Introduction to Neural Networks and Deep Learning

Giuseppe Castellucci

Web Mining & Retrieval 2015/2016

18/04/2016

Outline

What is Deep Learning

- It is a branch of machine learning based on a set of algorithms that attempt to model high-level abstractions in data by using multiple processing layer

- *Learning representations* of data
  - feature hierarchies with features from higher levels of the hierarchy formed by the composition of lower level features

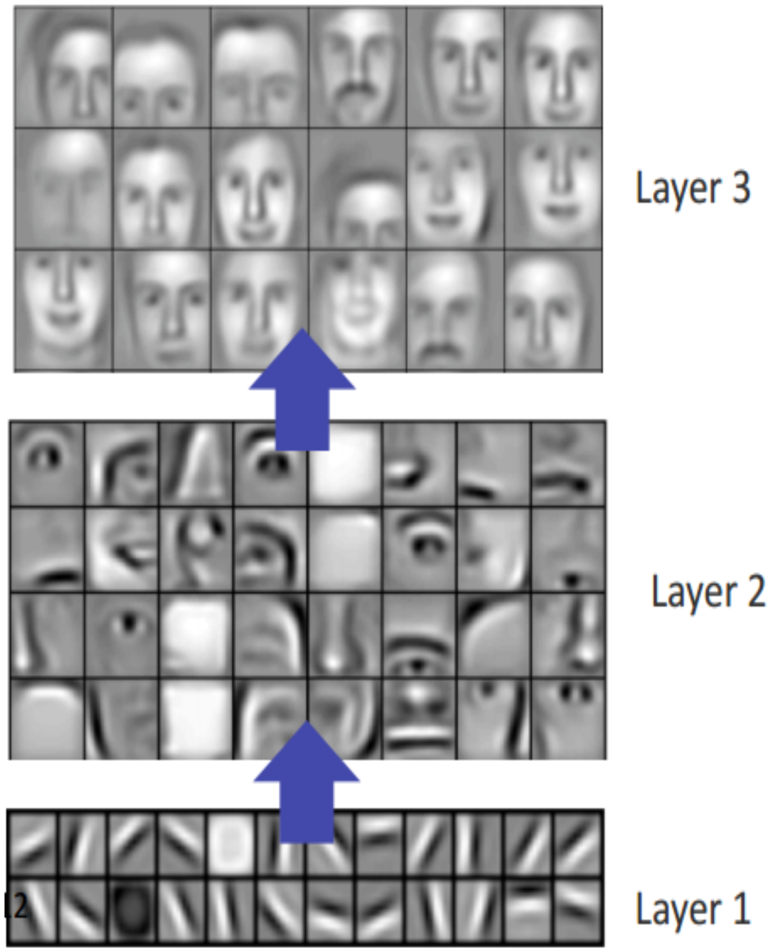
From Machine Learning…

- Machine Learning in general works well because of human-designed features
  - E.g. the so-called "Bag-of-Word" vector

- In this sense, machine learning is optimizing a set of parameters to obtain best performances
  - a costly operation
  - to be repeated for each new task

… to Deep Learning

- Representation Learning attempts at automatically learning the features (as well as the parameters)
- Deep Learning attempts at learning multiple levels (a hierarchy) of features of increasing complexity
- For example, in Face Detection
  - A face can be composed by eyes, nose, mouth
  - Each of them is composed from simpler shapes
- How to automatically learn these "features"?

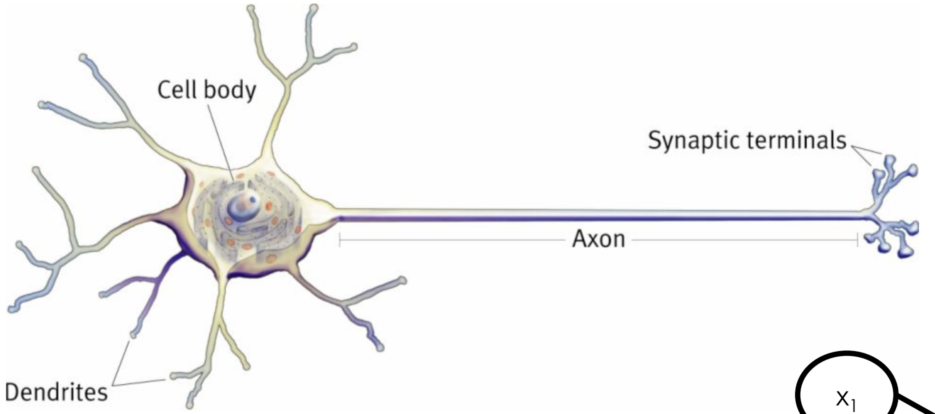


Neural Networks and DL

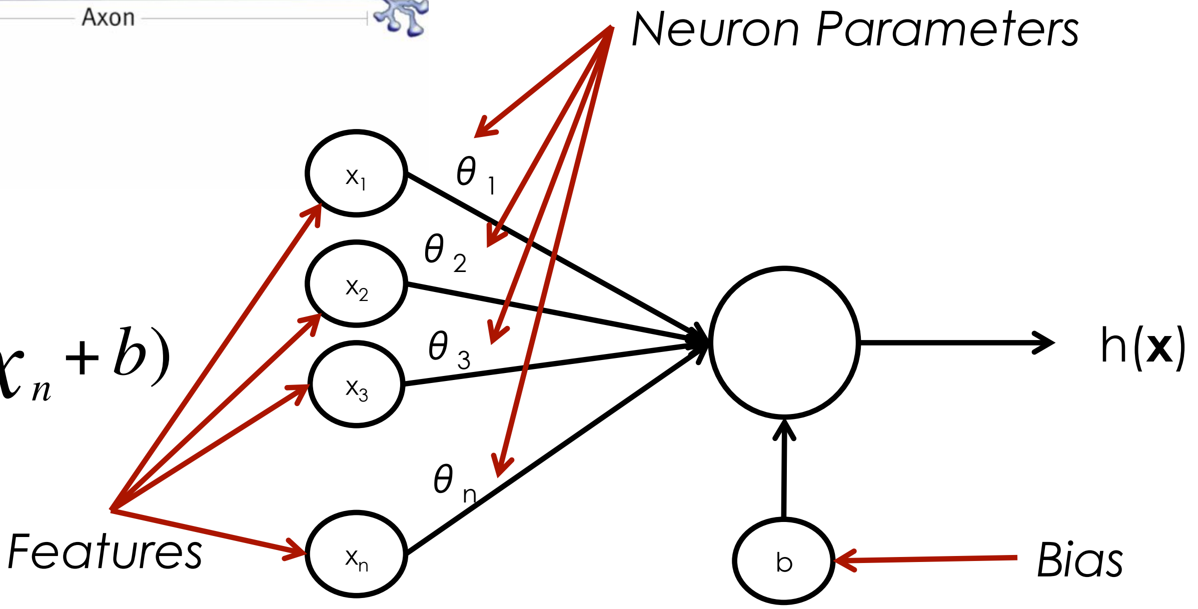
- Deep Learning models are mainly realized with Neural Networks
- Artificial Neural Networks dated back to 1950's (Rosenblatt, 1957)
- Until 2006 NNs were not adopted too much
- After that (see for details (Bengio, 2009))
  - more computational power with cheap hardware
  - new and more efficient training algorithms
  - exploiting of unlabeled data (pre-training)

Do You Remember the Perceptron?

- Linear Classifier mimicking a neuron



$$h(\vec{x}) = \operatorname{sgn}(\sum_n \theta_n x_n + b)$$

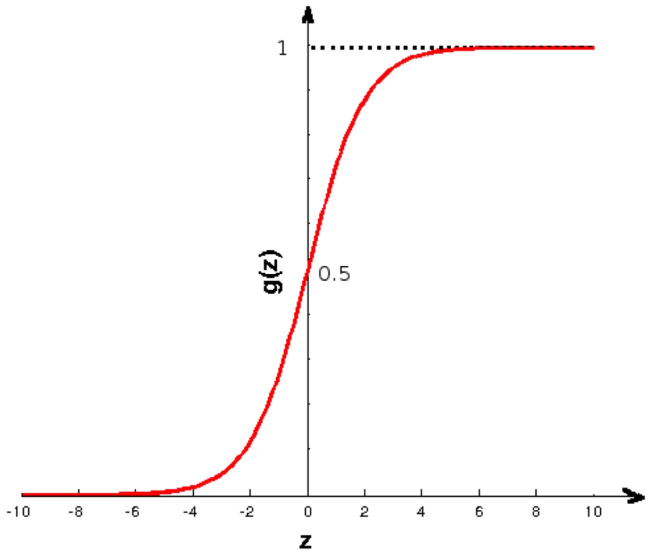


Perceptron and non-linear activation functions

- We can adopt the the *sigmoid* function instead of the *sgn*()
  - to bound the final values between 0 and 1
  - can be interpreted as probabilities of belonging to a class
  - belonging threshold is > 0.5
- It remains a linear classifier

$$g(z) = \frac{1}{1+e^{-z}}$$

$$h(\vec{x}) = g(\sum_n \theta_n x_n + b)$$

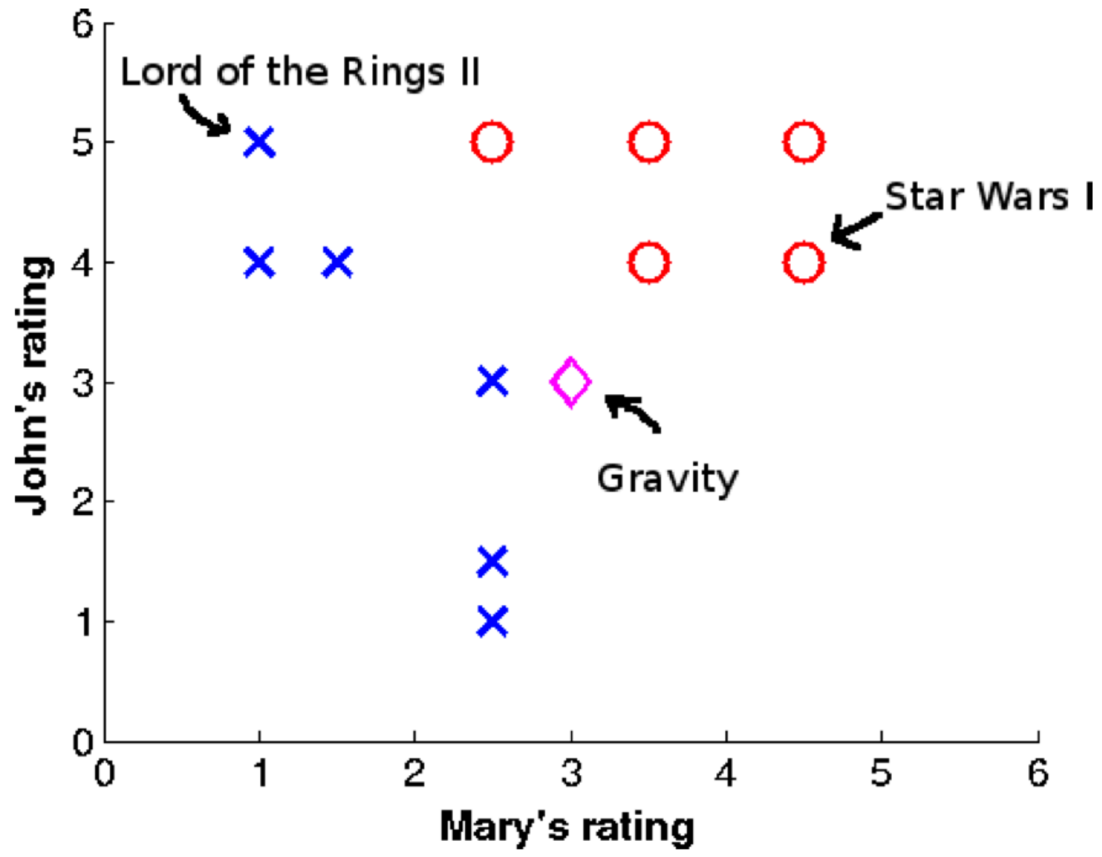


A machine learning problem (from Quoc Viet Le tutorial see references)

- Suppose we have to decide to watch *Gravity* or not based on the rates of two friends (John and Mary) on past films.
- For example,

Movie name	**Mary's rating**	**John's rating**	**I like?**
Lord of the Rings II	1	5	No
...	...	...	...
Star Wars I	4.5	4	Yes
Gravity	3	3	?

A machine learning problem (cont')



Building a linear classifier based on the perceptron

- Some notation
  - $\mathbf{x}_1$ is the Mary's rating
  - $\mathbf{x}_2$ is the John's rating
  - a past movie is associated to a label ***y***, 0 means we do not like the film, 1 means we like the film
- We want to approximate ***y*** with a computer program ***h*** that *decide* whether we'd like the film or not.
- The decision function will depend on the linear combination of $\mathbf{x}_i$

$$h = g(\theta_1 x_1 + \theta_2 x_2 + b) = g(\theta^T x + b)$$

How to induce *h* from examples

- Learn the parameters θ and b
- To find these we look at the past data (i.e. training data) optimizing an objective function
- **Objective function**: the error we make on the training data
  - the sum of differences between the decision function h and the label y
  - also called **Loss Function** or **Cost Function**

$$J(\theta,b)=\sum_{i=1}^{m}(h(x^{(i)};\theta,b)-y^{(i)})^2$$

A general training procedure: Stochastic Gradient Descent

- Optimizing J means minimizing it
  - it measures the errors we make on the training data.
- We can iterate over examples and update the parameters in the direction of smaller costs
  - we aim at finding the minimum of that function
- Concretely,

$$\theta_1 = \theta_1 - \alpha \Delta \theta_1$$

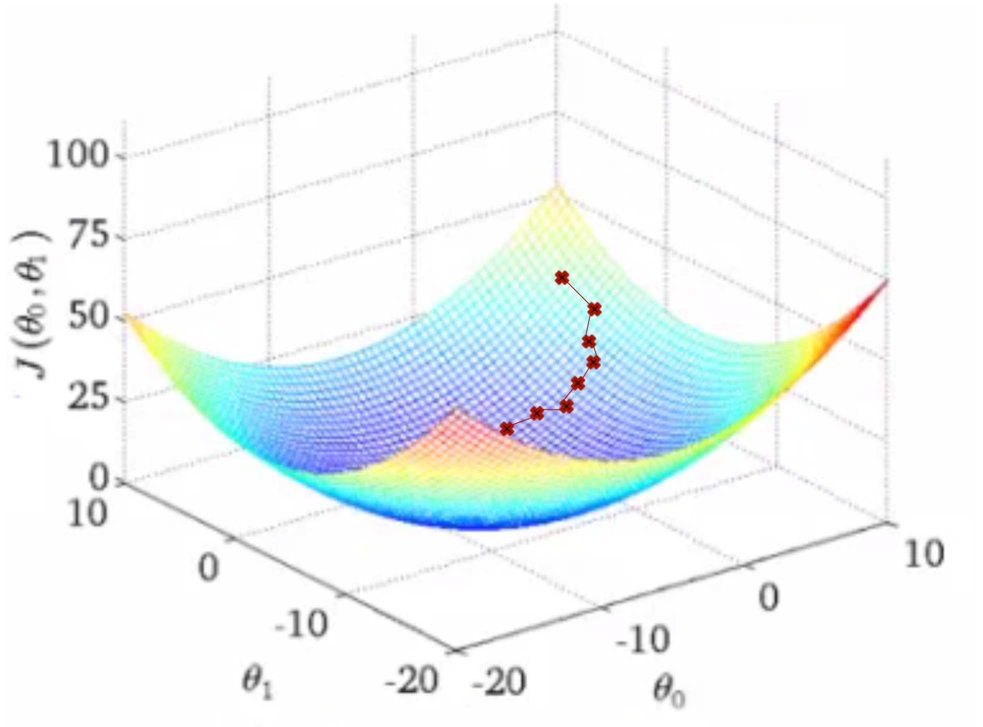
$$\theta_2 = \theta_2 - \alpha \Delta \theta_2$$

$$b = b - \alpha \Delta b$$

- α is a meta-parameter, the learning rate
- Δ are the partial derivatives of the cost function *wrt* each parameter

Why SGD?

- Weights are updated using the partial derivatives
- Derivative pushes down the cost following the steepest descent path on the error curve



SGD procedure

- Choose an initial random values for θand b
- Choose a learning rate
- Repeat until stop criterion is met:
  - Pick a random training example $x^{(i)}$
  - Update the parameters with

$$\theta_1 = \theta_1 - \alpha\Delta\theta_1$$

$$\theta_2 = \theta_2 - \alpha\Delta\theta_2$$

$$b = b - \alpha\Delta b$$

- We can stop
  - when the parameters do not change or,
  - the number of iteration exceeds a certain upper bound

Cost Function Derivative

- In order to update the parameters in SGD, we need to compute the **partial derivatives** *wrt* the learnable parameters.
- Remember the chain rule:
  - if h is a function of $z(x)$, then
  - the derivative of h *wrt* x is:

$$\frac{\vartheta h}{\vartheta x} = \frac{\vartheta h}{\vartheta z}\frac{\vartheta z}{\vartheta x}$$

- Thus, we need to compute
  - for an example $x^{(i)}$

$$\Delta\vartheta_1 = \frac{\vartheta}{\vartheta\theta_1}(h(x^{(i)};\theta,b) - y^{(i)})^2$$

$$\Delta\vartheta_2 = \frac{\vartheta}{\vartheta\theta_2}(h(x^{(i)};\theta,b) - y^{(i)})^2$$

$$\Delta b = \frac{\vartheta}{\vartheta b}(h(x^{(i)};\theta,b) - y^{(i)})^2$$

Cost Function Derivatives

$$\Delta\theta_1 = \frac{\vartheta}{\vartheta\theta_1}(h(x^{(i)};\theta,b) - y^{(i)})^2 =$$

$$= 2((h(x^{(i)};\theta,b) - y^{(i)})\frac{\vartheta}{\vartheta\theta_1}(h(x^{(i)};\theta,b)$$

$$= 2(g(\theta^T x^{(i)} + b) - y^{(i)})\frac{\vartheta}{\vartheta\theta_1}(g(\theta^T x^{(i)} + b))$$

$$g(z) = \frac{1}{1+e^{-z}}$$

$$\frac{\vartheta g}{\vartheta z} = (1 - g(z))g(z)$$

We have that:

$$\frac{\vartheta}{\vartheta\theta_1}(g(\theta^T x + b)) = \frac{\vartheta g(\theta^T x + b)}{\vartheta(\theta^T x + b)}\frac{\vartheta(\theta^T x + b)}{\vartheta\theta_1}$$

$$(1 - g(\theta^T x + b))g(\theta^T x + b)\frac{\vartheta(\theta_1 x_1 + \theta_2 x_2 + b)}{\vartheta\theta_1}$$

$$= (1 - g(\theta^T x + b))g(\theta^T x + b)x_1$$

Cost Function Derivatives

Then,

$$\Delta\theta_1 = 2[(g(\theta^T x^{(i)} + b) - y^{(i)})][(1 - g(\theta^T x^{(i)} + b))g(\theta^T x^{(i)} + b)x_1^{(i)}]$$

and we can do the same for θ_2

$$\Delta\theta_2 = 2[(g(\theta^T x^{(i)} + b) - y^{(i)})][(1 - g(\theta^T x^{(i)} + b))g(\theta^T x^{(i)} + b)x_2^{(i)}]$$

Cost Function Derivatives for b

- For the b parameters, the same steps apply:

$$\Delta b = \frac{\vartheta}{\vartheta b}(h(x^{(i)};\theta,b) - y^{(i)})^2 =$$

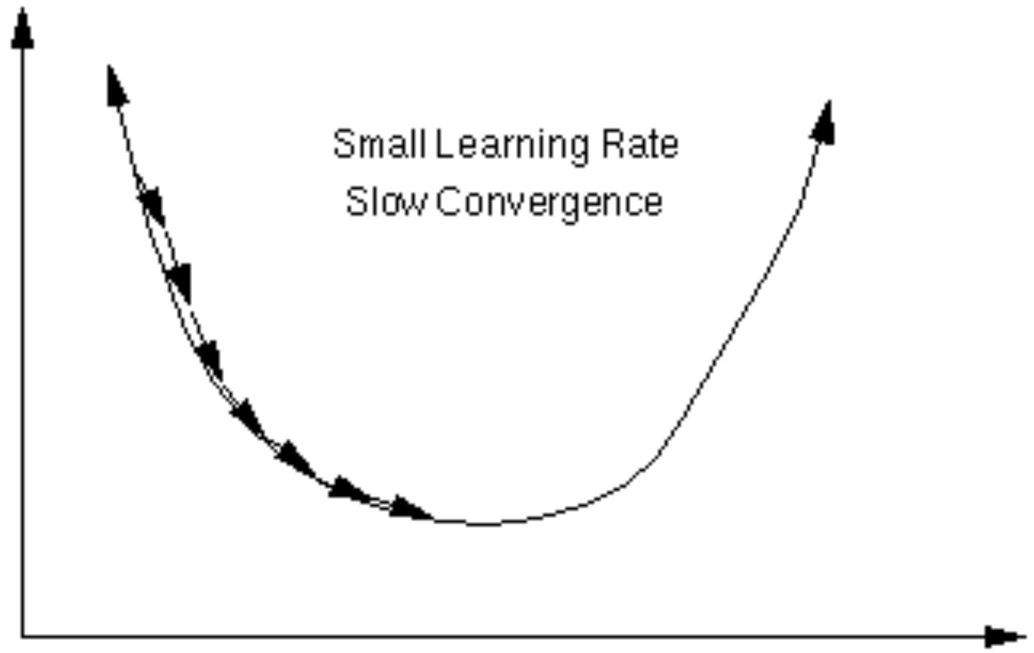
$$= 2((h(x^{(i)};\theta,b) - y^{(i)})\frac{\vartheta}{\vartheta b}(h(x^{(i)};\theta,b)$$

$$= 2(g(\theta^T x^{(i)} + b) - y^{(i)})\frac{\vartheta}{\vartheta b}(g(\theta^T x^{(i)} + b))$$

$$\frac{\vartheta}{\vartheta b}(g(\theta^T x + b)) = \frac{\vartheta g(\theta^T x + b)}{\vartheta(\theta^T x + b)}\frac{\vartheta(\theta^T x + b)}{\vartheta b} = (1 - g(\theta^T x + b))g(\theta^T x + b)$$

$$\Delta b = 2[(g(\theta^T x^{(i)} + b) - y^{(i)})][(1 - g(\theta^T x^{(i)} + b))g(\theta^T x^{(i)} + b)]$$

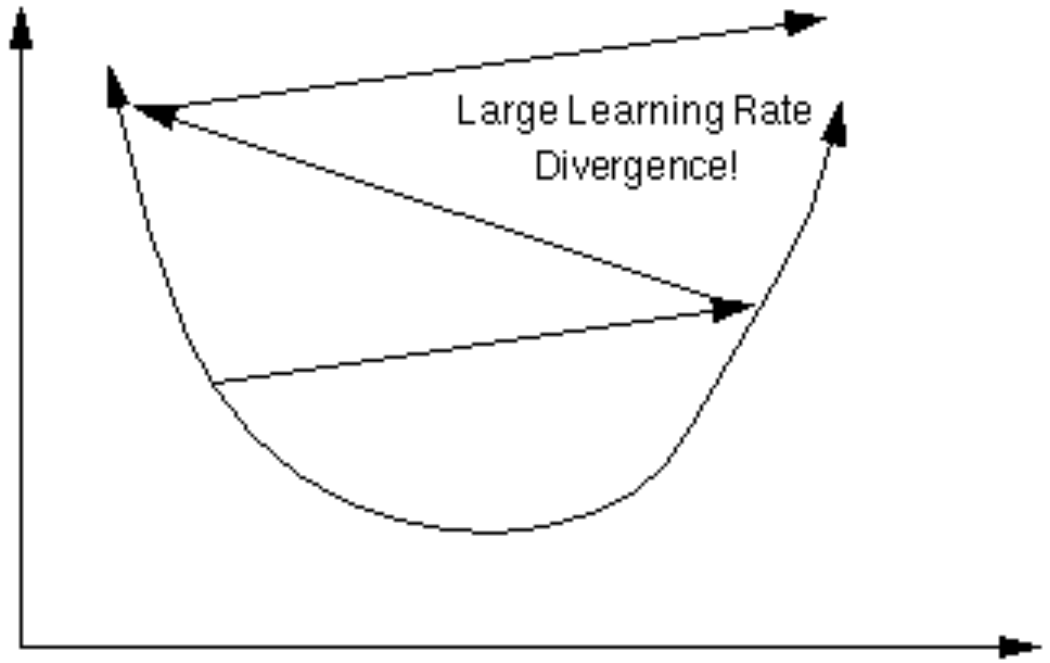
Learning rate: low values



- make the algorithm converge slowly
- it is a conservative and safer choice
- However, it implies very long training

$$\theta_1 = \theta_1 - \alpha\Delta\theta_1$$
$$\theta_2 = \theta_2 - \alpha\Delta\theta_2$$
$$b = b - \alpha\Delta b$$

Learning rate: high values

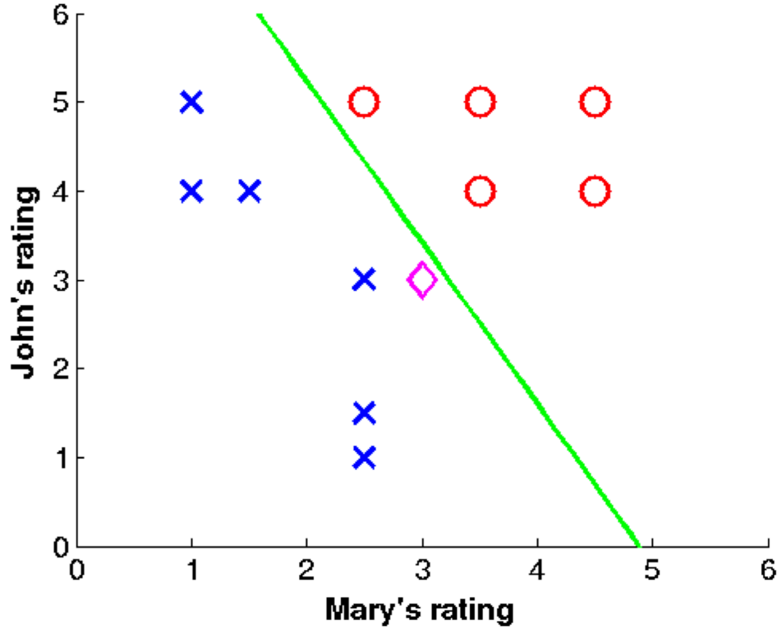


- make the algorithm converge quickly
- Training time is reduced
- it is a a less safer choice
  - risk of divergence

$$\theta_1 = \theta_1 - \alpha\Delta\theta_1$$
$$\theta_2 = \theta_2 - \alpha\Delta\theta_2$$
$$b = b - \alpha\Delta b$$

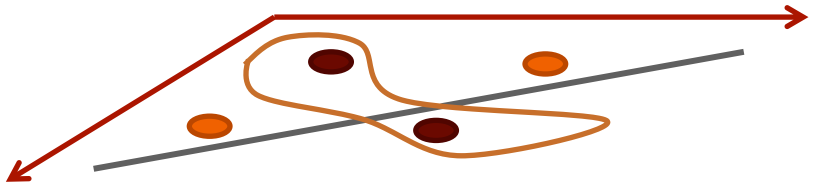
After training…

- We obtain a linear function *h*
- A plot of the decision function (boundary) will look like the green line in the figure
- A point lying above the line is a movie we do not like
- Else it is a movie we like



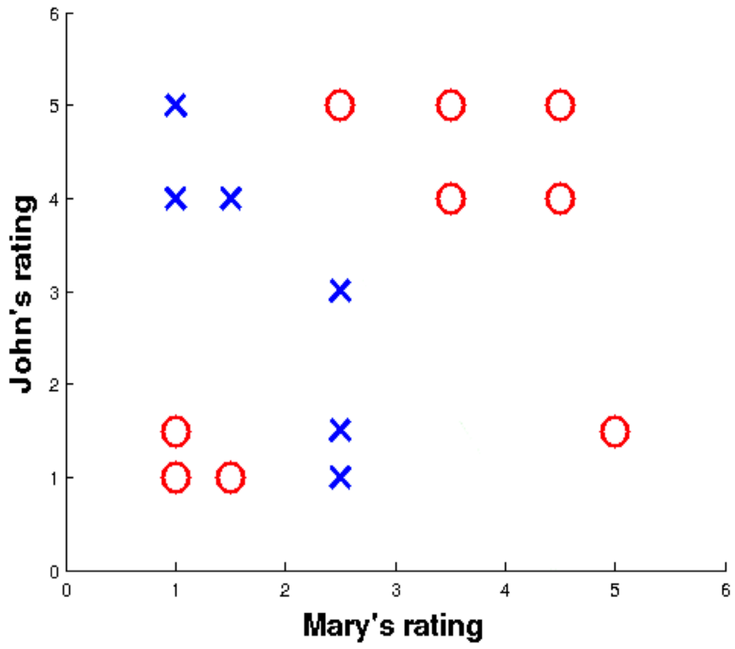
The problem with linear decision functions

- When the points are not linearly separable
  - linear decision functions cannot be adopted
- One solution is to project examples in a new space (the kernel solution)
- Another, is to adopt a more complex classifier

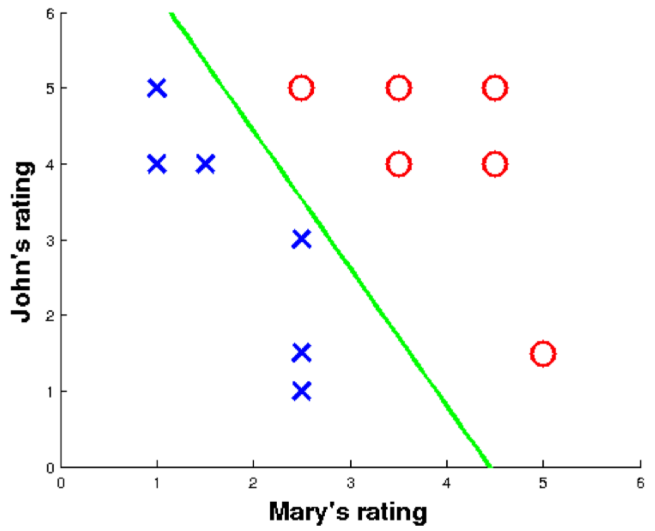


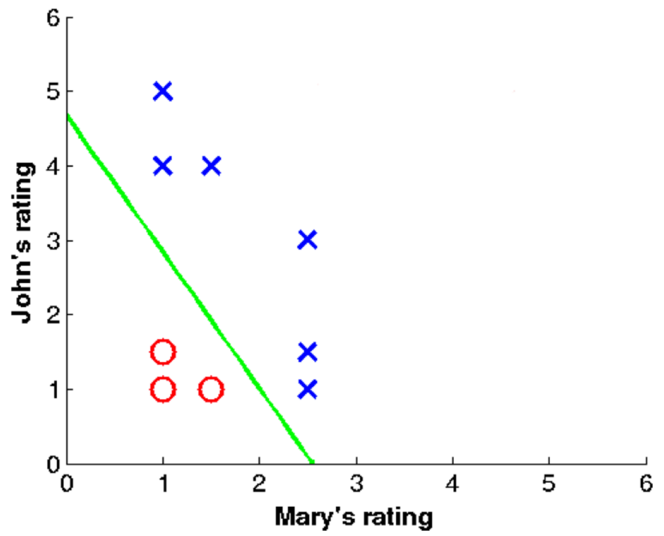
A non-linear problem

- Let us assume we know also the ratings of Susan in terms of Mary and John ratings
  - we don't add a third feature
  - a linear classifier in R^2 doesn't exist
- We can decompose the problem in
  - deciding in the bottom left
  - deciding in the top right
  - finally, combine the decisions with a new set of parameters



Decomposition

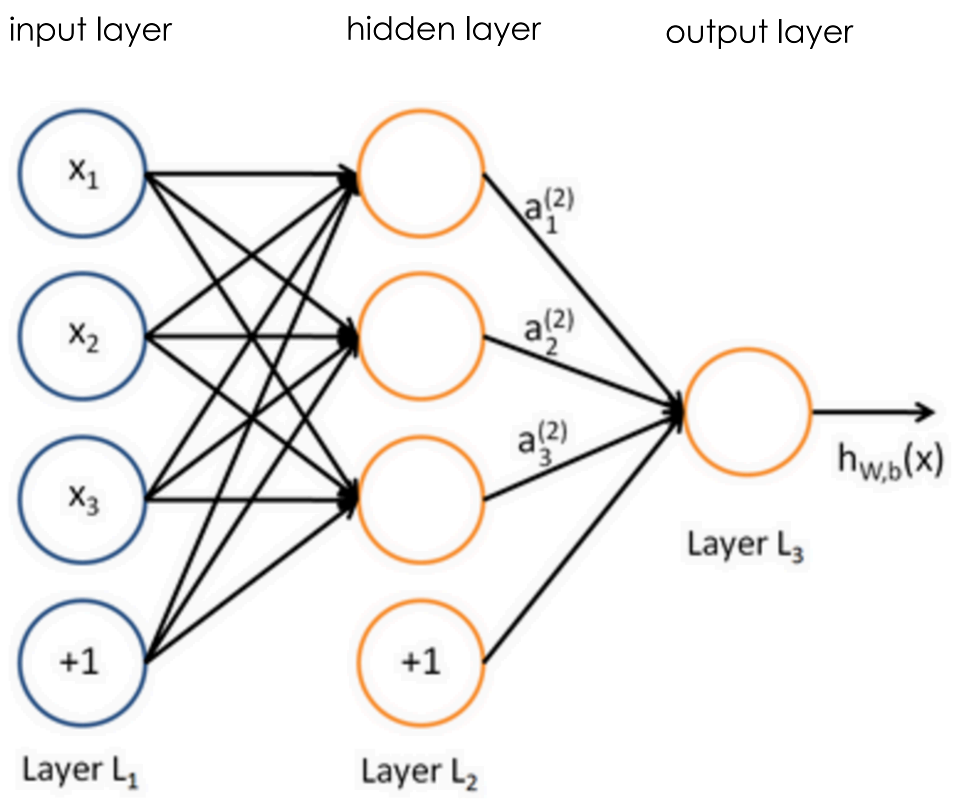




- The two functions must be adequately combined
  1. partition data and find h_1 and h_2
  2. combine h_1 and h_2, with a new set of parameters, to find the final decision function h

Neural Networks

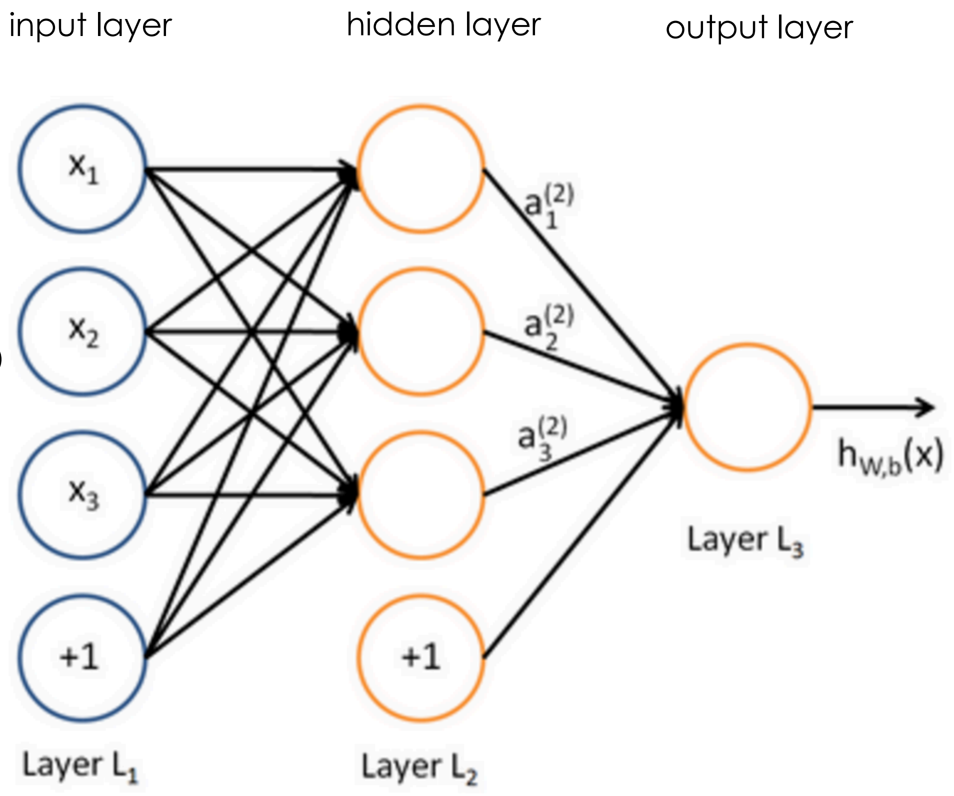
- Each circle represent a **neuron** (or unit)
  - 3 inputs, 3 hiddens and 1 output
- n_l=3 is the number of layers
- s_l denotes the number of units in layer l
- Layers:
  - Layer l is denoted as L_l
  - Layer l and l+1 are connected by a matrix of parameters $W^{(l)}$
    - $W^{(l)}_{i,j}$ connects neuron j in layer l with neuron I in layer l+1
- $b^{(l)}_i$ is the bias associated to neuron I in layer l+1



Neural Networks cont.

- $a^{(l)}_{I}$ is the activation of unit i in layer l
  - for $l=1$ $a^{(1)}_{i} = x_i$

$$a_1^{(2)} = f(W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2 + W_{13}^{(1)}x_3 + b_1^{(1)})$$
$$a_2^{(2)} = f(W_{21}^{(1)}x_1 + W_{22}^{(1)}x_2 + W_{23}^{(1)}x_3 + b_2^{(1)})$$
$$a_3^{(2)} = f(W_{31}^{(1)}x_1 + W_{32}^{(1)}x_2 + W_{33}^{(1)}x_3 + b_3^{(1)})$$
$$h_{W,b}(x) = a_1^{(3)} = f(W_{11}^{(2)}a_1^{(2)} + W_{12}^{(2)}a_2^{(2)} + W_{13}^{(2)}a_3^{(2)} + b_1^{(2)})$$



- We call $z^{(l)}_{i}$ the weighted sum of inputs to unit i in layer l, i.e.

$$z_i^{(2)} = \sum_{j=1}^{n} W_{ij}^{(1)} x_j + b_i^{(1)}$$
$$a_i^{(l)} = f(z_i^{(l)})$$

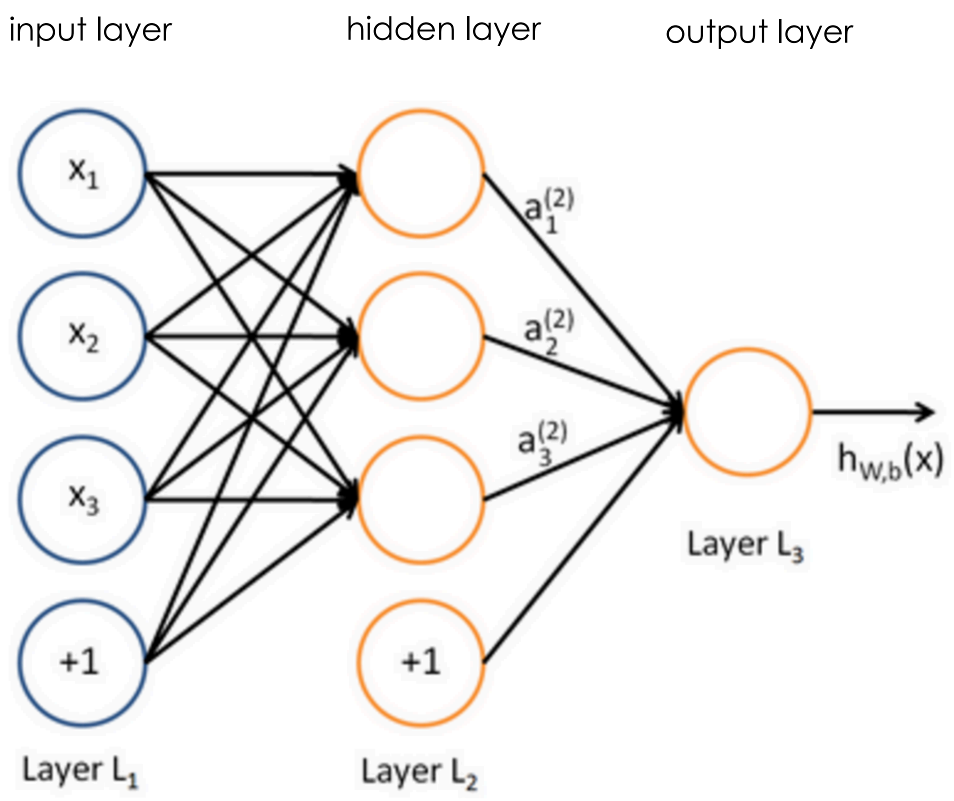
- f Is a non-linearity function
  - e.g. sigmoid

Neural Network Classification

- The classification corresponds in getting the value(s) in the output layer
- Propagating the input towards the network given W,b
- This process is called **forward propagation**

$$z^{(l+1)} = W^{(l)}a^{(l)} + b^{(l)}$$

$$a^{(l+1)} = f(z^{(l+1)})$$



How to Train a NN?

- We can re-use the gradient descent algorithm
  - define a cost function
  - compute the partial derivatives *wrt* to all the parameters
- As the NN models function composition
  - we are going to exploit the chain rule (again)
- Setup:
  - we have a training set of m examples
  - $\{(x^{(1)},y^{(1)}), \ldots, (x^{(m)},y^{(m)})\}$
  - x are the inputs and y are the labels

$h(z(x))$

$$\frac{\vartheta h}{\vartheta x} = \frac{\vartheta h}{\vartheta z}\frac{\vartheta z}{\vartheta x}$$

Cost Function of a NN

- Given a single training example (x,y) the cost is

$$J(W,b;x,y)=\frac{1}{2}\,|\,h_{W,b}(x)-y\,|^2$$

- For the whole training set J is the mean of the errors plus a regularization term (**weigth decay**)

$$J(W,b)=\frac{1}{m}\sum_{i=1}^{m}J(W,b;x^{(i)},y^{(i)})+\frac{\lambda}{2}\sum_{l=1}^{n_l-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(W_{ji}^{(l)})^2$$

$$=\frac{1}{m}\sum_{i=1}^{m}\left(\frac{1}{2}\,|\,h_{W,b}(x^{(i)})-y^{(i)}\,|^2\right)+\frac{\lambda}{2}\sum_{l=1}^{n_l-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(W_{ji}^{(l)})^2$$

- λcontrols the importance of the two terms (it has a similar role to the C parameter in SVM)

A GD step

- A GD step update the parameters according to

$$W_{ij}^{(l)} = W_{ij}^{(l)} - \alpha \frac{\vartheta}{\vartheta W_{ij}^{(l)}} J(W,b)$$

$$b_{i}^{(l)} = b_{i}^{(l)} - \alpha \frac{\vartheta}{\vartheta b_{i}^{(l)}} J(W,b)$$

- where α is the learning rate.
- The partial derivatives are computes with the **Backpropagation** algorithm

The backpropagation algorithm

- First, we compute for each example $\frac{\vartheta}{\vartheta W_{ij}^{(l)}} J(W,b,x^{(i)},y^{(i)})$
- Backpropagation works as follow:
  1. do a forward pass for an example $x^{(i)}, y^{(i)}$
  2. for each node i in layer l, compute an error term δ^{l}_{i}
     1. it measures how unit i is responsible for the error on the current example
  3. The error of an output node is the difference between the true output value and the predicted one
  4. For the intermediate layer l, a node receives a portion of the error based on the units it is linked to of the layer $l+1$
- Partial derivatives will be computed given the error terms

The backpropagation algorithm cont.

1. Perform a forward propagation for an example
2. For each unit i in the output layer (n_l)

$$\delta_i^{(n_l)} = \frac{\vartheta}{\vartheta z_i^{(n_l)}} | y - h_{W,b}(x) |^2 = -(y_i - a_i^{(n_l)}) \cdot f'(z_i^{(n_l)})$$

3. For $l=n_l-1,\ldots,2$
   1. for each node i in layer l $\delta_i^{(l)} = (\sum_{j=1}^{s_{l+1}} W_{ji}^{(l)} \delta_j^{(l+1)}) f'(z_i^{(l)})$
4. Compute the partial derivatives as:

$$\frac{\vartheta}{\vartheta W_{ij}^{(l)}} J(W,b;x,y) = a_j^{(l)} \delta_i^{(l+1)}$$

$$\frac{\vartheta}{\vartheta b_i^{(l)}} J(W,b;x,y) = \delta_i^{(l+1)}$$

The backpropagation algorithm (vectorial notation)

$$a = b \bullet c = b_i \bullet c_i$$

$$f'([z_1, z_2, z_3]) =$$
$$[f'(z_1), f'(z_2), f'(z_3)]$$

1. Perform a forward propagation for an example
2. For each unit i in the output layer (n_l)

$$\delta^{(n_l)} = -(y_i - a_i^{(n_l)}) \bullet f'(z_i^{(n_l)})$$

3. For $l = n_l - 1, \ldots, 2$
    1. for each node i in layer l $\delta^{(l)} = ((W^{(l)})^T \delta^{(l+1)}) \bullet f'(z_i^{(l)})$
4. Compute the partial derivatives as:

$$\nabla_{W^{(l)}} J(W, b; x, y) = \delta^{(l+1)} (a^{(l)})^T$$
$$\nabla_{b^{(l)}} J(W, b; x, y) = \delta^{(l+1)}$$

The full backpropagation algorithm

1. Set $\Delta W^{(l)}=0$, $\Delta b^{(l)}=0$ for all l
2. for $i=1$ to m
   1. Compute $\nabla_{W^{(l)}} J(W,b;x,y) = \delta^{(l+1)} (a^{(l)})^T, \nabla_{b^{(l)}} J(W,b;x,y) = \delta^{(l+1)}$
   2. Set $\Delta W^{(l)} = \Delta W^{(l)} + \nabla_{W^{(l)}} J(W,b;x,y)$

      $\Delta b^{(l)} = \Delta b^{(l)} + \nabla_{b^{(l)}} J(W,b;x,y)$
   3. Update the parameters with:

$$W^{(l)} = W^{(l)} - \alpha[(\frac{1}{m}\Delta W^{(l)}) + \lambda W^{(l)}]$$

$$b^{(l)} = b^{(l)} - \alpha[(\frac{1}{m}\Delta b^{(l)})]$$

NNs advices

- Randomly initialize the parameters of the network
  - for symmetry breaking
- Remember that the function f is a non-linear activation function
  - if f is the sigmoid

$$f(z) = \frac{1}{1+e^{-z}}$$

$$f'(z) = (1 - f(z))f(z)$$

- Moreover, during back-propagation we need to compute
  - activations values can be cached from the forward propagation step!

$$f'(z_i^{(l)}) = (1 - f(z_i^{(l)}))f(z_i^{(l)}) = (1 - a_i^{(l)})a_i^{(l)}$$

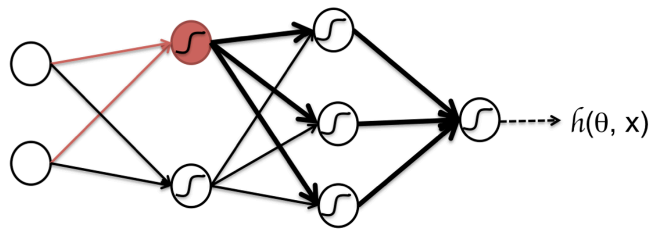
- If you must perform multi-classification
  - there will be an output unit for each of the labels

Some considerations

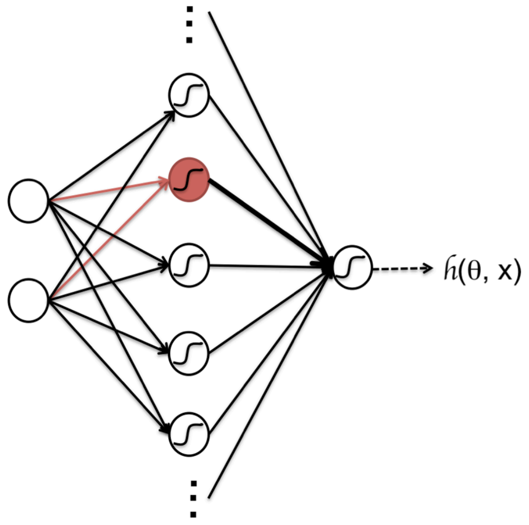
- How to stop and select the best model
  - Waiting the iteration in which the cost function doesn't change significantly
    - Risk of overfitting
- **Early stopping**
  - Provide hints as to how many iterations can be run before overfitting
  - Split the original training set into a new training set and a validation set
  - Train only on the training set and evaluate the error on the validation set
  - Stop training as soon as the error is higher than it was the last time
  - Use the weights the network had in that previous step
- **Dropout**
  - another form of regularization to avoid overfitting data
  - during training (only) randomly "turn off" some of the neurons of a layer
  - it prevents co-adaptation of units between layers

Deep vs Shallow Networks

- Deep networks should be preferred to Shallow ones
  - when problems are non-linear;
  - is has been observed that a shallow network needs about 10x number of neurons for reaching the expressivity of a deep one



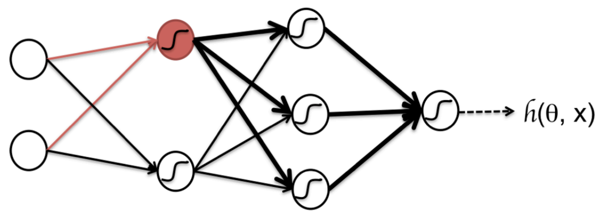
Deep Network



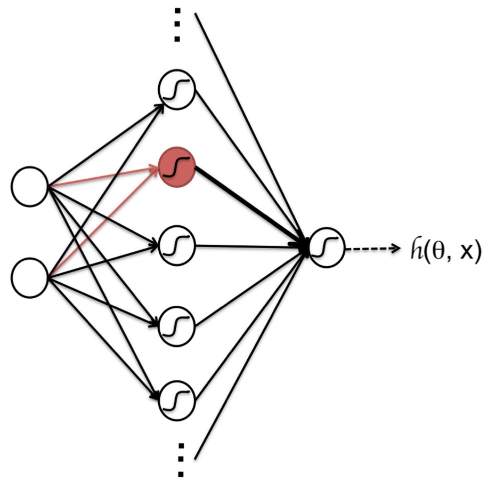
Shallow Network

Deep vs Shallow Networks: Intuition

- Think of a neuron as a program routine
  - in Deep Networks a neuron computation is re-used many times in the computation
  - in a Shallow Network it is used only once
- Using a shallow network is similar to writing a program without the ability of calling subroutines



Deep Network



Shallow Network

Deep Networks vs. Kernel

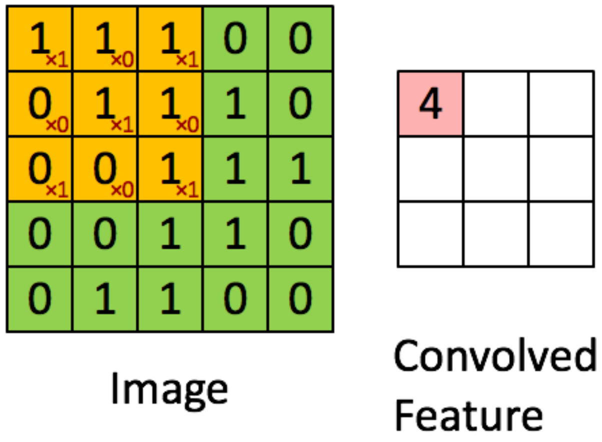
- A kernel machine can be thought of as a shallow network having a huge hidden layer
  - this hidden layer is never computed thanks to the kernel trick
- Kernel methods however are expensive
  - they rely on a set of examples, support vectors
  - for large dataset and complex problems this set can be large as well
- Neural networks computation
  - is independent on the dataset,
  - but only on the number of connections that have been choosen

Recent successes in Deep Learning

- Convolution Neural Networks
  - Images related tasks
- Recurrent Neural Networks
  - Language models
  - Speech to Text
  - Machine Translation
  - Conversation Models
- Advanced architectures
  - Image to Captions

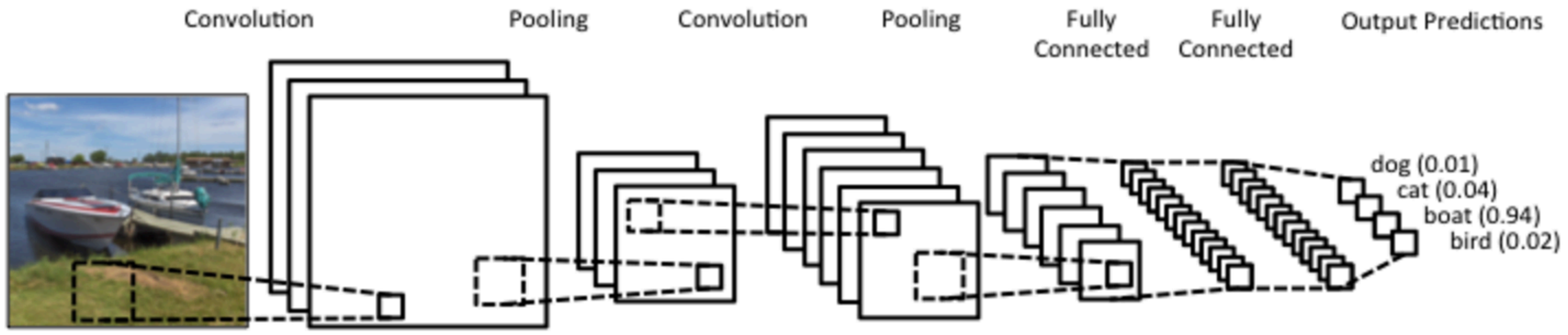
Convolutional Neural Networks

- Mainly used for images related tasks
  - image classification
  - face detection
  - etc…
- **Learn feature representations**
  - by ***convolving*** over the input of a layer
  - with a ***filter***, that slides over the input image
- **Compositionality** (local)
  - Each filter composes a local patch of lower-level features into a higher-level representation
- **Location Invariance**
  - the detection of specific patterns is independent of where it occurs



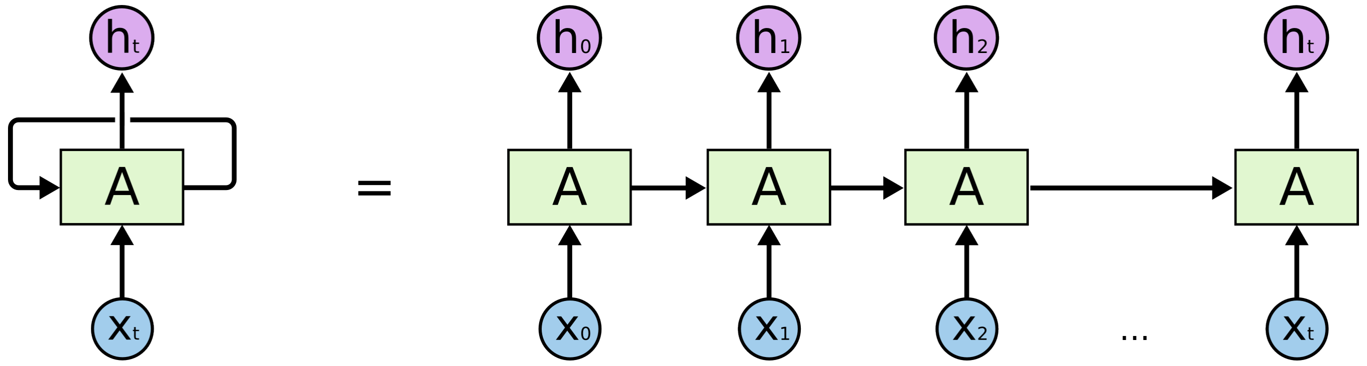
Convolutional Neural Networks

- CNNs automatically learn the parameters of the filters
  - a filter is a matrix of parameters
  - the key aspect is that a filter is adopted for the whole image
- Convolution can be applied in **multiple** layers
  - a layer $l+1$ is computed by convolving over output produced in layer l
- Pooling is an operation often adopted for taking the most informative features that are learned after a convolution step



Recurrent Neural Networks

- Used mainly to model sequences
  - naturally applied to textual and speech problems
- A representation at time step i is made dependent on the representations of the preceding steps
  - connections between units form a directed cycle

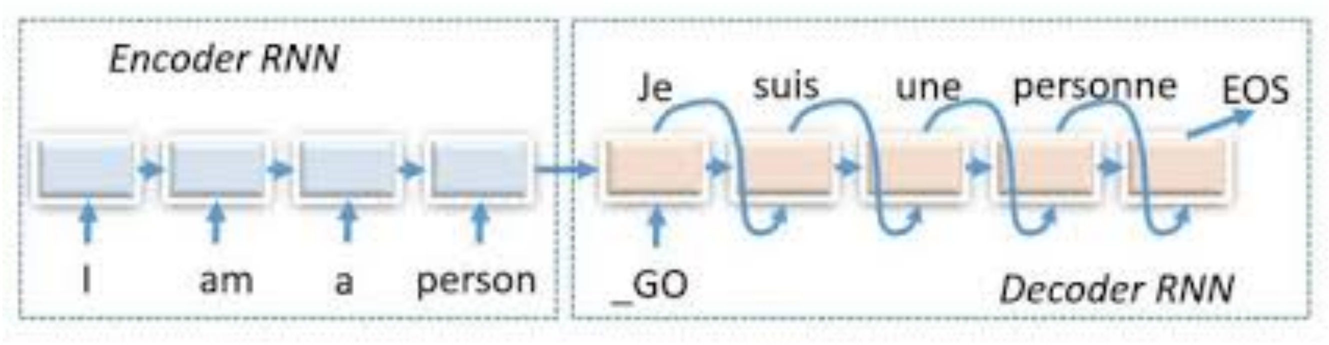


Recurrent Neural Networks

- Commons tasks are
  - ***language models***: predict the next word in a sentence given the already seen word
  - ***speech recognition***: predict a word given the current wave form and the preceding words
  - ***machine translation***: produce a sequence in a target language given an input sequence in a source language
- The most famous and effective model of RNNs are the Long-Short Term Memory (LSTM) Networks (Sepp Hochreiter and Jürgen Schmidhuber, 1997)
  - they are meant to better dealt with long-range dependencies

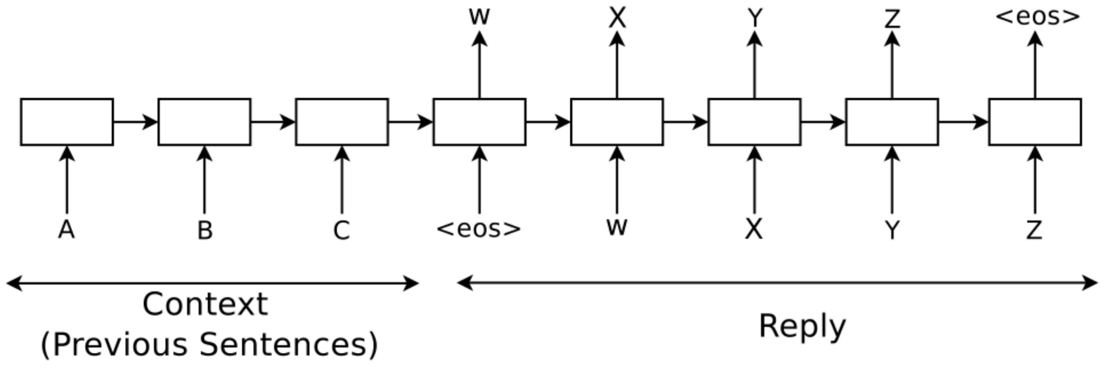
Recurrent Neural Networks: Machine Translation

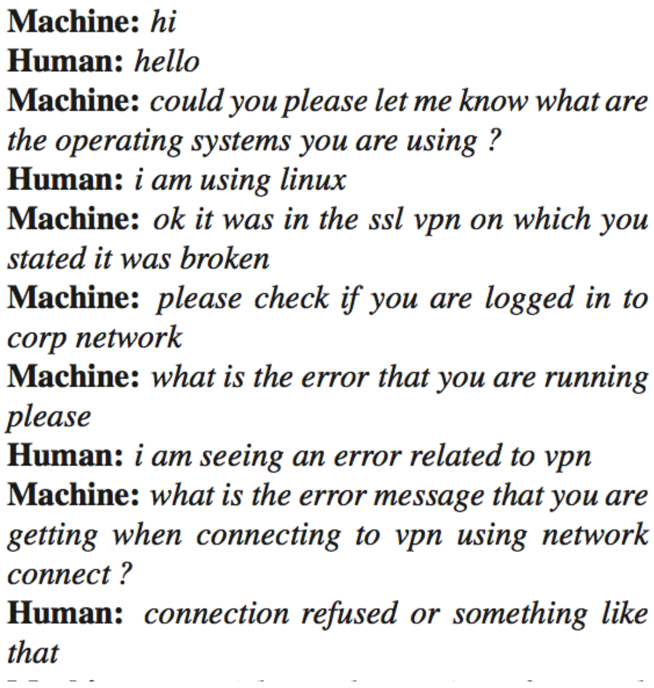
- A MT architecture is composed by a
  - ***Encoder***: it encodes the input x in the source language in a compact and fixed-size representation h
  - ***Decoder***: it decodes from h a new sequence in the target language (hopefully the translation of x)



RNNs: Neural Conversation Model

- Learn to answer question in a conversation
- For example, an IT help desk
  - model the input sequence
  - produce a output sequence

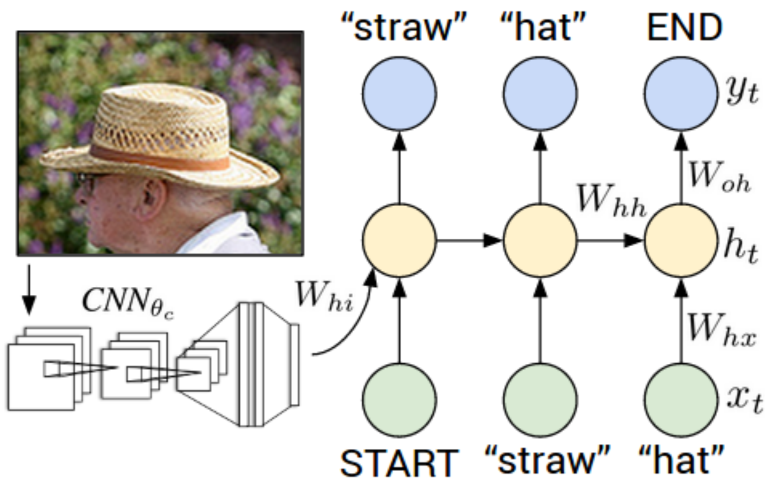




Machine: *hi*
Human: *hello*
Machine: *could you please let me know what are the operating systems you are using ?*
Human: *i am using linux*
Machine: *ok it was in the ssl vpn on which you stated it was broken*
Machine: *please check if you are logged in to corp network*
Machine: *what is the error that you are running please*
Human: *i am seeing an error related to vpn*
Machine: *what is the error message that you are getting when connecting to vpn using network connect ?*
Human: *connection refused or something like that*

Advanced architectures

- **Image to captions**
  - Convolutional Neural Network to learn a representation of the image
  - (Bi-directional) Recurrent Neural Network to generate a caption describing the image
    - its input is the representation computed from the CNN
    - its output is a sequence of words, i.e. the caption



"baseball player is throwing ball in game."

Bibliography

- Rosenblatt Frank. The Perceptron - a perceiving and recognizing automaton. Report 85-460-1, Cornell Aeronautical Laboratory
- Bengio Yoshua. Learning Deep Architectures for AI. Foundations and Trends in Machine Learning 2 (1): 1–127.
- Deep Visual-Semantic Alignments for Generating Image Descriptions. Andrej Karpathy, Li Fei-Fei, CVPR 2015
- Convolutional Neural Networks:
  - http://cs231n.github.io/ : stanford course on CNN for visual recognition with online (free) materials
  - http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/
  - Yann LeCun and Yoshua Bengio. 1998. Convolutional networks for images, speech, and time series. In The handbook of brain theory and neural networks, Michael A. Arbib (Ed.). MIT Press, Cambridge, MA, USA 255-258.
- Recurrent Neural Networks:
  - http://karpathy.github.io/2015/05/21/rnn-effectiveness/
  - http://colah.github.io/posts/2015-08-Understanding-LSTMs/
  - http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1/introduction-to-rnns/
  - Sepp Hochreiter and Jürgen Schmidhuber (1997). "Long short-term memory”. Neural Computation 9 (8): 1735–1780.
  - Graves, Alex; Mohamed, Abdel-rahman; Hinton, Geoffrey (2013). "Speech Recognition with Deep Recurrent Neural Networks". Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on: 6645–6649.
  - Vinyals, Oriol, and Quoc Le. "A neural conversational model." arXiv preprint arXiv:1506.05869 (2015).

Resources

- Most of this slides are based on
  - https://cs.stanford.edu/~quocle/tutorial1.pdf
  - http://web.stanford.edu/class/cs294a/sparseAutoencoder_2011new.pdf
- Software packages
  - Tensorflow
  - Theano
  - DeepLearning4j
  - Caffe
  - Torch
- Other useful resources can be found on the course website