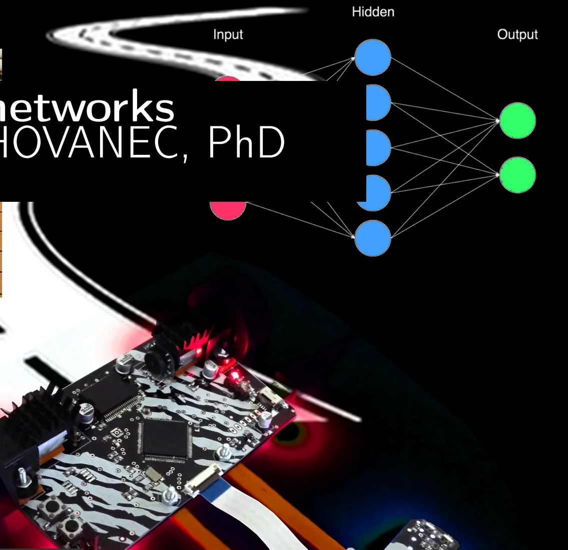
$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left[r_{t+1} + \lambda \max_{a} Q(s_{t+1}, a) - Q(s_t, a_t)\right]$$

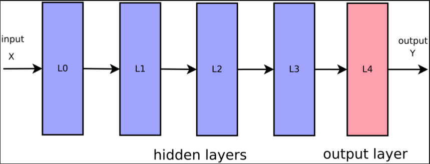
(The New Action Value = The Old Value) + The Learning Rate × (The New Information − the Old Information)

Neural networks

Michal CHOVANEC, PhD

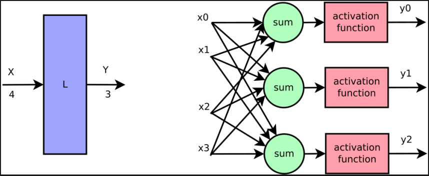


Neural network



- universal function approximator
- learning from examples
- training using parallel architectures NVIDIA GPU, TPU - from weeks to months
- inspired by human brain
- thousands of connected neurons

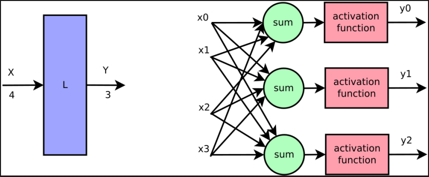
Layer



$$y_j = f(\sum_{i=0}^{N-1} X_i W_{ji} + b_j)$$

where
$f(x)$ is activation function
X input vector
W weights matrix
b bias vector
Y output vector

Layer

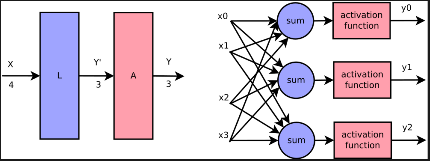


$$y_0 = f(\sum_{i=0}^{3} X_i W_{0i} + b_0)$$

$$y_1 = f(\sum_{i=0}^{3} X_i W_{1i} + b_1)$$

$$y_2 = f(\sum_{i=0}^{3} X_i W_{2i} + b_2)$$

Layer - split activation function



$$y'_0 = \sum_{i=0}^{3} X_i W_{0i} + b_0, \qquad y_0 = f(y'_0)$$

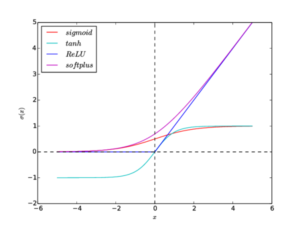
$$y'_1 = \sum_{i=0}^{3} X_i W_{1i} + b_1, \qquad y_1 = f(y'_1)$$

$$y'_2 = \sum_{i=0}^{3} X_i W_{2i} + b_2, \qquad y_2 = f(y'_2)$$

y' is called neuron **Activation**

Activation function

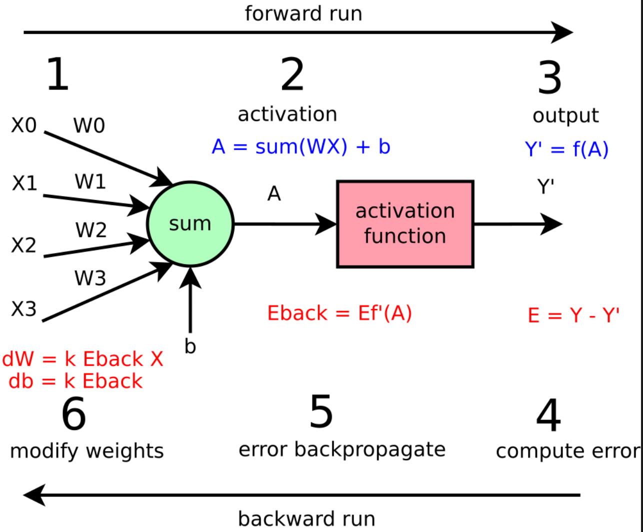
name	function	derivative
Linear	$y = x$	$y' = 1$
Sigmoid	$y = \frac{1}{1+e^{-x}}$	$y' = \frac{e^{-x}}{(1+e^{-x})^2}$
RBF	$y = e^{-x^2}$	$y' = xe^{-x^2}$
Relu	$y = \begin{cases} x, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases}$	$y' = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases}$
LeakyRelu	$y = \begin{cases} x, & \text{if } x > 0 \\ 0.01x, & \text{otherwise} \end{cases}$	$y' = \begin{cases} 1, & \text{if } x > 0 \\ 0.01, & \text{otherwise} \end{cases}$



Training

1. choose random training dataset sample (X, Y)
2. compute neural network output, $\hat{Y} = F(X)$
3. obtain error
   error = **target value** - **computed value**
   $E = Y - \hat{Y}$
4. update weights and biases using error

Training


forward run
1
2
3
activation
output
A = sum(WX) + b
Y' = f(A)
X0
W0
X1
W1
X2
W2
X3
W3
sum
A
activation function
Y'
b
Eback = Ef'(A)
E = Y - Y'
dW = k Eback X
db = k Eback
6
modify weights
5
error backpropagate
4
compute error
backward run

Training

One neuron weights updating example
- consider $Y, \hat{Y}, b$ are scalar values, and W, X are vectors

$$E = Y - \hat{Y}$$

$$J = \underbrace{\left(Y - f(\sum_{i=0}^{N-1} X_i W_i + b)\right)^2}_{\text{cost function to minimize}}$$

$$\frac{\partial J}{\partial W_j} = -2\underbrace{\left(Y - f(\sum_{i=0}^{N-1} X_i W_i + b)\right)}_{\text{Error E}} f'\underbrace{\left(\sum_{i=0}^{N-1} X_i W_i + b\right)}_{\text{Activation A}} X_j$$

$$\frac{\partial J}{\partial W_j} = -2Ef'(A)X_j$$

$$\frac{\partial J}{\partial W_j} \approx W_j(n) - W_j(n-1) = \Delta W_j$$

$$\Delta W_j = \eta E f'(A) X_j$$

Training

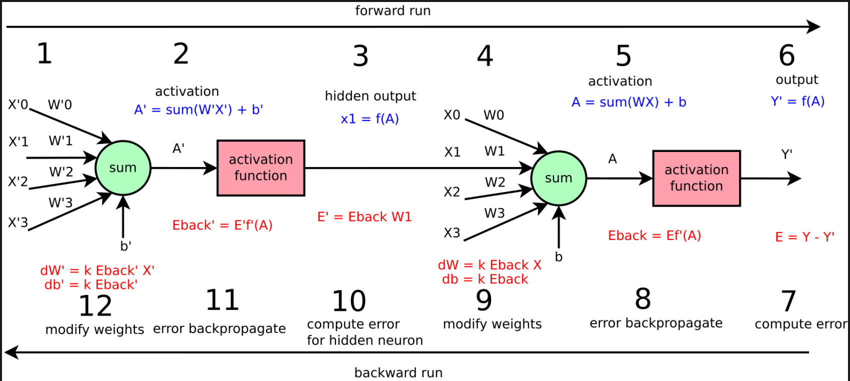
special case for $y(x) = x$

$$\Delta W_j = \eta E X_j$$

special case for $y(x) = RELU(x)$

$$\Delta W_j = \begin{cases} \eta E X_j \text{ if } \sum_{i=0}^{N-1} X_i W_{ji} + b_0 > 0 \\ 0 \text{ otherwise} \end{cases}$$

Error backpropagation



Mnist handwritten numbers classification

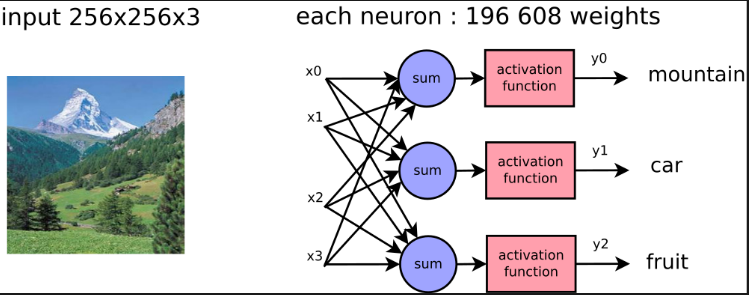
- handwritten numbers, 60000 for training, 10000 for testing
- 28x28 pixels, grayscale 9x9 pixels, mnist tiny
- 10 outputs

SCORING

- more than 99% **A** best networks
- more than 98% **B** average networks
- more than 97% **C** average networks
- more than 96.5% **D** pure networks
- more than 96% **E** pure networks
- otherwise **Fx** linear classifier

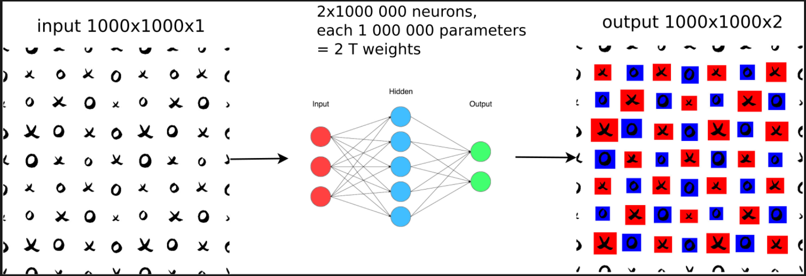
Convolutional neural networks

- goal : recognize image patterns
- input : RGB image, 256x256 pixels
- output : three classes mountain, car, fruit



Convolutional neural networks

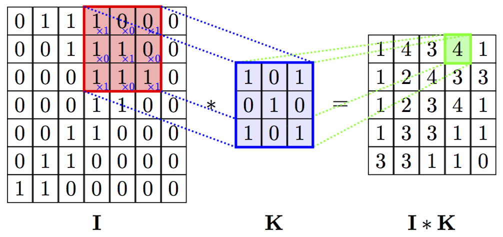
- goal : detect image patterns, X and Os
- input : grayscale image, 1000x1000 pixels
- output : two matrices 1000x1000
  - one for X desctions
  - second for Os detection
- total weights count $2 * (1000^2 * 1000^2) = \mathbf{2T}$



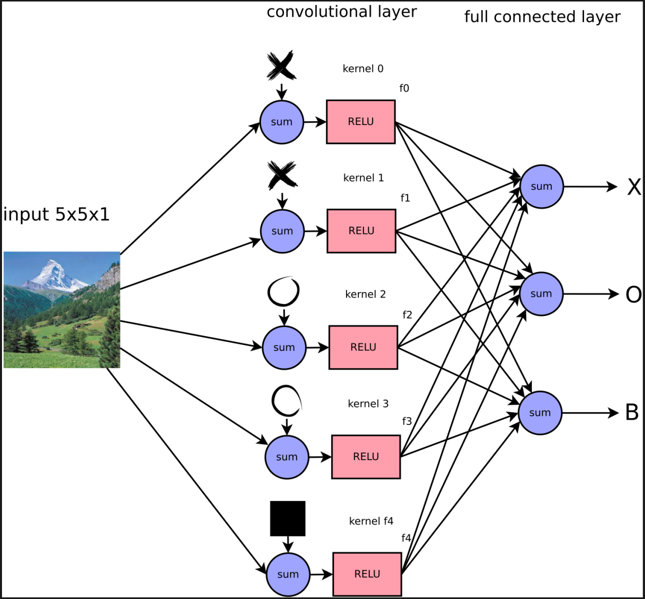
Convolution pattern search

- two neurons
- 25 weights each
- shift 5x5 window over whole 1000x1000 image

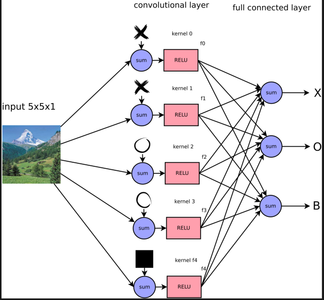
$$\begin{pmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix} \qquad \begin{pmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{pmatrix}$$



Minimal toy CNN

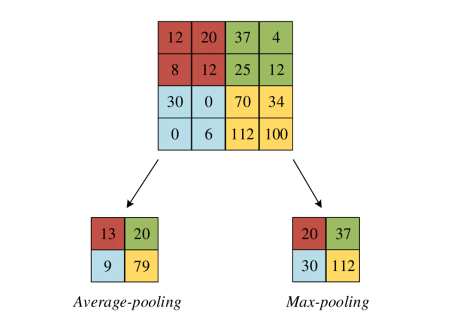


Minimal toy CNN parameters

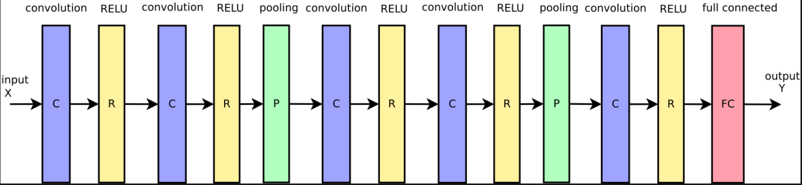


- 5 kernels, 5x5 + 1 parameters each, 130 weights
- 3 full connected, 5 + 1 parameters each, 18 weights
- total parameters to learn **148**
- parameters count is **INVARIANT** to input size
- recognition ability is also **SHIFT INVARIANT**

Pooling - subsampling



Convolutional network example

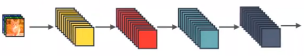


- layers count : 10 .. 100
- kernels size : 1x1, 3x3 or 5x5
- pooling size : 2x2
- activation : ReLU, LeakyReLU

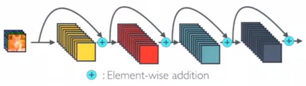
Advanced Convolutional network architectures

- Old methods (before NN) - 2011, 25.8%
- Convolutional, AlexNet - 2012, 16.4%
- Google inception - 2013, 6.7%
- Microsoft ResNet - 2015, 6.1%
- DenseNet - 2018, 5.17%

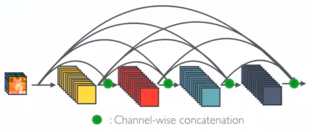
- CNN



- ResNET



- DenseNet



Usefull links

- CHRISTOPHER J.C.H. WATKINS : Q-learning
  http://www.gatsby.ucl.ac.uk/~dayan/papers/cjch.pdf
- Richard S. Sutton : Reinforcement Learning: An Introduction
  https://www.amazon.com/Reinforcement-Learning-Introduction-Adaptive-Computation/dp/0262193981
- Google DeepMind : Playing Atari with Deep Reinforcement Learning
  https://arxiv.org/pdf/1312.5602.pdf
- Google DeepMind : Dueling Network Architectures for Deep Reinforcement Learning
  https://arxiv.org/pdf/1511.06581.pdf
- Google DeepMind :Mastering the Game of Go without Human Knowledge
  https://deepmind.com/documents/119/agz_unformatted_nature.pdf
- Andrej Karpathy : Pong from pixels
  http://karpathy.github.io/2016/05/31/rl/
- Maxim Lapan : Deep reinforcement learning
  https://www.amazon.com/Practical-Reinforcement-Learning-Maxim-Lapan/dp/1788834240
- Mohit Sewak : Practical Convolutional Neural Networks
  https://www.amazon.com/Practical-Convolutional-Neural-Networks-Implement/dp/1788392302
- Densely Connected Convolutional Networks
  https://arxiv.org/pdf/1608.06993.pdf

Q&A

michal chovanec (michal.nand@gmail.com)
www.youtube.com/channel/UCzVvP2ou8v3afNiVrPAHQGg
github https://github.com/michalnand