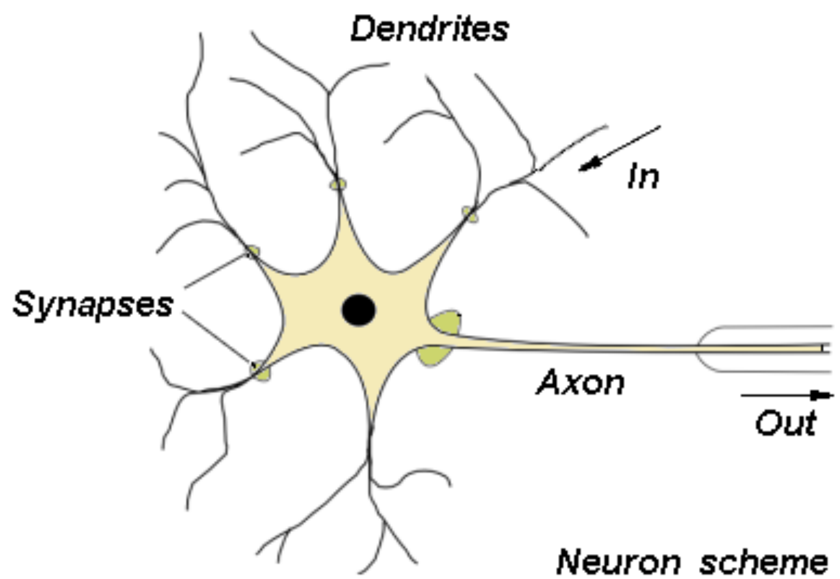
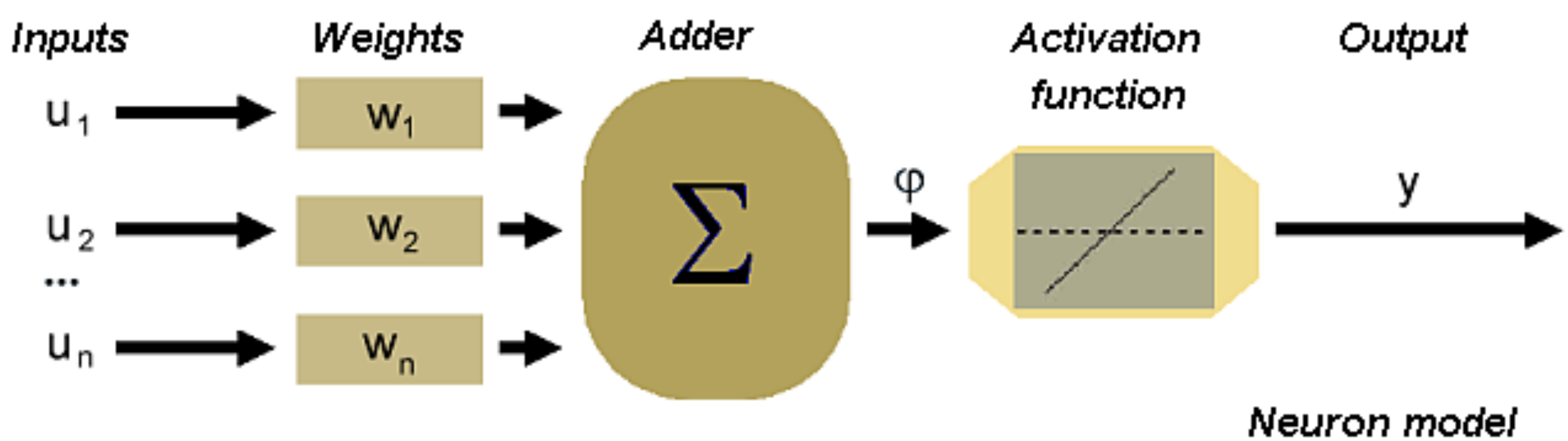
Machine Learning for Applications in Computer Vision

Neural Networks

Perceptron

- The human brain (10^{10} cells) is the archetype of neural networks

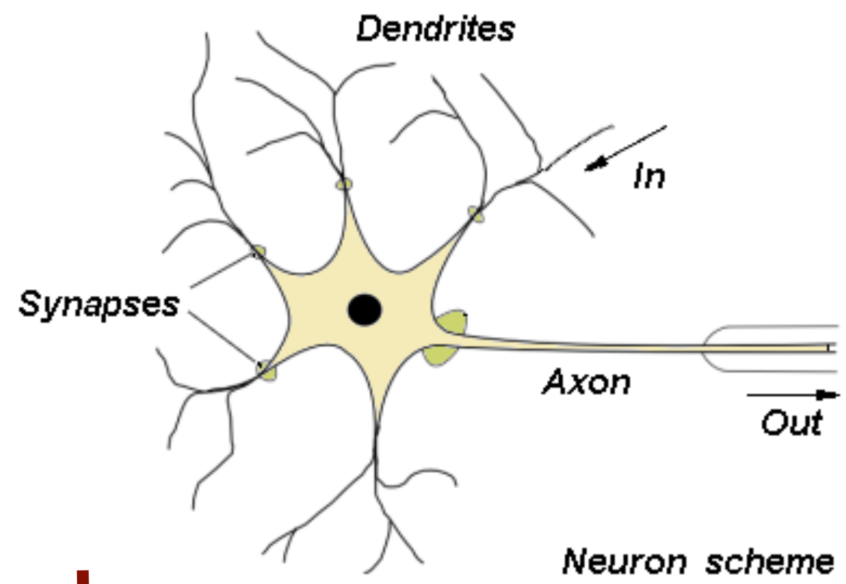




http://home.agh.edu.pl/~vlsi/AI/intro/

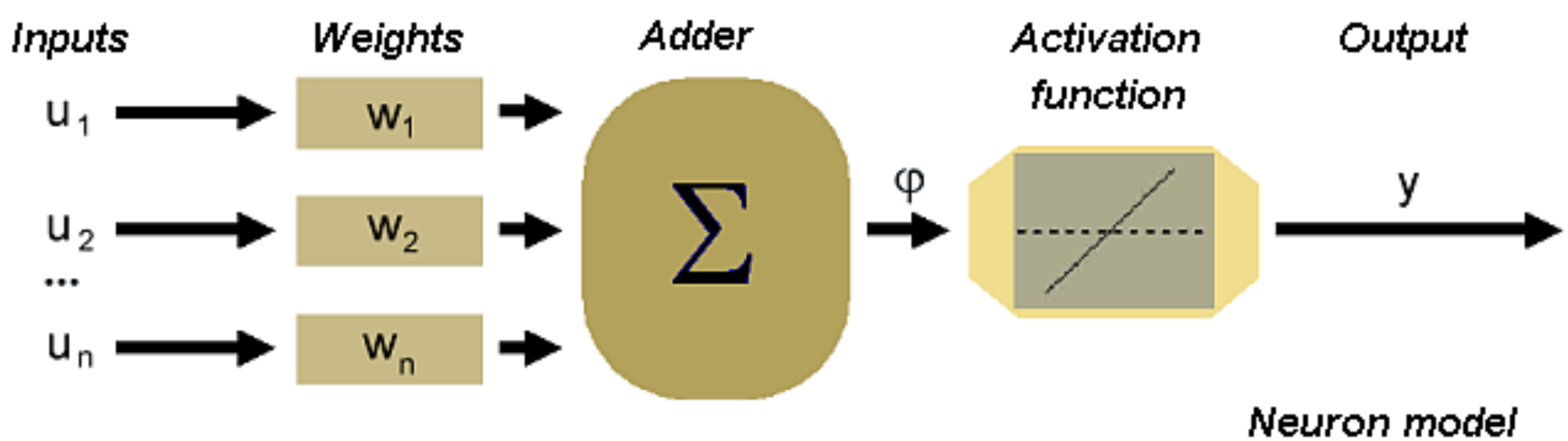
Perceptron

- The human brain (10^{10} cells) is the archetype of neural networks



Neuron scheme

Finds a hyperplane
to separate the classes !
Very similar to SVMs

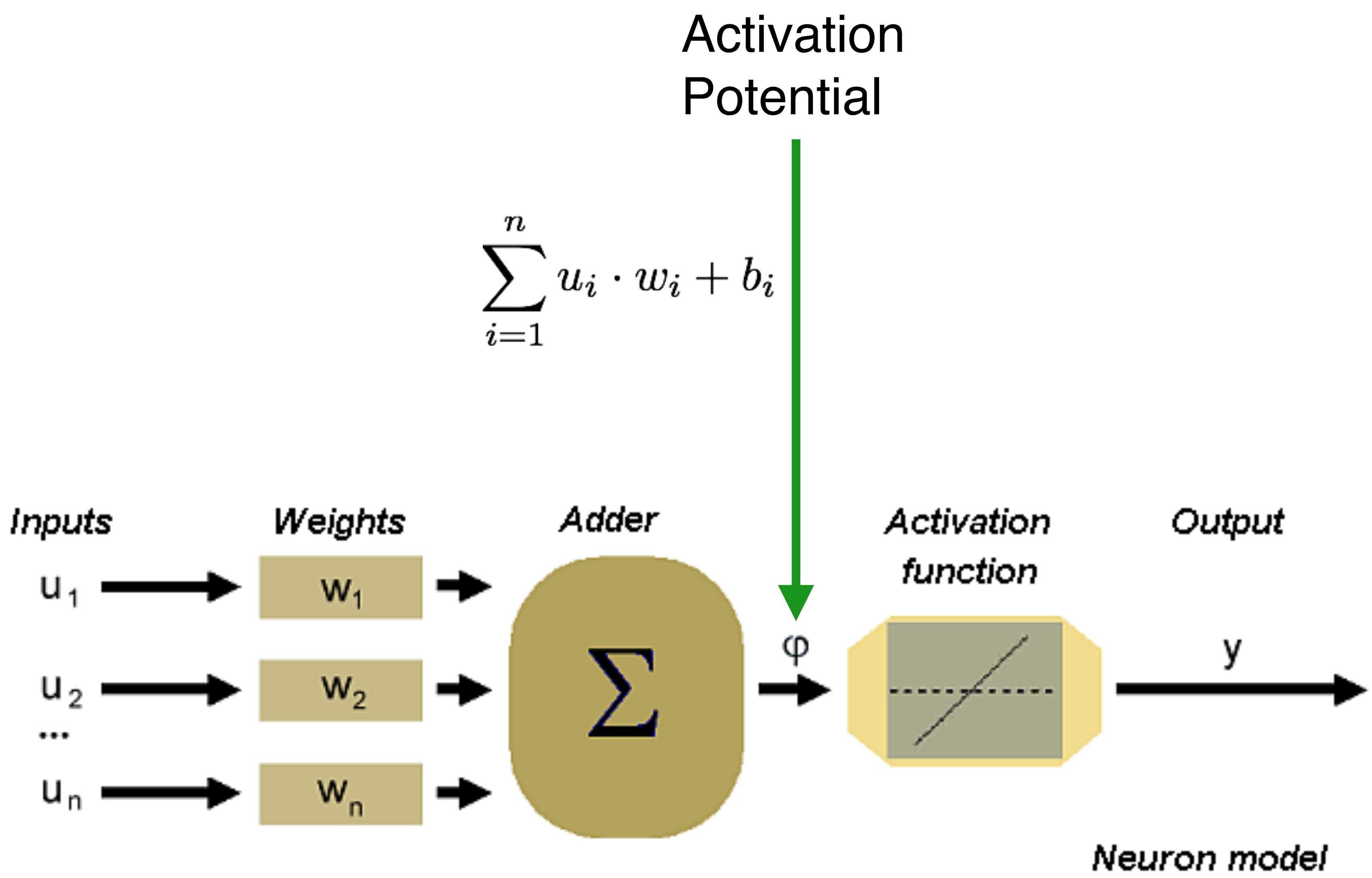


Neuron model

http://home.agh.edu.pl/~vlsi/AI/intro/

Perceptron

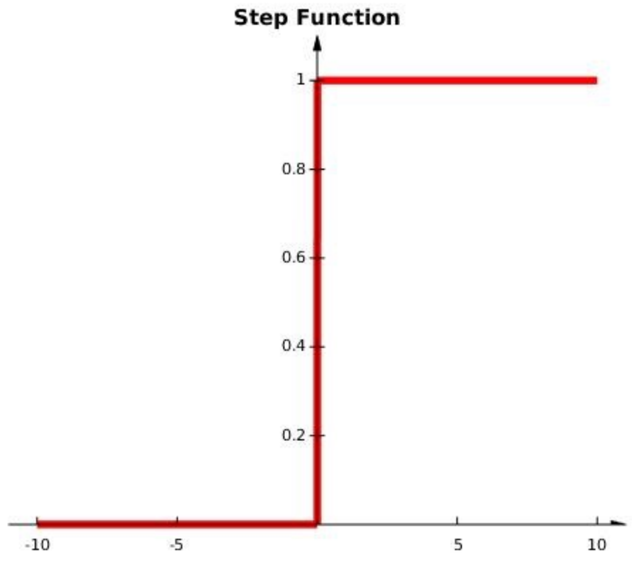
- The human brain (10^{10} cells) is the archetype of neural networks

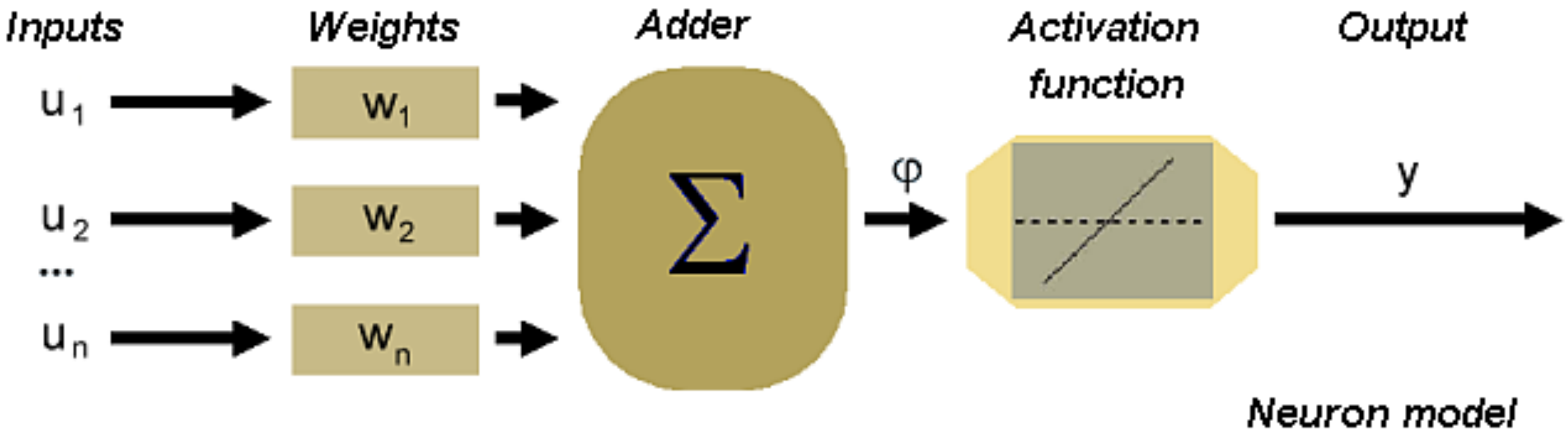


Neuron Activations

- Different type of activation functions

Threshold activation (binary classifier)



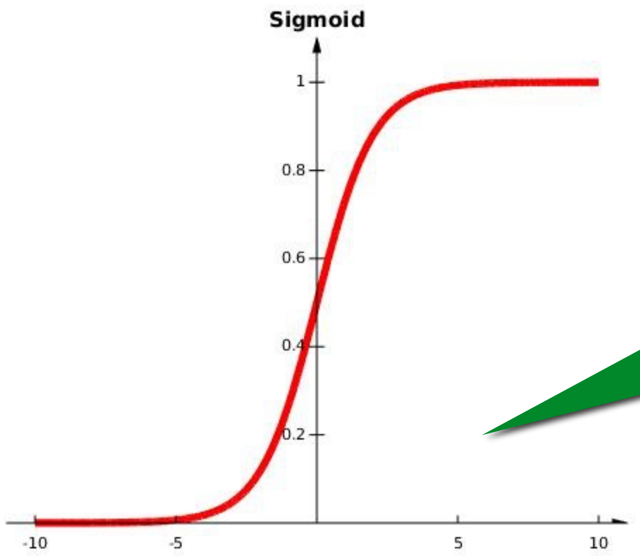


Neuron Activations

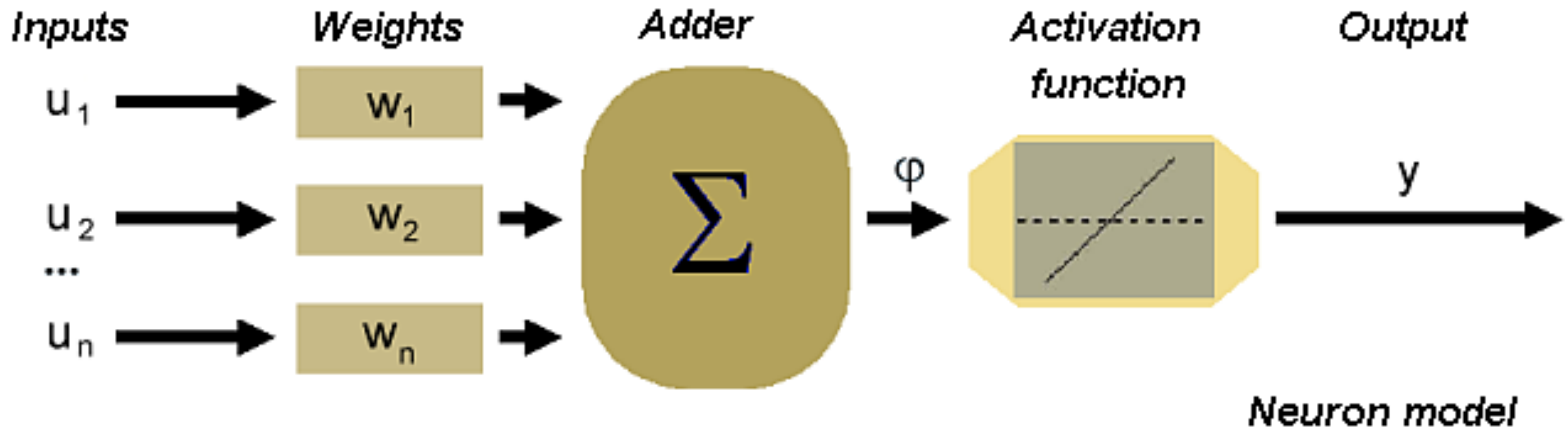
- Different type of activation functions

Sigmoid activation

$$y = \frac{1}{1 + \exp^{-\phi}}$$



More accurate !
Describe the non-linear characteristics of biological neurons

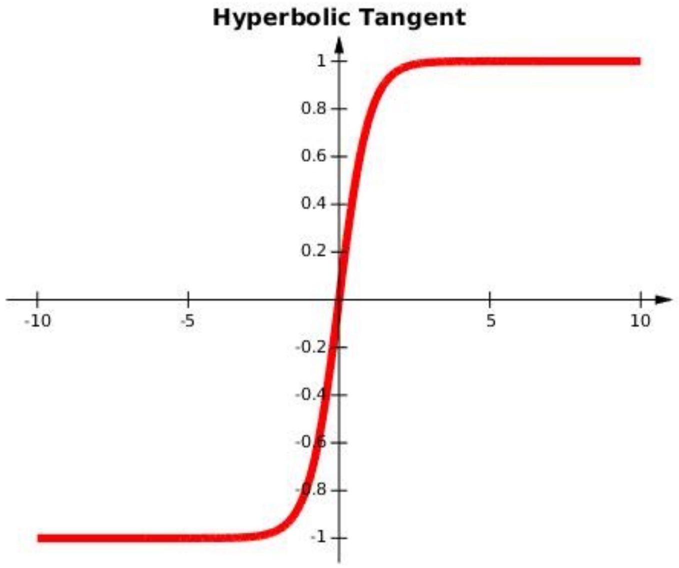


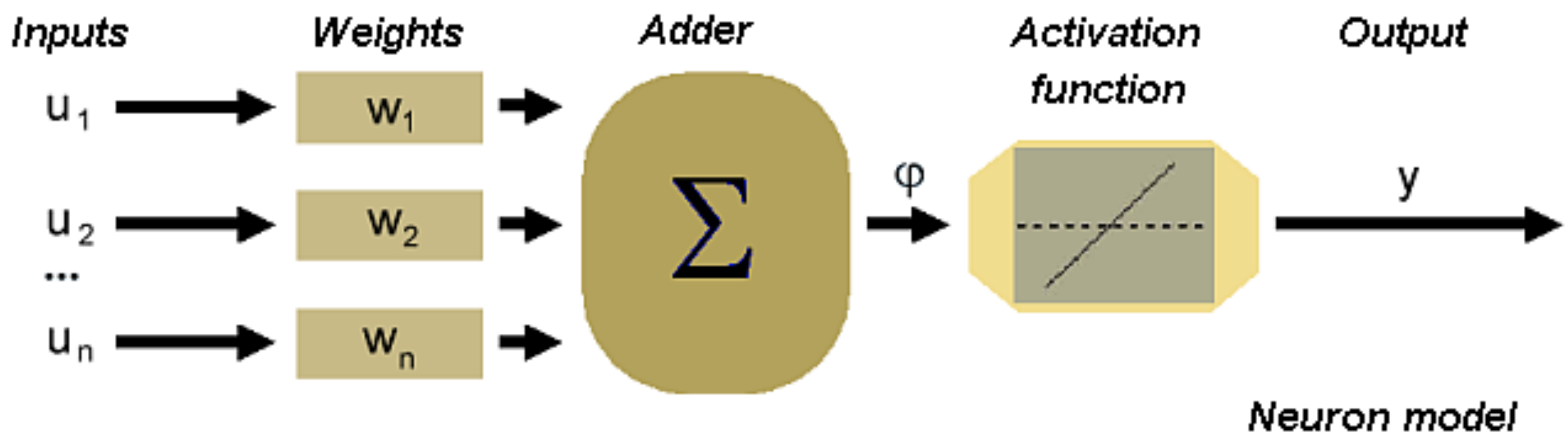
Neuron Activations

- Different type of activation functions

Tangent activation

$$y = \frac{\exp^{\phi} - \exp^{-\phi}}{\exp^{\phi} + \exp^{-\phi}}$$

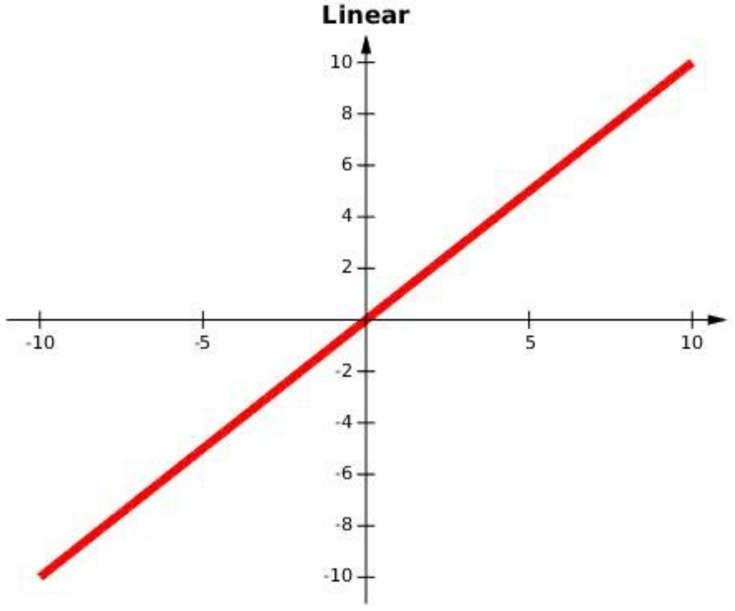


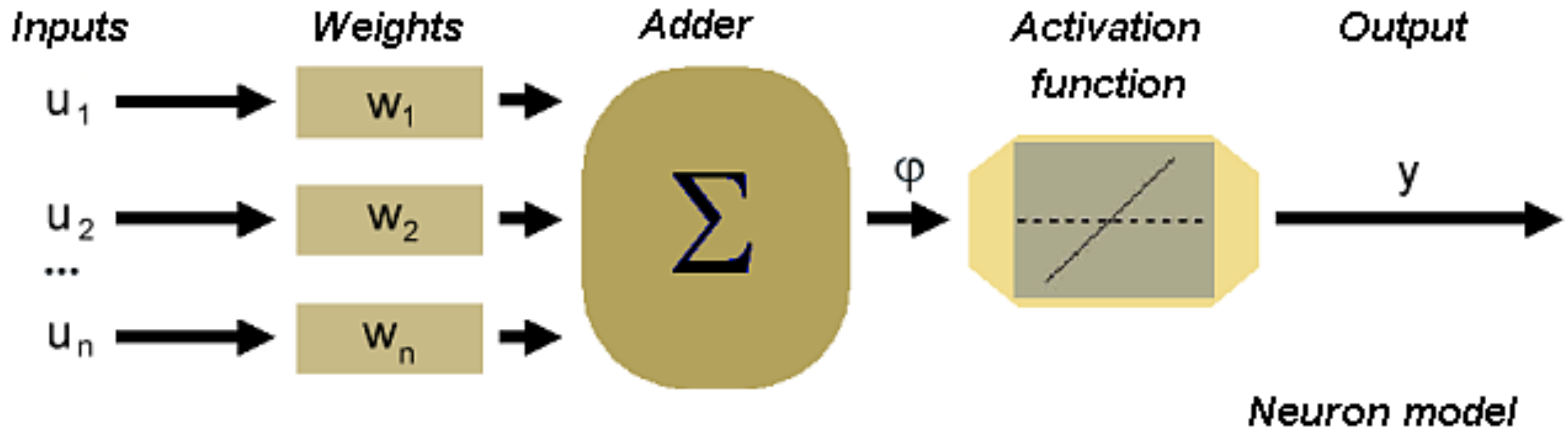


Neuron Activations

- Different type of activation functions

Linear activation



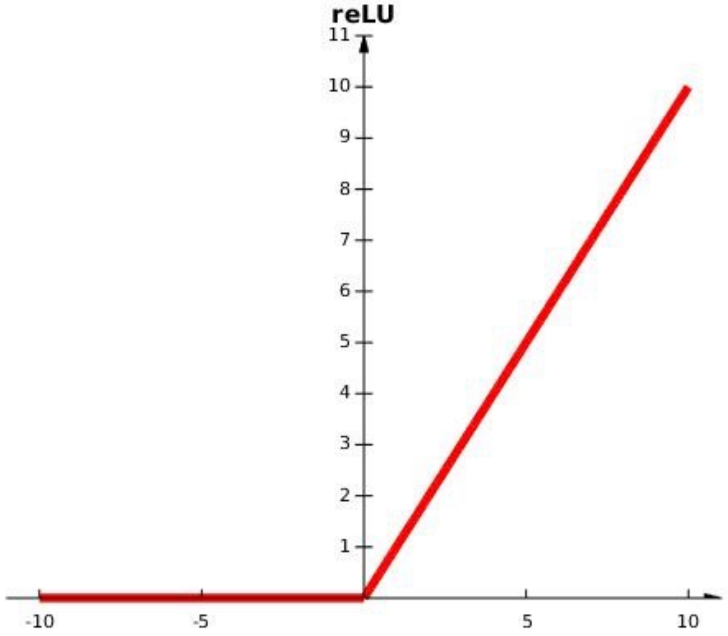


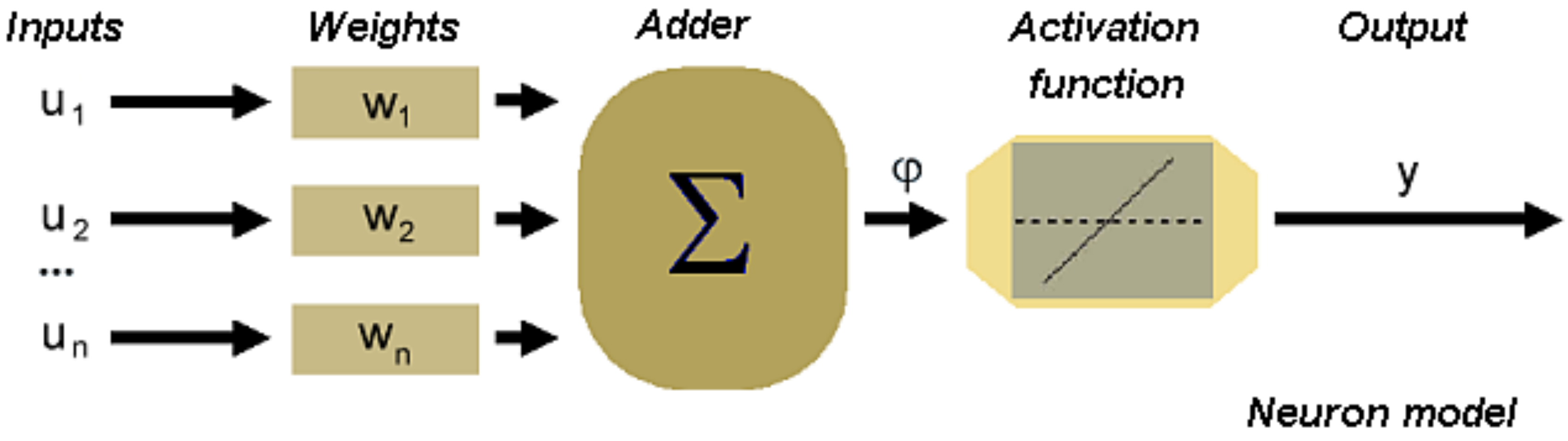
Neuron Activations

- Different type of activation functions

rectified Linear Unit activation

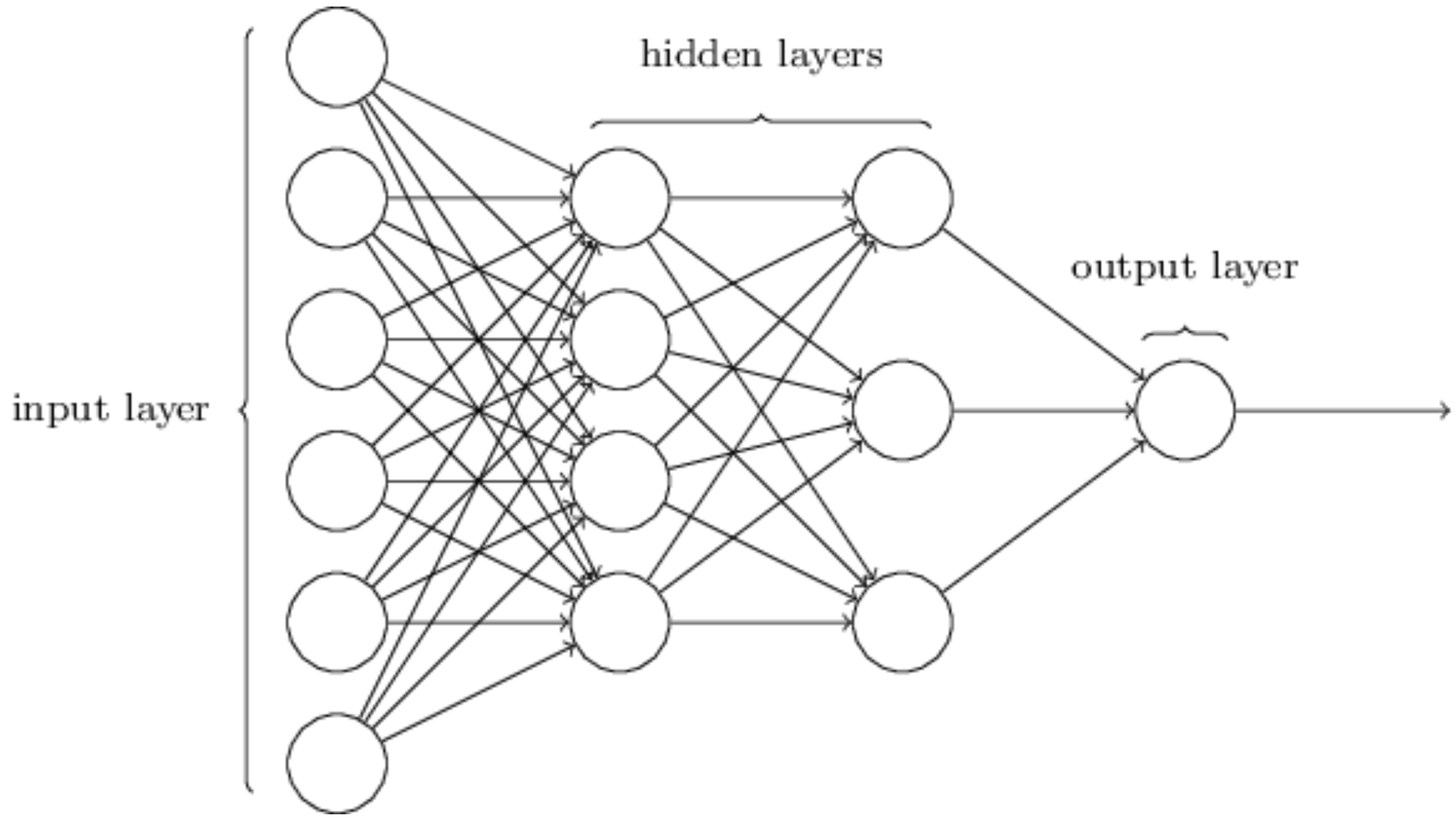
$$y = \max(0, \phi)$$





Simple Neural Network

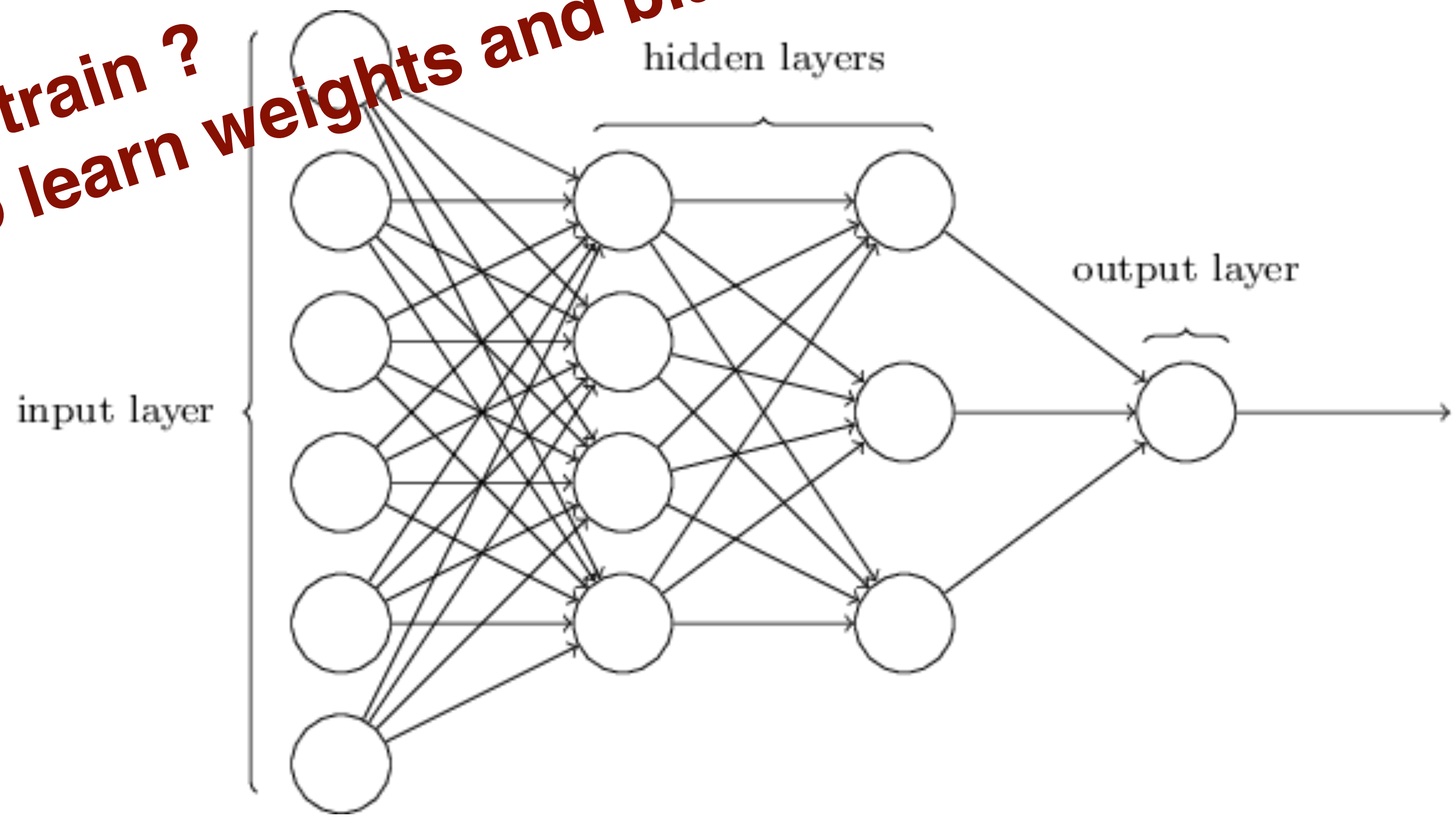
- Best reference: neuralnetworksanddeeplearning.com



Simple Neural Network

- Best reference: neuralnetworksanddeeplearning.com

How to train ?
How to learn weights and biases ?



Back Propagation

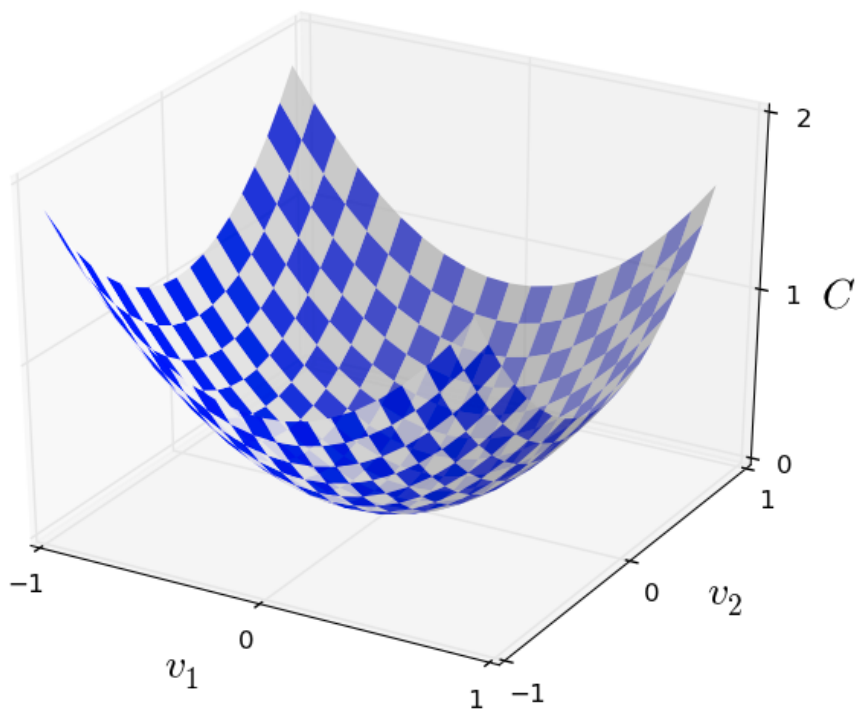
- Define a cost function

prediction

ground truth

$$C(w,b) \equiv \frac{1}{2n}\sum_x \|y(x) - a\|^2$$

- **Goal**: Find global minimum



Back Propagation

- Define a cost function

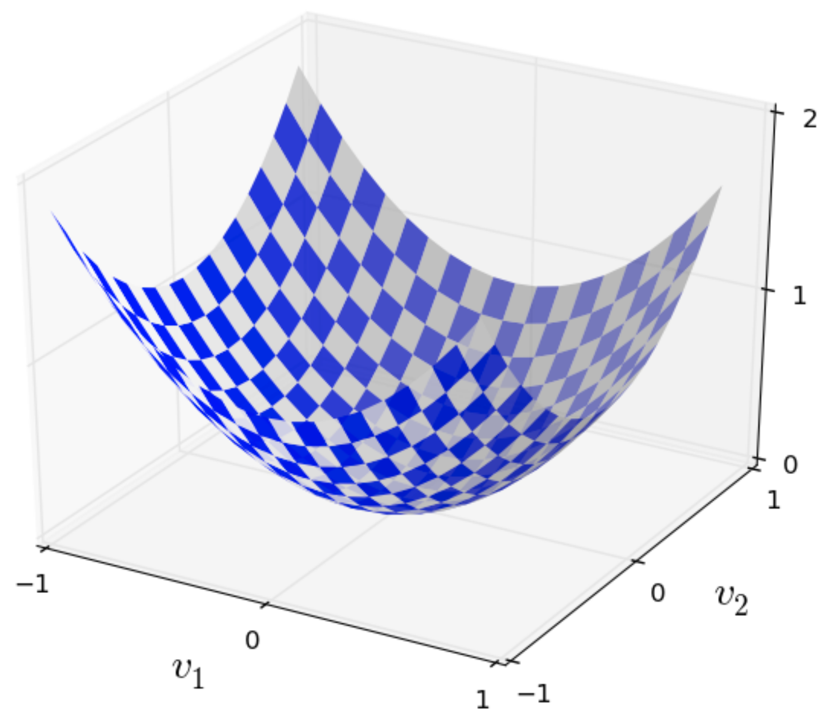
prediction

ground truth

$$C(w,b) \equiv \frac{1}{2n}\sum_x \|y(x) - a\|^2$$

- **Goal**: Find global minimum

Gradient Descent



Back Propagation

- Define a cost function

ground truth

prediction

$$C(w,b) \equiv \frac{1}{2n}\sum_x \|y(x) - a\|^2$$

- Minimize the error (derivative of the cost) with gradient descent
  - Gradient descent update rule:

$$w_k \rightarrow w'_k = w_k - \eta\frac{\partial C}{\partial w_k}$$

$$b_l \rightarrow b'_l = b_l - \eta\frac{\partial C}{\partial b_l}$$

Learning rate

Back Propagation

- Define a cost function

ground truth

prediction

$$C(w,b) \equiv \frac{1}{2n}\sum_x \|y(x) - a\|^2$$

- Propagate the error through layers
  - Take the derivate of the output of the layer w.r.t the input of the layer
  - Apply update rule for the parameters
- Repeat until convergence

Stochastic Gradient Descent

Gradient Descent

- Computing cost and derivative for the entire training set intractable.
- Not easy to incorporate new data in an ‘online’ setting

Solution: SGD

- Update parameters over a batch of images
- Mini-batch reduces the variance in the parameter update and can lead to more stable convergence
- Use momentum for faster convergence

$$v_k \to v_k' = \mu v_k - \eta \frac{\partial C}{\partial w_k} \quad , w_k \to w_k' = w_k + v_{k+1}$$

http://ufldl.stanford.edu/tutorial/supervised/OptimizationStochasticGradientDescent/

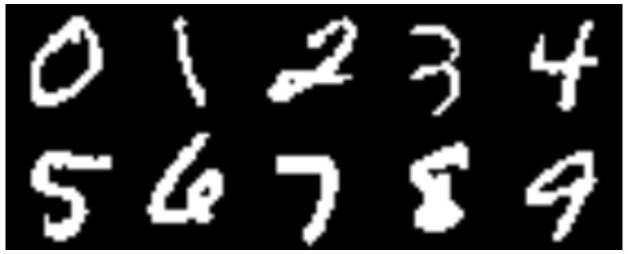
Other Optimisation Methods

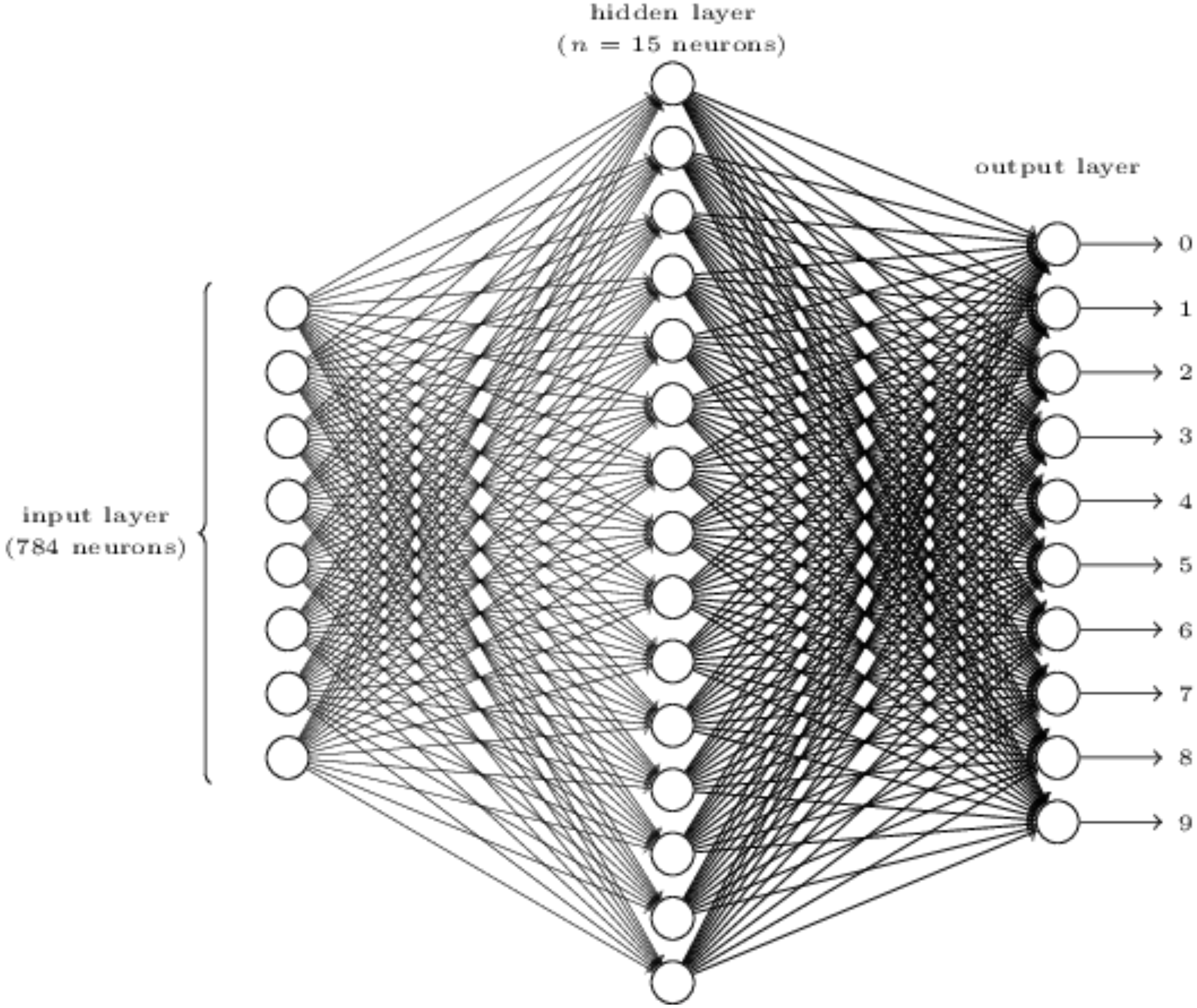
- AdaDelta
- AdaGrad
- NAG (Nesterov's Accelerated Gradient)
- RMSProb
- ADAM
  - Less sensitive to hyper-parameter initialisation

Back to Neuron Activations

- Sigmoid:
  - ◉ Dying gradients - saturated activation
  - ◉ Non-zero centered
- Tangent:
  - ◉ Dying gradients - saturated activation
  - ◉ Zero centered
- ReLU:
  - ◉ Greatly accelerated convergence
  - ◉ No expensive operation
  - ◉ Can be fragile during training : use Leaky-ReLU

MNIST Digit Classification with NN





Lets play around!

- cs.stanford.edu/people/karpathy/convnetjs

Quiz

Website

www.onlineted.com

Machine Learning Practical

Convolutional Neural Network in A Netshell

Lingni Ma
lingni@in.tum.de

Technische Universität München
Computer Vision Group

Summer Semester 2016

Unless otherwise stated, this slide takes most references from Standford lecture cs231n: "convolutional neural networks for visual recognition" (`http://cs231n.stanford.edu/`). The mathematical details are referenced from C. M. Bishop, "Pattern recognition and machine learning".

Outline

- Overview of convolutional neural network (CNN)
- Building blocks: basic layers of CNN
- Typical CNN architecture
- Training

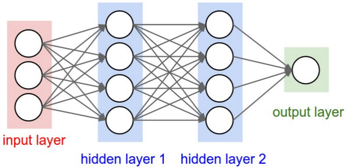
Applications of CNN

- classification and recognition
- image segmentation
- regression problems: optical flow, depth/normal estimation, loop closure, transformation, reconstruction
- feature extraction, encoding
- unsupervised problem

CNN Major Features

(recall) regular neural networks

- universal approximator
- fully-connected
  - not scale well to large input
  - easily overfit



regular neural network

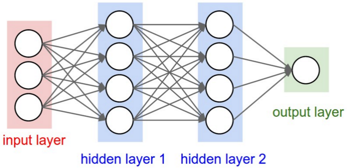
CNN Major Features

convolutional neural networks

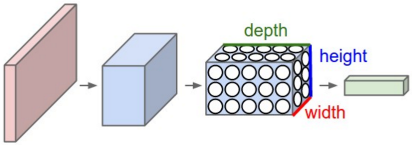
- feed-forward architecture, direct acyclic graph (DAG)

$$y_k^{l+1}(\mathbf{w},\mathbf{x}) = f\Big(\sum_{j=1}^{M} w_{kj}^{l+1} g\big(\sum_{i=1}^{D} w_{ji}^{l} x_i + b_j^l\big) + b_k^{l+1}\Big)$$

- error backpropagation
- 3D volume of neurons
- small filter, local fully-connected, parameter sharing

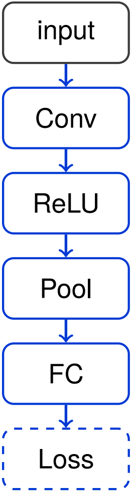


regular neural network



convolutional neural network

A Toy Example CIFAR-10 classification with CNN



input RGB image
(32 x 32 x 3)

compute the output of neurons, 12 filters of 3x3
(32 x 32 x 12)

element-wise activation $x_o = \max(0, x_i)$
(32 x 32 x 12)

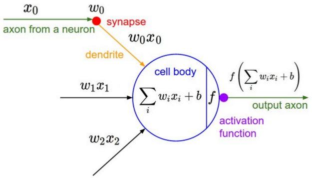
downsample operation along width and height
(16 x 16 x 12)

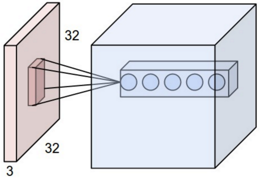
compute class score (SVM classifier)
(1 x 1 x 10)

loss computation during training

CNN: a direct acyclic graph (DAG) operates on volumes of neurons

- a set of learnable filters (or kernels)
- local fully connected: each filter is small in width and height, but go through the full depth of input volume
- parameter sharing: each filter slides along input width and depth
- feature map, output slice of one filter, stack into volume
- layer configuration
  - receptive field (F): filter spatial dimension
  - depth (D): number of filters
  - padding (P): zero-padding on boundary
  - stride (S): control the overlap of receptive field





input volume: $D_1 \times H_1 \times W_1$
conv layer: receptive field F, depth D, padding P, stride S

- how many parameters does this layer contain?

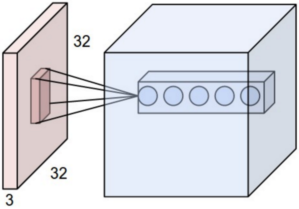
$$(F \times F \times D_1 + 1) \times D$$

- what is the output volume shape?

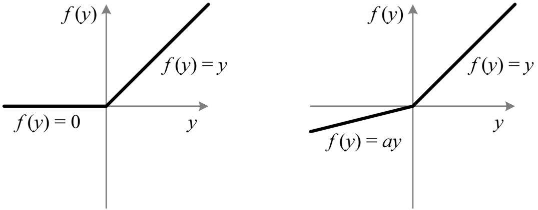
$$H_2 = (H_1 + 2P - F)/S + 1$$
$$W_2 = (W_1 + 2P - F)/S + 1$$
$$D_2 = D$$

- spatial resolution perservation

$S = 1$ and $F = 2P + 1$



- regular element-wise activation
- no hyper-parameter required
- ReLU[1] = max$(0, y)$
- PReLU[2] = max$(0, y) + \alpha$min$(0, y)$



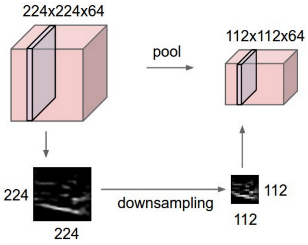
[1] Alex Krizhevsky, Ilya Sutskever and Geoff Hinton, "ImageNet classification with deep convolutional neural networks", NIPS 2012
[2] He et al. ICCV 2015, "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification"

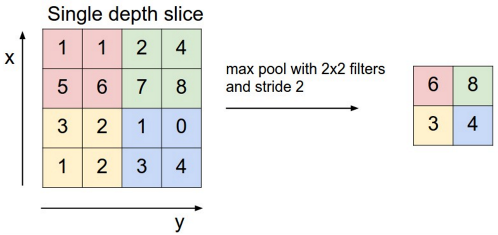
Basic CNN Layer Pooling

- drastically reduce the spatial parameters
- control over-fitting
- operate on each feature map
- common pooling layer specification

  $F = 2, S = 2, P = 0$, non-overlapping, most common

  $F = 3, S = 2, P = 0$, overlap pooling
- MAX pooling vs Average pooling





Basic CNN Layer — Fully Connected

- contain most of network parameters
- last layer of CNN compute class score $\Rightarrow$ act as SVM classifier
- net surgery: convert FC layer to Conv layer

  FC layer: $K = 4096$ operate on $7 \times 7 \times 512$ volume of neuron

  FC: 4096 inner product $\mathbf{w}^\mathsf{T}\mathbf{x}$, where $\mathbf{w}, \mathbf{x} \in \mathbb{R}^{7\times7\times512}$

  Conv: $D = 4096, F = 7, P = 0, S = 1$

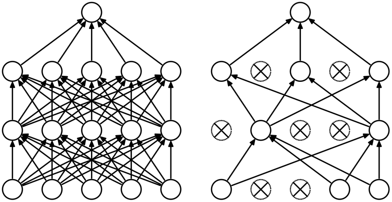
- practical advantages with Conv layer: efficiency

  input 224×224 image, downsample 32 times before FC.

  What if input 384×384?

  evaluating 224×224 crops of the 384×384 image in strides of 32 pixels is identical to forwarding the converted ConvNet one time

- batch normalization[1]
  - accelerate training and increase accuracy
  - append after activation
  - $y_i = \gamma \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} + \beta$, with $\mu_{\mathcal{B}} = \frac{1}{m}\sum_i^m x_i$, $\sigma_{\mathcal{B}}^2 = \frac{1}{m}\sum_i^m (x_i - \mu_{\mathcal{B}})^2$
- dropout[2]
  - randomly mask out neuros
  - control over-fitting
  - mostly append to FC layers



[1] Ioffe et al., "Batch normalization: accelerating deep network training by reducing internal covariate shift", NIPS 2015
[2] Srivastava et al., "Dropout: a simple way to prevent neural networks from overfitting", JMLR 2014

Loss Function

regression with Euclidean loss

- $E(\mathbf{w}) = \frac{1}{2N} \sum_i \|f(\mathbf{w}, x_i) - y_i\|^2$

classification with hinge loss

- $E(\mathbf{w}) = \frac{1}{N} \sum_{\mathbf{x}_n} \sum_{j \neq k} \max\left(0, y_j(\mathbf{w}, \mathbf{x}_n) - y_k(\mathbf{w}, \mathbf{x}_n) + 1\right)$

- cross-entropy $H(p,q) = \sum_x -p(x) \log q(x)$

  minimize KL-divergence between true distribution $p(x)$ and estimated distribution $q(x)$
- two-class classification (logistic regression)
  - $E(\mathbf{w}) = -\frac{1}{N} \sum_{\mathbf{x}_n} \Big(p_n \log \sigma\big(y(\mathbf{w}, \mathbf{x}_n)\big) + (1 - p_n) \log(1 - \sigma\big(y(\mathbf{w}, \mathbf{x}_n)\big)\Big)$
  - logistic sigmoid: $\sigma(y) = 1/\big(1 + \exp(-y)\big)$
- multi-class classification (multiclass logistic regression)
  - $E(\mathbf{w}) = -\frac{1}{N} \sum_{\mathbf{x}_n} \log \Big(\sigma\big(y_k(\mathbf{w}, \mathbf{x}_n)\big)\Big)$
  - softmax: $\sigma(y_k) = e^{y_k} / \sum_j e^{y_j}$

$E(\mathbf{w}) = E_{\text{data}}(\mathbf{w}) + \lambda E_{\text{regu}}(\mathbf{w})$

- regularizer is also known as weight decay
- control over-fitting by penalize parameter values
- $L_1 := \|\mathbf{w}\|_1$, not strictly differetiable
- $L_2 := \|\mathbf{w}\|_2^2$, differentiable, zero-mean Gaussian prior distribution

Building CNN patterns of layers and filters

Input

$\downarrow$

$\Big[\big[\,\text{Conv} \rightarrow \text{ReLU} \rightarrow (\text{BatchNorm})\big] \times N \rightarrow \text{Pool}\,\Big] \times M$

$\downarrow$

$\big[\,\text{FC} \rightarrow \text{ReLU} \rightarrow (\text{Dropout})\big] \times K$

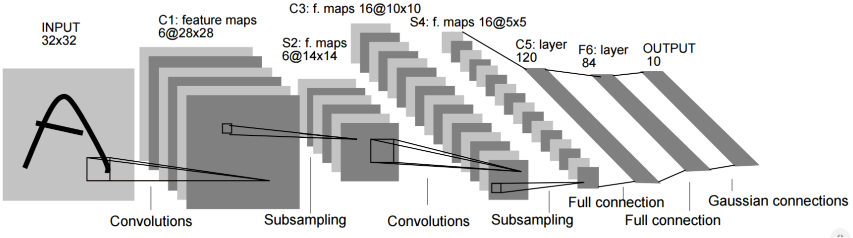
$\downarrow$

FC (score)

- usually $N \in (0, 3), M \geq 0$
- shape-preserve conv $F = 2P + 1, S = 1$, downsample with pooling
- better stack small conv filter to obtain large receptive field
  - stack three $F = 3$ filters vs. one $F = 7$ filter
- $K \in (0, 3)$

Classical CNN Architectures LeNet[1]

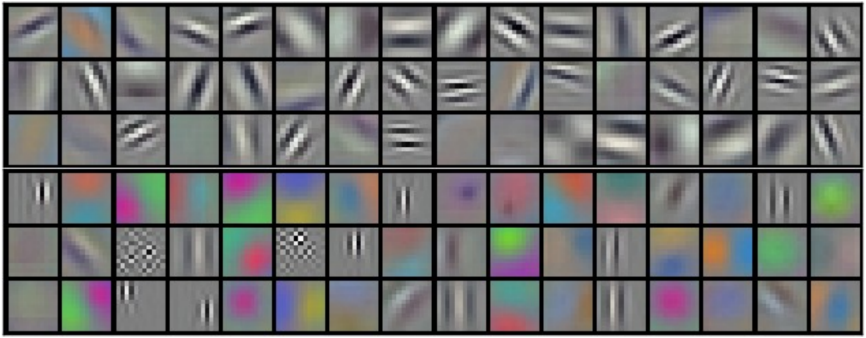
- first successful CNN example
- digit and letters recognition
- 3 Conv, 1 FC, 60K parameters



[1] Yann LeCun et al., "Gradient-based learning applied to document recognition", IEEE proceedings, 1998

Classical CNN Architectures AlexNet[1]

- first work to popularize CNN
- 5 Conv layers, 3 FC layers, 650K neurons, 60M parameters
- first 96 filters learned



[1] Alex Krizhevsky, Ilya Sutskever and Geoff Hinton, "ImageNet classification with deep convolutional neural networks", NIPS 2012

Classical CNN Architectures VGGNet[1]

- deep: 16 or 19 layers
- 13 or 16 Conv layers, small filters $F = 3, S = 1, P = 1$
- 5 max pooling layer, downsample by 32
- 240K neurons, 138M parameters

[1] Karen Simonyan and Andrew Zisserman, 2014 ICLR, "Very deep convolutional networks for large-scale image recognition"

Classical CNN Architectures

GoogLeNet[1]

- 22 layers
- inception module: combine convolutional layers

ResNet[2]

- very deep, e.g. 152 layers

Do networks get better simply by stacking more layers?

[1] Christian Szegedy et al, 2015 CVPR, "Going deeper with convolutions"
[2] He et al, 2015, "Deep Residual Learning for Image Recognition"

Training CNN

data preparation

- mean substraction, normalization
- augmentation

weight initialization

- Gaussain, xavier[1], msra[2]
- trainsfer learning

optimization algorithm

- SGD
- shuffle input

hyper-parameter

- learning rate policy
- batch size

[1] Xavier Glorot et al. "Understanding the difficulty of training deep feedforward neural networks", 2010
[2] He et al., "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification", ICCV 2015

Caffe[1]

- C++/CUDA, with Python and Matlab bindings
- (+) good for feedforward network (+) finetune (+) easy to start
- (-) bad at recurrent network (-) poor support for visualization

Torch[2]

- C and Lua
- used a lot by Facebook, DeepMind

TensorFlow[3]

- Google
- Python, source code not readable
- not limited for CNN, also good at RNN

[1] http://caffe.berkeleyvision.org/
[2] https://torch.ch
[3] https://www.tensorflow.org/

Question ?