LG Advanced Data Scientists Program
Deep Learning

[5: Artificial Neural Networks]

Prof. Sungroh Yoon

Electrical & Computer Engineering | Seoul National University

(last compiled at 21:30 on February 24, 2020)

Outline

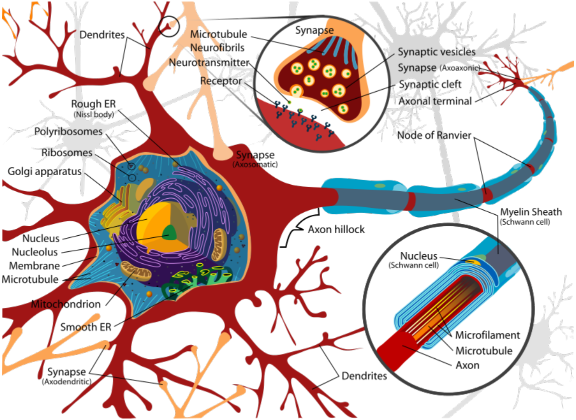
Readings

- *Deep Learning* by Goodfellow, Bengio, and Courville
  - Chapter 6: Deep Feedforward Networks
- *Neural Networks and Learning Machines* (3rd edition) by Haykin
  - Chapters 1–4
- *Pattern Recognition and Machine Learning* by Bishop
  - Chapter 5: Neural Networks
- *Elements of Statistical Learning* by Hastie, Tibshirani, and Friedman
  - Chapter 11: Neural Networks

Outline

Neuron

- electrically excitable cell that processes and transmits information through electrical and chemical signals



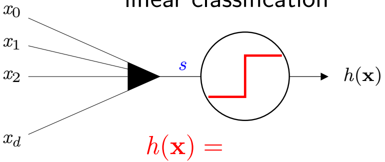
source: http://en.wikipedia.org/wiki/Neuron

Recall: linear models

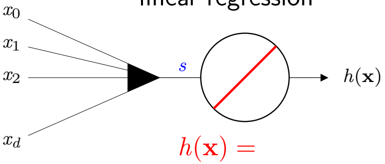
- based on "signal" s:

$$s = \sum_{i=0}^{d} w_i x_i$$

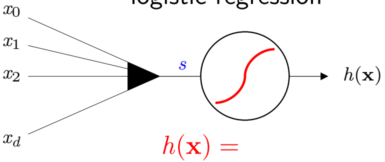
linear classification



linear regression



logistic regression



Artificial neural networks (ANNs)

- computational models inspired by brain
  - capable of machine learning and pattern recognition
  - popular until early 90's; popularity diminished in late 90's
  - renaissance: deep learning, AlphaGo, ...
- traditionally most studied model: feedforward neural network
  - comprises multiple layers of logistic regression models
  - also known as __________________ (MLP)

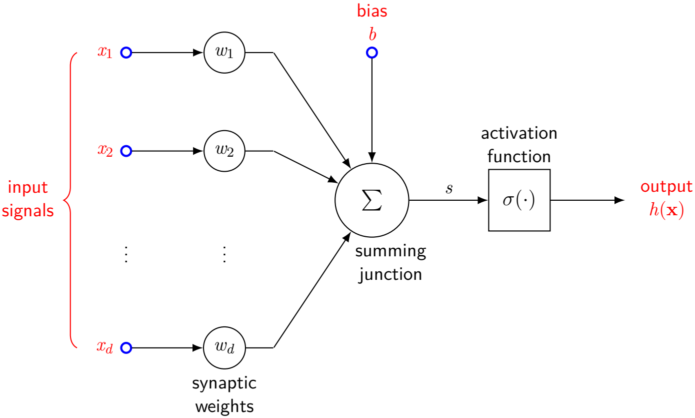
- multilayer perceptron: misnomer
  - network of multiple *logistic* models (continuous nonlinearity) rather than multiple *perceptrons* (discontinuous nonlinearity)
- central idea
  - extract linear combinations of inputs as derived features
  - then model the target as nonlinear function of these features

Outline

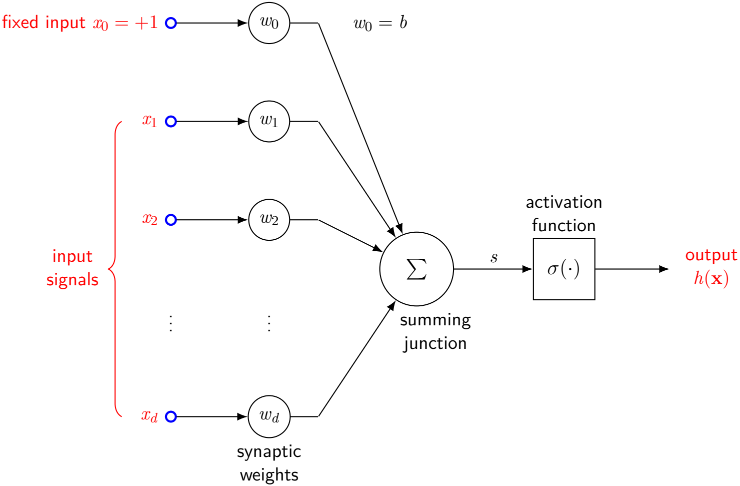
Models of a neuron

- three basic elements
    1. synapses (with weights)
    2. adder (input vector $\rightarrow$ scalar)
    3. ________ function (possibly nonlinear)

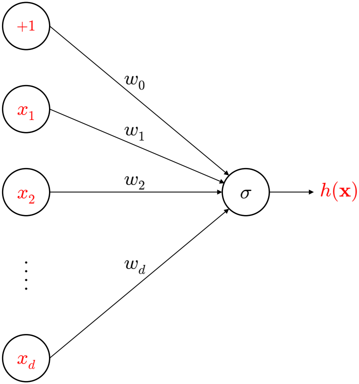
- model of a neuron:

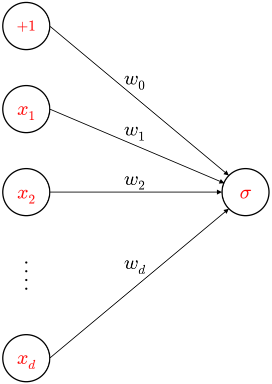


- alternative representation (w_0 for bias b):



- simplified representations:





Human neuron vs ANN neuron

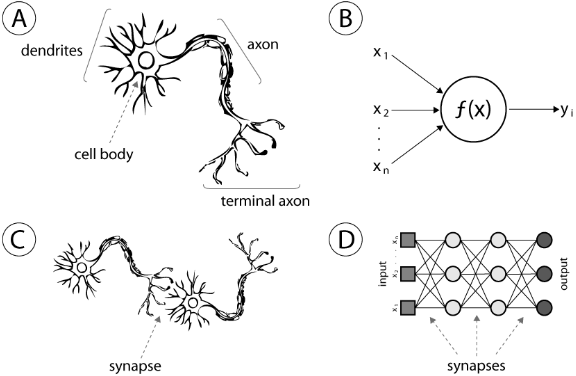


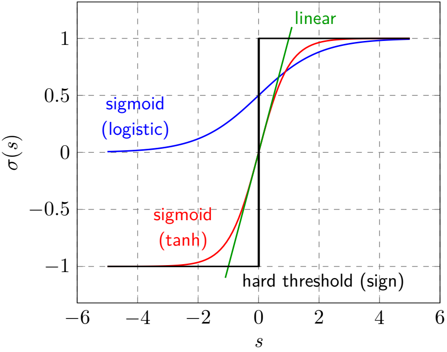
Figure 1: (a) human neuron (b) artificial neuron (c) biological synapse (d) ANN synapse

source: Maltarollo (2013)

Activation function

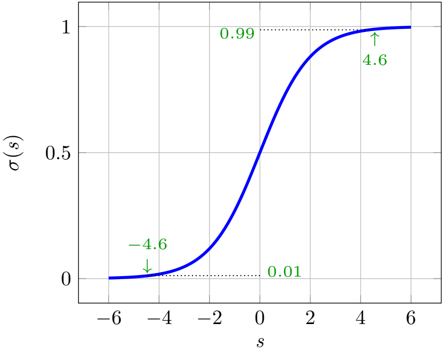
- function σ that defines output of neuron in terms of signal s

$$\sigma(s) = s \rightarrow \text{linear}$$
$$\text{sigmoid } \sigma(s) \rightarrow \underline{\hspace{3cm}} \text{ nonlinear}$$
$$\sigma(s) = \mathrm{sign}(s) \rightarrow \text{discontinuous nonlinear}$$



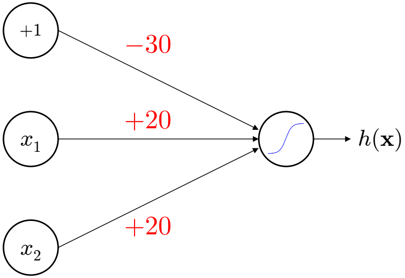
- simplifying assumption: all neurons use identical σ function
  - *e.g.* logistic sigmoid

$$\sigma(s) = \frac{1}{1 + e^{-s}}$$



Example 1

- let $x_1, x_2 \in \{0, 1\}$ and assume σ is logistic sigmoid
  - how to model $y = x_1 \wedge x_2$ (logical and) by neural net?

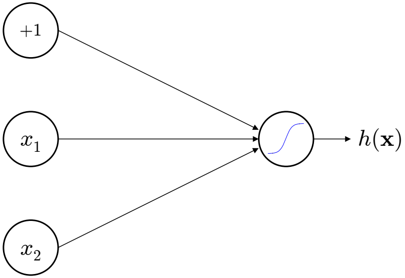


x_1	x_2	$h(\mathbf{x})$
0	0	$\sigma(-30) \approx$
0	1	$\sigma(-10) \approx$
1	0	$\sigma(-10) \approx$
1	1	$\sigma(10) \approx$

$$h(\mathbf{x}) = \sigma(-30 + 20x_1 + 20x_2)$$

Example 2

- let $x_1, x_2 \in \{0, 1\}$ and assume σ is logistic sigmoid
  - how to model $y = x_1 \vee x_2$ (logical or) by neural net?

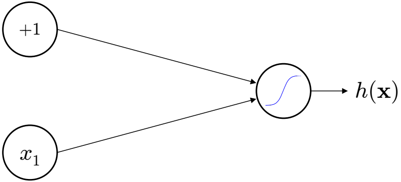


x_1	x_2	$h(\mathbf{x})$
0	0	$\sigma(-10) \approx 0$
0	1	$\sigma(10) \approx 1$
1	0	$\sigma(10) \approx 1$
1	1	$\sigma(30) \approx 1$

$$h(\mathbf{x}) = \sigma(-10 + 20x_1 + 20x_2)$$

Example 3

- let $x_1, x_2 \in \{0, 1\}$ and assume σ is logistic sigmoid
  - how to model $y = \neg x_1$ (logical not) by neural net?



x_1	$h(\mathbf{x})$
0	$\sigma(10) \approx 1$
1	$\sigma(-10) \approx 0$

$$h(\mathbf{x}) = \sigma(10 - 20x_1)$$

Rectifier activation

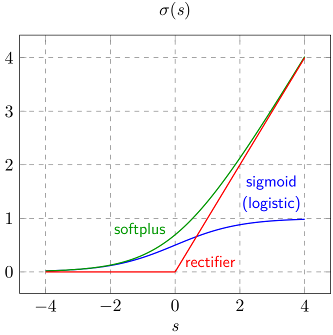
- rectifier: an activation function defined as

$$\sigma(s) = \max(0, s)$$

- softplus
  - a smooth approximation

$$\sigma(s) = \ln(1 + e^s)$$

$$\sigma'(s) = \frac{e^s}{1 + e^s} = \underbrace{\frac{1}{1 + e^{-s}}}_{\text{logistic function}}$$



- why rectifier?
  - arguably more ________ plausible than logistic sigmoid (cortical neurons: rarely in their maximum saturation regime)

ReLU (rectified linear unit)

- a unit employing the rectifier
  - very popular in deep neural nets
- ReLU advantages
  - allows _____ propagation of activations and gradients: only a small number of units have non-zero values
  - Nair and Hinton (2010): "unlike binary units, rectified linear units preserve information about relative intensities as information travels through multiple layers of feature detectors."
  - better for handling the ________ gradient problem: constant gradient propagation (more on this later)

Leaky ReLU

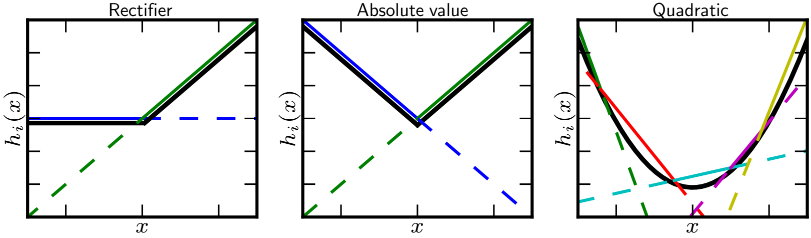
- limitation of ReLU
  - ReLU units can be fragile during training and can "die"
  - half of the operation regime is zero
- leaky ReLU: an attempt to fix the "dying ReLU" problem

$$\sigma(s) = \begin{cases} s & \text{if } s > 0 \\ \alpha s & \text{otherwise} \end{cases}$$

  - α: _____ constant (*e.g.* 0.01)

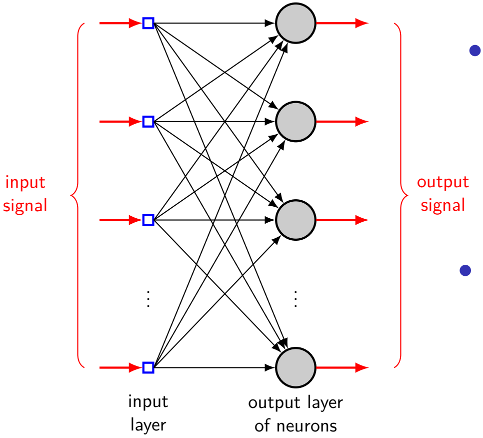
Maxout

- generalization of (leaky) ReLU (Goodfellow, 2013)
  - can be used as a universal approximator



Outline

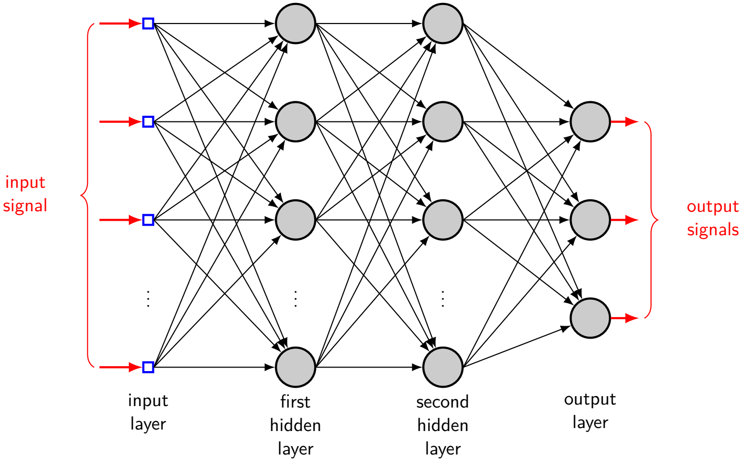
Single-layer feedforward network



- "layered"
  - input layer (of sources)
  - output layer (of neurons)
- "__________"
  - (function) signal direction: input $\rightarrow$ output
  - not vice versa

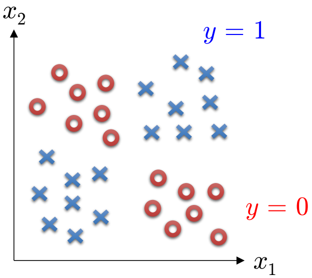
Multilayer feedforward network (multilayer perceptron)

- one or more ______ *layers* of neurons
  - not directly seen from either input or output

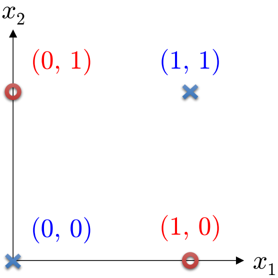


Example 4

- nonlinear classification by multilayer perceptron

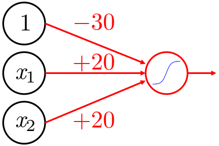


(a) nonlinear boundary needed

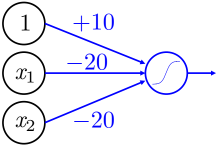


(b) simplified (XNOR)

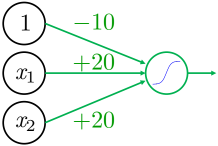
- $x_1 \wedge x_2$



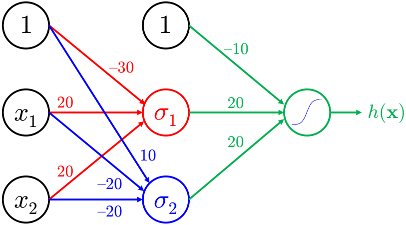
- $\neg x_1 \wedge \neg x_2$



- $x_1 \vee x_2$



- modeling $y = x_1 \ \texttt{XNOR} \ x_2$

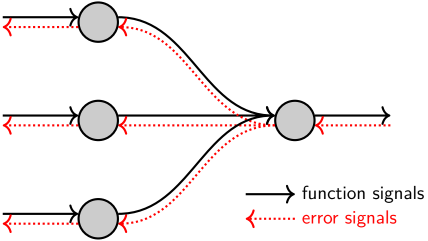


x_1	x_2	σ_1	σ_2	$h(\mathbf{x})$
0	0	0	1	
0	1	0	0	
1	0	0	0	
1	1	1	0	

Types of signals in neural networks

1. _______ signals: forward propagation
   - input signal comes in at input
   - propagates forward through network, and
   - emerges at output

2. ____ signals: back(ward) propagation
   - originates at an output neuron, and
   - propagates backward



Computations at hidden/output layer

- each hidden/output neuron performs two types of computations:

1. function signal appearing at neuron output
    - continuous nonlinear function of input and synaptic weights
2. estimate of gradient vector needed for back pass
    - gradients of error surface with respect to ______: $\nabla \mathcal{E}(\mathbf{w})$

Softmax function (aka normalized exponential)

- generalization of the logistic function
  - final layer of a network for multi-class classification

- definition: given a K-dimensional vector $\mathbf{h} = (h_1, h_2, \ldots, h_K)$
  - softmax function $\sigma : \mathbb{R}^K \mapsto \mathbb{R}^K$ s.t.

$$\sigma(\mathbf{h})_j = \frac{e^{h_j}}{\sum_{k=1}^{K} e^{h_k}} \qquad \text{for } j = 1, ..., K \tag{1}$$

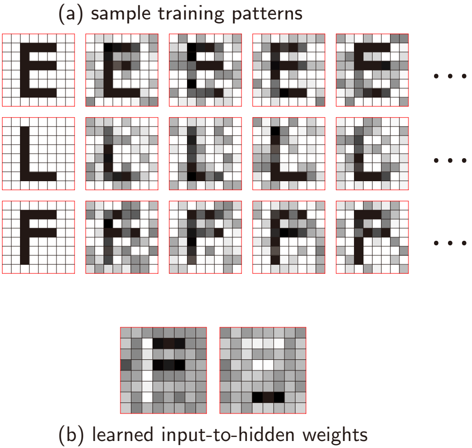
- components of vector $\sigma(\mathbf{h})$
  - sum to one and are all strictly between zero and one

$\Rightarrow$ represent a __________ probability distribution

Function of hidden neurons

- play critical role in operation of multilayer perceptron
  - each layer corresponds to "distributed representation"
- hidden neurons act as ______ detectors
  - as learning goes on, they gradually "discover" salient features characterizing training data
  - they do so by performing nonlinear transformation on input data into new space called *feature space*

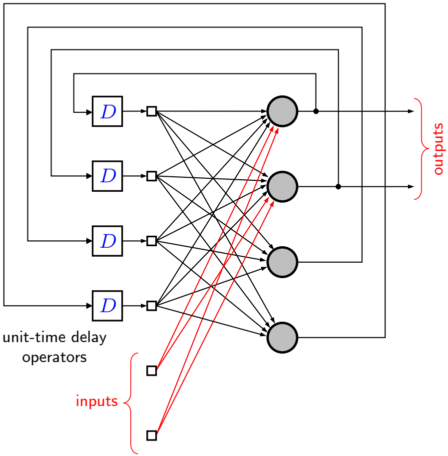
Example



source: Duda, Hart, and Stork (2001)

- 64-2-3 network for classifying 3 characters
  - 64-dim inputs
  - 2 hidden units
  - 3 output units
- learned i-to-h weights
  - describe feature groupings useful for classification

Recurrent neural networks (RNN)



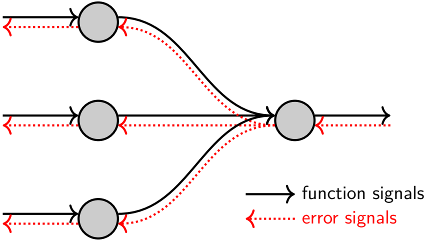
- have feedback loop(s)
  - fully recurrent
  - long short-term memory (LSTM) nets
  - gated recurrent unit (GRU) nets

Outline

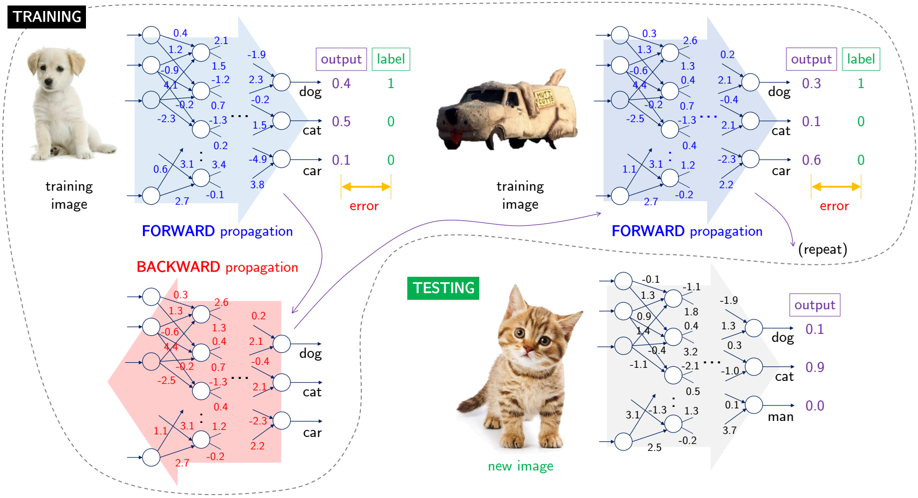
Recall: types of signals in neural networks

1. _______ signals: forward propagation
   - input signal comes in at input
   - propagates forward through network, and
   - emerges at output

2. ____ signals: back(ward) propagation
   - originates at an output neuron, and
   - propagates backward

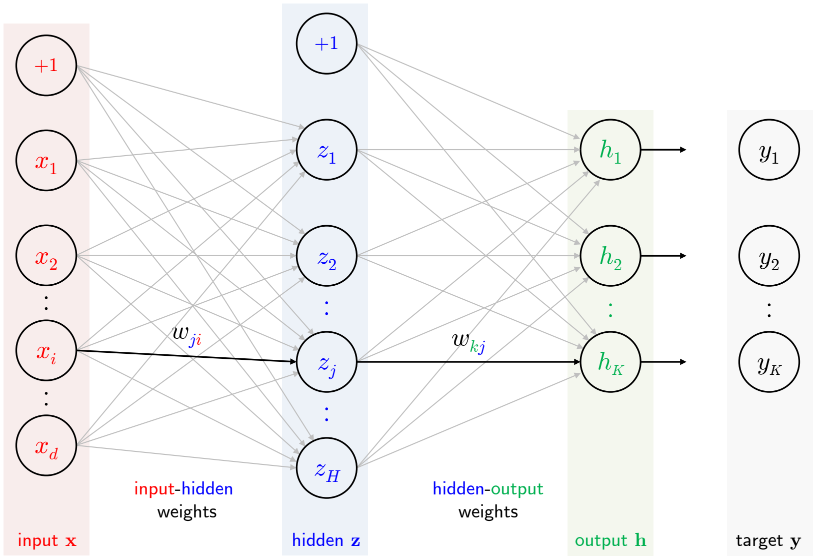


Training by forward/backward propagation



Assumptions and notations

- feedforward neural network with single hidden-layer
  - for the sake of simplicity in explanation
  - but there is no loss of generality
- d-H-K neural network
  - d-dimensional input
  - H units in hidden layer
  - K output signals



+1
x_1
x_2
x_i
x_d
+1
z_1
z_2
z_j
z_H
h_1
h_2
h_K
y_1
y_2
y_K
w_{ji}
w_{kj}
input-hidden weights
hidden-output weights
input $\mathbf{x}$
hidden $\mathbf{z}$
output $\mathbf{h}$
target $\mathbf{y}$

- notations
  - input: $\mathbf{x} = (x_1, \ldots, x_i, \ldots, x_d) \in \mathbb{R}^d$
  - hidden: $\mathbf{z} = (z_1, \ldots, z_j, \ldots, z_H) \in \mathbb{R}^H$
  - output: $\mathbf{h} = (h_1, \ldots, h_k, \ldots, h_K) \in \mathbb{R}^K$
- synaptic weights: $\mathbf{w} = \{w_{ji}, w_{kj}\}$
  - input-hidden weights: $w_{ji} \quad (1 \leq i \leq d, 1 \leq j \leq H)$
  - hidden-output weights: $w_{kj} \quad (1 \leq j \leq H, 1 \leq k \leq K)$

Activations of hidden/output units

- hidden unit:

$$z_j(\mathbf{x}) = \sigma\left(\sum_{i=0}^{d} w_{ji} x_i\right)$$

- output unit:

$$\begin{aligned} h_k(\mathbf{x}) &= \sigma\left(\sum_{j=0}^{H} w_{kj} z_j\right) \\ &= \sigma\left(\sum_{j=0}^{H} w_{kj}\ \sigma\left(\sum_{i=0}^{d} w_{ji} x_i\right)\right) \end{aligned}$$

Error measures

- e_k: error signal on k-th output

$$\underbrace{e_k}_{\text{error}} \triangleq \underbrace{h_k}_{\text{NN output}} - \underbrace{y_k}_{\text{correct label}}$$

- $\mathcal{E}_n$: error energy on example $(\mathbf{x}_n, y_n)$ $\leftarrow$ sum of squared errors

$$\mathcal{E}_n = \frac{1}{2}\sum_{k=1}^{K} e_{k,n}^2$$

- $\mathcal{E}_\mathcal{D}$: mean-squared error on data $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$

$$\mathcal{E}_\mathcal{D} = \frac{1}{N}\sum_{n=1}^{N}\mathcal{E}_n = \frac{1}{2N}\sum_{n=1}^{N}\sum_{k=1}^{K} e_{k,n}^2$$

Dependence of per-sample error $\mathcal{E}_n(\mathbf{w})$

1. error $\mathcal{E}_n$ depends on ______ n

$$\mathcal{E}_n = \frac{1}{2}\sum_{k=1}^{K} e_{k,n}^2 = \frac{1}{2}\sum_{k=1}^{K}(h_{k,n} - y_{k,n})^2 = \frac{1}{2}||\mathbf{h}_n - \mathbf{y}_n||^2 \quad (2)$$

$$= \frac{1}{2}\sum_{k=1}^{K} e_k^2 = \frac{1}{2}\sum_{k=1}^{K}(h_k - y_k)^2 = \frac{1}{2}||\mathbf{h} - \mathbf{y}||^2 \quad (3)$$

- (2) $\to$ (3): for the sake of simplicity, we sometimes omit n

2. error $\mathcal{E}_n$ depends on **all** weights $\mathbf{w} = \{w_{ji}, w_{kj}\}$

$$\begin{aligned}
\mathcal{E}_n = \mathcal{E}_n(\mathbf{w}) &= \frac{1}{2}\sum_{k=1}^{K}(h_k - y_k)^2 \\
&= \frac{1}{2}\sum_{k=1}^{K}\left(\sigma\left(\sum_{j=0}^{H} w_{kj}\ \sigma\left(\sum_{i=0}^{d} w_{ji} x_i\right)\right) - y_k\right)^2
\end{aligned}$$

$\Rightarrow$ lots of __________

Learning objective

- find $\mathbf{w}$ that minimizes training error $\mathcal{E}_{\mathcal{D}}$

$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}} \frac{1}{N} \sum_{n=1}^{N} \mathcal{E}_n(\mathbf{w})$$

- training neural network: non-trivial
  - neural network model is over-____________
  - optimization is non-convex

Selecting optimization method for neural networks

- neural network model typically has many parameters
  - $\Rightarrow$ often need very large training data sets
    - second-order techniques (*e.g.* Newton's method) are usually fast but not attractive here ($\because$ Hessian of $\mathcal{E}$ can be very large)
  - $\Rightarrow$ first-order online methods (*e.g.* stochastic gradient descent) are commonly used

- iterative optimization: $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla \mathcal{E}(\mathbf{w})$

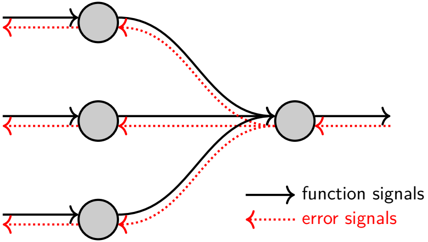
- key component: $\boxed{\text{evaluating}\qquad\qquad}$
  - can be used directly for sequential (online) optimization
  - can be accumulated over $\mathcal{D}$ for batch optimization

Outline

Back-propagation algorithm

- aka ________
  - popular method to evaluate $\nabla \mathcal{E}(\mathbf{w})$ for multilayer perceptrons
  - means 'back propagation of errors'
- popular method for training multilayer perceptrons
  - landmark in neural networks (mid-1980s): "*computationally efficient* method for training multilayer perceptrons"
- training proceeds in two phases

1. forward phase
   - apply input $\mathbf{x}$ and forward propagate through network
   - find activations of all hidden and output units
2. backward phase
   - successive adjustments to synaptic weights
   - easy for output layer; challenging for ______ layers



Credit assignment problem

- for output layer
  - explicit '______' (correct output y_k) exists for h_k
  - error signal can be evaluated directly: $e_k = h_k - y_k$
  - it is straightforward to find how output (thus error) depends on hidden-to-output weights

$\Rightarrow$ hidden-to-output 'sensitivity' $\dfrac{\partial \mathcal{E}}{\partial w_{kj}}$: easy to evaluate

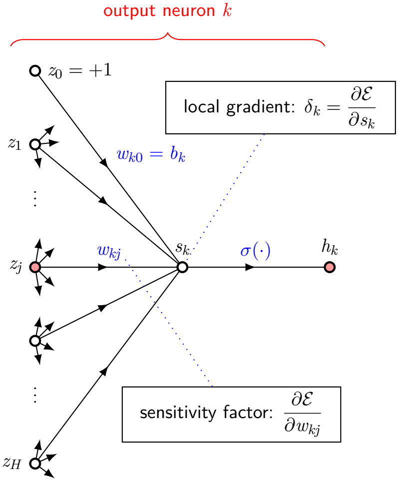
- for hidden layer
  - no explicit teacher to tell what hidden unit's output should be
  - this is called *credit assignment* problem

  $\Rightarrow$ input-to-hidden sensitivity $\dfrac{\partial \mathcal{E}}{\partial w_{ji}}$: difficult to evaluate

- power of back-propagation
  - it allows us to calculate effective error for each hidden unit
  - thus we can derive learning rule for ______________ weights

Outline

Sensitivity factor and delta error (output layer)



- sensitivity factor

$$\frac{\partial \mathcal{E}}{\partial w_{kj}} \tag{4}$$

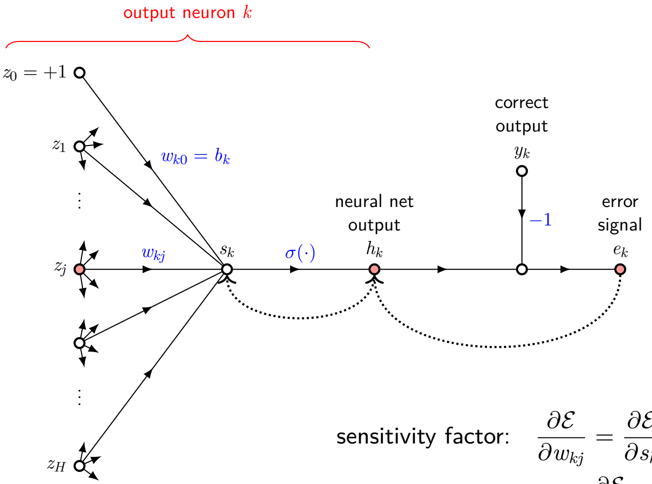
determines direction of search in weight space for weight w_{kj}

- "delta error"

$$\delta_k = \frac{\partial \mathcal{E}}{\partial s_k} \tag{5}$$

describes how $\mathcal{E}$ changes with unit's activation s_k

Output layer: explicit error signal $e_k = h_k - y_k$



sensitivity factor: $$\frac{\partial \mathcal{E}}{\partial w_{kj}} = \frac{\partial \mathcal{E}}{\partial s_k} \frac{\partial s_k}{\partial w_{kj}}$$

delta error: $$\delta_k = \frac{\partial \mathcal{E}}{\partial s_k} = \frac{\partial \mathcal{E}}{\partial e_k} \frac{\partial e_k}{\partial h_k} \frac{\partial h_k}{\partial s_k}$$

- sensitivity factor

$$\frac{\partial \mathcal{E}}{\partial w_{kj}} = \frac{\partial \mathcal{E}}{\partial s_k}\frac{\partial s_k}{\partial w_{kj}} = \frac{\partial \mathcal{E}}{\partial s_k} \cdot \frac{\partial}{\partial w_{kj}}\left(\sum_{j=1}^{H} w_{kj} z_j\right) = \delta_k \cdot z_j$$

- delta error δ_k

$$\begin{aligned}
\delta_k \triangleq \frac{\partial \mathcal{E}}{\partial s_k} &= \frac{\partial \mathcal{E}}{\partial e_k}\frac{\partial e_k}{\partial h_k}\frac{\partial h_k}{\partial s_k} \\
&= \underbrace{\frac{\partial}{\partial e_k}\left(\frac{1}{2}\sum_{k=1}^{K} e_k^2\right)}_{=e_k} \cdot \underbrace{\frac{\partial}{\partial h_k}\left(h_k - y_k\right)}_{=1} \cdot \underbrace{\frac{\partial}{\partial s_k}\left(\sigma(s_k)\right)}_{=\sigma'(s_k)} \\
&= e_k \cdot \sigma'(s_k)
\end{aligned}$$

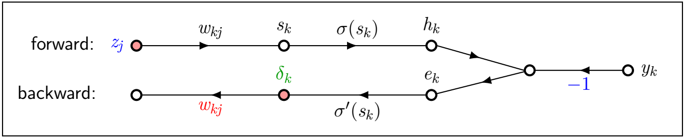
- weight update rule: $w_{kj} \leftarrow w_{kj} + \Delta w_{kj}$

$$\Delta w_{kj} = -\eta \frac{\partial \mathcal{E}}{\partial w_{kj}} = \boxed{}$$

Remarks

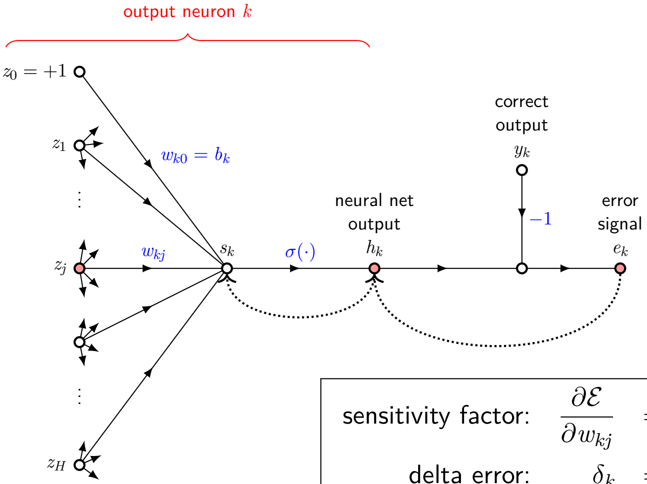
- we have derived 'delta rule' for multilayer perceptron:

$$\underbrace{\qquad\qquad}_{\text{weight correction}} = -\underbrace{\eta}_{\text{learning rate}} \times \underbrace{\qquad}_{\substack{\text{delta error}\\ \text{at } s_k}} \times \underbrace{\qquad}_{\substack{\text{input signal}\\ \text{to neuron } k}}$$



- Widrow-Hoff learning rule (aka LMS rule)
  - special case of back-propagation algorithm
  - uses identity activation func: $\sigma(s) = s \Rightarrow \delta_k = e_k\sigma'(s_k) = e_k$
  - weight update: $\Delta w_{kj} = -\eta \cdot e_k \cdot z_j$

Summary: output layer



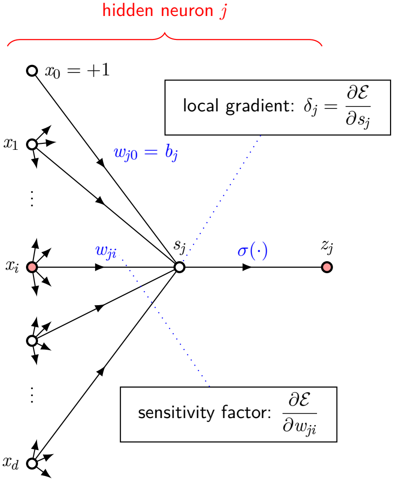
$$\text{sensitivity factor:} \quad \frac{\partial \mathcal{E}}{\partial w_{kj}} = \delta_k \cdot z_j$$

$$\text{delta error:} \quad \delta_k = e_k \cdot \sigma'(s_k)$$

$$\text{learning rule:} \quad \Delta w_{kj} = -\eta \cdot \delta_k \cdot z_j$$

Outline

Sensitivity factor and delta error (hidden layer)



- sensitivity factor

$$\frac{\partial \mathcal{E}}{\partial w_{ji}} \tag{6}$$

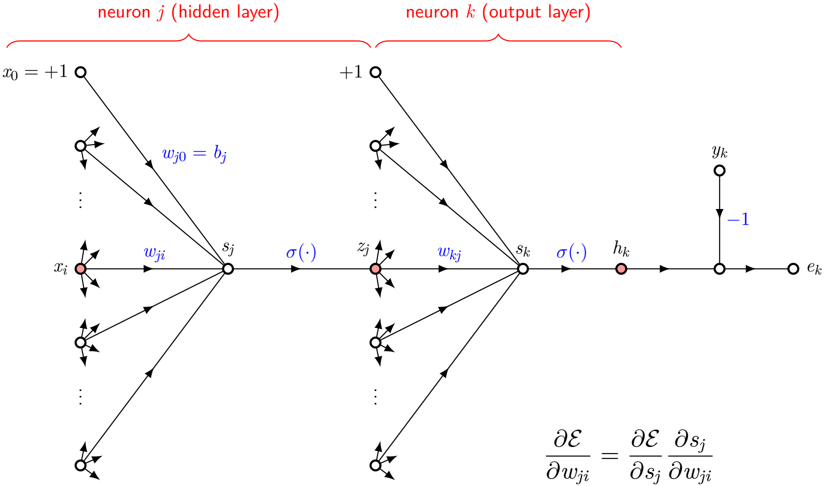
determines direction of search in weight space for weight w_{ji}

- delta error

$$\delta_j = \frac{\partial \mathcal{E}}{\partial s_j} \tag{7}$$

describes how $\mathcal{E}$ changes with unit's activation s_j

Hidden layer: no explicit error signal



$$\frac{\partial \mathcal{E}}{\partial w_{ji}} = \frac{\partial \mathcal{E}}{\partial s_j} \frac{\partial s_j}{\partial w_{ji}}$$

$$\delta_j = \frac{\partial \mathcal{E}}{\partial s_j} = \frac{\partial \mathcal{E}}{\partial z_j} \frac{\partial z_j}{\partial s_j}$$

- delta error:

$$\delta_j \triangleq \frac{\partial \mathcal{E}}{\partial s_j} = \frac{\partial \mathcal{E}}{\partial z_j}\frac{\partial z_j}{\partial s_j} = \frac{\partial}{\partial z_j}\left(\frac{1}{2}\sum_{k=1}^{K} e_k^2\right) \cdot \frac{\partial}{\partial s_j}\left(\sigma(s_j)\right)$$

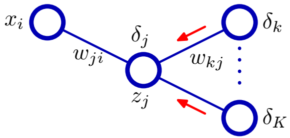
$$= \left(\sum_{k=1}^{K} e_k \frac{\partial e_k}{\partial z_j}\right) \cdot \sigma'(s_j)$$

$$= \sigma'(s_j)\sum_{k=1}^{K} e_k \frac{\partial e_k}{\partial s_k}\frac{\partial s_k}{\partial z_j}$$

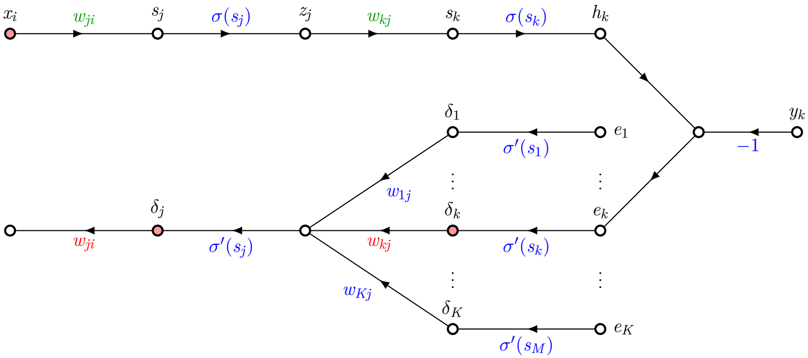
$$= \sigma'(s_j)\sum_{k=1}^{K} e_k \frac{\partial(\sigma(s_k) - y_k)}{\partial s_k}\frac{\partial}{\partial z_j}\left(\sum_{j=0}^{H} w_{kj} z_j\right)$$

$$= \sigma'(s_j)\sum_{k=1}^{K} \underbrace{e_k \sigma'(s_k)}_{=\delta_k} w_{kj}$$

$$= \sigma'(s_j)\sum_{k=1}^{K} \delta_k w_{kj}$$



- ‘back propagation of errors’



$$\boxed{\delta_j =}$$

- sensitivity factor

$$\frac{\partial \mathcal{E}}{\partial w_{ji}} = \frac{\partial \mathcal{E}}{\partial s_j}\frac{\partial s_j}{\partial w_{ji}} = \frac{\partial \mathcal{E}}{\partial s_j} \cdot \frac{\partial}{\partial w_{ji}}\left(\sum_{i=1}^{d} w_{ji}x_i\right) = \delta_j \cdot x_i$$

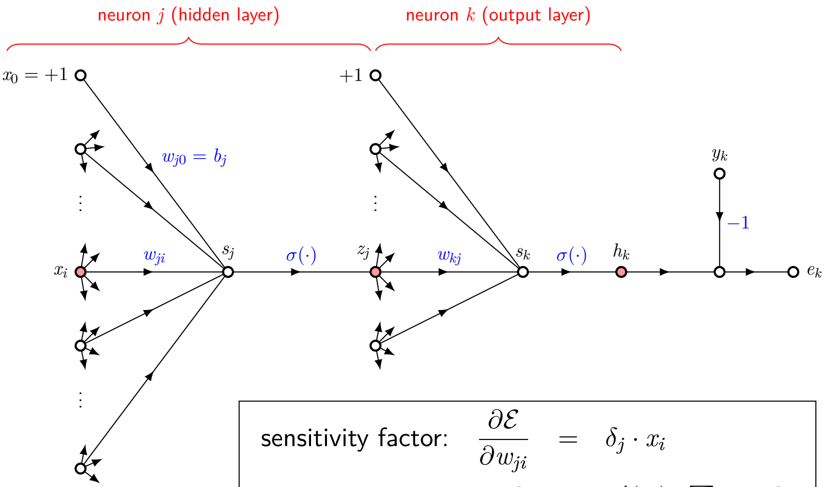
- delta error δ_j

$$\delta_j = \sigma'(s_j)\sum_{k=1}^{K} \delta_k w_{kj}$$

- weight update rule: $w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$

$$\Delta w_{ji} = -\eta \frac{\partial \mathcal{E}}{\partial w_{ji}} = \boxed{}$$

Summary: hidden layer



$$\text{sensitivity factor:} \quad \frac{\partial \mathcal{E}}{\partial w_{ji}} = \delta_j \cdot x_i$$

$$\text{delta error:} \quad \delta_j = \sigma'(s_j) \cdot \sum_k w_{kj}\delta_k$$

$$\text{learning rule:} \quad \Delta w_{ji} = -\eta \cdot \delta_j \cdot x_i$$

Back-propagation algorithm

stochastic back propagation

1: initialize all weights $\{w_{ji}, w_{kj}\}$
2: **repeat**
3: pick $n \in \{1, 2, \ldots, N\}$
4: *forward:* compute all
5: *backward:* compute all
6: update weights:

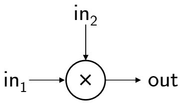
$$\begin{array}{rcccccc} \text{hidden}(j)\text{-to-output}(k): & w_{kj} & \leftarrow & w_{kj} & - & \eta \cdot \delta_k \cdot z_j \\ \text{input}(i)\text{-to-hidden}(j): & w_{ji} & \leftarrow & w_{ji} & - & \eta \cdot \delta_j \cdot x_i \end{array}$$

7: **until** it is time to stop
8: return final weights $\{w_{ji}^*, w_{kj}^*\}$

Multiplication

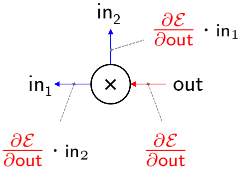
- $\text{out} = \text{in}_1 \cdot \text{in}_2$

forward



$$\frac{\partial \mathcal{E}}{\partial \text{in}_1} = \frac{\partial \mathcal{E}}{\partial \text{out}} \cdot \frac{\partial \text{out}}{\partial \text{in}_1} = \underbrace{\frac{\partial \mathcal{E}}{\partial \text{out}}}_{\substack{\text{output} \\ \text{gradient}}} \cdot \underbrace{\text{in}_2}_{\substack{\text{local} \\ \text{gradient}}}$$

backprop

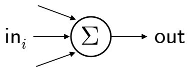


$$\frac{\partial \mathcal{E}}{\partial \text{in}_2} = \frac{\partial \mathcal{E}}{\partial \text{out}} \cdot \frac{\partial \text{out}}{\partial \text{in}_2} = \frac{\partial \mathcal{E}}{\partial \text{out}} \cdot \text{in}_1$$

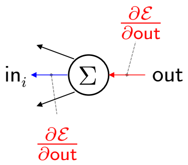
Summation

- $\text{out} = \sum_i \text{in}_i$

forward



backprop



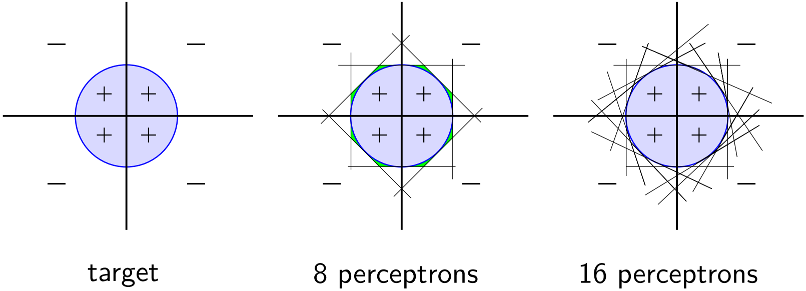
$$
\begin{aligned}
\frac{\partial \mathcal{E}}{\partial \text{in}_i} &= \frac{\partial \mathcal{E}}{\partial \text{out}} \cdot \frac{\partial \text{out}}{\partial \text{in}_i} \\
&= \underbrace{\frac{\partial \mathcal{E}}{\partial \text{out}}}_{\text{output gradient}} \cdot \underbrace{1}_{\text{local gradient}} \\
&= \frac{\partial \mathcal{E}}{\partial \text{out}}
\end{aligned}
$$

- sum (forward) $\Leftrightarrow$ fanout (backprop)

Outline

Neural networks: powerful model

- universal approximators
  - GMM: universal *density* estimator (given enough Gaussians)
  - ANN: universal _______ estimator (given enough neurons)

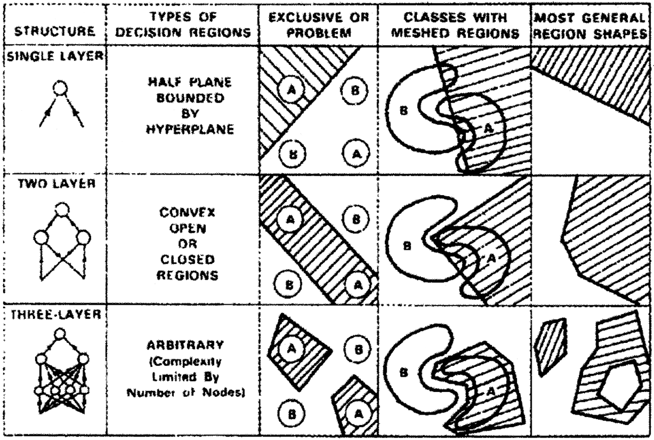


- types of decision regions that can be formed
  - more layers → more "intelligent"

hidden layer

hidden layer

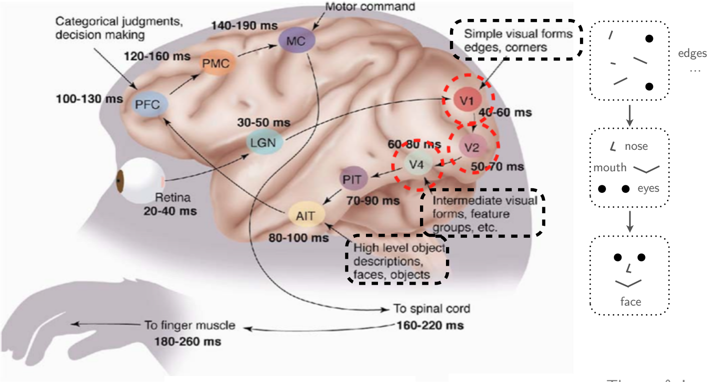
hidden layers



source: Lippmann (1987)

Inspiration from visual cortex

- human brain: at least 5–10 layers for visual processing
  - hierarchical model needed for human-level intelligence



source: Thorpe & Larochelle

Theoretical justification of deep learning

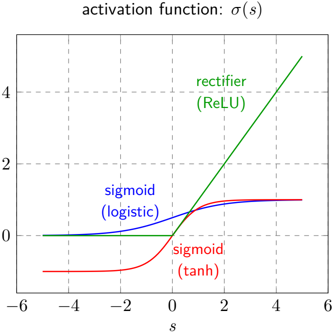
- DL can represent certain functions (exponentially) more compactly
- example: Boolean functions
  - a sort of feed-forward network (hidden units are logic gates)
  - any Boolean function can be represented by a two-layer circuit with an exponential number of hidden units
  - the same function can be represented by a ________ number of hidden units if we can adapt the number of layers

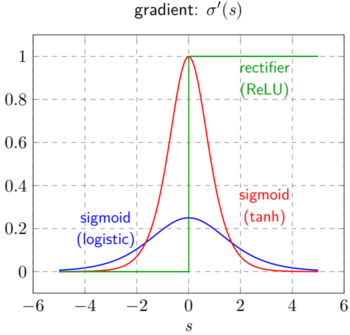
Huston, we have a problem

- training becomes significantly harder with more layers
- ANN history in a nutshell
  - breakthrough #1: backpropagation
  - doomed again: *e.g.* vanishing gradients, overfitting, runtime
  - breakthrough #2: ___________ pretraining + fine-tuning
- to understand more of the story
  - let's first study MLP and backpropagation
  - and then learn recent advances in deep architectures later

The vanishing gradient problem

- errors ‘vanish’ with backpropagation
    - true for not only feed-forward but also recurrent neural nets





- comparison
  - logistic: suffers from the vanishing gradient problem
  - tanh: better than logistic but the problem exists
  - rectifier (ReLU): gradients do not vanish

Outline

Summary

- artificial neural network: universal approximator of functions
  - neuron model: synapse (weights), adder, activation functions
  - conventional: feed-forward multilayer perceptrons (w/ backprop)
  - recent: RBM, AE, CNN, RNN, GAN and many other variants

- hidden layers act as feature detectors
  - # hidden layers $\uparrow \Longrightarrow$ intelligence $\uparrow$ but training difficulty $\uparrow$
  - practical solutions exist for effective training of deep neural networks

- two types of signals for training feed-forward multilayer perceptrons
  1. function signals: forward propagation
  2. error signals: backward propagation