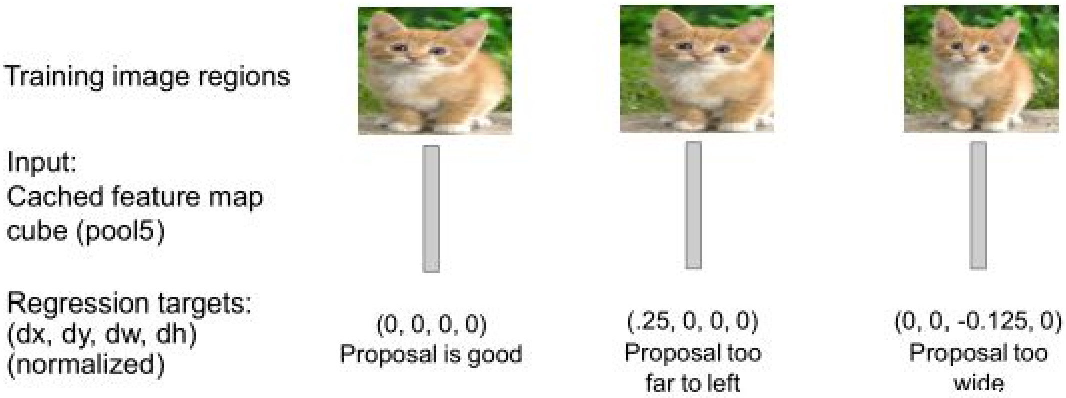
https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47

- **(5) Ranking / selecting / merging -> detection**
    - **Greedy non-maximum suppression**
        - Regions with low classification probabilities are rejected
        - Regions with high Intersection over Union values (within the same category)
    - **Result**: localized and classified object

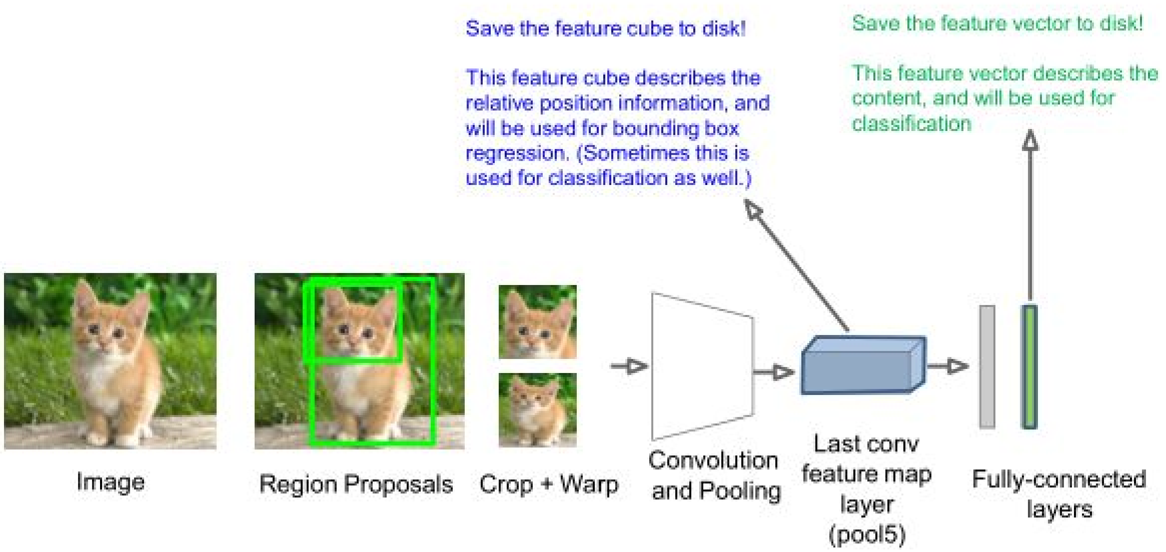




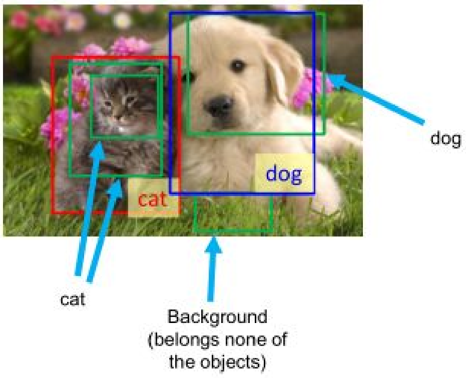
- **(6) Bounding Box Regression**
    - Linear regression model
    - One per object category
    - **Input**: last feature map cube of the convent (pool5)
    - **Output**: size and position modification to the bounding box
        - dx, dy, dw, dh



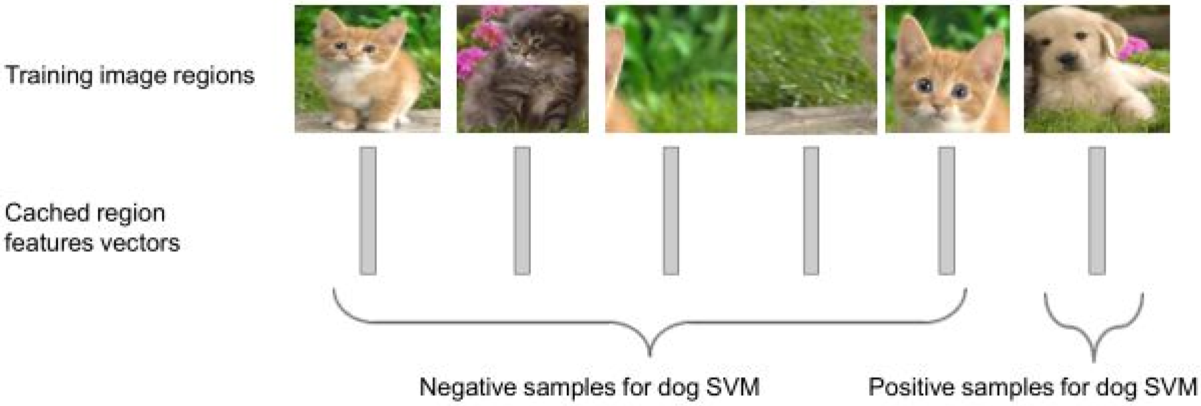
- **RCNN Training steps**
    - **Step 1**
        - **Take a pre trained CNN**
        - Reusing a pre-trained network is useful if there is not enough data to train or if it provides good enough result. (fine tuning is typically needed)
    - **Step 2**
        - Extract features
        - Go through database
        - Use region proposal
        - Calculate features for each proposed region


Save the feature cube to disk!
This feature cube describes the relative position information, and will be used for bounding box regression. (Sometimes this is used for classification as well.)
Save the feature vector to disk!
This feature vector describes the content, and will be used for classification
Image
Region Proposals
Crop + Warp
Convolution and Pooling
Last conv feature map layer (pool5)
Fully-connected layers

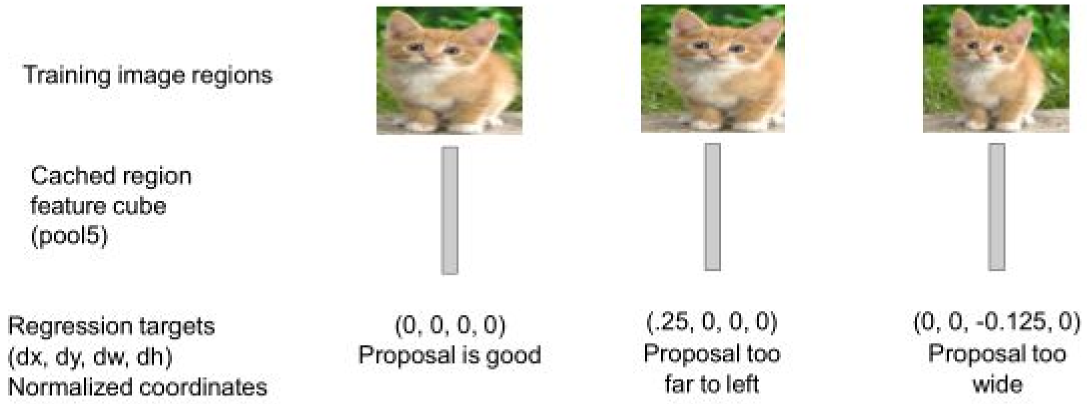
- **Step 3**
  - Identify which proposed region belongs to which object class
  - Based on the annotated image



- **Step 4**
  - Train one SVM per class to classify region features


Training image regions
Cached region features vectors
Negative samples for dog SVM
Positive samples for dog SVM

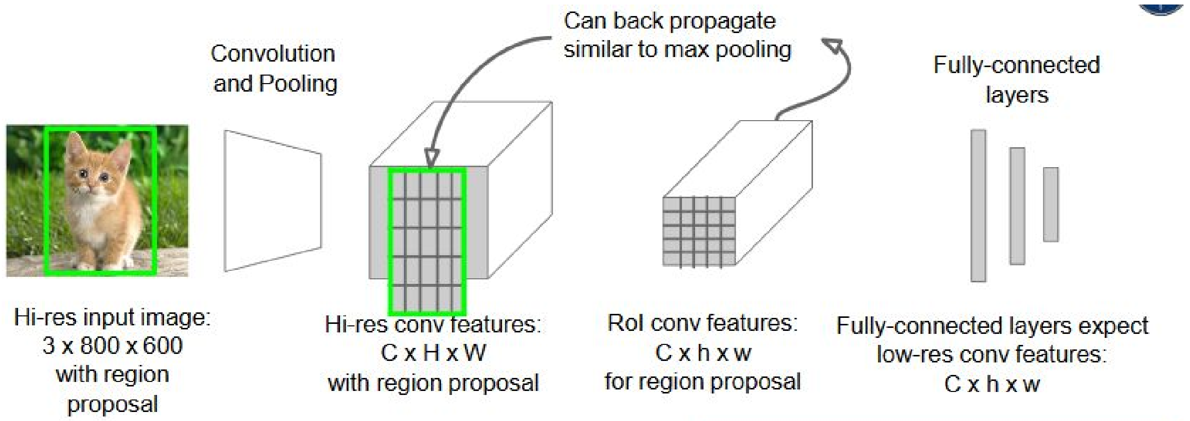
- **Step 5**
  - (bbox regression): For each class train a linear regression model to map from cached features cubes to offsets / size of the boxes to fix “slightly wrong” position proposals


Training image regions
Cached region feature cube (pool5)
Regression targets (dx, dy, dw, dh) Normalized coordinates
(0, 0, 0, 0) Proposal is good
(.25, 0, 0, 0) Proposal too far to left
(0, 0, -0.125, 0) Proposal too wide

- RCNN has much better results, and a grate improval to pre-CNN methods. Bounding box regression can also help.
- Problems with R-CNN
    - **Slow at test-time: need to run full forward pass of CNN for each region proposal**
        - Recalculate the features again-and-again in the overlapping regions
        - **Solution**: share computation of convolutional layers between proposals for an image
    - **SVMs and bbox regressors are post-hoc**
        - CNN features not updated in response to SVMs and regressors
    - **Complex multistage training pipeline**
        - Calculate the features for all the regions for all the training image first
        - Then train for SVM and bbox regressor separately
        - **Solution**: just train the whole system end-to-end all at once

Fast R-CNN

- **Problem with R-CNN**
    - It still takes a **huge amount of time** to train the network as you would have to **classify 2000 region proposals per image.**
    - It **cannot** be implemented **real time** as it takes around 47 seconds for each test image.
    - The **selective search algorithm** is a fixed algorithm. Therefore, **no learning** is happening at that stage. This could lead to the generation of **bad candidate region proposals.**


Convolution and Pooling
Can back propagate similar to max pooling
Fully-connected layers
Hi-res input image: 3 x 800 x 600 with region proposal
Hi-res conv features: C x H x W with region proposal
RoI conv features: C x h x w for region proposal
Fully-connected layers expect low-res conv features: C x h x w

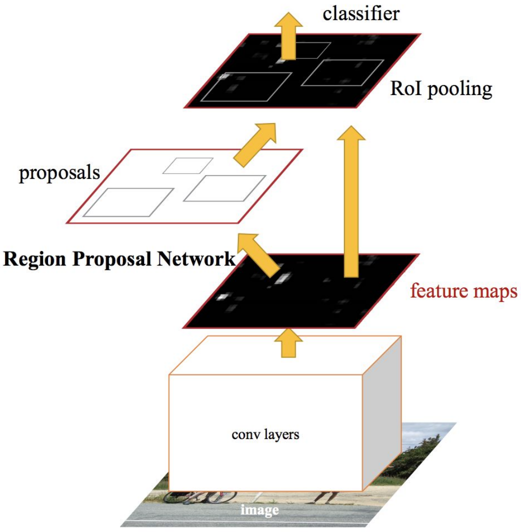
Instead of SVM, a SoftMax layer makes the decision at Fast R-CNN.

- The approach is similar to the R-CNN algorithm. But**, instead of feeding the region proposals** to the **CNN**, we **feed** the **input image** to the **CNN** to **generate a convolutional feature map.**

- From the **convolutional feature map**, we **identify** the **region of proposals** and warp them into squares and by **using a RoI pooling layer we reshape them into a fixed size so that it can be fed into a fully connected layer.**
- From the RoI feature vector, we use a **softmax layer to predict the class of the proposed region and also the offset values for the bounding box.**
- The **reason** "Fast R-CNN" is **faster** than R-CNN is **because** you **don't have to feed 2000 region proposals to the convolutional neural network every time**. Instead, the **convolution operation is done only once** per image and a **feature map is generated from it.**

Faster R-CNN

- **Problem with Fast R-CNN:**
    - When you look at the **performance** of Fast R-CNN **during testing time**, including **region proposals slows down the algorithm significantly** when compared to not using region proposals. Therefore, **region proposals become bottleneck**s in Fast R-CNN algorithm affecting its performance.
- Both of the above algorithms(R-CNN & Fast R-CNN) uses selective search to find out the region proposals. **Selective search is a slow and time-consuming process** affecting the performance of the network.
- Therefore we use an **object detection algorithm that eliminates the selective search algorithm and lets the network learn the region proposals.**
- **Region Proposal Network**
    - Slide a small window on the feature map
    - Build a small network for:
        - classifying object or not-object, regressing bbox locations
    - Positions of the sliding window provides localization information with reference to the image.
    - Box regression provides finer localization information with reference to this sliding window.
    - Use N anchor boxes at each location.
    - Anchors are translation invariant: use the same ones every location.
    - Regression gives offsets from anchor boxes
    - Classification gives the probability that each (regressed) anchor shows an object



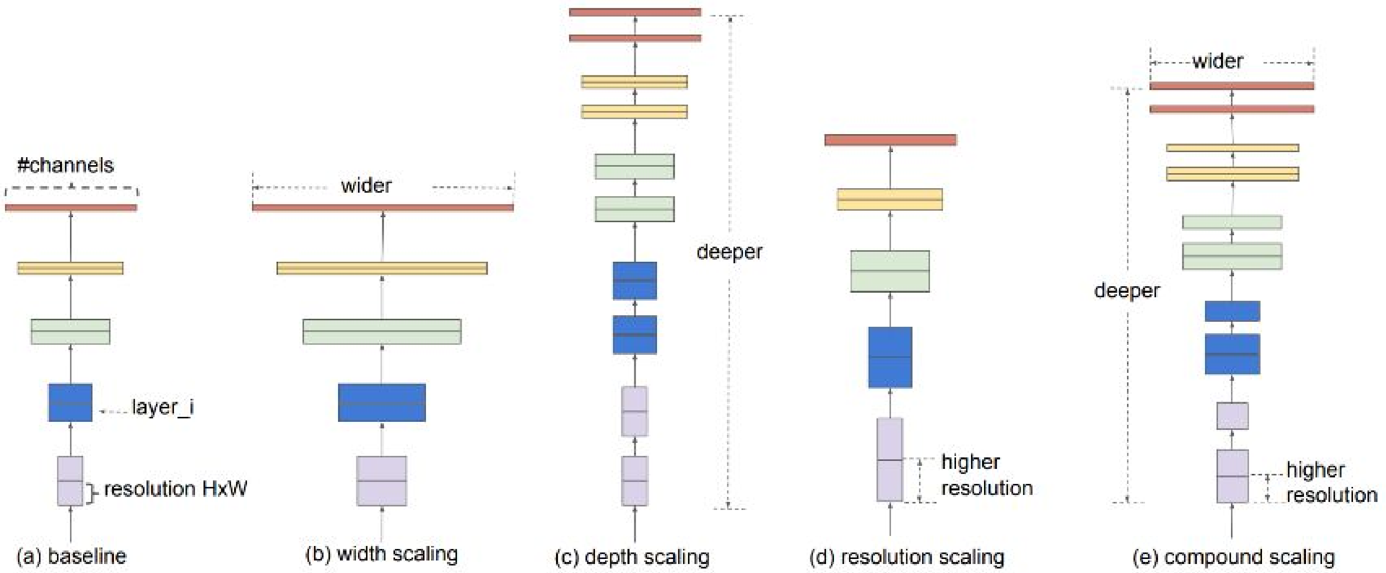
- **One network, four losses**: RPN classification (anchor good / bad), RPN regression (anchor -> proposal), Fast R-CNN classification (over classes), Fast R-CNN regression (proposal -> box)

https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e

Supervised vs. unsupervised learning

- 3. topic Machine learning vs traditional programmingban leírva.

EfficientNet



- **Recent network architecture**
- Based on **scaling the width, the depth and the resolution uniformly.**
- Can be used for any existing architecture and the efficiency will be significantly better with the same performance
- Best performance can be reached by using NN to generate the optimal baseline ConvNet

https://medium.com/@nainaakash012/efficientnet-rethinking-model-scaling-for-convolutional-neural-networks-92941c5bfb95

10. topic

Optimization problem of objective functions of neural network

Learning in practice

- Learning based on the **training set:**

$$\tau^{(K)} = \{(\mathbf{x}_k, d_k); k = 1, \ldots, K\}$$

- **Minimize** the **empirical error function:**

$$\mathbf{w}_{opt}^{(K)} : \min_{\mathbf{w}} \frac{1}{K} \sum_{k=1}^{K} \underbrace{\left(d_k - Net(\mathbf{x}_k, \mathbf{w})\right)^2}_{E_k} = \min_{\mathbf{w}} R_{emp}(\mathbf{w})$$

- Learning is a **multivariate optimization task**

Objective function

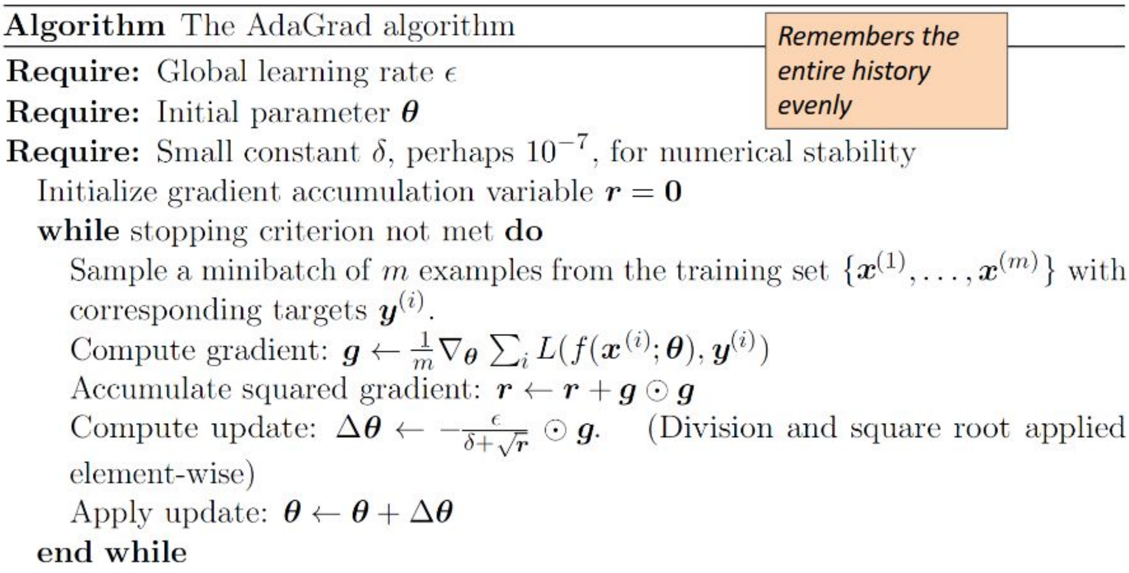
- **Objective function** or **error function, cost function, loss function, criterion** is a special function that we have to **minimize** for a neural network by **modifying its parameters. (optimization)**
- The loss is calculated from the actual value and the predicted value by the network.
- **Tells the net the size of the error, and penalize according to it.**
- 3-as topicban már beszéltünk róla milyen lossok vannak.
- **Optimization**:
    - Find the optimal weights:

$$\mathbf{w}_{opt} = min\ f(\mathbf{x}, \mathbf{d}, Net(\mathbf{x}, \mathbf{w}))$$

    - Stochastic Gradient Descent method (there are more advanced ones)

AdaGrad optimizer

- Individually **adapts the learning rates** of all model parameters by **scaling them inversely proportional to the square root of the sum of all of their historical squared values.**
- The **parameters** with the **largest partial derivative** of the loss have a **correspondingly rapid decrease in their learning rate, while parameters with small partial derivatives have a relatively small decrease in their learning rate.**
- **AdaGrad performs well for some but not all deep learning models**

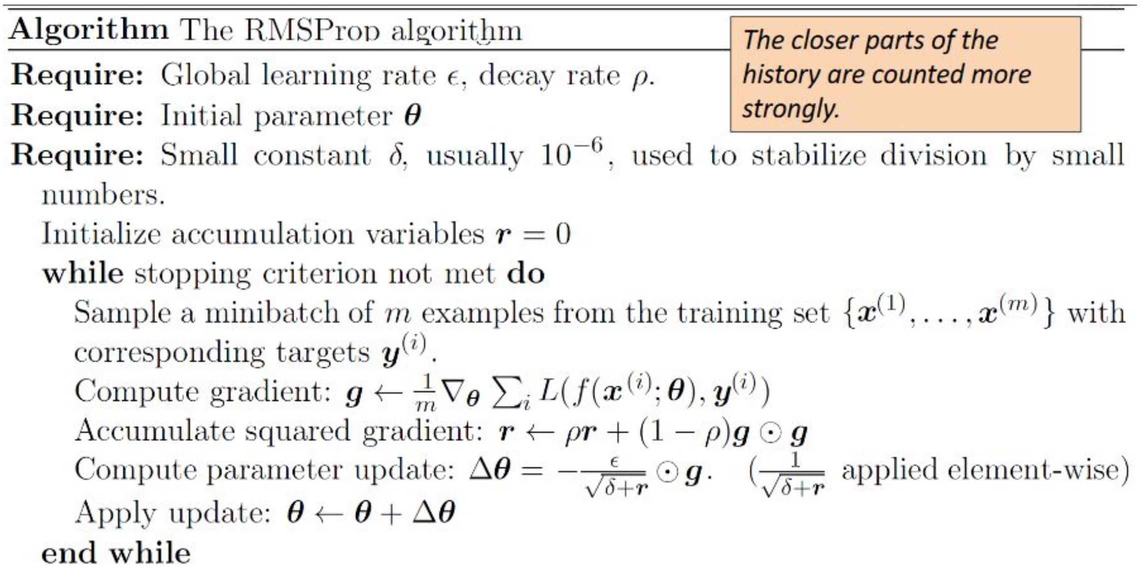


```
Algorithm The AdaGrad algorithm
Require: Global learning rate $\epsilon$
Require: Initial parameter $\boldsymbol{\theta}$
Require: Small constant $\delta$, perhaps $10^{-7}$, for numerical stability
  Initialize gradient accumulation variable $\boldsymbol{r} = \mathbf{0}$
  while stopping criterion not met do
    Sample a minibatch of $m$ examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Accumulate squared gradient: $\boldsymbol{r} \leftarrow \boldsymbol{r} + \boldsymbol{g} \odot \boldsymbol{g}$
    Compute update: $\Delta\boldsymbol{\theta} \leftarrow -\frac{\epsilon}{\delta + \sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$.  (Division and square root applied element-wise)
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
  end while
```

Remembers the entire history evenly

RMSP algorithm

- **Modifies the AdaGrad** to **perform better in non-convex setting**, by **changing the gradient accumulation into an exponentially weighted moving average.**
- **Adagrad reduces the learning rate in each step → After a while it stops.**
- Adagrad shrinks the learning rate according to the entire history of the squared gradient and may have made the learning rate too small before arriving at such a convex structure



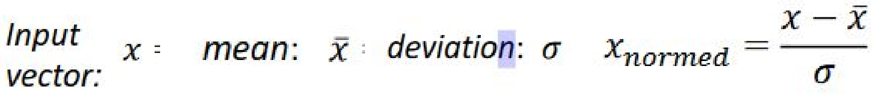
```
Algorithm The RMSProp algorithm
Require: Global learning rate $\epsilon$, decay rate $\rho$.
Require: Initial parameter $\boldsymbol{\theta}$
Require: Small constant $\delta$, usually $10^{-6}$, used to stabilize division by small numbers.
  Initialize accumulation variables $\boldsymbol{r} = 0$
  while stopping criterion not met do
    Sample a minibatch of $m$ examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Accumulate squared gradient: $\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1 - \rho) \boldsymbol{g} \odot \boldsymbol{g}$
    Compute parameter update: $\Delta \boldsymbol{\theta} = -\frac{\epsilon}{\sqrt{\delta + \boldsymbol{r}}} \odot \boldsymbol{g}$.   ($\frac{1}{\sqrt{\delta + \boldsymbol{r}}}$ applied element-wise)
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta \boldsymbol{\theta}$
  end while
```

The closer parts of the history are counted more strongly.

- RMSProp uses an **exponentially decaying average to discard history from the extreme past so that it can converge rapidly after finding a complex bowl,** as if it were an instance of the AdaGrad algorithm initialized within that bowl.

Input vector normalization

- **If the input vector contains high and small values in different vector positions it is useful to normalize them.**
- Squeeze the number to the same range
- **Speeds up training**



Input vector: x : *mean:* $\bar{x}$: *deviation:* σ $\quad x_{normed} = \frac{x - \bar{x}}{\sigma}$

- **Different normalization strategies exist for different input types**
- Showing it in two dimension, it shapes the input vector

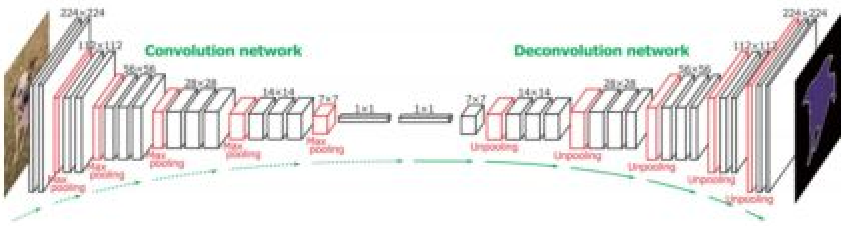
DeconvNet, U-Net

- **Semantic Image Segmentation architectures**

DeconvNet

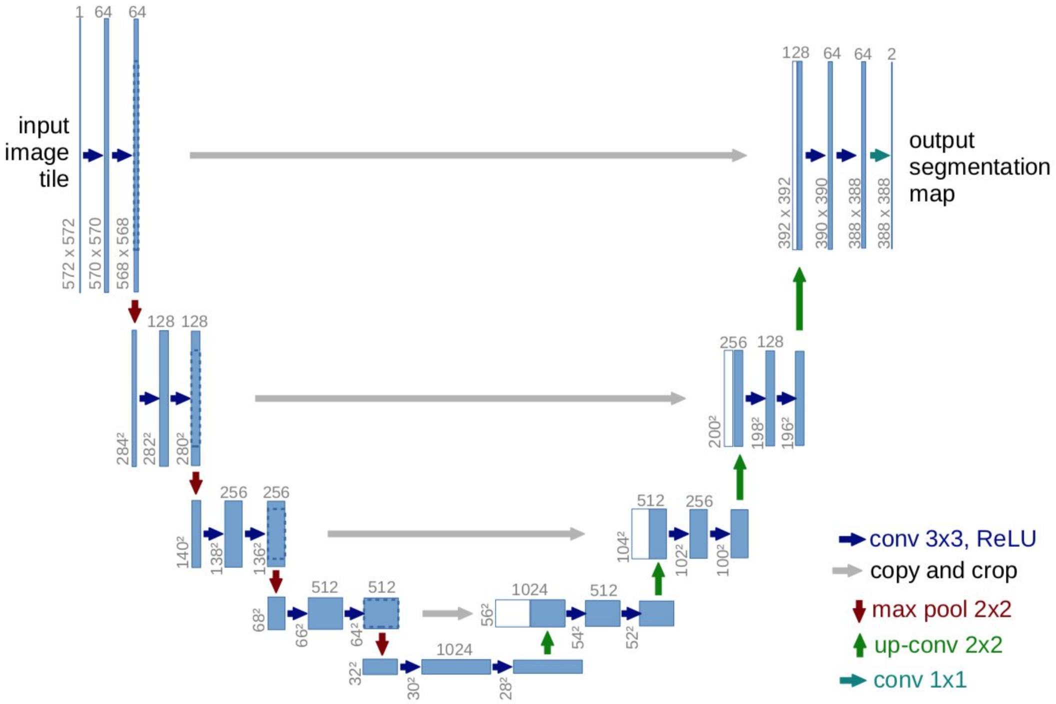
- Instance-wise segmentation
- Two-stage training
  - train on **easy example** (cropped bounding boxes centered on a single object) first and
  - then more **difficult examples**
- **Fully symmetrical convolutional network**
  - All **convolution and pooling layers are reversed**
- Two stage training (first side trained for classification first)

- Output probability map same size as input



U-Net

- Designed for **biomedical image processing**: **cell segmentation**
- **Data augmentation** via applying **elastic deformations,**
    - Natural since deformation is a common variation of tissue
    - **Smaller dataset is enough**
- **Concatenate features** from encoder network (instead of reusing pooling indices)
- Relatively shallow network with **low computational demand**
    - 3x3 convolution kernel size only
    - 2x2 max pooling
- **No fully connected layer in the middle**

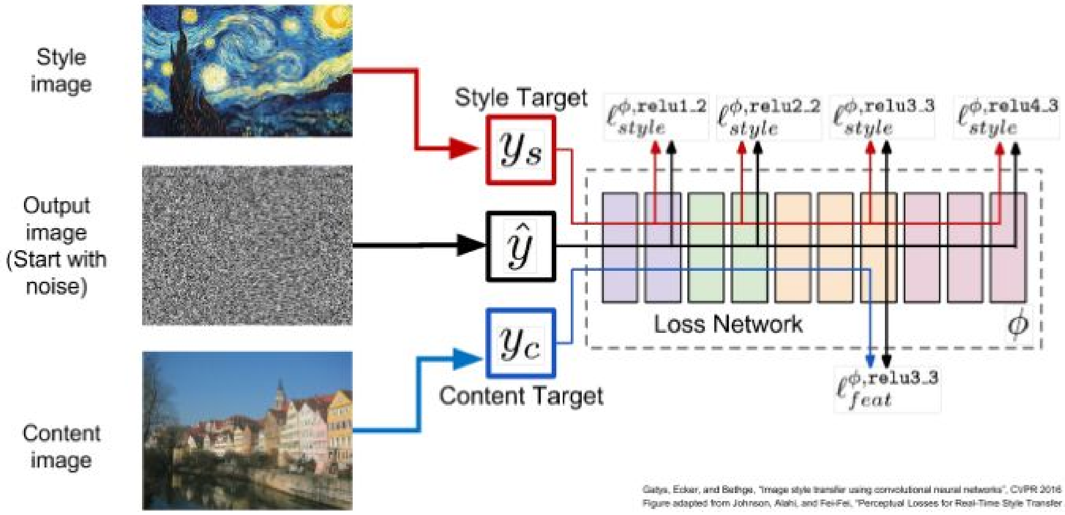


http://users.itk.ppke.hu/~ekacs/anyagok/felev5/NeurHalok/Lab%20reports/2019-11-14_Evelin_Remetehegyi_1_Barnabas_Benko_2_Csaba_Ekart_3_lab_report.pdf

Neural style transfer

- **Uses gradient ascend**
- **Take input image and transform it into the style of an other input image**

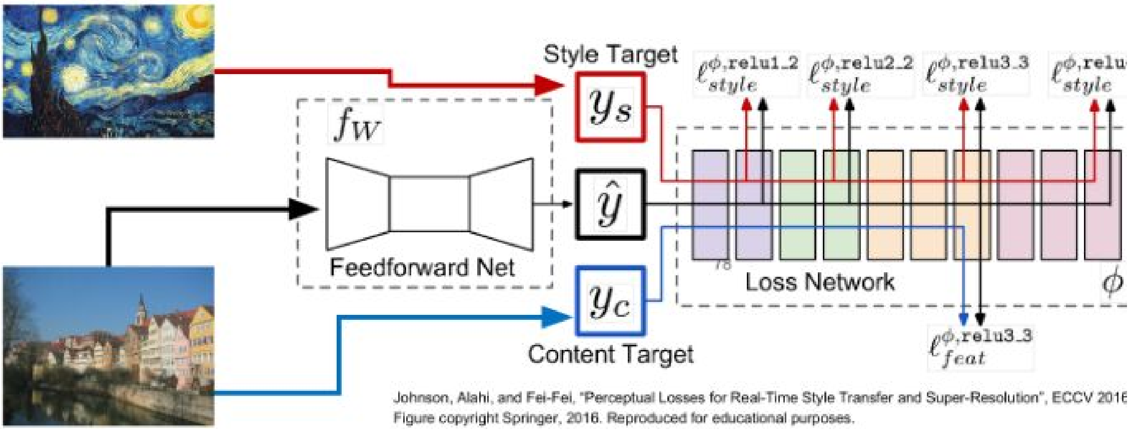
- Gradient ascent transforms the image according to a **loss function**
- Can we find a **loss function which** would **preserve objects** and another which **preservers features connected to a style**?



Style image
Output image (Start with noise)
Content image
Style Target
y_s
$\hat{y}$
y_c
Content Target
$\ell_{style}^{\phi,relu1\_2}$
$\ell_{style}^{\phi,relu2\_2}$
$\ell_{style}^{\phi,relu3\_3}$
$\ell_{style}^{\phi,relu4\_3}$
Loss Network
ϕ
$\ell_{feat}^{\phi,relu3\_3}$

Gatys, Ecker, and Bethge, "Image style transfer using convolutional neural networks", CVPR 2016
Figure adapted from Johnson, Alahi, and Fei-Fei, "Perceptual Losses for Real-Time Style Transfer and

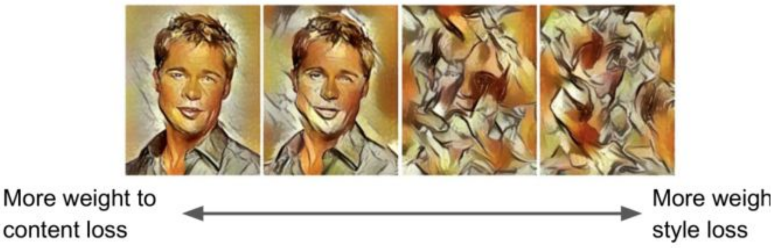
- **Lot of time** to generate an image
- **Many forward and backward passes** are needed
- How to **fast** things up?
  - We could **train a network that learns the result of this iterative transformation and tries to predict it. Only a single pas is needed.**



f_W
Feedforward Net
Style Target
y_s
$\hat{y}$
y_c
Content Target
$\ell_{style}^{\phi,relu1\_2}$
$\ell_{style}^{\phi,relu2\_2}$
$\ell_{style}^{\phi,relu3\_3}$
$\ell_{style}^{\phi,relu}$
Loss Network
ϕ
$\ell_{feat}^{\phi,relu3\_3}$

Johnson, Alahi, and Fei-Fei, "Perceptual Losses for Real-Time Style Transfer and Super-Resolution", ECCV 2016
Figure copyright Springer, 2016. Reproduced for educational purposes.

- **Loss function for content**
  - Can the **same objects** be found on both images?
  - Content loss, perceptual loss
    - **distance** between **two embedded image vectors in the last features layers**
- **Style loss**
  - Can the **same low level features, edges, structures, simple patterns** be found on both images?
  - Style loss
    - **distances** between **lower level representations of the images.**

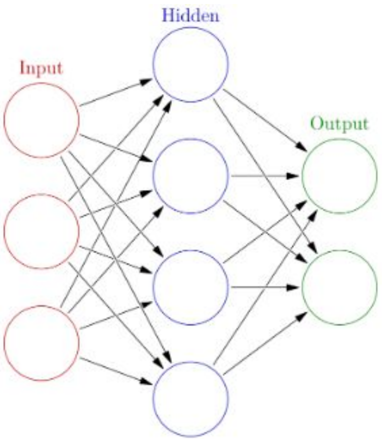


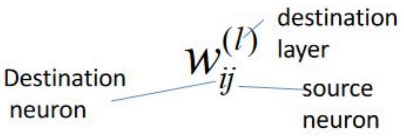
11. topic

Multilayer perceptron

$$Net(\mathbf{x},\mathbf{W}) = \varphi^{(L)}\left(\mathbf{w}^{(L)}\varphi^{(L-1)}\left(\mathbf{w}^{(L-1)} \quad \dots \quad \varphi^{(2)}\left(\mathbf{w}^{(2)}\varphi^{(1)}\left(\mathbf{w}^{(1)}\mathbf{x}\right)\right)\right)\right)$$

- aka **Feed Forward Neural Networks** or **Fully Connected Neural Networks**
- Used for: **classification and approximation**
- Built from:
  - **Input layer**
  - **Hidden layer(s)**
  - **Output layer**
- **Many hidden layers: deep network**
- The outputs is **typically not binary**
- **Can solve linearly non-separable problems**

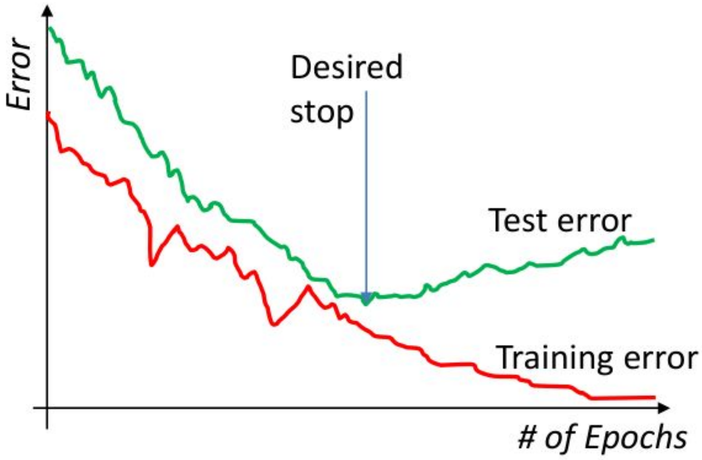




- **Multilayer perceptrons are used for**
  - **Classification**: Supervised learning for classification (input and class labels given)
  - **Approximation**: Approximate an arbitrary function with arbitrary precision

Early stopping

- Regularization and optimization methods
- **Idea:**
  - **Split data into training and test sets**
  - At the end of each epoch (or, every N epochs):
    - **evaluate the network performance on the test set**
    - **if the network outperforms the previous best model: save copy of the network parameters at the current epoch**
  - **The best sub optimum is selected finally**
  - **Since the error function is not necessarily monotonic the optimization goes on but the suboptima are saved**



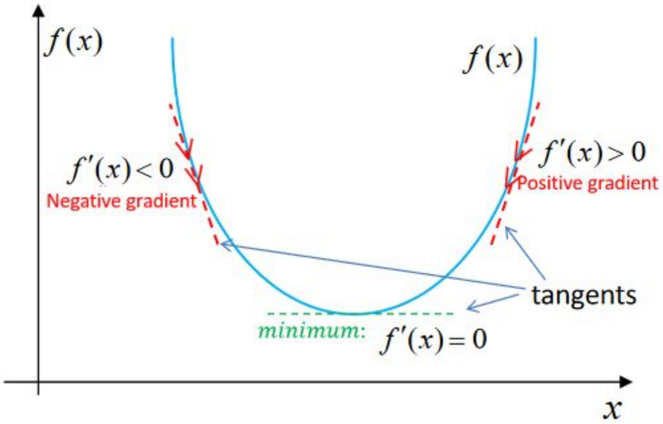
Gradient descent (multidimensional cases as well)

- **Find a function local minimum**
- We **start out from one point** (say x_1) and with an **iterative method**, we need to go towards the minimum
- We **follow the descending gradient**
- For small ε

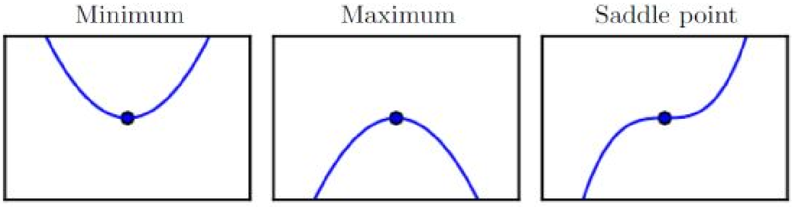
$$f(x+\varepsilon) \approx f(x) + \varepsilon f'(x)$$

- therefore

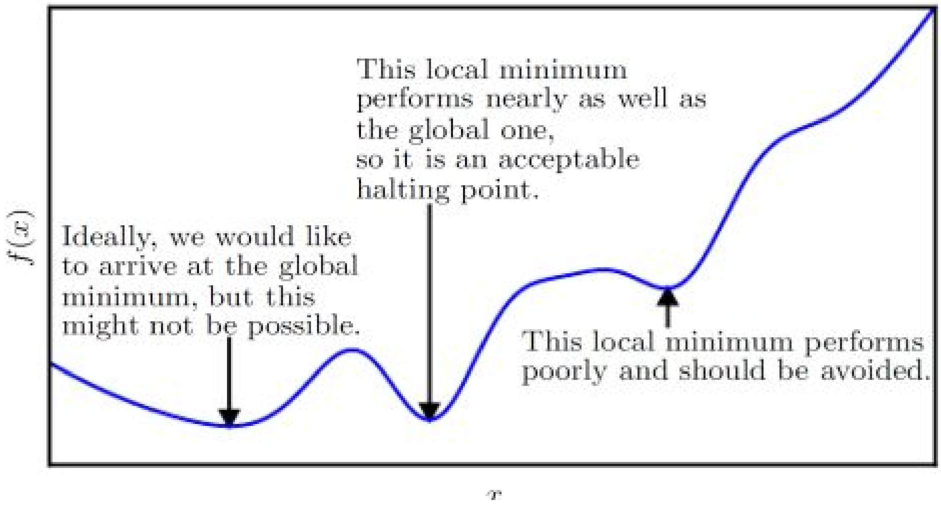
$$f\left(x-\varepsilon\, sign\left(f'(x)\right)\right) \leq f(x)$$



- **Stationary points: (f'(x) = 0)**
  - **Local minimum**: f(x) smaller than all neighbors
  - **Local maximum**: f(x) larger than all neighbors
  - **Saddle points**: neither



- We **don't have to find a global minimum, we just need a minimum, that performs very well:**



This local minimum performs nearly as well as the global one, so it is an acceptable halting point.
Ideally, we would like to arrive at the global minimum, but this might not be possible.
This local minimum performs poorly and should be avoided.
$f(x)$
x

Multidimensional input functions

- In case of a vector scalar function
- In 2D, directional derivatives (slope towards x1 and x2)

- **Definition of gradient:**

$$f : R^2 \to R$$

$$\nabla f(x_1, x_2) := \left(\frac{\partial f}{\partial x_1} \quad \frac{\partial f}{\partial x_2} \right)$$

- The **gradient defines (hyper) plane approximating the function infinitesimally at point x(x1, x2)**

$$\Delta z = \frac{\partial f(x_1, x_2)}{\partial x_1} \cdot \Delta x_1 + \frac{\partial f(x_1, x_2)}{\partial x_2} \cdot \Delta x_2$$

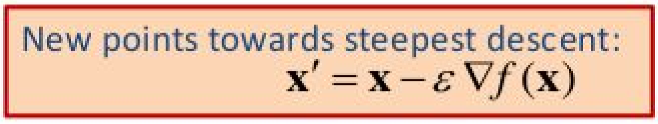
- Directional derivative to arbitrary direction u (u is unit vector) is the slope of f in that direction at point x(x1, x2)

$$\mathbf{u}^{\mathrm{T}} \nabla f(\mathbf{x})$$

- Itt **keresünk egy olyan u-t amire ez minimális:**

$$\min_{u, u^{\mathrm{T}} u = 1} \mathbf{u}^{\mathrm{T}} \nabla f(\mathbf{x})$$

- **Steepest gradient descent:**



New points towards steepest descent:

$$\mathbf{x}' = \mathbf{x} - \varepsilon \nabla f(\mathbf{x})$$

- Steepest gradient descent iteration:

$$\mathbf{x}(n+1) = \mathbf{x}(n) - \varepsilon \nabla f(\mathbf{x}(n))$$

- ε is the learning rate
  - Small constant
  - Decreases as the iteration goes ahead
- **Line search**: checked with several values, and the one selected, where f(x) is the smallest
- **Stop when it's close enough to zero**

Jacobian matrix

- Partial derivative of a vector -> vector function
- Specifically if we have a function

$$\mathbf{f} : \Re^m \to \Re^n$$

then the Jacobian matrix

$$\mathbf{J} \in \Re^{n \times m}$$

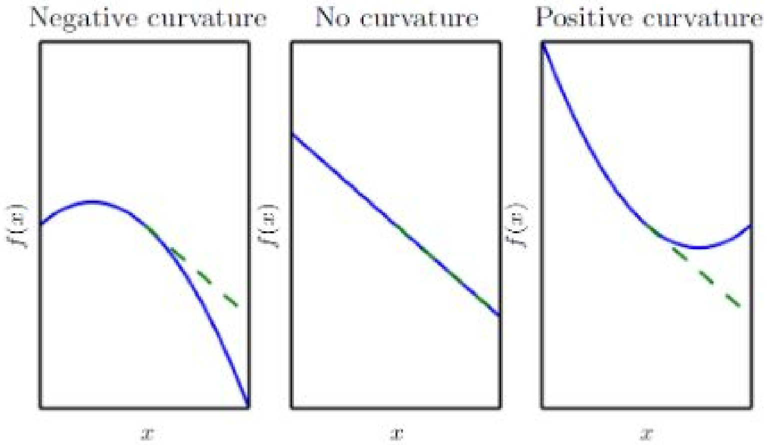
$$\mathbf{J} = \begin{bmatrix} \frac{\partial \mathbf{f}}{\partial x_1} & \cdots & \frac{\partial \mathbf{f}}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

2nd derivatives

- 2nd derivative determines the curvature of a line in 1D
- In ND it is described by the Hessian Matrix:

$$H\left(f(x_{i,j})\right) = \frac{\partial^2}{\partial x_i \partial x_j} f(x) = \frac{\partial^2}{\partial x_j \partial x_i} f(x)$$

- The Hessian is the Jacobian of the gradient.



Itt ez van tovább is, de nem írom le, mert szerintem az egész csak arról szól, hogy hogyan vezetjük le a Newton Methodot. Ami szép és jó, de nem is kapcsolódik ide, másrészt meg úgyis csak a szuper okosak akarják azt is megtanulni.

Weight regularization (L1, L2)

- **Modifies the weights**
  - Rather than using MSE const function
  - **Done on minibatch level**
  - Can be used with **other cost functions**
- Differentiable: **back-prop fine**
- **Why?**
  - **Network prefers smaller weights**
  - If a few **large weights dominate the decision**, the network will **lose fine generalization properties**
  - In case of **large weights, the decisions are less distributed, the network is less error tolerant**
- A lényeg hogy a cost functionhöz hozzáadunk valamit, amivel büntetni tudjuk a nagy weighteket. A lambda az egy kis paraméterke, ami megadja ennek az értékét.

$$C_0 = \frac{1}{2n} \sum_x \|y - a^L\|^2$$

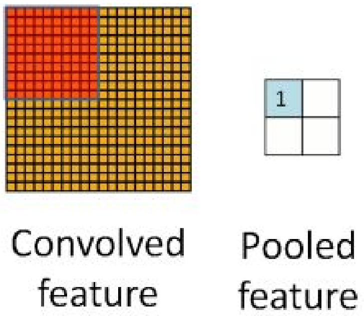
$$C_{L_1} = \frac{1}{2n} \sum_x \|y - a^L\|^2 + \frac{\lambda}{n} \sum_w |w|$$

$$C_{L_2} = \frac{1}{2n} \sum_x \|y - a^L\|^2 + \frac{\lambda}{2n} \sum_w w^2$$

https://towardsdatascience.com/l1-and-l2-regularization-methods-ce25e7fc831c
https://www.youtube.com/watch?v=iuJgyiS7BKM

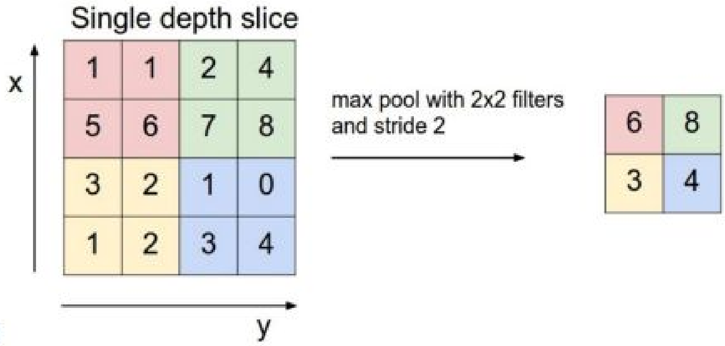
Pooling

- Pooling **summarizes statistically the extracted features from the same location on the feature map**
- Mathematically it is a local function over 1D or 2D data
  - **Input**: segment of a vector in 1D / rectangular neighborhood in 2D
  - **Function**: Statistical (max-pool), L2 norm, weight average etc.
- **In most cases stride > 1, which leads to downsampling.**
- Pooling introduces **some shift invariance**



Max pooling

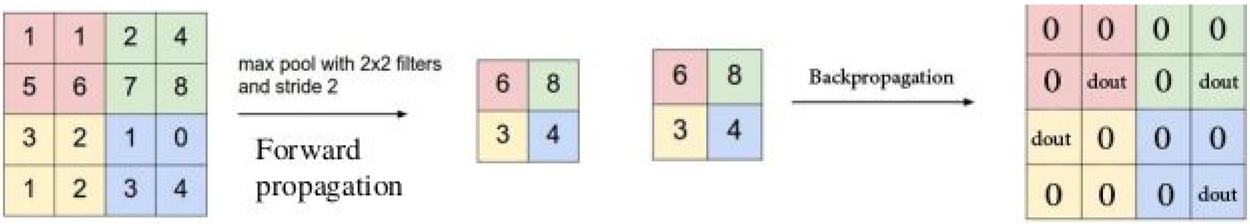
- **Most used pooling in CNNs**
- Pick the **largest value from a neighborhood**
- **Non-linear, statistical filter**
- **Downsampling depends on the stride**



1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

6	8
3	4

- **Backpropagation through max pooling layer:**
  - **Store the maximum positions:**



1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

6	8
3	4

6	8
3	4

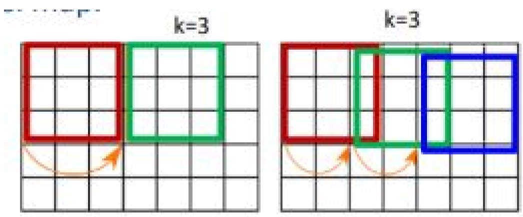
0	0	0	0
0	dout	0	dout
dout	0	0	0
0	0	0	dout

Average pooling

- **Ugyanaz, mint a max csak átlaggal**

Overlapping pooling (nem is biztos, hogy a tételhez tartozik)

- **Regularization technique**
- Pooling layers summarize the outputs in the same kernel map.



Unpooling

- Ez volt korábban, nem árt megemlíteni

12. topic

Perceptron convergence theorem - no proof required

- **Assumptions:**
  - $w(0) = 0$
  - input space **linearly separable**, therefore $\exists\, w_o$

$$x \in X^+ : \quad w_o^T x > 0 : \; d = 1$$
$$x \in X^- : \quad w_o^T x < 0 : \; d = 0$$

  - Let us denote $\tilde{x} = -x$

$$\tilde{x} \in \tilde{X}^- : \quad w_o^T \tilde{x} > 0 : \; d = 1$$

- **The perceptron convergence theorem basically states that the perceptron learning algorithm converges in finite number of steps, given a linearly separable dataset.**

$$n_{\max} = \frac{\beta \|w_0\|^2}{\alpha^2} \qquad \alpha = \min_{x(n) \in \{X^+, \tilde{X}^-\}} w_o^T x(n) \qquad \beta = \max_{x(k) \in \{X^+, \tilde{X}^-\}} \|x(k)\|^2$$

Momentum optimizer

- Stimulate a **unity weight mass, having v velocity** (follow Newton's laws of dynamics)
- **Update rule:**

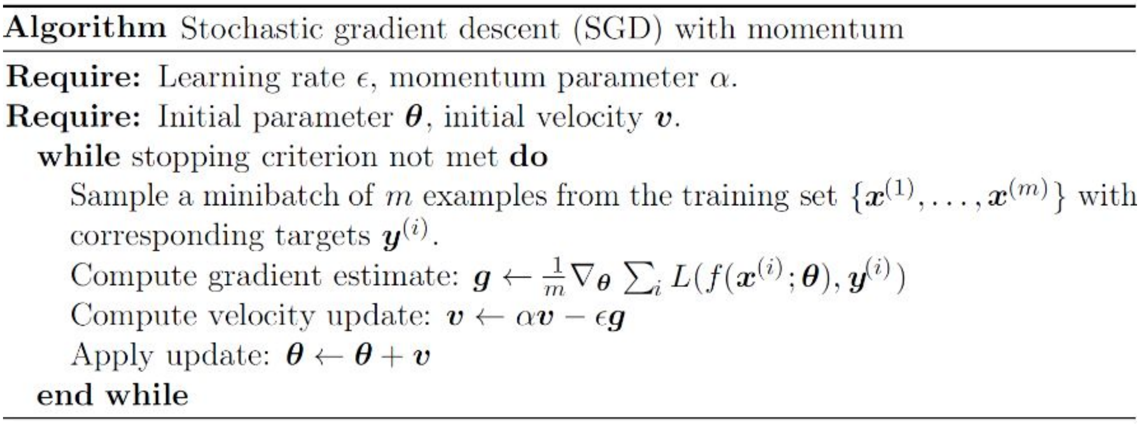
$$\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \nabla_{\boldsymbol{\theta}} \left(\frac{1}{m} \sum_{i=1}^{m} L(\boldsymbol{f}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)}) \right),$$
$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}.$$

- The **v accumulates the gradient elements:**

$$\nabla_{\boldsymbol{\theta}} \left(\tfrac{1}{m} \sum_{i=1}^{m} L(\boldsymbol{f}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)}) \right)$$

- The **larger** α **is relative to** ε **the more previous gradients affect the current direction.**
- **Terminal velocity** is applied when it finds **descending gradient permanently:**

$$\frac{\epsilon ||\boldsymbol{g}||}{1 - \alpha}$$



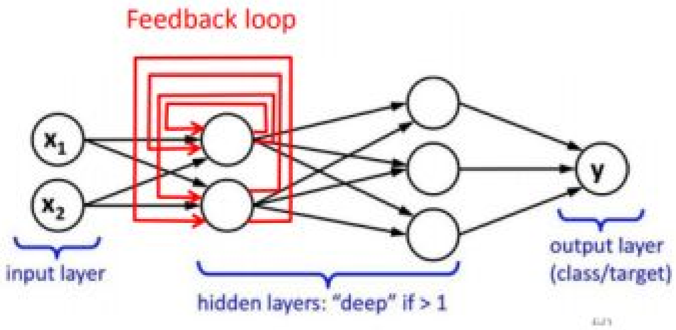
```
Algorithm Stochastic gradient descent (SGD) with momentum
Require: Learning rate $\epsilon$, momentum parameter $\alpha$.
Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$.
  while stopping criterion not met do
    Sample a minibatch of $m$ examples from the training set $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ with
    corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient estimate: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{g}$
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$
  end while
```

https://mlfromscratch.com/optimizers-explained/#/

Recurrent neural network examples: predicting the next letter, image captioning

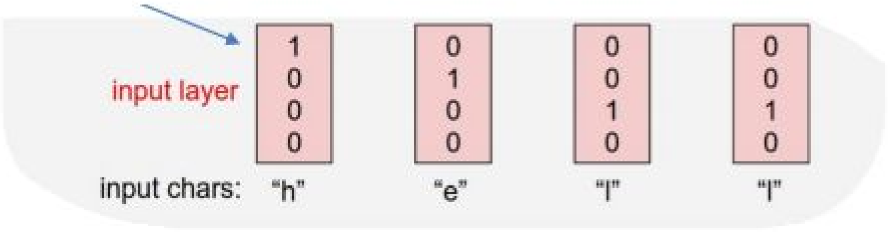
RNN

- **Unlike traditional neural networks the output of the RNN depends on previous inputs.**
  - State
- RNN contains **feedback**
- Theoretically: **directed graph with cyclic loops**
- From now time has a role in execution
  - **Time steps, delays**

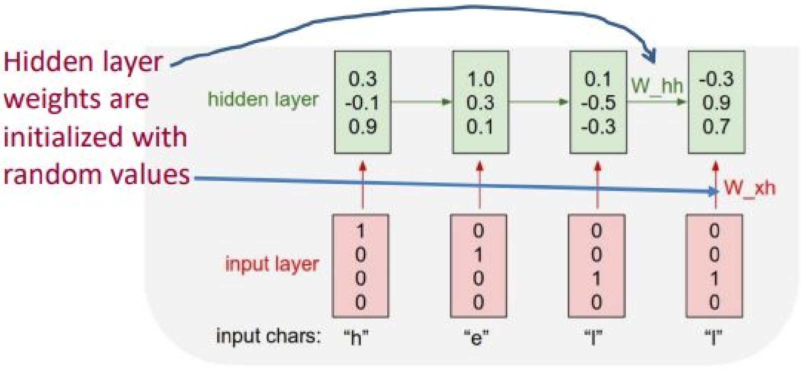


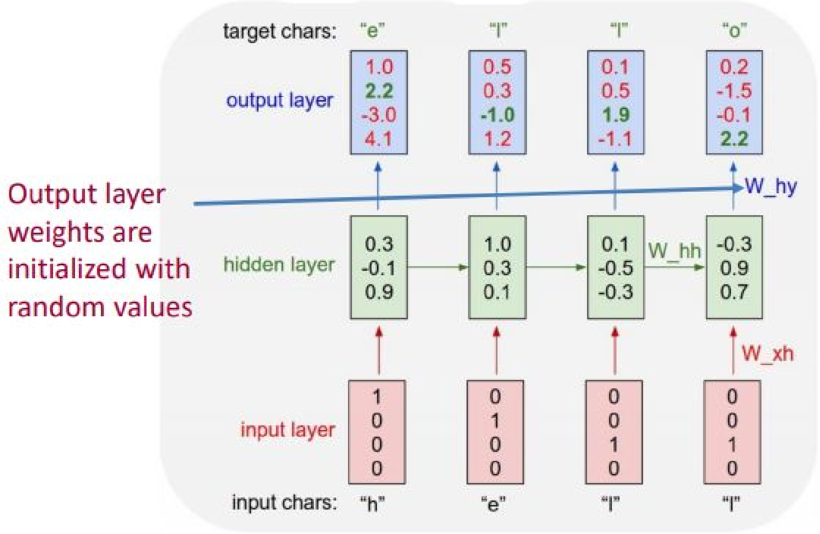
Predicting the next letter

- **Character-level Language Model**
- Vocabulary: [h, e, l, o]
- **Example training sequence: "hello"**

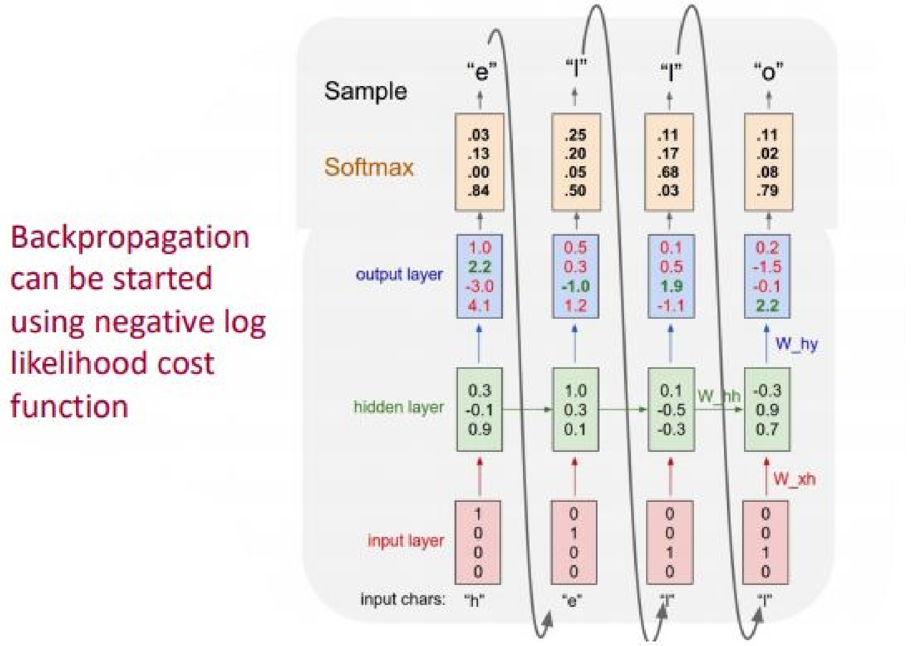


$$\boxed{h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)}$$





Output layer weights are initialized with random values



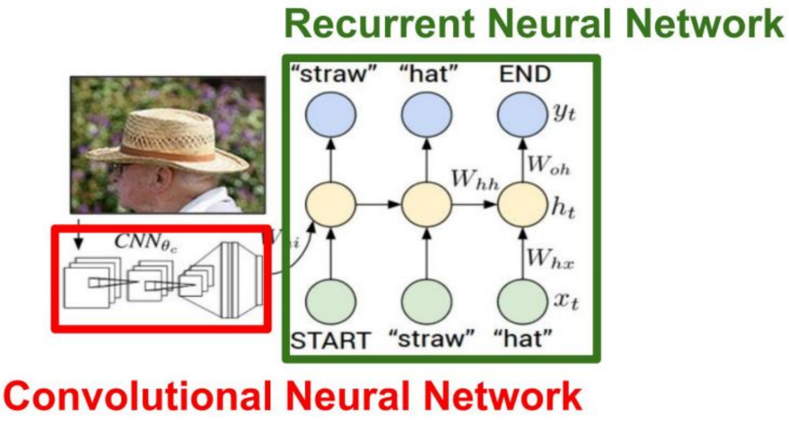
Backpropagation can be started using negative log likelihood cost function

- **At test time sample characters one at a time, feed back to model**

Image captioning (itt nem teljesen értem mit kéne mondani)

- A cél az, hogy írjuk le szép kereken több szóban, hogy mi a lényeg a képen.
- Itt valami olyasmi a lényeg, hogy több komponensből áll a hálózatunk. Van egy része, ami egy egyszerű CNN, ez az ami fel tud ismerni dolgokat és egy RNN ami meg érti a nyelvet. Ők ügyesen összedolgoznak és ki adnak valami értelmeset.

https://www.tensorflow.org/tutorials/text/image_captioning?fbclid=IwAR2AeOV1YPz3EkiOrcJsqDqDLxerqh7aVBXF1UY0quDxKUlf5kxEsKE2nv0

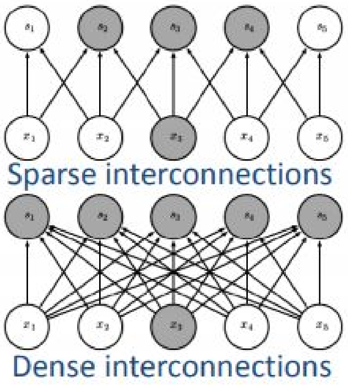


$$h = \tanh(Wxh * x + Whh * h + Wih * v)$$

Properties of convolutional neural networks - sparsity, parameter sharing, equivariance, invariance to shifting

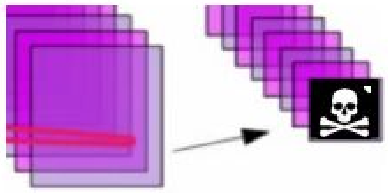
Sparsity

- **The interconnection weights are just a fraction of the fully connected NN**. (the weight matrix between two layers are sparse)
- A few dozen free parameter describes the operation layer
- Receptive field organization similar to neural neuron vision systems



Parameter sharing

- **Same parameters everywhere in the layer**
- Contribution to the gradient of a weight from many positions
- **Reduces the risk of overfitting**
- **Reduces the risk of dying RELU**
  - When it happens, an entire feature extractor on a layer is dying



Variable input size

- **The input size is either resized or padded**
- Input images are resized to the same size

Equivariance

- **Equivariance to translation**
  - The **output shifts with an input shift**
- In a fully connected neural network **each input is a dedicated channel for certain input parameter**-therefore the **inputs cannot be swapped.**
  - Like bank example, one cannot replace the age input with the salary input
- **In CNN, the image can be shifted** because the **inputs are not dedicated and the features are identified anywhere**

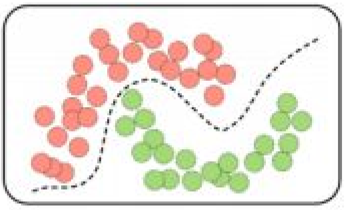
Adversarial attacks

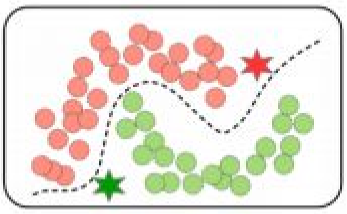
Adversarial Samples

- **Optical illusions for NNs**
- **Special constructed elements which can not be found in the normal input set.**

Adversarial attacks

- **High number of parameters** to optimize & higher dimensional input
- **Network works well, but can't cover all the possible inputs.**
- **One can exploit that there will be regions in the input domain, which were not seen during training.**





Adversarial noise

- A lényeg, hogy valami **kis amplitúdójú gonosz zaj** teljesen megváltoztatja a háló eredményét, pedig az **emberi szemnek még csak nem is látható.**
- Knowing a trained network one can identify modifications (which not happen in real life), which **change the network output completely**
- Luckily this **low amplitude noise is not robust enough in real life** (lens distortion etc.)

Sticker based adversarial attacks

- **High intensity noise concentrated on a small region of the input image:** positions: (x,y), size: (w,h) of stickers

$$C_d = N\left(I + \sum_{i=1}^{k} St_i\left(x_i, y_i, w_i, h_i\right) + \sum_{j=1}^{l} St_j\left(x_j, y_j, w_j, h_j\right) \right)$$

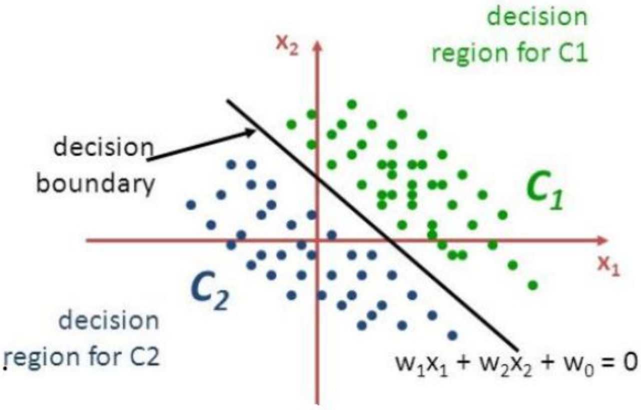
- These attacks are **robust enough to be applied in practical applications.**
- Convnet can't be used?

Itt egy szép példa ilyen matricás támadásra:
https://www.youtube.com/watch?v=MIbFvK2S9g8
tök jól látszik benne, hogy ez a kis kép teljesen láthatatlanná tud tenni a hálók számára.

13. topic

Elementary set separation by a single neuron

- (volt már amúgy 4. topicban részben)
- Use **step function as activation.** (binary output)
- In a 2D input space the **hyperplane is a straight line.**
- **Above the line is classified 1**
- **Below the line is classified 0**



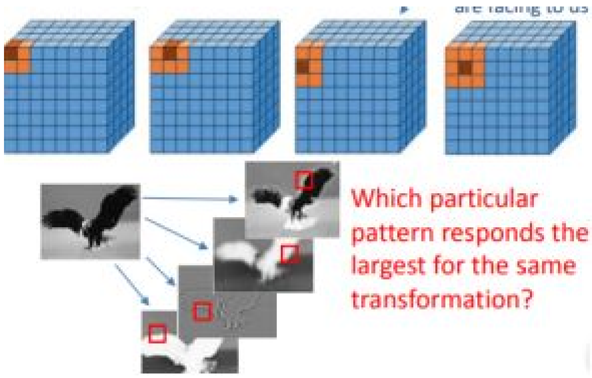
- Neuron with m inputs has an m dimensional input space.
- Neuron makes a linear decision for a **2 class problem**
- The decision boundary is a hyperplane defined:

$$\mathbf{w}^T\mathbf{x} = 0$$

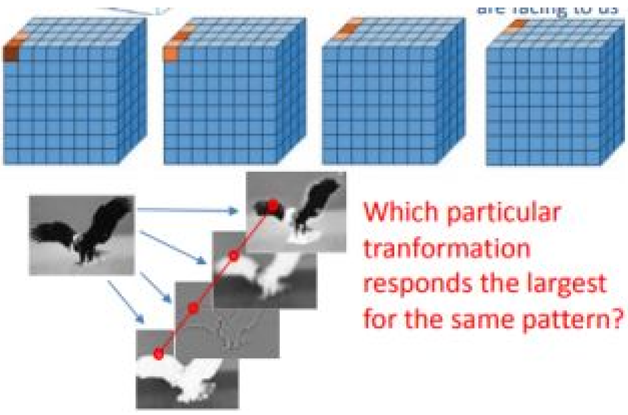
- **Why hyperplane?**
  - Most logic functions has this complexity.
  - Common in mathematical and computational tasks
  - Using multiple hyperplanes -> more complex decision boundary.
- **Two sets are linearly separable if there exists at least one hyperplane in the space with all of the blue points on one side of the line and all the red points on the other side.**

Local response normalization

- Implementation of the Lateral inhibition from neurobiology
  - If a neuron **starts spiking strongly in a layer it inhibits (suppresses) the of the neighboring cells**
  - Winner take all (have a strong decision)
  - Balances the asymmetric responses in neurons in different areas of the layer
- **Useful when we are dealing with ReLU neurons**
  - **Normalizes the unbounded activation of the ReLU neurons**
    - **Avoids concentrating and delivering large values through layers**
  - It enhances high spatial frequencies by suppressing the local neighbors of the strongest neuron
- **Intra map normalization:**
  - 2D normalization within the same feature map
  - Balancing for different areas
  - Winner-take-all for neighbouring neurons in the same feature map (strongest response to the same transformation should win)


are facing to us
Which particular pattern responds the largest for the same transformation?

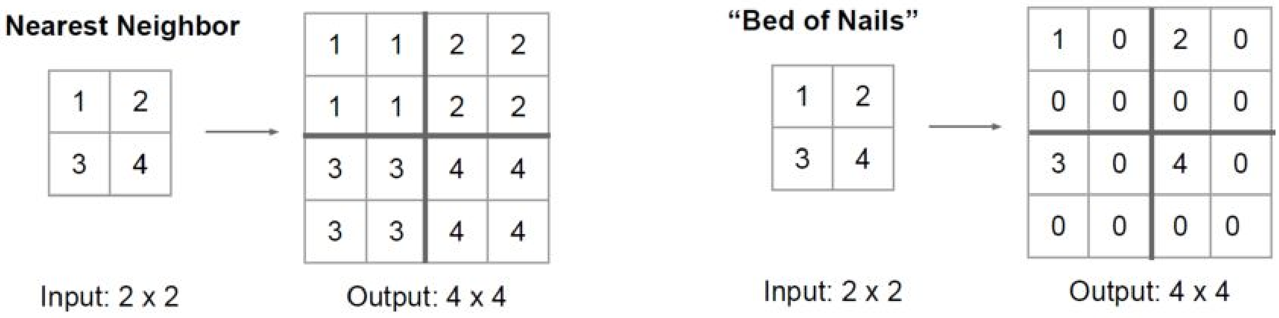
- **Inter map normalization:**
  - Normalization between the neighboring feature maps
  - Winner take all for the largest response with different transformation for the same input location


are facing to us
Which particular tranformation responds the largest for the same pattern?

https://towardsdatascience.com/difference-between-local-response-normalization-and-batch-normalization-272308c034ac

Unpooling

- Upsamplinghez jó.
- Transposed convolution-nél a 8-as topicban volt



Nearest Neighbor

Input: 2 x 2

1	2
3	4

Output: 4 x 4

1	1	2	2
1	1	2	2
3	3	4	4
3	3	4	4

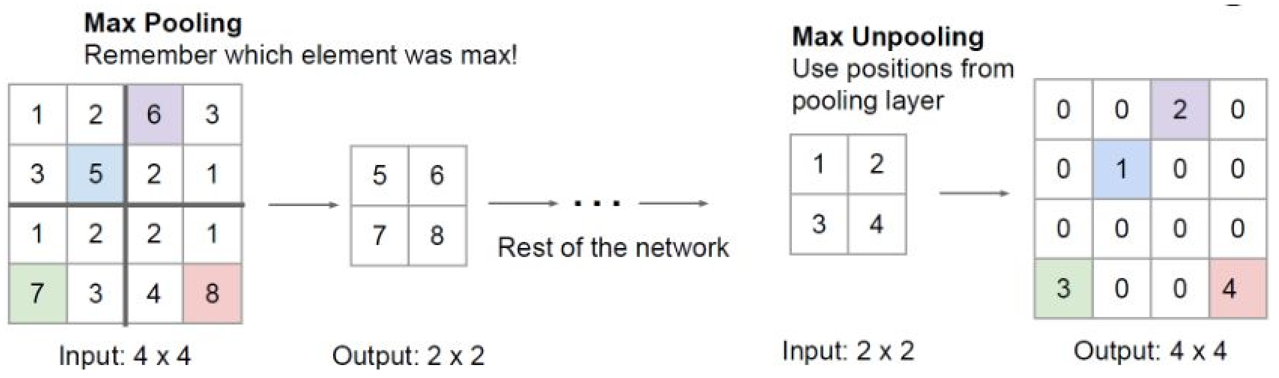
"Bed of Nails"

Input: 2 x 2

1	2
3	4

Output: 4 x 4

1	0	2	0
0	0	0	0
3	0	4	0
0	0	0	0



Max Pooling
Remember which element was max!

Input: 4 x 4

1	2	6	3
3	5	2	1
1	2	2	1
7	3	4	8

Output: 2 x 2

5	6
7	8

Rest of the network

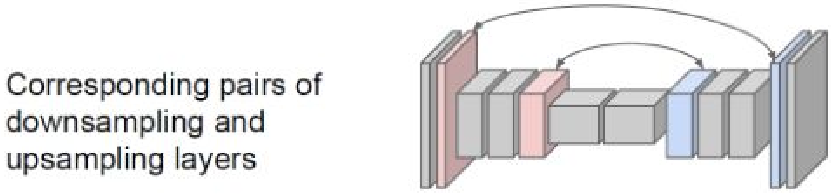
Max Unpooling
Use positions from pooling layer

Input: 2 x 2

1	2
3	4

Output: 4 x 4

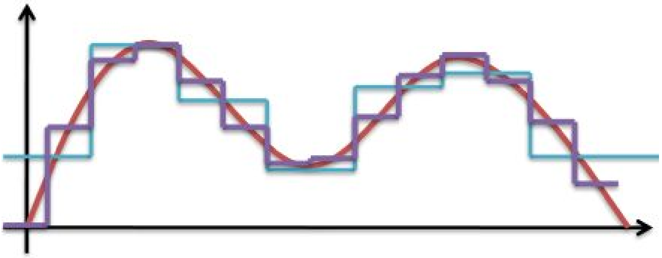
0	0	2	0
0	1	0	0
0	0	0	0
3	0	0	4



Representations - Blum and Li theorem, construction

$$F(x) \in L^2$$
$$\forall \varepsilon > 0, \exists \mathbf{w}$$
$$\int \underset{\mathbf{X}}{\cdots} \int \left(F(\mathbf{x}) - Net(\mathbf{x}, \mathbf{w}) \right)^2 \mathbf{d}x, \ldots \mathbf{d}x_N < \varepsilon$$

- **Every function in L^2 can be represented arbitrarily closely approximation by a neural net.**
- **Proof**:
  - From **elementary integral theory**: Every function can be approximated by appropriate step function sequence.
    - The step function can have arbitrary narrow steps
    - Eg. each step could be divided into two sub-steps
    - Therefore we can synthetize a function with arbitrary precision



$$I(X) = \begin{cases} 1 & \text{if } \mathbf{x} \in X \\ 0 & \text{else} \end{cases}$$
$$F(x) \cong \underbrace{\sum_i F(x_i) I(x_i)}_{s(x)}$$

- The construction:
  - Has **no dimensional limit**
  - Has no equidistance restrictions on tiles (partitions)
  - **can be further fined, and the approximation can be any precise**.
- **Limitations**
  - **The size of the FFNN is quite big**
  - **The network is synthetized, the weights are generated**
- Úgy amúgy az L^P definíciója mellékesen:

$$L^1 : \int \underset{\mathbf{X}}{\cdots} \int \left(F(x) \right) \mathbf{d}x, \ldots \mathbf{d}x_N < \infty$$
$$L^2 : \int \underset{\mathbf{X}}{\cdots} \int \left(F(x) \right)^2 \mathbf{d}x, \ldots \mathbf{d}x_N < \infty$$
$$L^p : \int \underset{\mathbf{X}}{\cdots} \int \left(F(x) \right)^p \mathbf{d}x, \ldots \mathbf{d}x_N < \infty$$

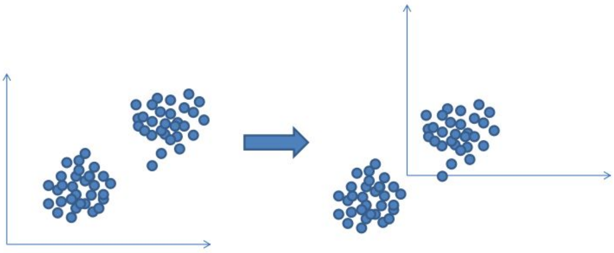
14. topic

Principal component analysis (PCA)

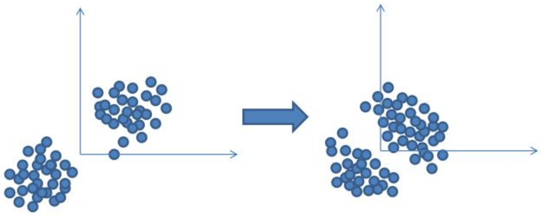
- **Technique for dimensionality reduction**
  - **Goal**: improve ML **performance**, **compress** data, **visualise** data etc.
- Linear coordinate transformation
  - **converts a set of observations of possibly correlated variables**
  - into a **set of values of linearly uncorrelated orthogonal variables: principal components**
- **Deterministic** algorithm
- The idea is to **project the data onto a subspace which compresses most of the variance in as little dimensions as possible**
- Each new dimension is a **principle component**
- The principal components **are ordered** according to **how much variance in the data** they capture.

Steps of PCA:

1. **Mean normalization**: For every value in the data, subtract its mean dimension value. This makes the average of each dimension zero.



2. **Standardization (optional)**: Only if you want to have each features in the same variance



3. **Covariance matrix**: Calculate the covariance matrix
4. **Eigenvectors and eigenvalues of the covariance matrix**
5. **Rank** eigenvectors by eigenvalues
6. **Keep top k eigenvectors** and stack them to form a feature vector
7. **Transform data to PCs**

   New data = feature vectors (transposed) * original data

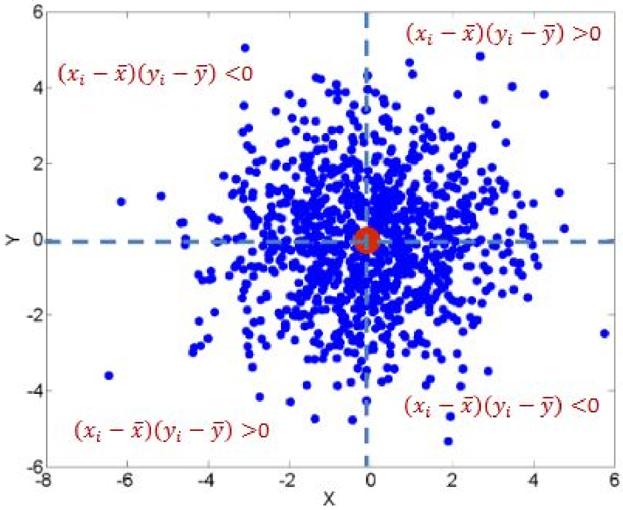
Covariance calculation

$$\text{Variance(x)}=\frac{1}{n}\sum_{i=1}^{n}(x_i-\bar{x})^2$$

$$=\frac{1}{n}\sum_{i=1}^{n}(x_i-\bar{x})(x_i-\bar{x})$$

$$\text{Covariance}(x,y)=\frac{1}{n}\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y})$$

Positive covariance if y_1 -y < 0 and x_1 - x < 0, negative is >0, or no covariance if its zero.



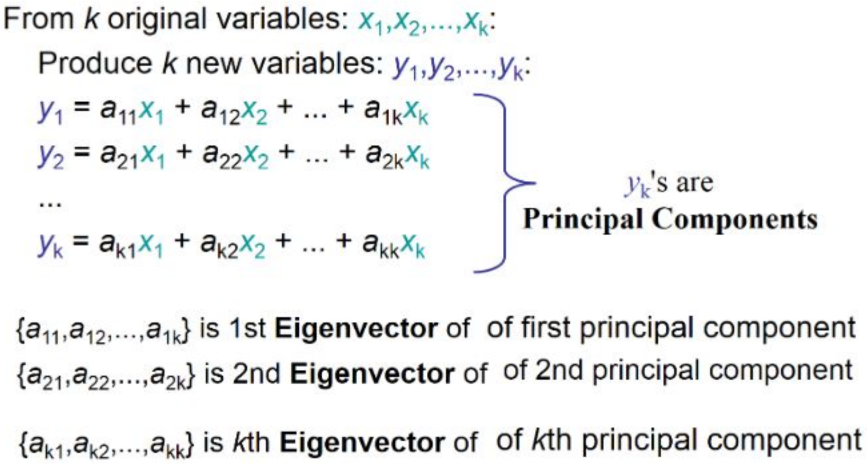
Covariance matrix

- **Diagonal elements are variances.**
- The matrix is **symmetric**.
- m: dimension, n: number of vectors

$$Cov\ (\Sigma) = \begin{bmatrix} cov(x_1,x_1) & cov(x_1,x_2) & \cdots & cov(x_1,x_m) \\ cov(x_2,x_1) & cov(x_2,x_2) & \cdots & cov(x_2,x_m) \\ \vdots & \vdots & \vdots & \vdots \\ cov(x_m,x_1) & cov(x_m,x_2) & \cdots & cov(x_m,x_m) \end{bmatrix}$$

$$Cov\ (\Sigma) = \frac{1}{n}(X-\bar{X})(X-\bar{X})^T; where\ X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}$$

Eigenvectors and principal components



From k original variables: $x_1,x_2,\ldots,x_k$:

Produce k new variables: $y_1,y_2,\ldots,y_k$:

$y_1 = a_{11}x_1 + a_{12}x_2 + \ldots + a_{1k}x_k$

$y_2 = a_{21}x_1 + a_{22}x_2 + \ldots + a_{2k}x_k$

...

$y_k = a_{k1}x_1 + a_{k2}x_2 + \ldots + a_{kk}x_k$

y_k's are **Principal Components**

$\{a_{11},a_{12},\ldots,a_{1k}\}$ is 1st **Eigenvector** of of first principal component

$\{a_{21},a_{22},\ldots,a_{2k}\}$ is 2nd **Eigenvector** of of 2nd principal component

$\{a_{k1},a_{k2},\ldots,a_{kk}\}$ is kth **Eigenvector** of of kth principal component

How many PCs to use?

- **Proportion of variance for each feature**: (where **lambda_i** are the **eigenvalues**)

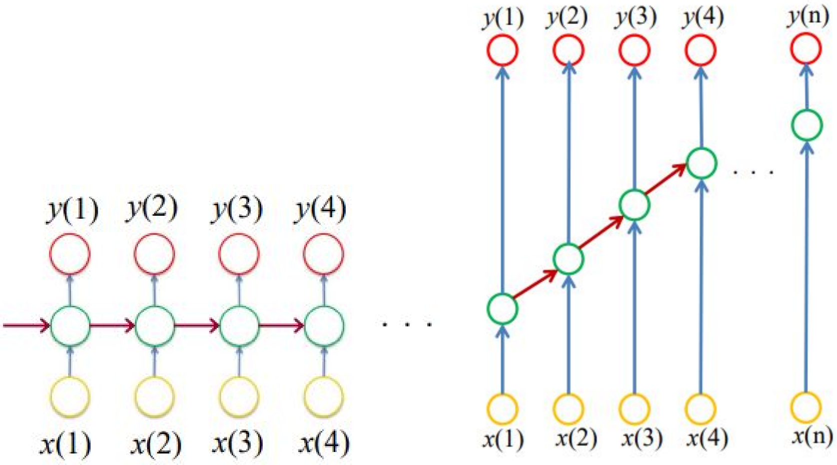
$$\frac{\lambda_i}{\sum_{i=1}^{n}\lambda_i}$$

- Reach a **predefined threshold**
- Or find the elbow of the Scree plot

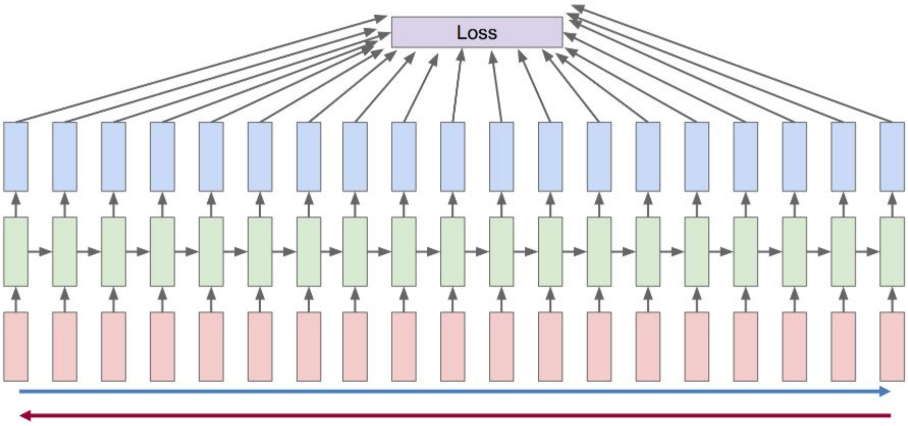
https://www.youtube.com/watch?v=TJdH6rPA-TI
https://towardsdatascience.com/a-one-stop-shop-for-principal-component-analysis-5582fb7e0a9c
http://setosa.io/ev/principal-component-analysis/

Backpropagation through time

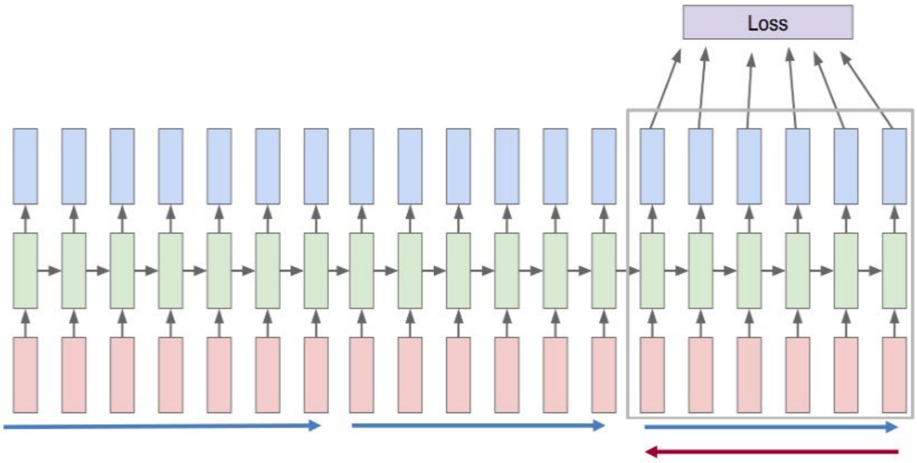
- Backpropagation through time (BPTT) is a gradient-based technique for training certain types of recurrent neural networks
- **Assuming that the length of the input vector sequence is limited**
- It became a feedforward neural net
- Possible to apply back propagation
- We need **multiple vector sequences to train!**



- **Forward through entire sequence to compute loss, then backward through entire sequence to compute gradient.**



- **Truncated backpropagation: Run forward and backward through chunks of the sequence instead of whole.**



https://machinelearningmastery.com/gentle-introduction-backpropagation-time/

Stochastic gradient descent optimizer

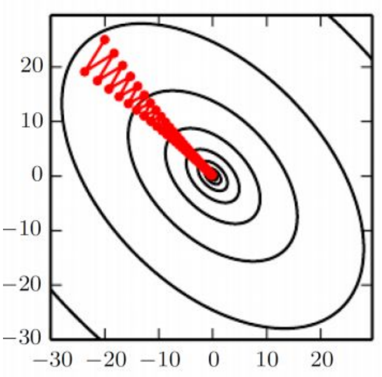
- **Gradient descent, with mini-batches and changing learning rate during the iteration**
- Decreases learning rate during the training to reduce overshoot.
- **Very slow** :(
- **Sufficient conditions to guarantee convergence of SGD:**

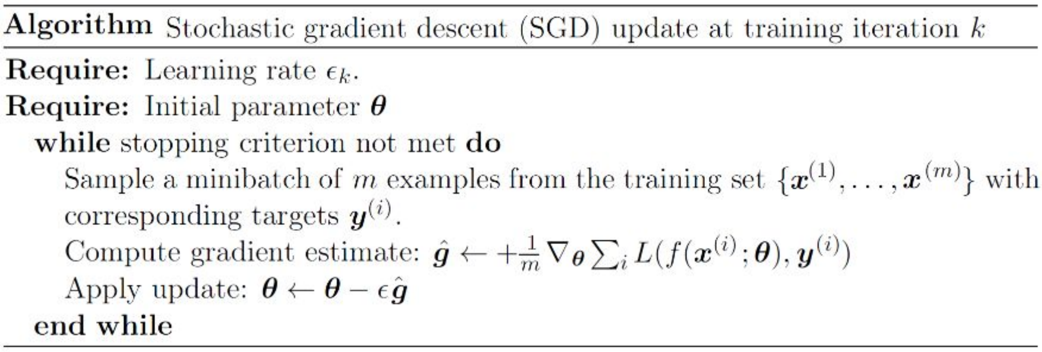
$$\sum_{k=1}^{\infty} \epsilon_k = \infty, \quad \text{and} \quad \sum_{k=1}^{\infty} \epsilon_k^2 < \infty$$

- **In practice:**

$$\epsilon_k = (1-\alpha)\epsilon_0 + \alpha\epsilon_\tau \qquad \alpha = \frac{k}{\tau}$$

- After iteration TAU, it is common to leave the epsilon constant
- Let's consider a very **elongated quadratic function resembles a long canyon.**
- Gradient descent **wastes time repeatedly descending canyon walls, because they are the steepest feature.**
- Because the **step size is somewhat too large, it has a tendency to overshoot the bottom of the function and thus needs to descend the opposite canyon wall on the next iteration.**



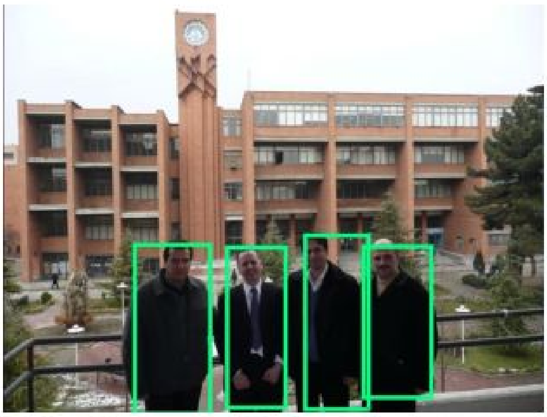


```
Algorithm Stochastic gradient descent (SGD) update at training iteration $k$
Require: Learning rate $\epsilon_k$.
Require: Initial parameter $\boldsymbol{\theta}$
  while stopping criterion not met do
    Sample a minibatch of $m$ examples from the training set $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ with
    corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient estimate: $\hat{\boldsymbol{g}} \leftarrow +\frac{1}{m}\nabla_{\boldsymbol{\theta}}\sum_i L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon\hat{\boldsymbol{g}}$
  end while
```

Object detection problem explained

- **Object classification**: we make only one decision per image (what's on the picture) (eg. Alexnet)

- **Detection and localization is more complex**: we make multiple decision per image (regressions for localization and classification for detection)
    - PASCAL object recognition database and challenge
        - Annotated image database
    - eg. R-CNN, Fast R-CNN, Faster R-CNN :D



- **The goal of object detection is to detect the presence of object from a certain set of classes, and locate the exact position in the image.**
- We can informally divide all objects into two big groups: **things and stuff.** Things are objects of certain size and shape like **cars, bicycles, people, animals, planes**. We can specify where object is located in image with a **bounding box. Stuff is more likely a region of image** which correspond to objects like **road, or grass, or sky, or wate**r. It is easier to specify the location of a sky **by marking the region in an image, not by a bounding box.**
- **Chicken and egg problem:**
    - You need to know that it is a **bicycle** before able to say that **both a wheel part and a pipe segment belongs to the same object.**
    - You need to **know that the red box contains an object before you can recognize it. (cannot recognize a bicycle if you try it from separated parts)**
    - Our brain does it parallel
    - How neural nets can solve it?
        - **Detection by regression?**
            - **Bounding boxes**
            - **Region proposals** (find "blobby" image regions that are likely to contain objects)
        - **Detection by classification?**

Effects of filter size on convolution

Filter size considerations

- **Small** field of view -> accurate **localization**
- **Large** field of view -> context **assimilation**
- Effective filter size increases

$$n_o\colon k \times k \quad \rightarrow \quad n_a\colon \big(k + (k-1)(r-1)\big) \times \big(k + (k-1)(r-1)\big)$$

n_o : original convolution kernel size

n_a : atrous convolution kernel size

r: rate

- However we take into account only the non-zero filter values:
    - Number of filter parameters is the same
    - Number of operations per position is the same

15. topic

Rosenblatt perceptron training algorithm

Learning:

- **Annotated dataset**: $x_j \to d_j$
- Given the parametric **equation of the perceptron**: $y = sign(w^T x)$
- **Goal**: find the **optimal** w_{opt} values, **where** $d_j = sign({w_{opt}}^T x_j)$
- Training set: $X^+ = \{x : \ d =+ 1\}$ $X^- = \{x \ : \ d = 0\}$ (these **are linearly separable)**
- You can have a test set, which will be used for testing and scoring the result.

$$X^+ = \left\{\mathbf{x} : \ \mathbf{w}_{\text{opt}}^{\text{T}}\mathbf{x} > 0\right\}$$
$$X^- = \left\{\mathbf{x} : \ \mathbf{w}_{\text{opt}}^{\text{T}}\mathbf{x} < 0\right\}$$

The algorithm

1. **Initialization**
   Set $w(0) = 0$ $w(0) = rand$
2. **Activation**
   Select $x_k \to d_k$ pair
3. **Computation of actual response**
   $y_k = sign(w^T(k) \cdot x(k))$
4. **Adaptation of the weight vector**
   $w(k+1) = \Psi(x(k),\ w(k),\ d(k),\ y(k))$
5. **Continuation**
   Until all responses of the perceptron are OK

Back-propagation

Sajnos itt nem teljesen pontosan 100%ig tudom, hogy pontosan mi az amire kíváncsiak. :(

Introduction

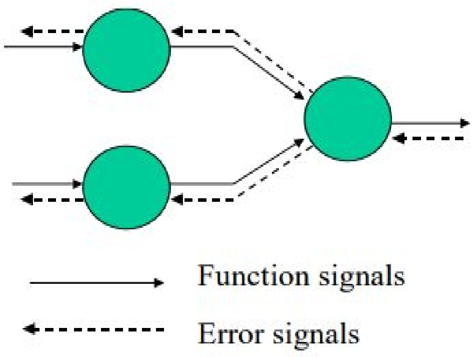
- Back propagation is a technique to train NN-s, an algorithm, to adjust the program itself, according to it's past function.
- We calculate the gradient of the loss function at output, as a result we see how much each element affects the output.
- Backpropagation refers to two things:
  - The mathematical method used to calculate derivatives and an application of the derivative chain rule.
  - The training algorithm for updating network weights to minimize error.

The goal of the backpropagation training algorithm is to modify the weights of a neural network in order to minimize the error of the network outputs compared to some expected output in response to corresponding inputs.

Hmmm

- The **Rosenblatt algorithm is inapplicable**,

  - **the error and desired output in the hidden layers of the FFNN is unknown**
- Someway the **error of the whole network has to be distributed to the internal neurons, in a feedback way**



- **Forward propagation of function signals and back-propagation of errors signals**
- **Sequential back propagation**
  - Adapting the weights of the FFNN (recursive algorithm)

$$\mathbf{w}_{ij}^{(l)}(k+1) = w_{ij}^{(l)}(k) + \Delta w_{ij}^{(l)}(k)$$

$$\Delta w_{ij}^{(l)}(k) = ?$$

- The **weights are modified towards the differential of the error function (delta rule):**

$$\Delta w_{ij}^{(l)} = -\eta \frac{\partial R_{emp}}{\partial w_{ij}^{(l)}}$$

- The elements of the training set adapted by the **FFNN sequentially:**

$$R_{emp} = R_{emp}(y(\mathbf{x}), d)$$

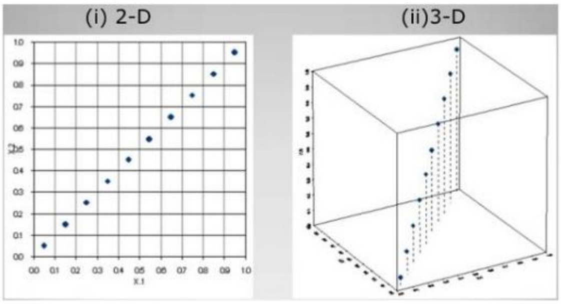
Back-propagation

- Tough we showed how to modify the weights with back propagation, its most important value that **it can calculate the gradient.**
- The **weight updates can be calculated with different optimization methods, after the gradients are calculates**
- **These methods can speed up training drastically.**

https://www.youtube.com/watch?v=XE3krf3CQls

Curse of dimensionality

- **What is it?**
  - A name for **various problems that arise** when analyzing data in **high dimensional space.**
  - **Dimensions = independent features in ML**
    - Input vector size (different measurements, or number of pixels in an image)
  - **Occurs when d (# dimensions) is large in relation to n (number of samples).**
- **Real life examples:**
  - **Genomics**
    - We have ~20k genes, but disease sample size are often in the 100s or 1000s
- **Sparse data:**
  - When dimensionality **d increases**, the **volume of the space increases so fast**, that the available **data becomes sparse** (i.e. few points in large space)



- **Noisy data:**
    - More features can lead to increased noise -> it is harder to find the true signal
- **Less clusters:**
    - **Neighborhoods** with fixed k points **are less concentrated** as **d increases**.
- **Complex features**:
    - High dimensional functions tend to have **more complex features than low dimensional functions and hence harder to estimate**
- Running complexity:
    - Many data points (labeled measurements) are needed
    - **Complexity (running time) increase with dimension d**
    - A lot of methods have at least O(n*d^2) complexity), where n is the number of samples
    - As d becomes large this complexity becomes **very costly**
- **Distances in high dimensions:**
    - 2D vs 100D
    - **Euclidean distance become meaningless, most two points are "far" from each other**

Mathematical effects

- **Ratio between volume of a sphere and a cube**

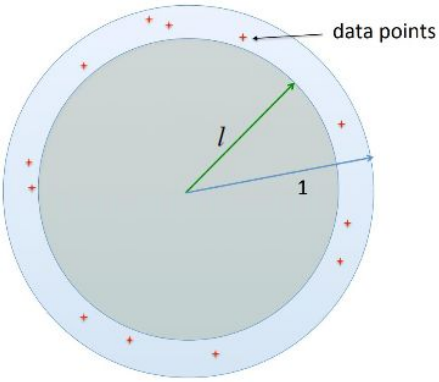
$$\frac{(\frac{4}{3})\pi r^3}{(2r)^3} \approx \frac{4r^3}{8r^3} \approx 0.5$$

d	3	5	10	20	30	50
ratio	0.52	0.16	0.0025	2.5E-08	2.0E-14	1.5E-28

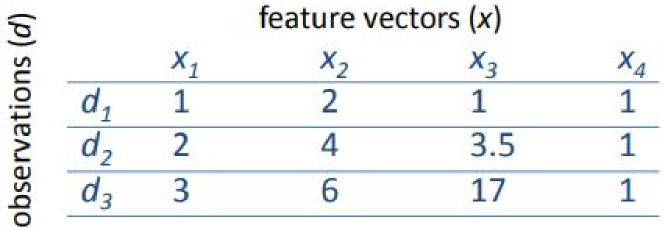
- Most of data is in the corner of the cube
    - **Euclidean distance is meaningless, most two points are "far" from each other**
- k-NN classification and k-means becomes problematic, most of the neighbors are equidistant

The nearest neighbor problem in a sphere

- **Assume randomly distributed points in a sphere with a unit diameter**
- **The median of the nearest neighbors is l**
- **As dimension tends to infinity, the median converges to 1**



How to calculate dimensionality?



feature vectors (x)

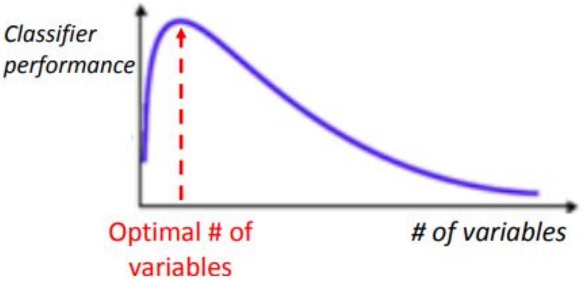
observations (d)

	x_1	x_2	x_3	x_4
d_1	1	2	1	1
d_2	2	4	3.5	1
d_3	3	6	17	1

- Basically **how many independent coordinates.** (x1 és x2 fele egymásnak -> kuka, x4 konstans (semmi infó benne)

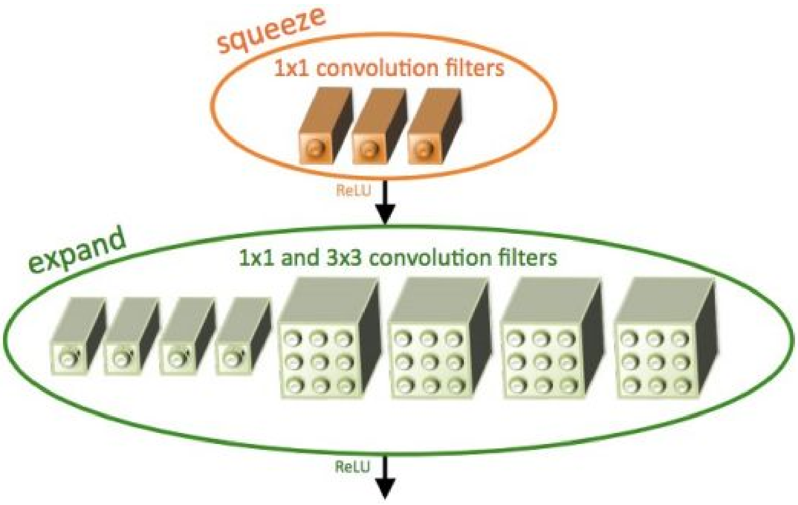
How to avoid the curse?

- **Reduce dimensions**
  - **Feature selection - Choose only a subset of features**
  - Use algorithms that transform the data into lower dimensional space **(PCA, t-SNE)**
- **Less is more**
  - In many cases the **information that is lost by discarding variables is made up by a more accurate mapping / sampling in the lower dimensional space**



SqueezeNet

- **Network architecture**
- **Depths are squeezed before each operation**
- The expand is done by the **concatenation of the 1x1 and 3x3 convolutions.**
- **Advantage: the expand layer is saved.**



- In a SqueezeNet architecture we will use a linear approximation of 128 feature maps, using 16 independent feature maps
- From the linear combination of these elements the new maps are created

Backpropagation and gradient-based optimizers

- Backpropagation már volt.
- Optimizerek is: Momentum, AdaGrad, Adam, RMSProp etc.
- Én azokról beszélnék sokat szerintem.
- **The backpropagation algorithm is an instruction set for computing the gradient of a multivariable function.**
- The **Adam optimizer** is a specialized gradient-descent algorithm that **uses the computed gradient, its statistics, and its historical values** to take small steps in its opposite direction inside the input parameter space as a means of minimizing a function. It is used for **optimization in neural network training.**
- In other words, the **Adam optimizer** would need to use an algorithm like the **backpropagation** algorithm to first **compute the gradient of the function**. Then the Adam optimizer would use this computation to perform gradient-descent in a specialized manner