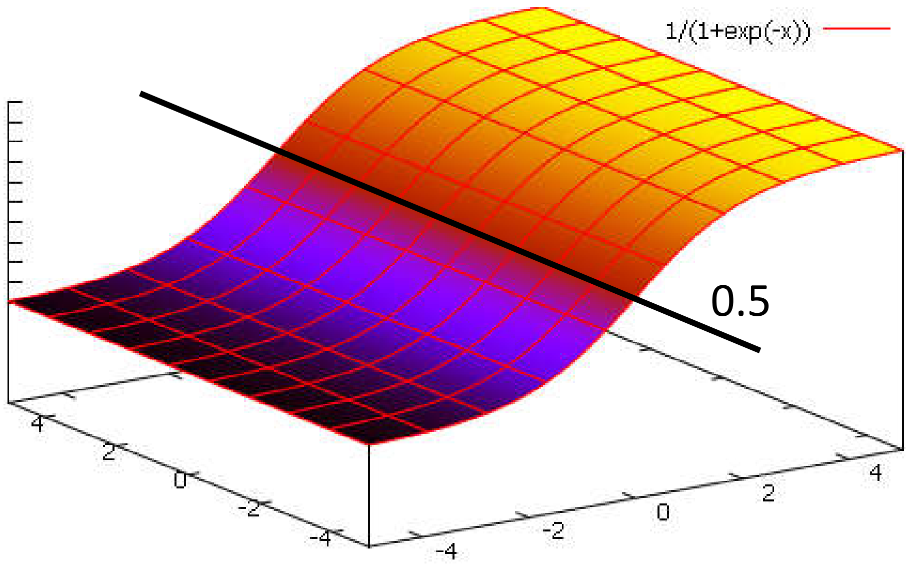
Radial Basis Function Networks

MLP classification (recap)

- MLPs consist of sigmoidal weighted summation units
- The discriminant formed by a single unit takes the shape of a hyperplane
  - It's shape (a hyperplane) is due to the weighted sum
  - The sigmoid makes it fuzzy, but does not (alone) change the discriminant’s shape
- The next layer combines the hyperplanes into regions
- Can we define a unit which forms a region on its own?

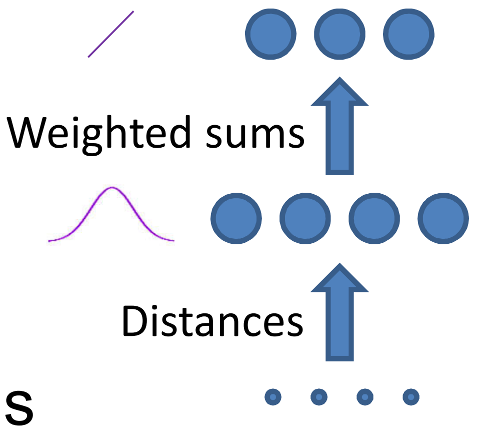


UPPSALA UNIVERSITET

Radial Basis Function Networks

RBFNs are feedforward networks, where:

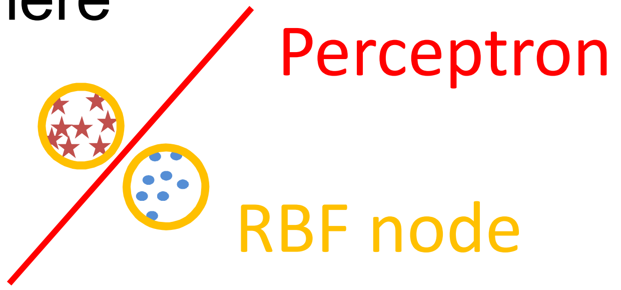
- The hidden nodes (the RBF nodes) compute distances, instead of weighted sums



- The hidden layer activation functions are Gaussians (or similar), instead of sigmoids
- The output layer nodes are linear weighted sum units, i.e. a linear a combination of Gaussians
- One hidden layer only

RBFNs for classification

- The hidden nodes now form hyperspheres instead of hyperplanes
- The weight vector of a hidden node represents the centre of the sphere



- Local 'receptive fields' (regions) instead of global dividers
  - An RBF node only reacts (significantly) for inputs within its region
- Note: In Competitive Learning we also compute distances, to find the closest one. In RBFNs, the nodes do not compete – they are combined

UPPSALA UNIVERSITET

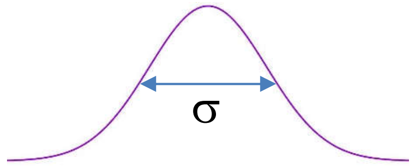
RBFN implementation

- Each hidden node, j, computes $h_j = f(r_j)$ where r_j is the distance between the current input vector $\boldsymbol{x}$ and the node's weight vector, $\boldsymbol{t}_j$ (= the centre of a hypersphere)

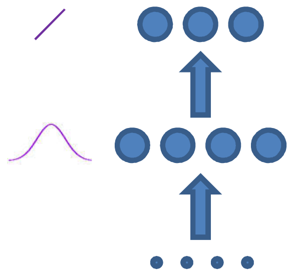
$$r_j = \sum_{i=1}^{N} \left(x_i - t_{ji}\right)^2$$

- $f(r_j)$ should have a maximum at 0, i.e. when the input vector is at the centre of the sphere, e.g. a Gaussian:

$$f\left(r_j\right) = e^{-\left(\frac{r_j}{\sigma}\right)^2}$$



where σ is the standard deviation (width)



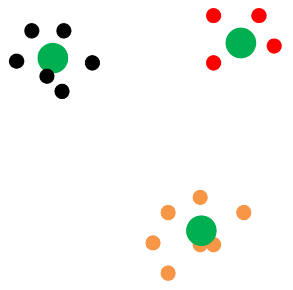
RBFN learning

- Learning: To find the position and size of the spheres (hidden layer) and how to combine them (output layer)
- We <u>could</u> train this supervised (deriving a new version of Backprop – everything is still differentiable!)
  - Slow, and does not exploit the localized properties
- Hidden and output layer are conceptually different now ➔ two separate learning problems
- Output layer is just a layer of (linear) perceptrons.
  - can use the Delta rule
  - but we must train the hidden layer first!

UPPSALA UNIVERSITET

RBFN Learning (positions)

- Could be random and fixed (i.e. not trained at all)
  - Any problem can be solved this way
  - But it requires a huge number of nodes
- Train unsupervised! (K-Means or Competitive Learning)
  - Allows the basis functions to move around, to where they are most likely needed
- Can later be fine-tuned by supervised learning (gradient descent/Backprop)
  - Backprop can be optimized for this (only a few nodes have to be updated for a given input vector)

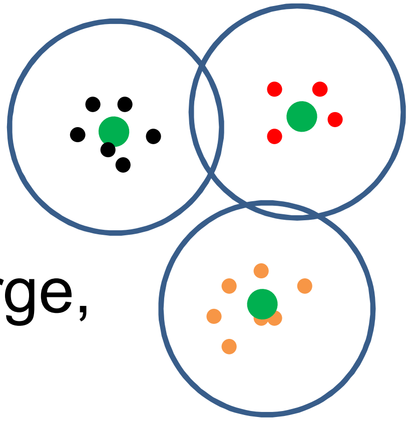


RBFN Learning (widths)

- Often equal and fixed (i.e. σ is a global constant)
- Two common ways to set σ (train positions first!):
  - The average distance between a basis function and its closest neighbor
  - or
  - $\sigma = d/\sqrt{2M}$

    d = max distance between two nodes

    M = number of basis functions



- Do <u>not</u> train widths with Backprop!
  - tends to make the hyperspheres very large, destroying the localized properties

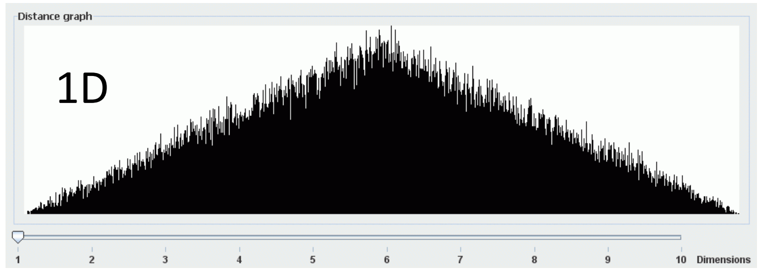
RBFN v.s. MLP

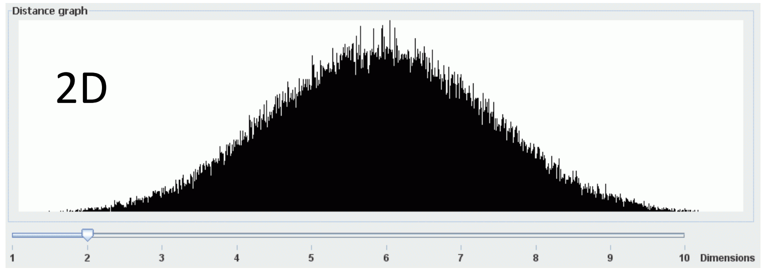
RBFN hidden nodes form local regions. The hyperplanes of a MLP are global ➔

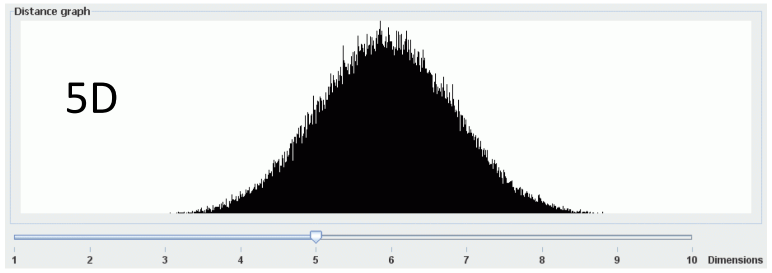
- MLPs often do better in regions where little data is available
  - and therefore also when little data is available in total
- MLPs usually require fewer hidden nodes and tend to generalize better (i.e. RBFNs are more likely to overfit)
- RBFNs learn faster
  - Sometimes, only the output layer has to be trained
  - Even if the hidden nodes <u>are</u> trained, only a few of them are affected by a given input and need to be updated
- MLPs often do better for problems with many input variables
  - A consequence of the “curse of dimensionality”

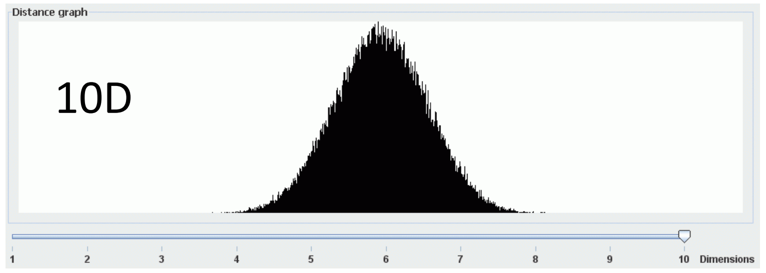
The Curse of Dimensionality

- Measuring distances becomes increasingly more difficult in higher dimensionality spaces!
- Test: Measure the average distance between two randomly selected points in range 0..1 and plot a histogram with the average in the middle:









RBFN v.s. MLP

- RBFNs are less sensitive to the order of presentation
  - Good for on-line learning (e.g. in reinforcement learning)
- RBFNs make less false-yes classification errors
  - The basis functions tend to respond with a low value for data far from their receptive fields
  - MLPs can respond with very high output also for data from uncharted terrain (which is also why they may extrapolate better than RBFNs)

To summarize: If data is hard to obtain, use MLP. If you have lots of data and/or if you want the network to learn continuously (on-line), use RBFN.

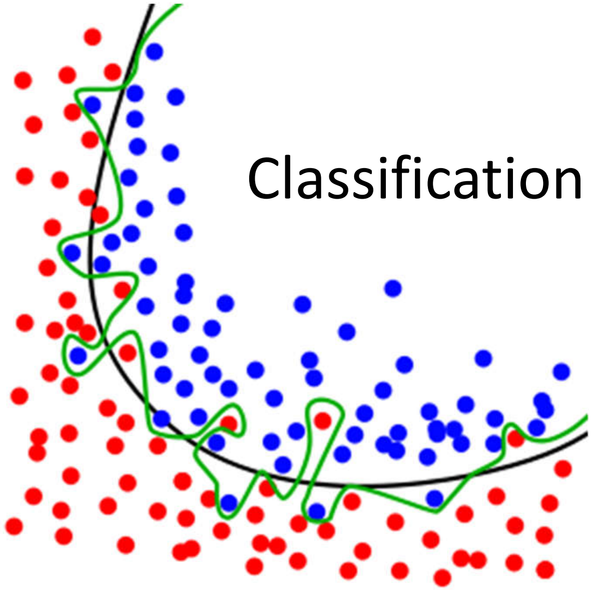
Regularization techniques

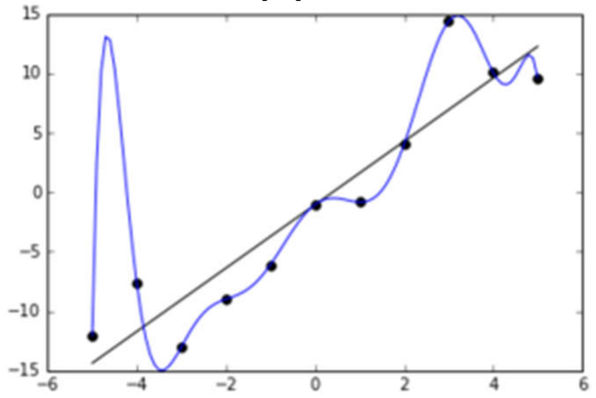
to avoid overfitting

UPPSALA UNIVERSITET

Regularization

- Overtraining/overfitting (here considered synonyms)





Regularization = any method which tries to prevent this

Images from Wikipedia:Overfitting

UPPSALA UNIVERSITET

Regularization methods (examples)

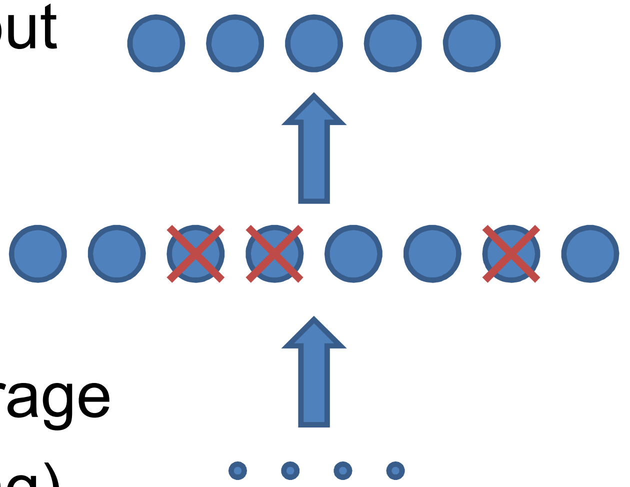
- Early stopping (to avoid training for too long)
- Trying to minimize the network size (parameters/weights)
- Noise injection (to force the network to generalize)
- Weight decay (with or without removal of weights)
- Lagrangian optimization (constraint terms in the loss func.)
- Multitask learning (constrains the hidden layer)
- Averaging (over several approximations)
- Dropout

Averaging: Wisdom of the Crowds

- Galton’s ox (1907)
- 800 people on a country fair guessed the weight of an ox
- Average was within 1% of the true weight!
  - Better than most individual guesses
  - Better than any of the experts! (breeders and traders)
- Requires that the individuals do not influence each other!
- In ML: *Bagging* (averaging over several predictors)

Dropout

- Introduced in 2014 (Srivastava, Hinton, *et al)*
- For each training example, switch off hidden nodes at random (prob. 0.5)
- When testing, use all hidden nodes, but halve the hidden-to-output weights



- In effect, the network outputs the average of several networks (similar to bagging)
- Can let inputs drop out too (but, if so, with lower prob.)
- Takes longer to train

UPPSALA UNIVERSITET

Deep Learning

A very brief introduction

UPPSALA UNIVERSITET

Deep Learning

- Simplest definition: any neural network with more than one hidden layer
- Nowadays, usually <u>much</u> deeper, and with different types of (often hand crafted) layers. For example CNNs:

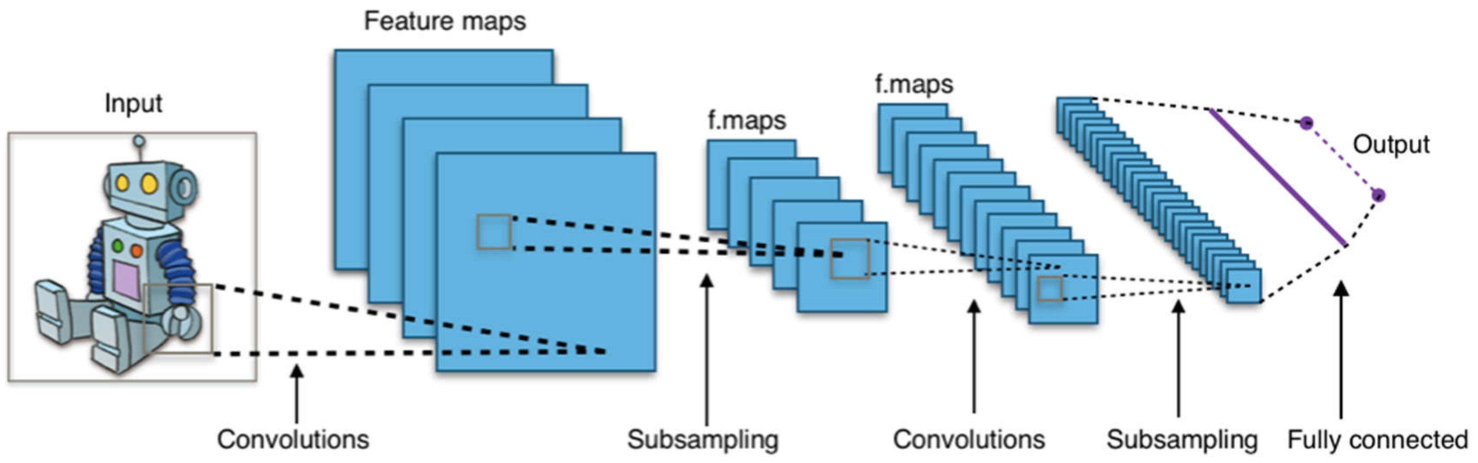
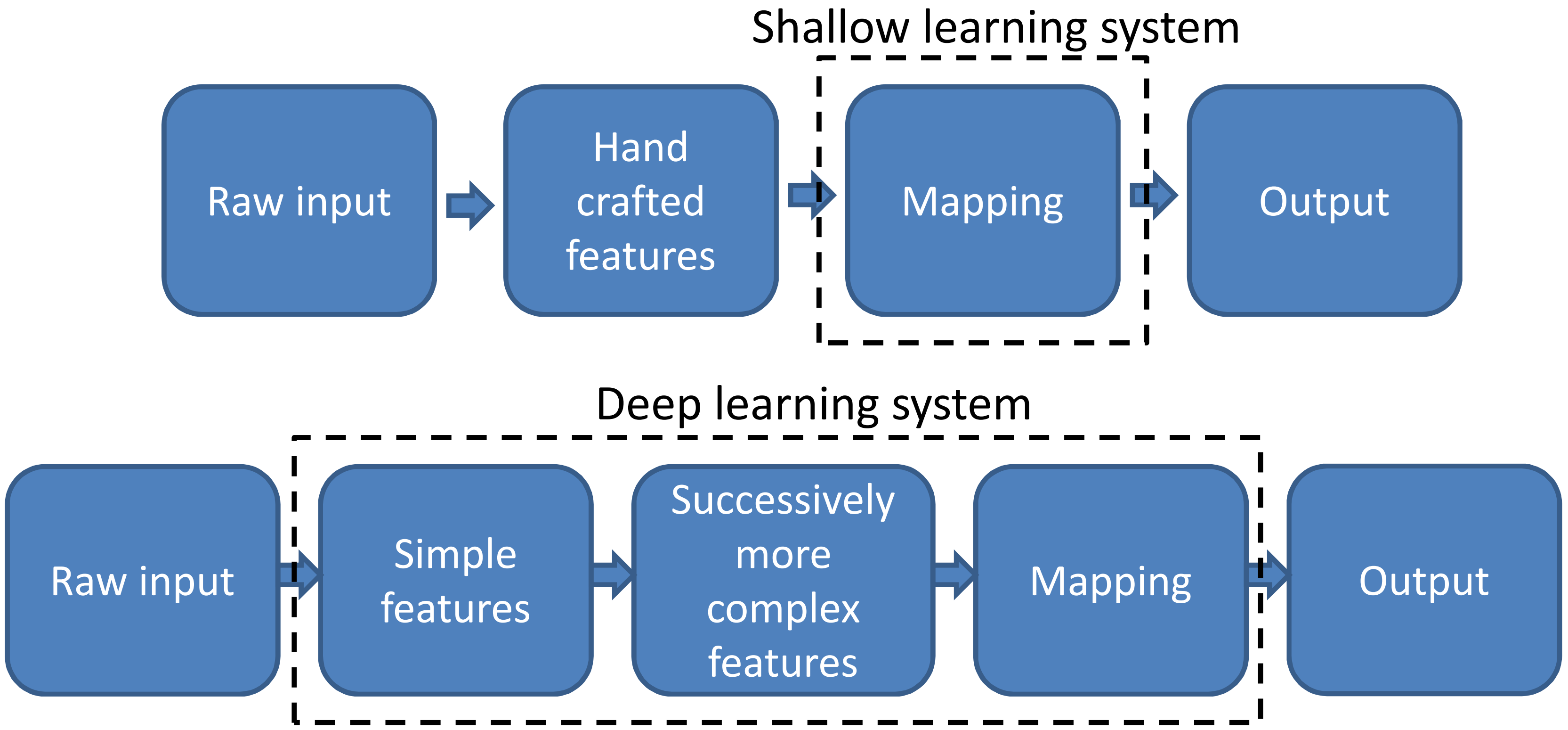


Image from Wikipedia: Convolutional Neural Networks

Why more layers when one hidden layer should suffice?

- More layers ➔ more levels of abstraction
- More levels of abstraction ➔ automatic feature selection



Block diagrams inspired by Goodfellow, Bengio et al

Deep Learning: Why now?

Deep Learning challenges

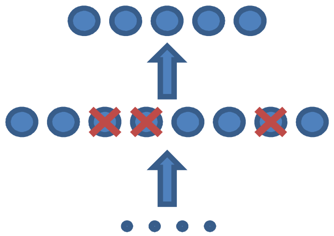
- Severe risk of overfitting (huge number of parameters)
- Requires huge amounts of data
- Long training times. Requires very fast computers
- Vanishing gradients

Lots of Data

+

Suitable hardware

+



New methods
(Dropout and ReLUs in particular)

+ Big companies are involved, and they own lots of data!

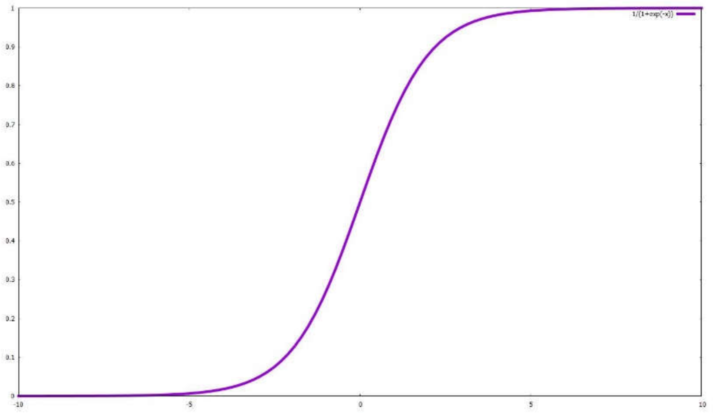
Vanishing gradients

- Backprop computes weight changes with the chain rule
- Chain rule ➔ multiplying many (small) gradients
- More layers ➔ longer chain ➔ <u>very</u> small gradients

Even worse

- Backprop tends to push the nodes towards their extreme values (towards either end of the activation function)
- For sigmoids, the derivative f’(S) is very close to 0 for large positive and negative values
- f’(S) is part of the chain ...

$$\frac{\partial E}{\partial w} = \frac{\partial E}{\partial y_{out}} \frac{\partial y_{out}}{\partial S_{out}} \frac{\partial S_{out}}{\partial y_{hid3}} \frac{\partial y_{hid3}}{\partial S_{hid3}} \frac{\partial S_{hid3}}{\partial y_{hid2}} \dots \frac{\partial y_{hid1}}{\partial S_{hid1}} \frac{\partial S_{hid1}}{\partial w}$$

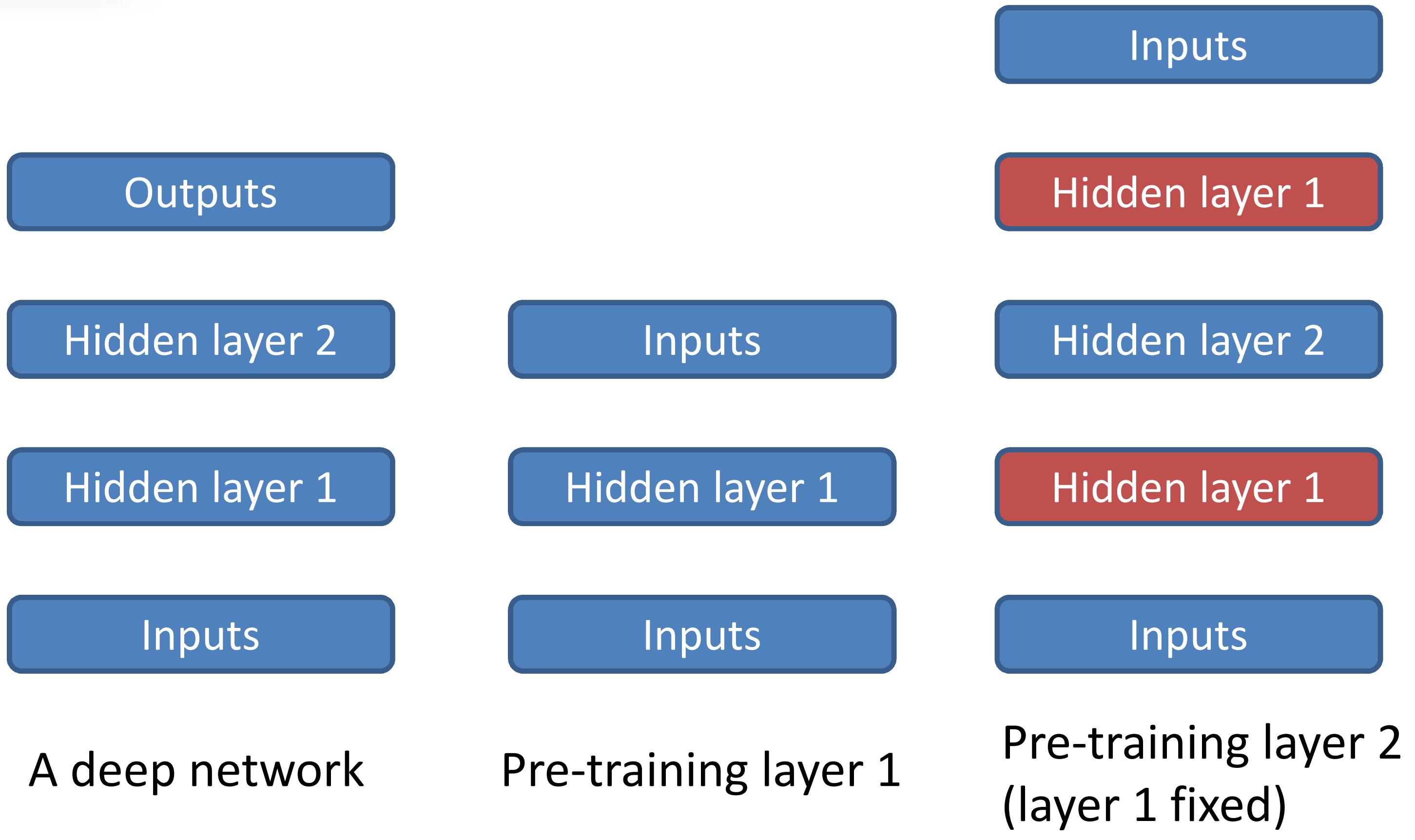


UPPSALA UNIVERSITET

Deep Learning milestones

- **2006** Hinton shows how unsupervised pre-training (using auto-encoders) makes deep nets trainable by gradient descent
- **2010** Martens show that deep nets can be trained with second-order methods, without requiring pre-training
- **2010** It is discovered that the use of *Rectified Linear Units* makes a big difference for gradient descent (no more vanishing gradients!)
- **2013** Sutskever et al show that regular gradient descent (e.g. Backprop) can outperform second-order methods, with clever selection of initial weights and momentum
- **2014** Dropout is introduced as a regularization method

Auto-encoders for pre-training in DL

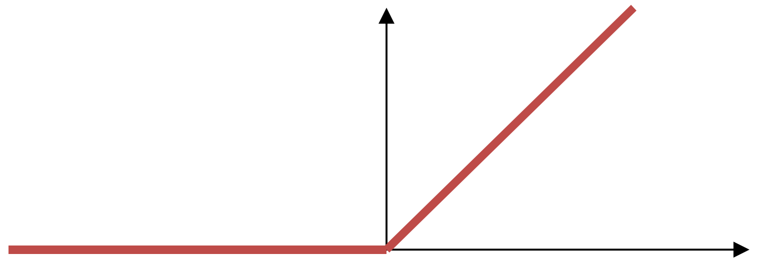


Auto-encoding this way, is another form of regularization!
(since it restricts the hidden layers)

UPPSALA UNIVERSITET

Rectified Linear Units (ReLUs)

- $y = \max(x, 0)$



Pros

- Very easy to compute
- Almost linear (but non-linear enough)
- Does not saturate (no vanishing gradients)
- Very simple derivative (0 for x<0, 1 for x>0)

Cons

- Derivative undefined for x=0
- Nodes with negative weighted sum (0 output) will not learn
- Dead units
- Probably not so good for shallow networks?

UPPSALA UNIVERSITET

Common ReLU variants

- Leaky ReLU

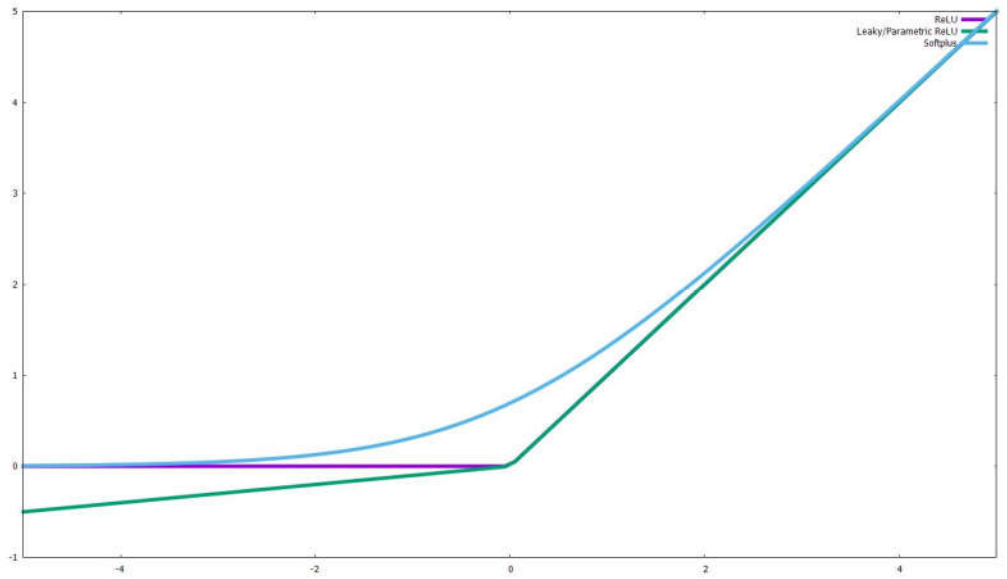
$$y = \begin{cases} x, \text{if } \mathrm{x} > 0 \\ 0.01x, \text{otherwise} \end{cases}$$

- Parametric ReLU

$$y = \begin{cases} x, \text{if } \mathrm{x} > 0 \\ ax, \text{otherwise} \end{cases}$$

- Softplus

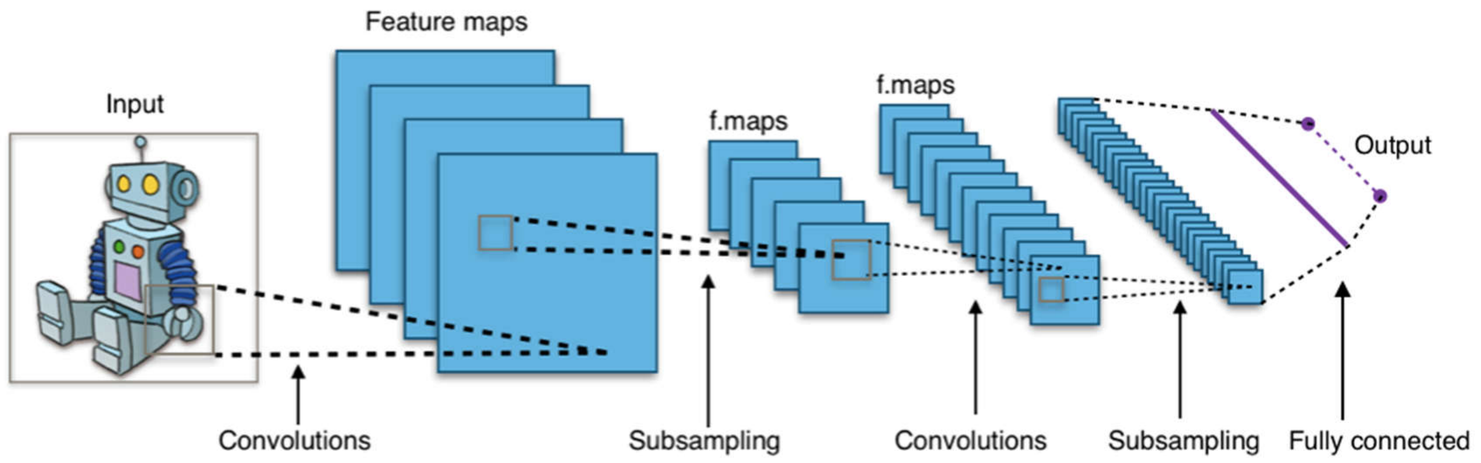
$$y = \log(1 + e^{x})$$



UPPSALA UNIVERSITET

Convolutional Neural Networks

to convolute = to fold

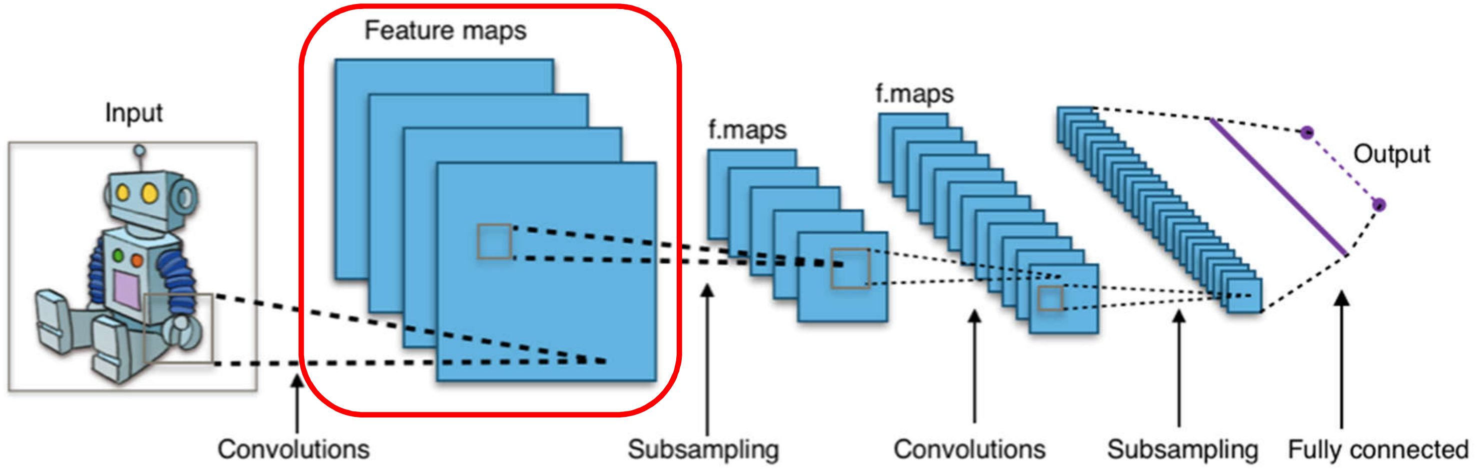


CNN characteristics

- Receptive fields (windows)
- Shared weights
- Pooling (subsampling)

UPPSALA UNIVERSITET

Convolutional Neural Networks

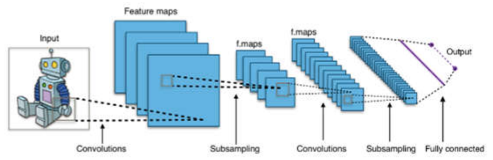


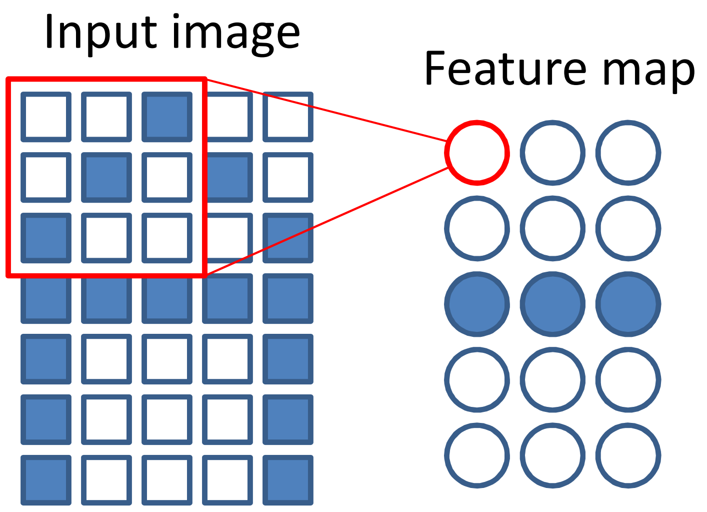
Convolutional layer = layer of feature maps

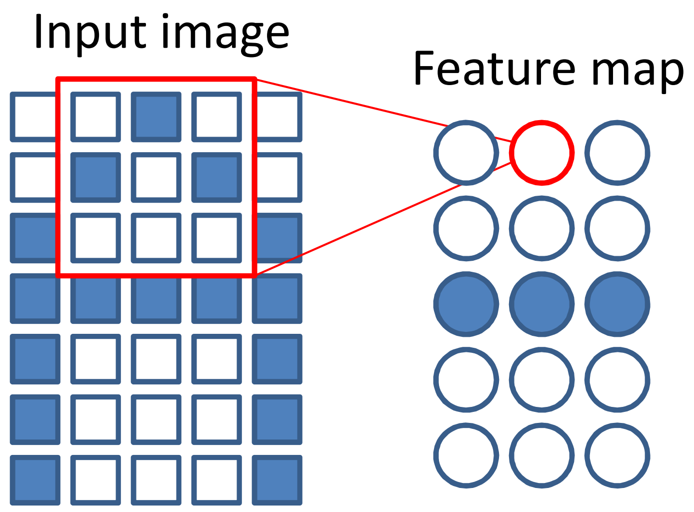
Each feature map is a grid of <u>identical</u> feature detectors
(so each map looks for <u>one</u> feature, but all over the image at once)

Feature detector = Filter = Neuron

CNN Feature Maps

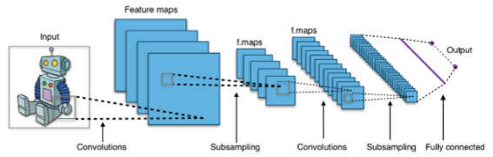




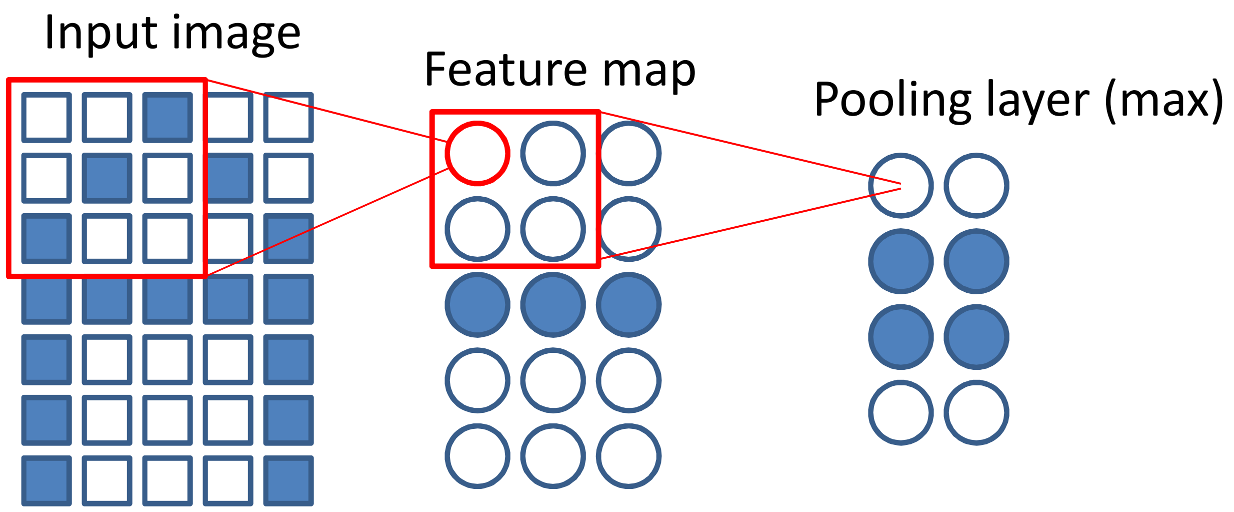


- Here, each neuron looks for a horizontal line in the centre of its field
- The field/window is small, so the neuron only has a few weights (here 3x3+1(bias) = 10 weights)
- All neurons in the map detect the same feature, so they all share those 10 weights
- So, the total number of weights for the whole map, is just 10
- *Stride* = step length when moving the window (here, the stride is 1)

CNN Pooling (Subsampling)

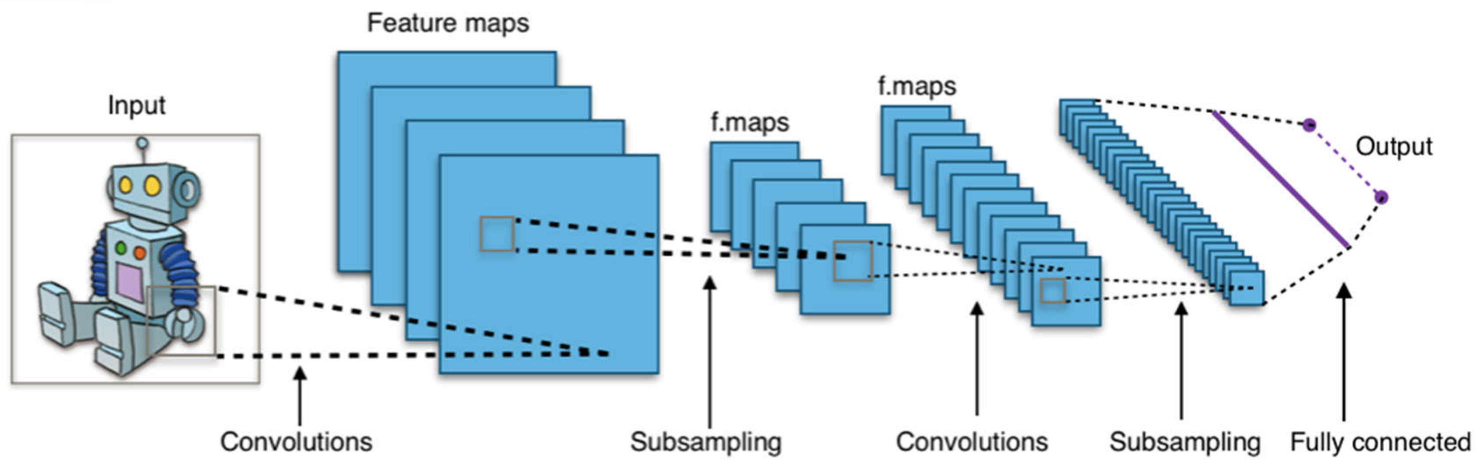


- Interleaved with the convolutional layers
- Reduces the resolution of the preceeding feature map
- Same principle (moving window) but predefined function (for example max, average, etc. It is not trained)
- Effect depends on the function (e.g. for max, don't care <u>where</u> the feature is within the pooling window)



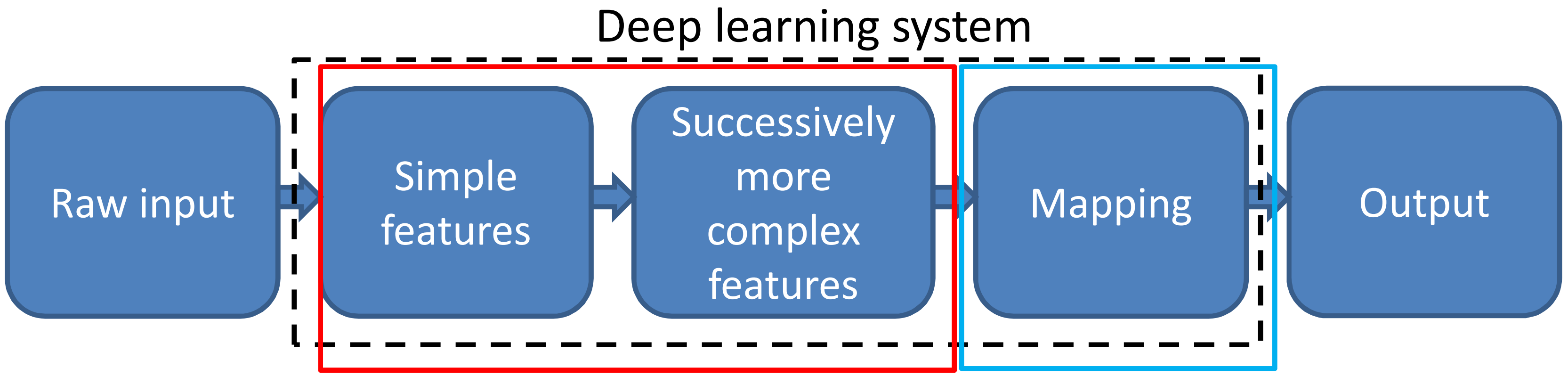
UPPSALA UNIVERSITET

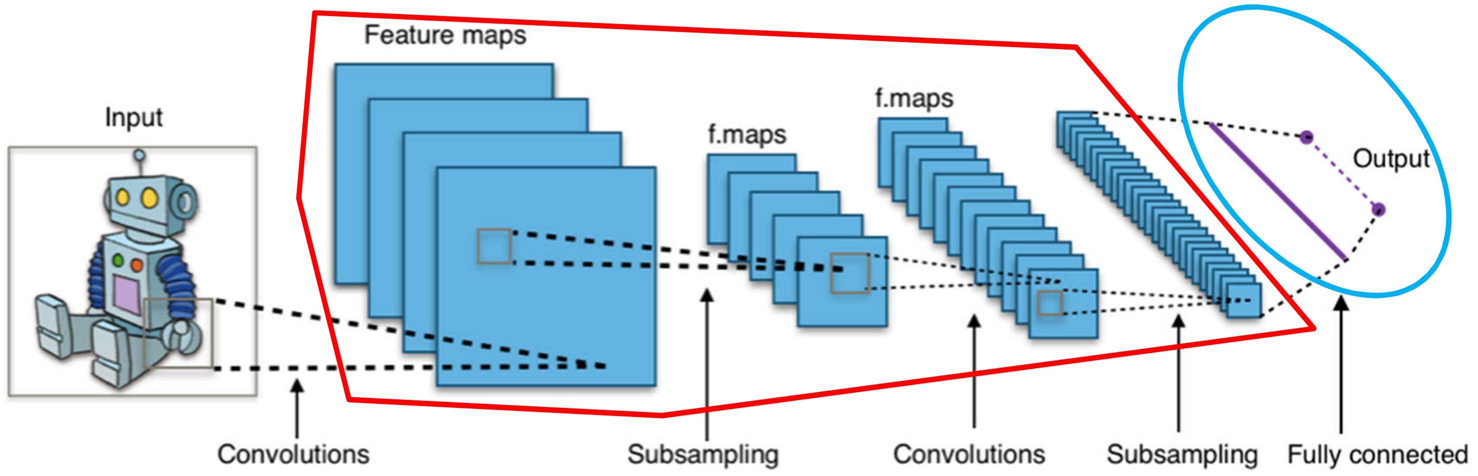
Convolutional Neural Networks



- conv→pool→conv→pool→…→MLP
- CNN = MLP with automatic feature extraction layers
- (but we’re still hand-crafting the structure)
- Structure is deep and wide with a huge number of connections, but much fewer unique parameters due to shared weights

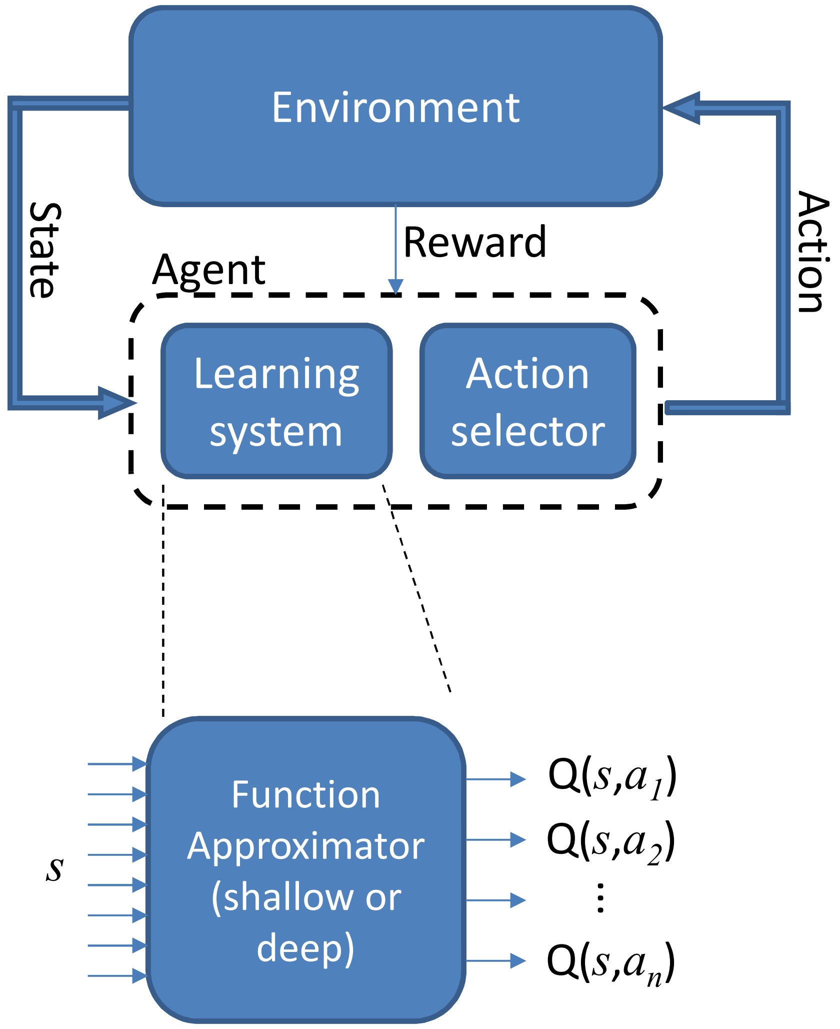
Transfer Learning





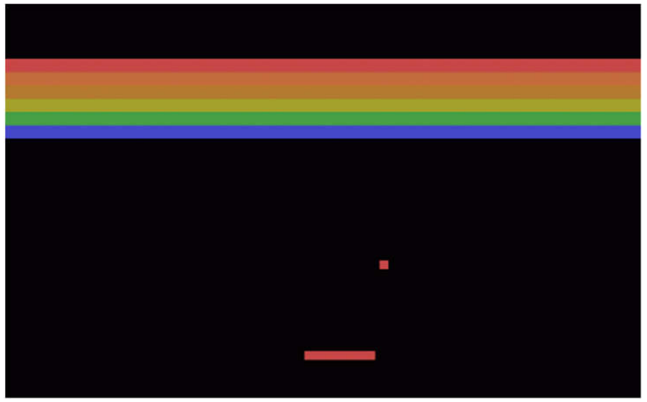
(Deep) Q-Learning

- Select and do action a_i
- Train that FA output, Q(s,a_i), on target r+γmaxQ(s',a')
- The other FA outputs can not be trained (have no target)
- Shallow or Deep FAs treated the same way
- But deep networks may give us automatic feature extraction ...
- Deep Atari



Deep Atari
DeepMind, Mnih *et al,* 2013

- Trained Deep Q-Learning system on 7 Atari 2600 games
- State: raw pixel data! (no manual feature extraction)
- The same configuration/hyperparameters for all games!
  – All previous attempts were specialized for one
- Outperformed all previous attempts on 6 games, outperformed human experts on 3
- Deep Q-Learning with CNNs
  – with some added useful tricks
    for example Experience Replay


Breakout

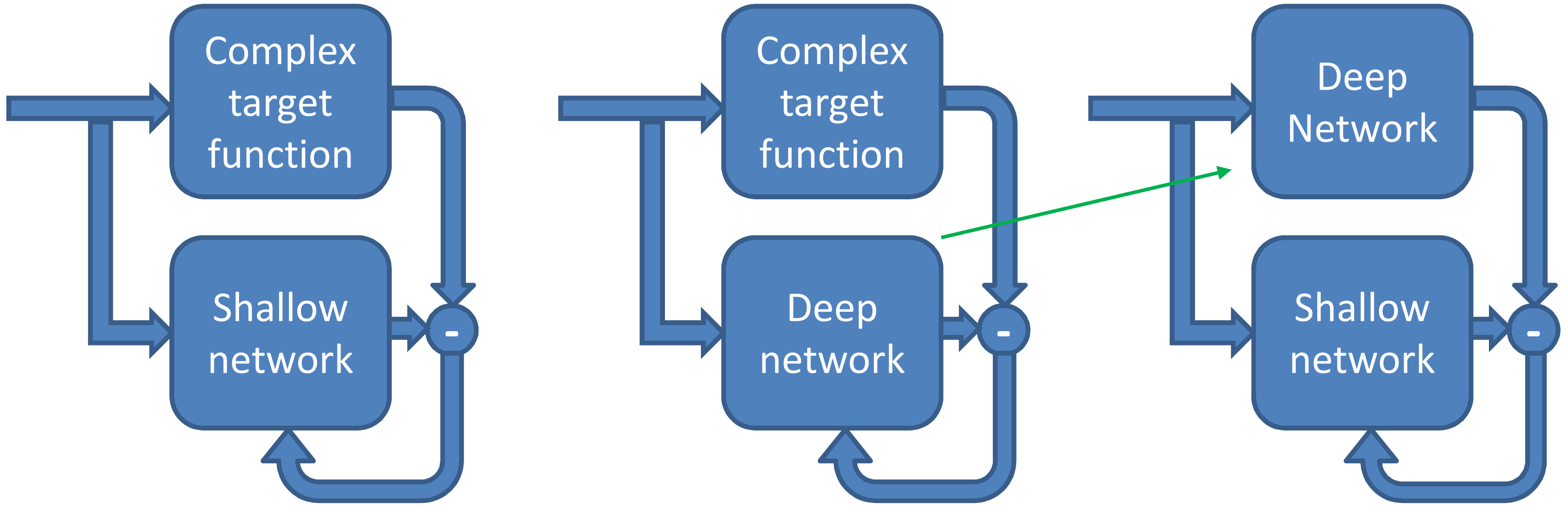
Experience Replay

(a neat trick also for shallow Q-Learning)

- After every transition, store the experience $<s_t, a_t, r_t, s_{t+1}>$
- This is everything we need to update Q(s,a)
  - but we don’t do it yet
- Instead, pick (randomly) a previously stored experience to update instead (the one we just stored will sooner or later be picked this way too)
- Better: select a small set of previous experiences, a 'mini-batch' to update (Deep Atari did that)
- Effect: Learning is now more off-line – better for the ANN

Do Deep Nets Have to be Deep?

(Ba & Caruana, 2014)



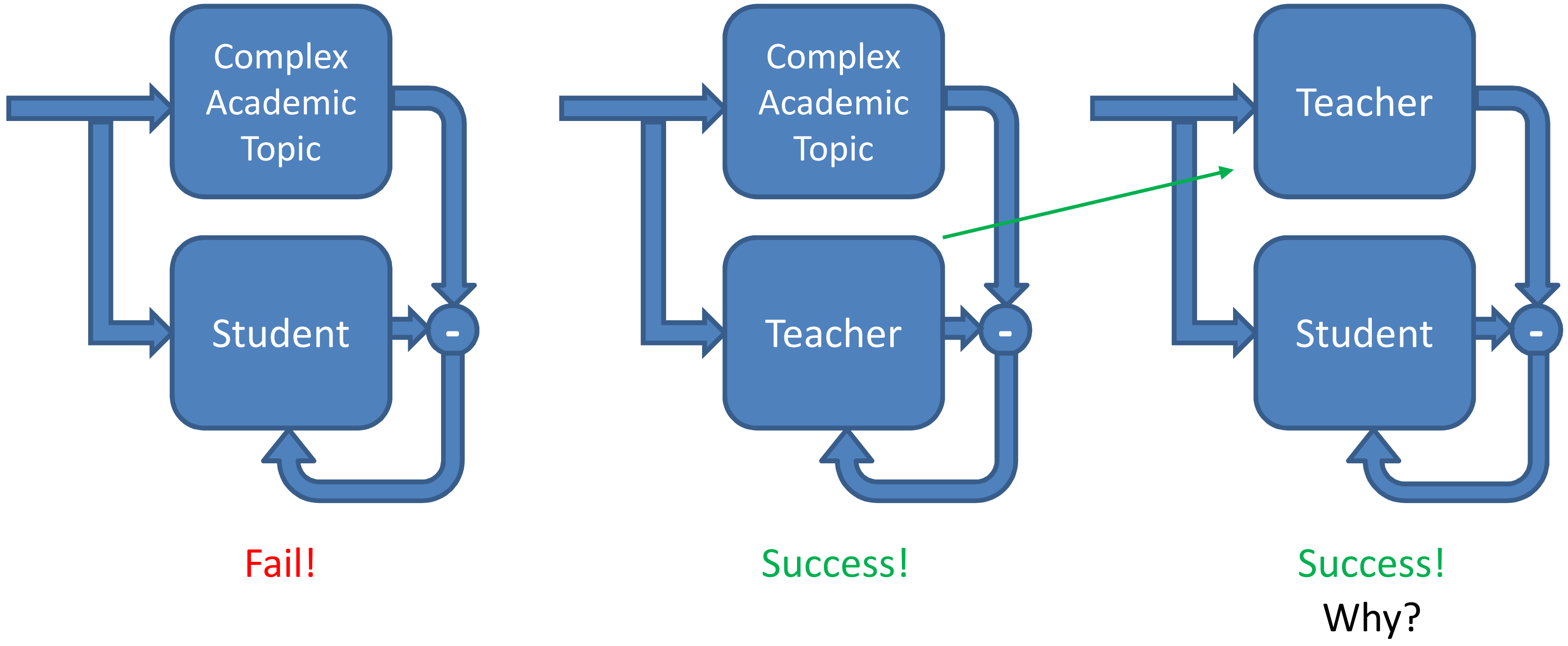
Fail!

Success!

Success!
Why?

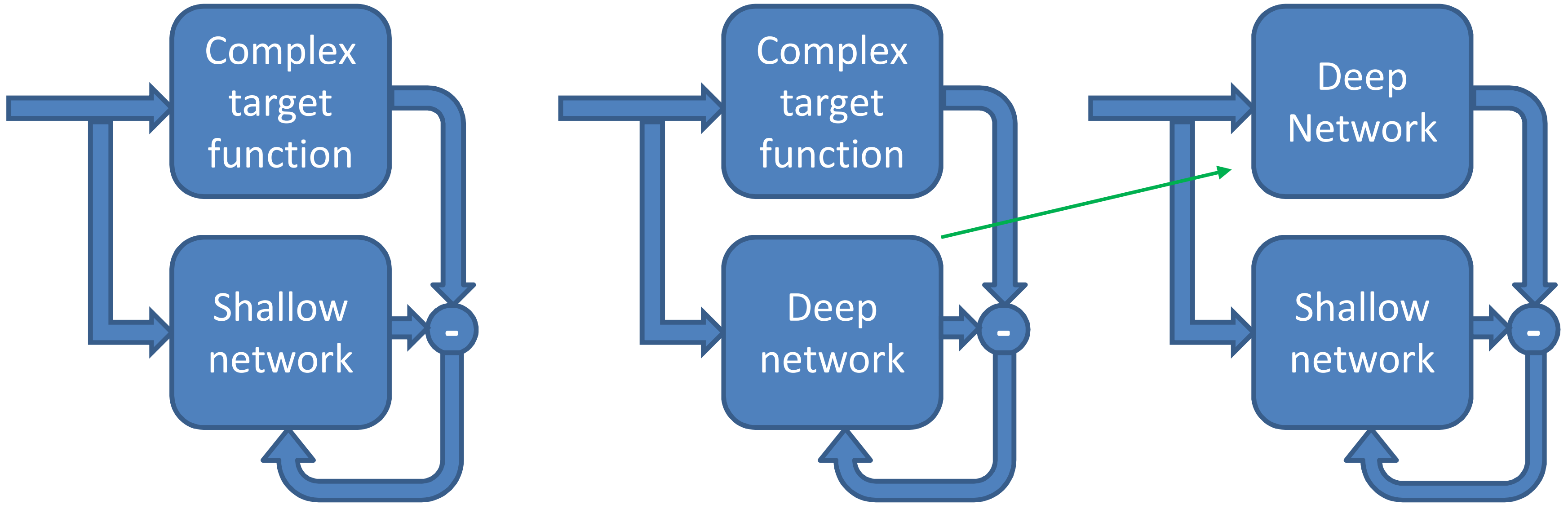
Do Deep Nets Have to be Deep?

(Ba & Caruana, 2014)



Do Deep Nets Have to be Deep?

(Ba & Caruana, 2014)



Fail!

Success!

Success!
Why?

- The representations do not have to be deep, but may be <u>easier to find</u>
- The shallow network often learns from the deep network faster, than the deep network did ➔ The deep network has 'cleaned' the problem
- The shallow network responds much faster! (fewer operations)