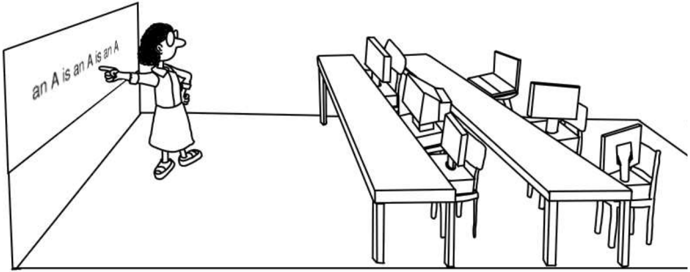
Machine Learning

Krystian Mikolajczyk & Deniz Gunduz

Department of Electrical and Electronic Engineering

Imperial College London



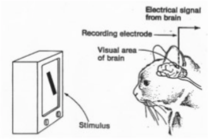
Part 4.2 summary

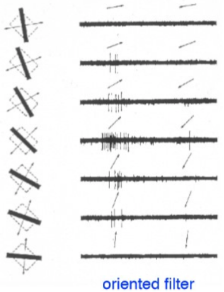
Department of Electrical and Electronic Engineering
Imperial College London

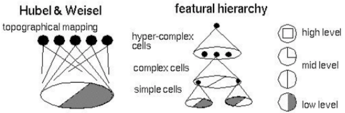
- Neural Networks

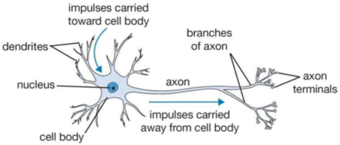
Neural Network: Binary class perceptron

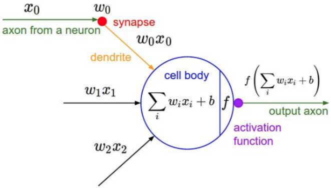
- Machine for binary classifications (Rosenblatt 1958)
- Receptive fields (Hubel and Wiesel 1959)



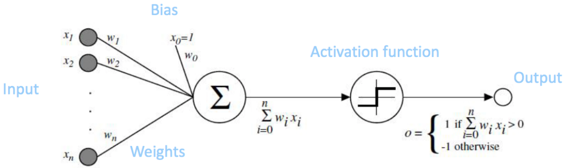




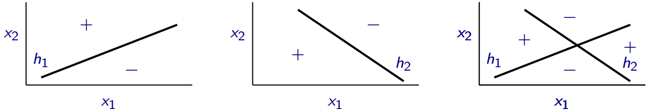


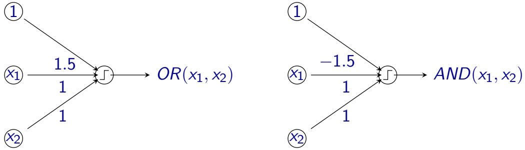


Neural Network: Perceptron



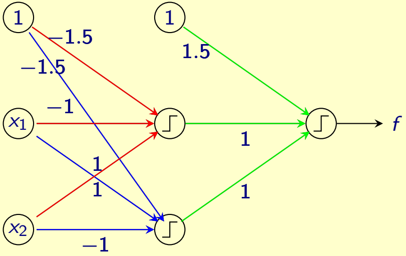
- perceptron is the model of a single neuron





However, the exclusive or (XOR) cannot be solved by a single layer

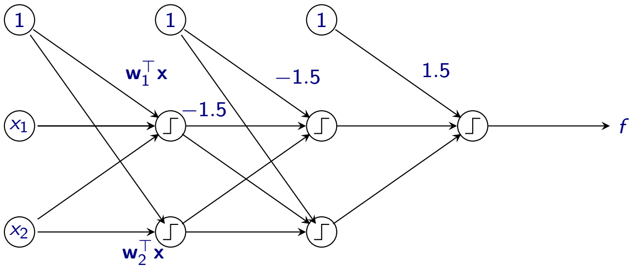
Example: Multilayer perceptron

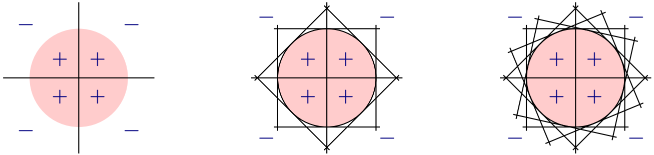


$$h_1 \bar{h}_2 + \bar{h}_1 h_2$$

XOR with 2 layer MLP

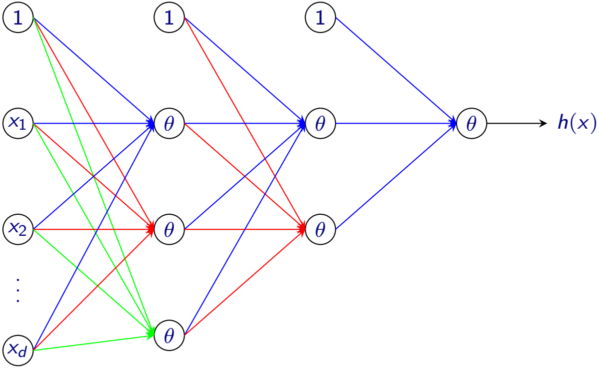
Neural Network: Multilayer perceptron

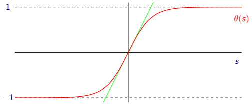




Neural network

- Replace sign with some other non-linear function θ.

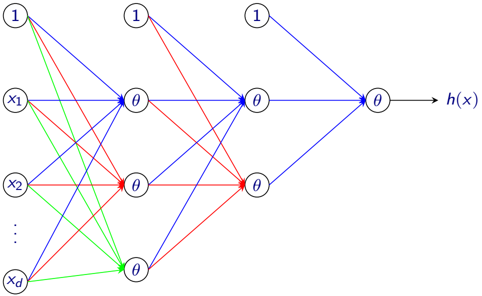




Input x hidden layers $1 \leqslant l < L$ output layer L

Neural Network: Formal description

- Weights $w_{ij}^{(l)}$ $\begin{cases} 1 \leqslant l \leqslant L \text{ layers;} \\ 0 \leqslant i \leqslant d^{(l-1)} \text{ inputs;} \\ 1 \leqslant j \leqslant d^{(l)} \text{ outputs} \end{cases}$



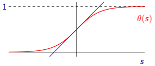
- Outputs $x_j^{(l)} = \theta(s_j^{(l)}) = \theta\left(\sum_{i=0}^{d^{(l-1)}} w_{ij}^{(l)} x_i^{(l-1)}\right)$

- Input $\mathbf{x}$ is applied to the input layer $x_1^{(0)}, \ldots, x_{d^{(0)}}^{(0)} \quad \Rightarrow \quad x_1^{(L)} = h(\mathbf{x})$.

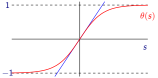
- Modifications: e.g., multiclass classification: $x_j^{(L)}$ log-probability of class j.

Neural Network: Nonlinearity

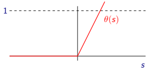
- sigmoid: $\theta(s) = \frac{e^s}{1+e^s}$.



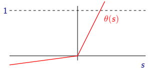
- tanh: $\tanh(s) = \frac{e^s - e^{-s}}{e^s + e^{-s}} = 2\theta(2s) - 1$.



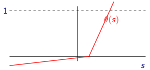
- ReLU: Rectified Linear Unit $\theta(s) = \max\{0, s\}$.



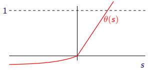
- Leaky ReLU: $\theta(s) = \max\{\alpha s, s\}$.



- Maxout $\theta(s) = \max\{\alpha_1 s + \beta_1, \alpha_2 s + \beta_2\}$.



- ELU $\theta(s) = \max\{\alpha(e^s - 1), s\}$.

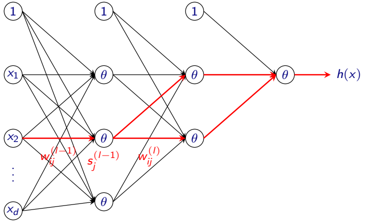


Neural Network: (stochastic) gradient descent for ERM

- Output $h(\mathbf{x})$ is determined by all the weights $\mathbf{w} = \{w_{i,j}^{(l)}\}$ (or weight matrices $W^{(l)}$)
- Error on output: $\sum_{k=1}^{n} \ell(h(\mathbf{x}_k), y_k)$
- Just consider on a single data point: $\ell(h(\mathbf{x}_k), y_k) = \ell_k(\mathbf{w})$
- Gradient: $\dfrac{\partial \ell_k(\mathbf{w})}{\partial w_{ij}^{(l)}}$ for all i, j, l.
- Easy trick:

$$s_j^{(l-1)} = \sum_{i=0}^{d^{(l-2)}} w_{ij}^{(l-1)} x_i^{(l-2)}$$

$$\frac{\partial \ell_k(\mathbf{w})}{w_{ij}^{(l-1)}} = \frac{\partial \ell_k}{\partial s_j^{(l-1)}} \cdot \frac{\partial s_j^{(l-1)}}{\partial w_{ij}^{(l-1)}}$$



Neural Network: Computing the gradients

We need:

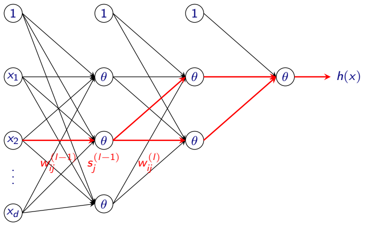
$$\frac{\partial \ell_k(\mathbf{w})}{w_{ij}^{(l-1)}} = \frac{\partial \ell_k}{\partial s_j^{(l-1)}} \cdot \frac{\partial s_j^{(l-1)}}{\partial w_{ij}^{(l-1)}}$$

Second term:

$$\frac{\partial s_j^{(l-1)}}{\partial w_{ij}^{(l-1)}} = \frac{\partial \left[\sum_{i=0}^{d^{(l-2)}} w_{ij}^{(l-1)} x_i^{(l-2)}\right]}{\partial w_{ij}^{(l-1)}} = x_i^{(l-2)}$$

How to compute

$$\delta_j^{(l-1)} = \frac{\partial \ell_k}{\partial s_j^{(l-1)}} ?$$



Neural Network: Backward computation

Final layer:

$$\delta_1^{(L)} = \frac{\partial \ell_k}{\partial s_1^{(L)}} = \frac{\partial \ell_k}{\partial x_1^{(L)}} \cdot \frac{\partial x_1^{(L)}}{\partial s_1^{(L)}}$$

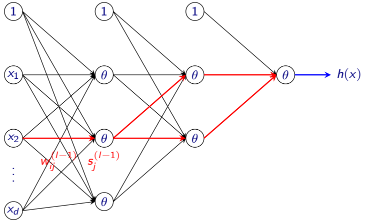
for

$$\ell_k = \frac{1}{2}(x_1^{(L)} - y_k)^2 = \frac{1}{2}(\theta(s_1^{(L)}) - y_k)^2$$

$$\frac{\partial \ell_k}{\partial x_1^{(L)}} = x_1^{(L)} - y_k$$

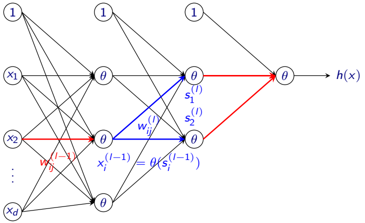
for $\theta(s) = \tanh(s)$,
$\frac{\partial x_1^{(L)}}{\partial s_1^{(L)}} = \theta'(s_1^{(L)}) = 1 - \theta^2(s_1^{(L)}) = 1 - (x_1^{(L)})^2$ Thus

$$\delta_1^{(L)} = \left(x_1^{(L)} - y_k\right)\left(1 - (x_1^{(L)})^2\right)$$



Neural Network: Backward computation

$$
\begin{aligned}
\delta_i^{(l-1)} &= \frac{\partial \ell_k}{\partial s_i^{(l-1)}} \\
&= \sum_{j=1}^{d^{(l)}} \frac{\partial \ell_k(\mathbf{w})}{\partial s_j^{(l)}} \cdot \frac{\partial s_j^{(l)}}{\partial x_i^{(l-1)}} \cdot \frac{\partial x_i^{(l-1)}}{\partial s_i^{(l-1)}} \\
&= \sum_{j=1}^{d^{(l)}} \delta_j^{(l)} \cdot w_{ij}^{(l)} \cdot \theta'(s_i^{(l-1)}) \\
&= \theta'(s_i^{(l-1)}) \sum_{j=1}^{d^{(l)}} \delta_j^{(l)} \cdot w_{ij}^{(l)} \\
&= (1 - (x_i^{(l-1)})^2) \sum_{j=1}^{d^{(l)}} \delta_j^{(l)} \cdot w_{ij}^{(l)}
\end{aligned}
$$



Neural Network: Backpropagation algorithm

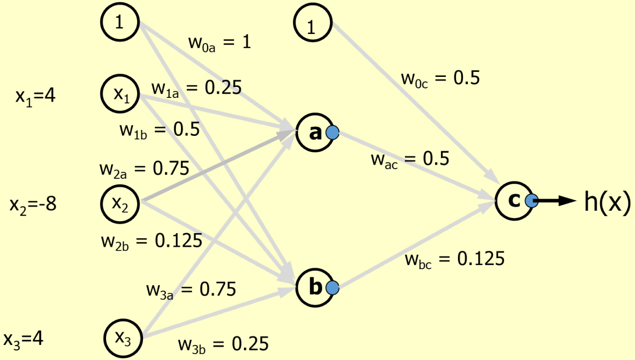
Backpropagation

- Intialize all weights $w_{ij}^{(l)}$ at random
- for $t = 1, 2, \ldots$ do
  - Pick a data point (x_k, y_k)
  - **Forward:** Compute all $x_j^{(l)}$
  - **Backward:** Compute all $\delta_j^{(l)}$
  - **Update:** $w_{ij}^{(l)} \leftarrow w_{ij}^{(l)} - \eta_t x_i^{(l-1)} \delta_j^{(l)}$
    - single point (SGD), minibatch, batch
- Return final weights $w_{ij}^{(l)}$

Regularization: $L_2, L_1, \ldots$, dropout (in each iteration randomly select weights which are not updated)

Example

Consider neural network architecture given in the Figure



- Propose a non-linearity activation function, draw this function and its gradient in a figure.

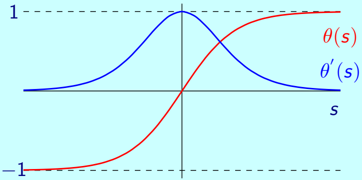
Example

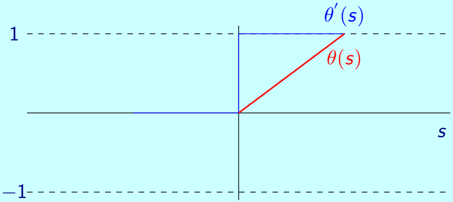
Solution:

$$\theta(s) = \tanh(s) = \frac{e^s - e^{-s}}{e^s + e^{-s}}, \qquad \frac{\partial \tanh(s)}{\partial s} = 1 - \tanh^2(s)$$

or

$$\theta(s) = \max\{0, s\}, \qquad \frac{\partial \max\{0, s\}}{\partial s} = 1$$





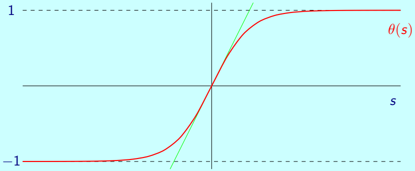
Example

- Discuss what happens if we use $\mathbf{x}$ as input, $\theta(s) = s$ as the activation function and W_l as a matrix of weights in layer l. Can $\theta(s) = s$ be achieved with $\tanh(s)$?

Example

Solution: The network is reduced to a simple linear predictor as all weights between layers can be directly multiplied i.e., product of matrices $\mathbf{w}_L^\top W_{L-1} W_{L-2} \dots W_1 \mathbf{x} = \mathbf{w}_*^\top \mathbf{x}$.

It can be achieved by forcing signal and weights to be small with regularization.



Example

- Given input vector $\mathbf{x} = (4, -8, 4)$ use the proposed non-linearity from question i) to calculate outputs of all neurons and $h(x)$.

Example

Solution:

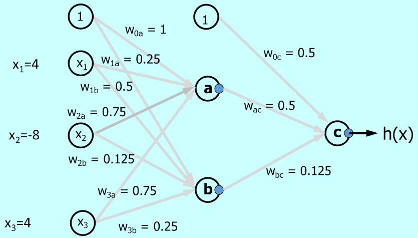
Outputs $x_j^{(l)} = \theta(s_j^{(l)}) = \theta\left(\sum_{i=0}^{d^{(l-1)}} w_{ij}^{(l)} x_i^{(l-1)}\right)$.

ReLU: $\max(0, s_j)$

$f_a = \max(0, 1 + 0.25 \cdot 4 + 0.75 \cdot (-8) + 0.75 \cdot 4) = 0$

$f_b = \max(0, 1 + 0.5 \cdot 4 + 0.125 \cdot (-8) + 0.25 \cdot 4) = 3$

$h(\mathbf{x}) = f_c = \max(0, 0.5 + 0.5 \cdot (0) + 0.125 \cdot 3) = 0.875$



Example

- Write down the main steps of the backpropagation algorithm.

  Solution: See the slide before the example.

- What is the role of the learning rate in gradient descent and what are the risks of setting the learning rate too large or too small?

Example

Solution:

The learning rate can affect the speed of convergence. Too small η will result in very long optimization process. Too large η can lead to oscillations near the minimum of the function or divergence.

Example

- Apply the backpropagation algorithm to calculate the update of weight $w_{3,b}^{(2)}$. Use L_2 loss without regularization, ReLU activation, learning rate of 0.1 and training example of $\mathbf{x} = (4, -8, 4)$ with its label $y = 2$.

Example

Solution: Using the outputs from forward propagation in a).

$\eta_t = 0.1$

Loss $\ell_2(f_c, y) = \frac{1}{2}(f_c - y)^2 = \frac{1}{2}(0.875 - 2)^2 = 0.63$

$\theta'(s) = \frac{\partial \max(0,\mathbf{c})}{\partial \mathbf{c}} = 1$

gradient $\frac{\partial \ell}{\partial f_c} = (f_c - y) = (0.875 - 2) = -1.125$

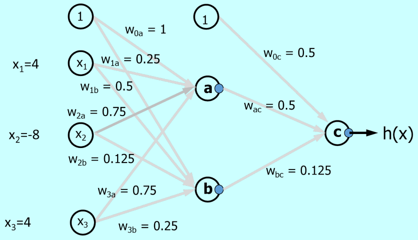
$\delta_c = \frac{\partial \ell}{\partial f_c} \cdot \frac{\partial f_c}{\partial \mathbf{c}} = \frac{\partial \ell}{\partial f_c} \cdot \frac{\partial \max(0,\mathbf{c})}{\partial \mathbf{c}} = \frac{\partial \ell}{\partial f_c}$

$\delta_c = (f_c - y) = (0.875 - 2) = -1.125$

$\delta_b = \delta_c w_{bc} = -1.125 \cdot 0.125 = -0.14$

weight updates $w_{3b} \leftarrow w_{3b} - \eta_t x_3 \delta_b$

weight updates $w_{3b} \leftarrow w_{3b} + 0.1 \cdot 4 \cdot 0.14$



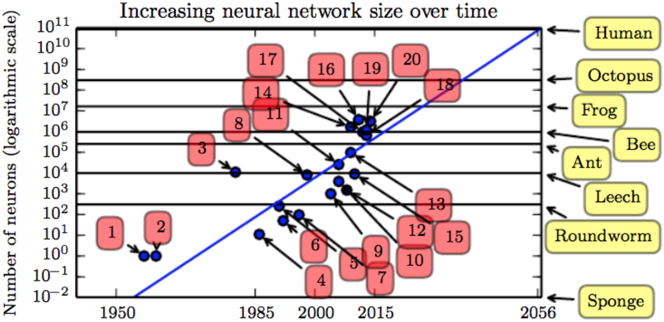
Neural Network: Remarks

- Features are learned!
- Hard to interpret
- VC-dimension is high–bad generalization properties
- Hard to optimize
- Universal approximator
  - adding an extra layer may reduce exponentially the number of nodes needed
- State-of-the art performance in many areas:
  - Convolutional neural networks: image processing
  - Recurrent NN: speech and natural language processing
- Lots of data helps
- Use various regularizers, different target functions, etc.

Why now?

1. Perceptron (Rosenblatt, 1958, 1962)
2. Adaptive linear element (Widrow and Hoff, 1960)
3. Neocognitron (Fukushima, 1980)
4. Early back-propagation network (Rumelhart et al., 1986b)
5. Recurrent neural network for speech recognition (Robinson and Fallside, 1991)
6. Multilayer perceptron for speech recognition (Bengio et al., 1991)
7. Mean field sigmoid belief network (Saul et al., 1996)
8. LeNet-5 (LeCun et al., 1998b)
9. Echo state network (Jaeger and Haas, 2004)
10. Deep belief network (Hinton et al., 2006)
11. GPU-accelerated convolutional network (Chellapilla et al., 2006)
12. Deep Boltzmann machine (Salakhutdinov and Hinton, 2009a)
13. GPU-accelerated deep belief network (Raina et al., 2009)
14. Unsupervised convolutional network (Jarrett et al., 2009)
15. GPU-accelerated multilayer perceptron (Ciresan et al., 2010)
16. OMP-1 network (Coates and Ng, 2011)
17. Distributed autoencoder (Le et al., 2012)
18. Multi-GPU convolutional network (Krizhevsky et al., 2012)
19. COTS HPC unsupervised convolutional network (Coates et al., 2013)
20. GoogLeNet (Szegedy et al., 2014a)

Since the introduction of hidden units, artificial neural networks have doubled in size roughly every 2.4 years.



Review

- Content
  - Part 1, KM: Components of learning, tasks, types of learning, ML problem formulation, simple predictors
  - Part 2, KM: Feasibility of learning, error function, Empirical Risk Minimization, generalisation bounds, performance vs complexity, bias/variance trade off, Hoeffding/VC inequalities
  - Part 3, KM: Feature transformations, noisy data, overfitting, regularisation, validation
  - Part 4, KM: Logistic regression, gradient descent, Perceptron, Multi Layer Perceptron, Neural Network, backpropagation
  - Part 5, DG: Hyperplane, separation with hard margin, soft margin, support vector machines,
  - Part 6, DG: Nearest neighbour classification, linear unsupervised learning, principle component analysis
  - Part 7, DG: K-means clustering, kernel K-means, advanced clustering algorithms