

ETC3555

Statistical Machine Learning

Neural networks

22 August 2018

Outline (subject to change)

Week Topic

- 1 Introduction/The learning problem
- 2 The learning problem
- 3 Linear models
- 4 Gradient descent
- 5 Neural Networks
- 6 Neural Networks and Backpropagation
- 7 Deep Neural Networks
- 8 Support vector machines
- 9 Recommender systems and matrix completion

Semester break

- 10 Text mining
- 11 Social networks
- 12 Project presentation

1 The neural network model

2 Backpropagation algorithm

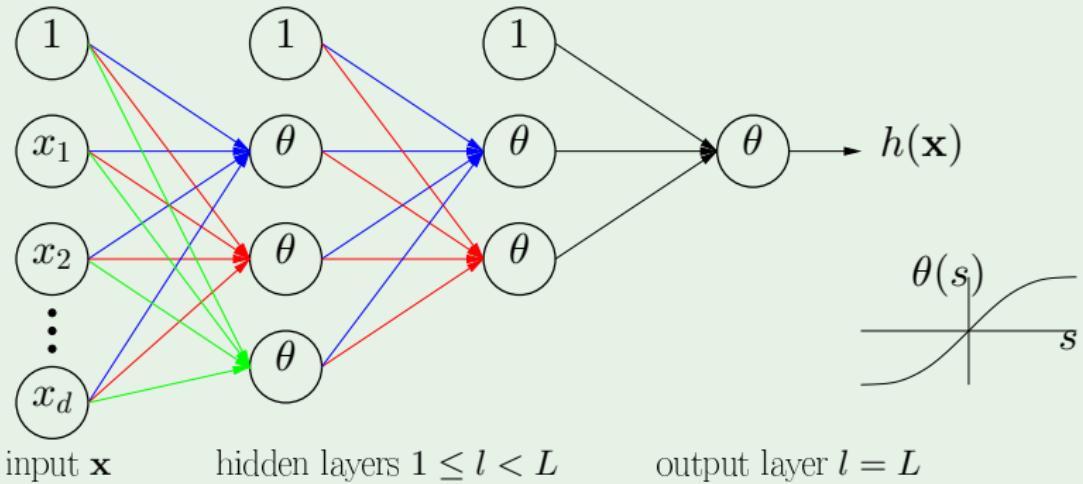
Outline

1 The neural network model

2 Backpropagation algorithm

The neural network

The neural network



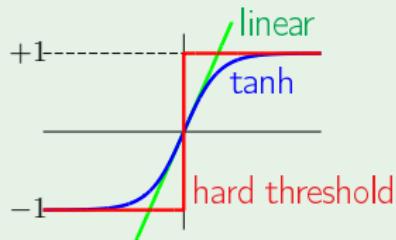
How the network operates

How the network operates

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

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

Apply \mathbf{x} to $x_1^{(0)} \cdots x_{d^{(0)}}^{(0)} \rightarrow \rightarrow x_1^{(L)} = h(\mathbf{x})$



$$\theta(s) = \tanh(s) = \frac{e^s - e^{-s}}{e^s + e^{-s}}$$

The activation function

- Every node with an input has a *transformation* or *activation* function, θ . In the previous example:
 $\theta(s) = \text{sign}(s)$.
- $\text{sign}(s)$ function can be approximated with $\tanh(s)$, the hyperbolic tangent function, a soft threshold. A smooth function → easier to optimize, e.g. gradient descent.
- For the nodes in the hidden layers, arbitrary activation functions can be used. They are often chosen to allow fast computation and efficient optimization.
- For the output node, a different sigmoid will be used depending on the learning problem
 - Classification: $\hat{y} = \theta(s) = \tanh(s)$ or $\text{logistic}(s)$
 - Regression: $\hat{y} = \theta(s) = s$

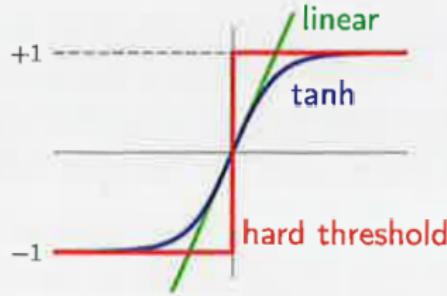
Tanh vs logistic

Exercise 3.5

Another popular soft threshold is the hyperbolic tangent

$$\tanh(s) = \frac{e^s - e^{-s}}{e^s + e^{-s}}.$$

- How is \tanh related to the logistic function θ ? [Hint: shift and scale]
- Show that $\tanh(s)$ converges to a hard threshold for large $|s|$, and converges to no threshold for small $|s|$. [Hint: Formalize the figure below.]



Tanh vs logistic

- (a)** $\tanh(s) = 2 \times \text{logistic}(2s) - 1$
- (b)** Use a taylor expansion around zero.

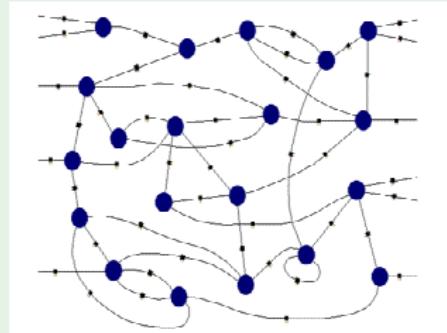
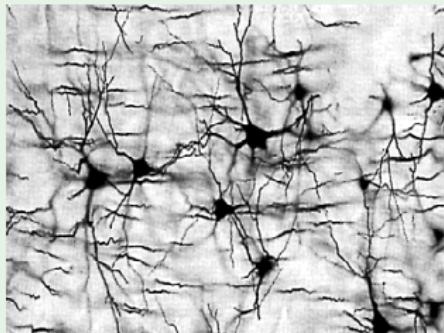
Biological inspiration

Biological inspiration

biological function



biological structure



Outline

1 The neural network model

2 Backpropagation algorithm

Applying SGD

Applying SGD

All the weights $\mathbf{w} = \{\mathbf{w}_{ij}^{(l)}\}$ determine $h(\mathbf{x})$

Error on example (\mathbf{x}_n, y_n) is

$$\mathbf{e}(h(\mathbf{x}_n), y_n) = \mathbf{e}(\mathbf{w})$$

To implement SGD, we need the gradient

$$\nabla \mathbf{e}(\mathbf{w}): \frac{\partial \mathbf{e}(\mathbf{w})}{\partial \mathbf{w}_{ij}^{(l)}} \text{ for all } i, j, l$$

Computing partial derivatives efficiently

Computing $\frac{\partial \mathbf{e}(\mathbf{w})}{\partial w_{ij}^{(l)}}$

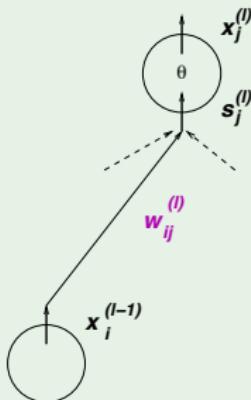
We can evaluate $\frac{\partial \mathbf{e}(\mathbf{w})}{\partial w_{ij}^{(l)}}$ one by one: analytically or numerically

A trick for efficient computation:

$$\frac{\partial \mathbf{e}(\mathbf{w})}{\partial w_{ij}^{(l)}} = \frac{\partial \mathbf{e}(\mathbf{w})}{\partial s_j^{(l)}} \times \frac{\partial s_j^{(l)}}{\partial w_{ij}^{(l)}}$$

We have $\frac{\partial s_j^{(l)}}{\partial w_{ij}^{(l)}} = x_i^{(l-1)}$

We only need: $\frac{\partial \mathbf{e}(\mathbf{w})}{\partial s_j^{(l)}} = \delta_j^{(l)}$



A $\delta_j^{(l)}$ will be associated to each $\frac{\partial \mathbf{e}(\mathbf{w})}{\partial s_j^{(l)}}$.

Computing partial derivatives efficiently

δ for the final layer

$$\delta_j^{(l)} = \frac{\partial e(\mathbf{w})}{\partial s_j^{(l)}}$$

For the final layer $l = L$ and $j = 1$:

$$\delta_1^{(L)} = \frac{\partial e(\mathbf{w})}{\partial s_1^{(L)}}$$

$$e(\mathbf{w}) = (x_1^{(L)} - y_n)^2$$

$$x_1^{(L)} = \theta(s_1^{(L)})$$

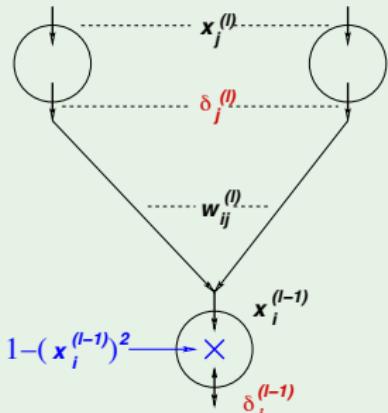
$$\theta'(s) = 1 - \theta^2(s) \quad \text{for the tanh}$$

$$e(\mathbf{w}) = e(h(\mathbf{x}_n), y_n) = e(x_1^{(L)}, y_n) = (x_1^{(L)} - y_n)^2$$

Computing partial derivatives efficiently

Back propagation of δ

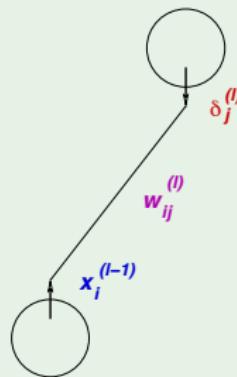
$$\begin{aligned}\delta_i^{(l-1)} &= \frac{\partial \mathbf{e}(\mathbf{w})}{\partial s_i^{(l-1)}} \\ &= \sum_{j=1}^{d^{(l)}} \frac{\partial \mathbf{e}(\mathbf{w})}{\partial s_j^{(l)}} \times \frac{\partial s_j^{(l)}}{\partial x_i^{(l-1)}} \times \frac{\partial x_i^{(l-1)}}{\partial s_i^{(l-1)}} \\ &=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



Backpropagation algorithm

Backpropagation algorithm

- 1: Initialize all weights $w_{ij}^{(l)}$ at random
- 2: **for** $t = 0, 1, 2, \dots$ **do**
- 3: Pick $n \in \{1, 2, \dots, N\}$
- 4: **Forward:** Compute all $x_j^{(l)}$
- 5: **Backward:** Compute all $\delta_j^{(l)}$
- 6: Update the weights: $w_{ij}^{(l)} \leftarrow w_{ij}^{(l)} - \eta x_i^{(l-1)} \delta_j^{(l)}$
- 7: Iterate to the next step until it is time to stop
- 8: Return the final weights $w_{ij}^{(l)}$

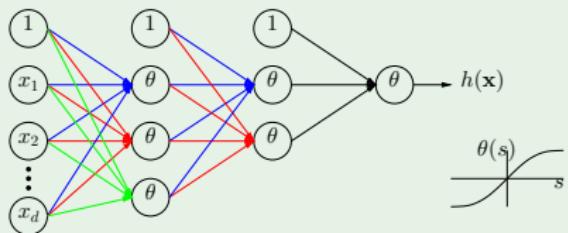


Automatic feature extraction

Final remark: hidden layers

learned nonlinear transform

interpretation?



Matrix notations

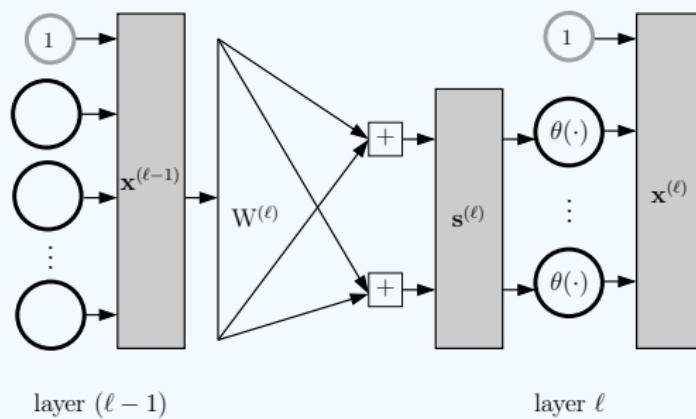
layer ℓ parameters

signals in outputs	$s^{(\ell)}$	$d^{(\ell)}$ dimensional input vector
weights in	$x^{(\ell)}$	$d^{(\ell)} + 1$ dimensional output vector
weights out	$W^{(\ell)}$	$(d^{(\ell-1)} + 1) \times d^{(\ell)}$ dimensional matrix
	$W^{(\ell+1)}$	$(d^{(\ell)} + 1) \times d^{(\ell+1)}$ dimensional matrix

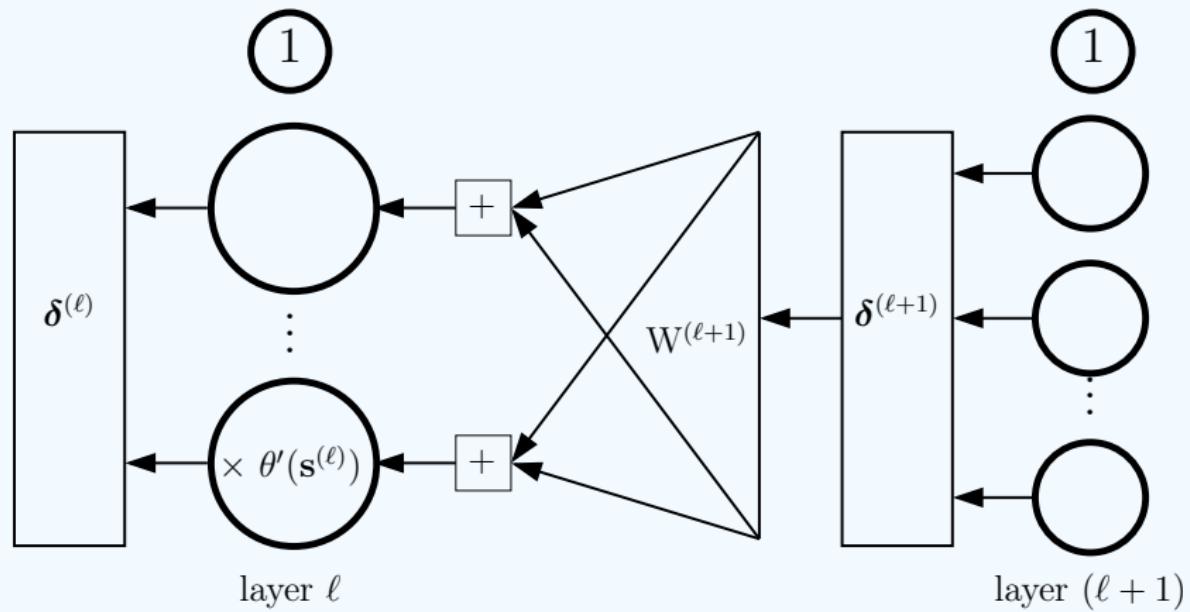
Forward propagation

Forward propagation to compute $h(\mathbf{x})$:

- 1: $\mathbf{x}^{(0)} \leftarrow \mathbf{x}$ [Initialization]
- 2: **for** $\ell = 1$ to L **do** [Forward Propagation]
- 3: $\mathbf{s}^{(\ell)} \leftarrow (\mathbf{W}^{(\ell)})^T \mathbf{x}^{(\ell-1)}$
- 4: $\mathbf{x}^{(\ell)} \leftarrow \begin{bmatrix} 1 \\ \theta(\mathbf{s}^{(\ell)}) \end{bmatrix}$
- 5: $h(\mathbf{x}) = \mathbf{x}^{(L)}$ [Output]



Backpropagation



Backpropagation

$$\frac{\partial e}{\partial W^{(\ell)}} = \mathbf{x}^{(\ell-1)} (\boldsymbol{\delta}^{(\ell)})^T$$

Backpropagation to compute sensitivities $\boldsymbol{\delta}^{(\ell)}$.

Input: a data point (\mathbf{x}, y) .

0: Run forward propagation on \mathbf{x} to compute and save:

$$\begin{aligned}\mathbf{s}^{(\ell)} &\quad \text{for } \ell = 1, \dots, L; \\ \mathbf{x}^{(\ell)} &\quad \text{for } \ell = 0, \dots, L.\end{aligned}$$

1: $\delta^{(L)} \leftarrow 2(x^{(L)} - y)\theta'(s^{(L)})$ [Initialization]

$$\theta'(s^{(L)}) = \begin{cases} 1 - (x^{(L)})^2 & \theta(s) = \tanh(s); \\ 1 & \theta(s) = s. \end{cases}$$

2: **for** $\ell = L - 1$ to 1 **do** [Back-Propagation]

3: Let $\theta'(\mathbf{s}^{(\ell)}) = [1 - \mathbf{x}^{(\ell)} \otimes \mathbf{x}^{(\ell)}]_1^{d^{(\ell)}}$.

4: Compute the sensitivity $\boldsymbol{\delta}^{(\ell)}$ from $\boldsymbol{\delta}^{(\ell+1)}$:

$$\boldsymbol{\delta}^{(\ell)} \leftarrow \theta'(\mathbf{s}^{(\ell)}) \otimes [\mathbf{W}^{(\ell+1)} \boldsymbol{\delta}^{(\ell+1)}]_1^{d^{(\ell)}}$$