Bil 470 / YAP 470

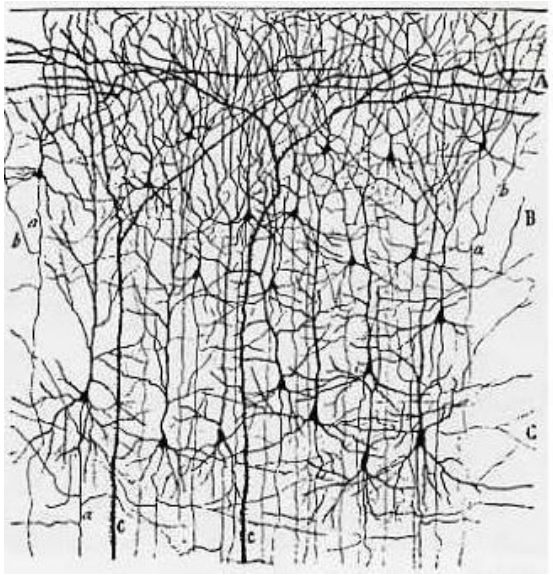
Introduction to Machine Learning (Yapay Öğrenme)

Batuhan Bardak

Neural Networks

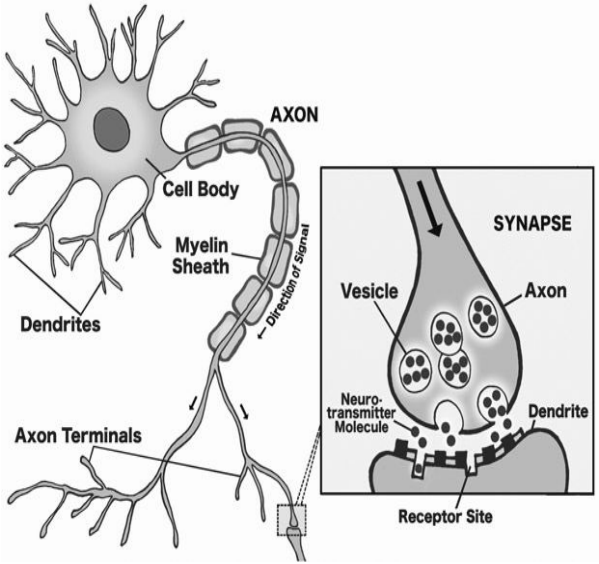
Date: 10.10.2022

Discovery of Neurons



Ramón y Cajal, 1900

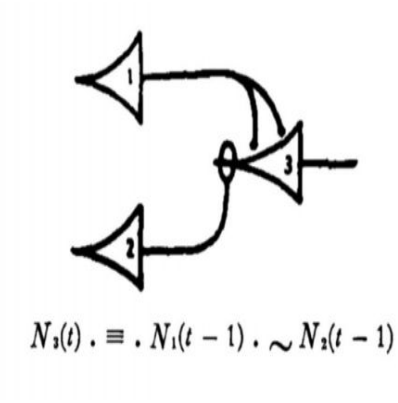
How the Human Brain Learns



Inputs arrive at dendrites, and axon serve as output channel

A Mathematical Model of Brain: McCulloch & Pitts Neuron

Artificial Neurons



MCP, 1943

BULLETIN OF
MATHEMATICAL BIOPHYSICS
VOLUME 5, 1943

A LOGICAL CALCULUS OF THE IDEAS IMMANENT IN NERVOUS ACTIVITY

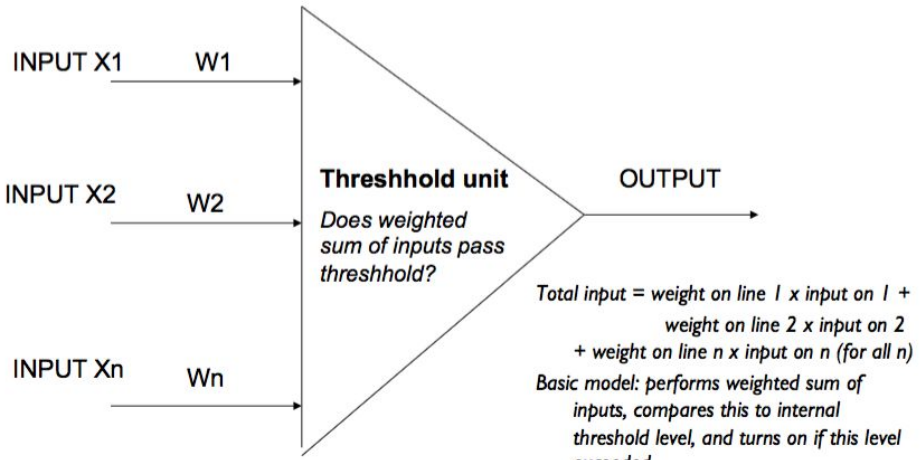
WARREN S. MCCULLOCH AND WALTER PITTS

FROM THE UNIVERSITY OF ILLINOIS, COLLEGE OF MEDICINE,
DEPARTMENT OF PSYCHIATRY AT THE ILLINOIS NEUROPSYCHIATRIC INSTITUTE,
AND THE UNIVERSITY OF CHICAGO

Because of the "all-or-none" character of nervous activity, neural events and the relations among them can be treated by means of propositional logic. It is found that the behavior of every net can be described in these terms, with the addition of more complicated logical means for nets containing circles; and that for any logical expression satisfying certain conditions, one can find a net behaving in the fashion it describes. It is shown that many particular choices among possible neurophysiological assumptions are equivalent, in the sense that for every net behaving under one assumption, there exists another net which behaves under the other and gives the same results, although perhaps not in the same time. Various applications of the calculus are discussed.

McCulloch & Pitts Neuron

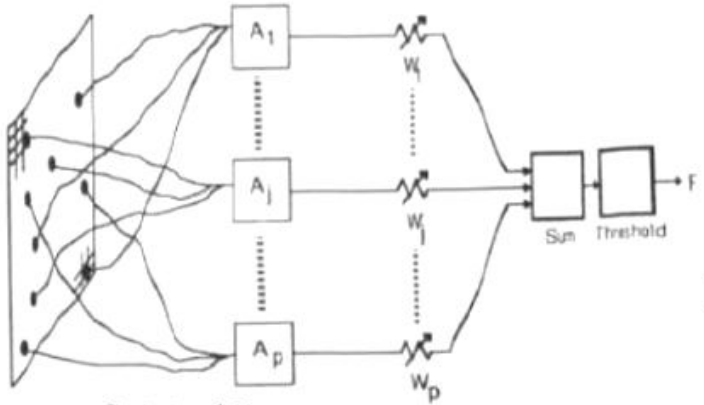
McCulloch Pitts Neuron



Total input = weight on line 1 x input on 1 + weight on line 2 x input on 2 + weight on line n x input on n (for all n)

Basic model: performs weighted sum of inputs, compares this to internal threshold level, and turns on if this level exceeded.

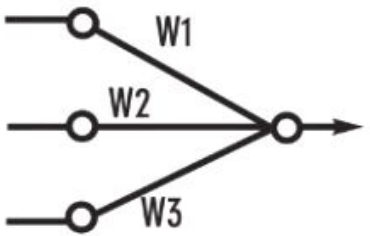
Perceptron



Original Perceptron

(From Perceptrons by M. L Minsky and S. Papert, 1969, Cambridge, MA: MIT Press. Copyright 1969 by MIT Press.

Frank Rosenblatt
(1928-1971)

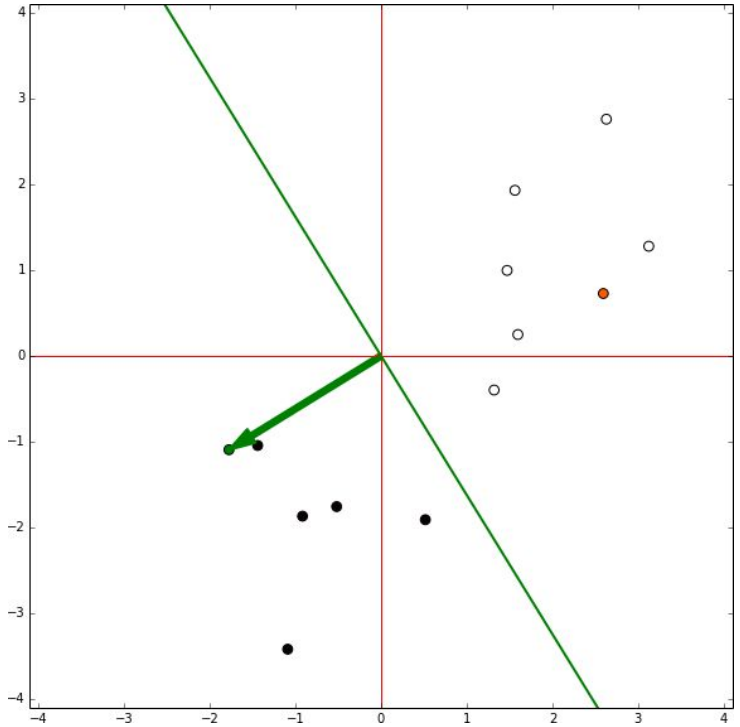


Simplified model:

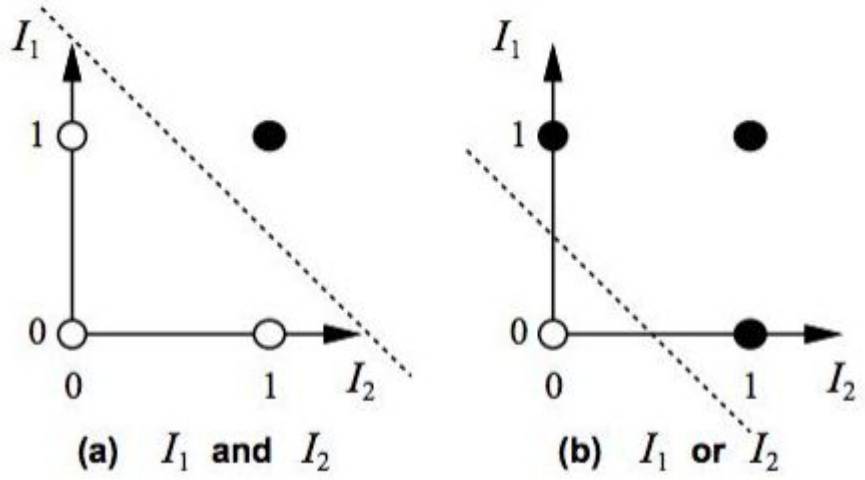
23

Frank Rosenblatt (1957)

Perceptron Learning Rule



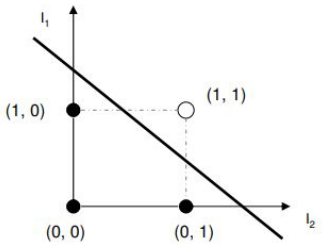
Linear Seperable Case



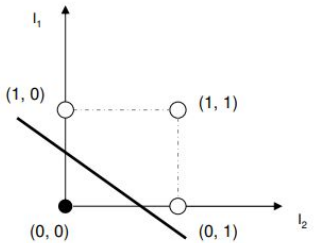
Xor Case

Use more than one neuron!

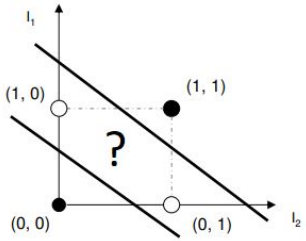
AND		
I_1	I_2	out
0	0	0
0	1	0
1	0	0
1	1	1



OR		
I_1	I_2	out
0	0	0
0	1	1
1	0	1
1	1	1



XOR		
I_1	I_2	out
0	0	0
0	1	1
1	0	1
1	1	0



Backpropagation (1985)

A bit of history...

$$\frac{\partial E_p}{\partial w_{ji}} = \frac{\partial E_p}{\partial o_{pj}} \frac{\partial o_{pj}}{\partial w_{ji}}$$

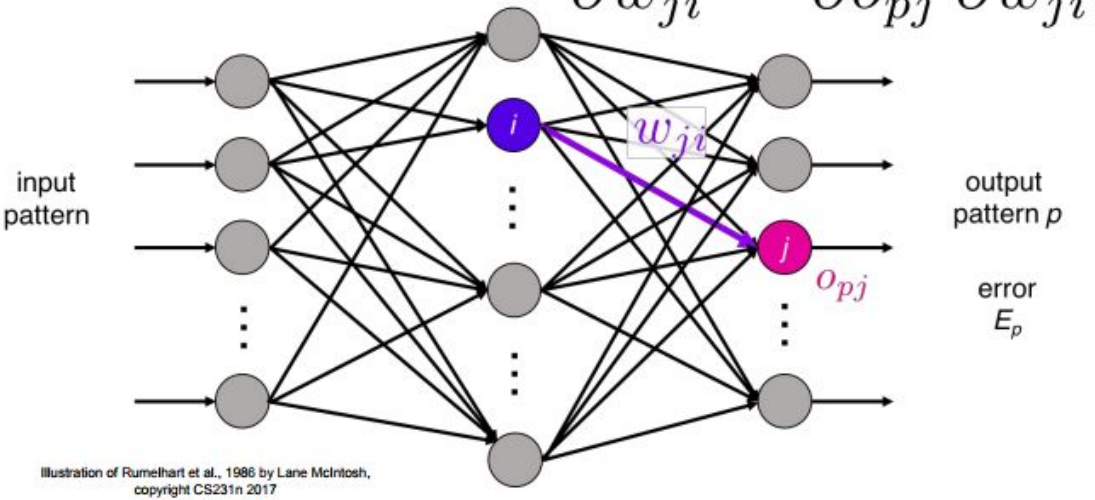


Illustration of Rumelhart et al., 1986 by Lane McIntosh, copyright CS231n 2017

nature > letters > article

MENU

nature
International journal of science

Letter | Published: 09 October 1986

Learning representations by back-propagating errors

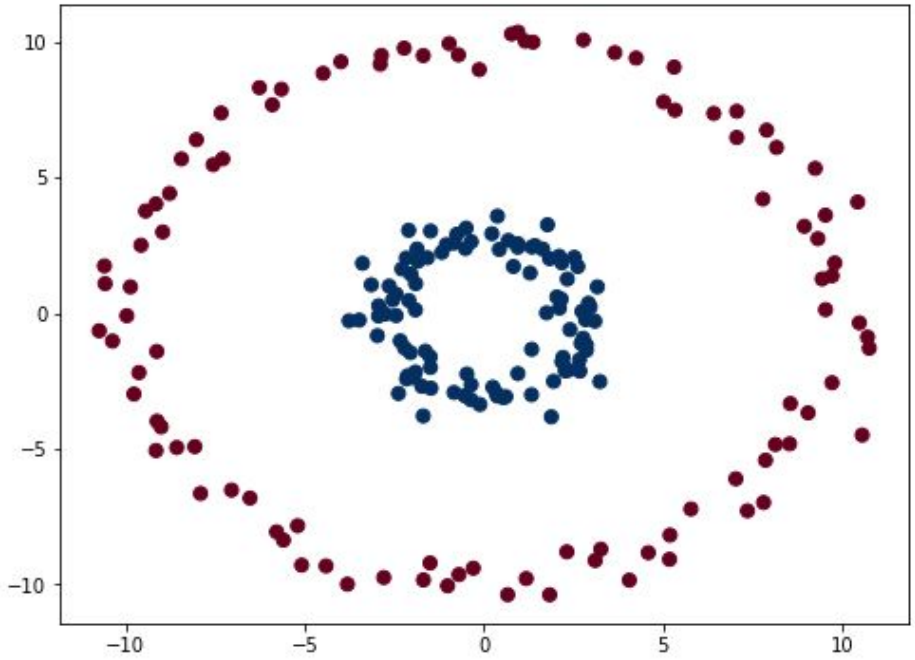
David E. Rumelhart, Geoffrey E. Hinton & Ronald J. Williams

Nature **323**, 533–536 (09 October 1986) | Download Citation

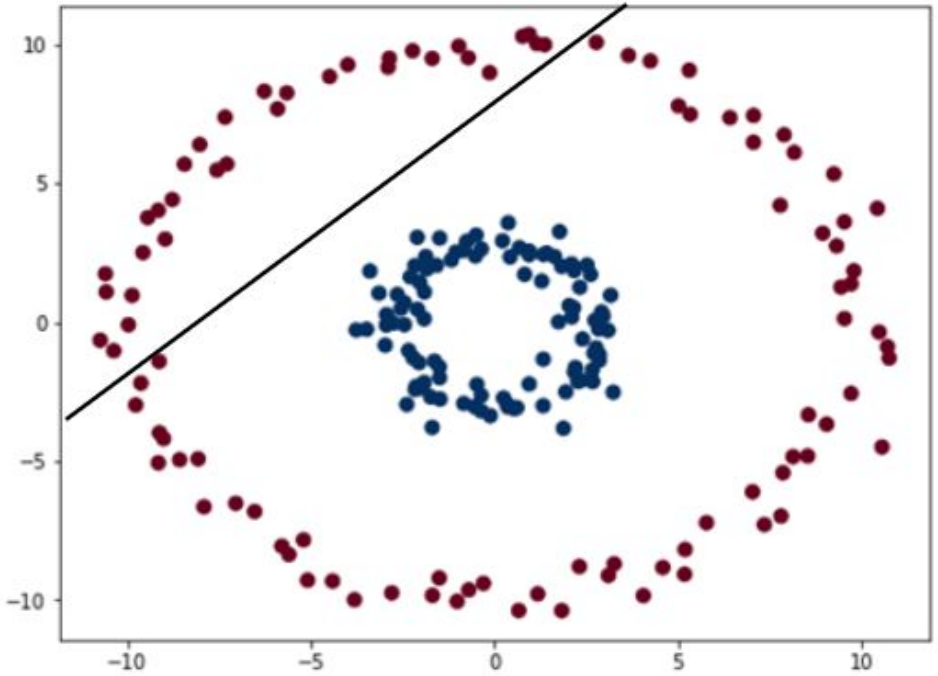
Abstract

We describe a new learning procedure, back-propagation, for networks of neurone-like units. The procedure repeatedly adjusts the weights of the connections in the network so as to minimize a measure of the difference between the actual output vector of the net and the desired output vector. As a result of the weight adjustments, internal 'hidden' units which are not part of the input or output come to represent important features of the task domain, and the regularities in the task are captured by the interactions of these units. The ability to create useful new features distinguishes back-propagation from earlier, simpler methods such as the perceptron-convergence procedure[1].

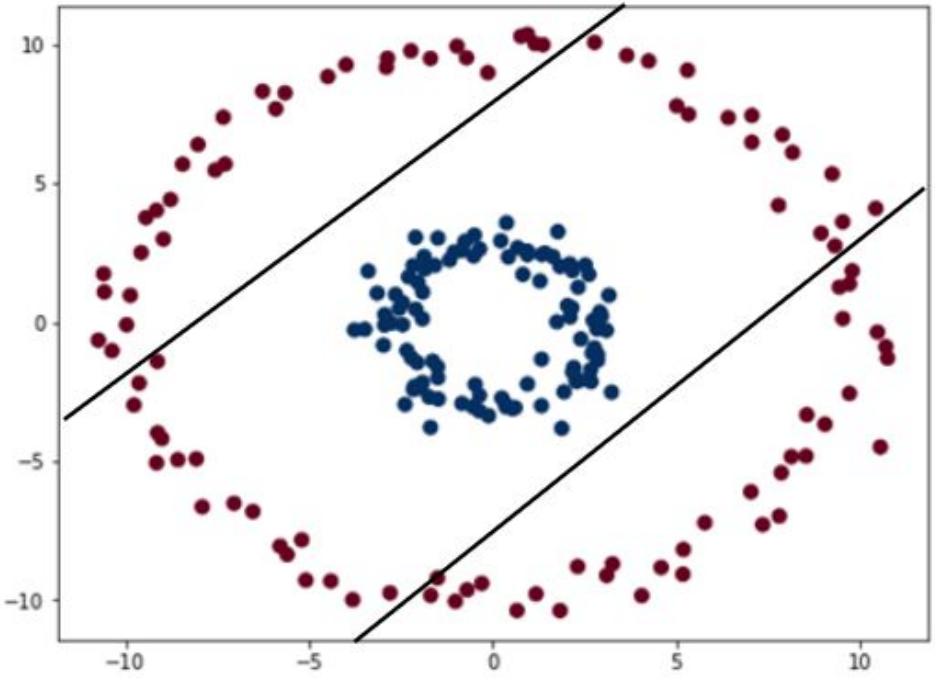
What to do in a More Complex Scenario?



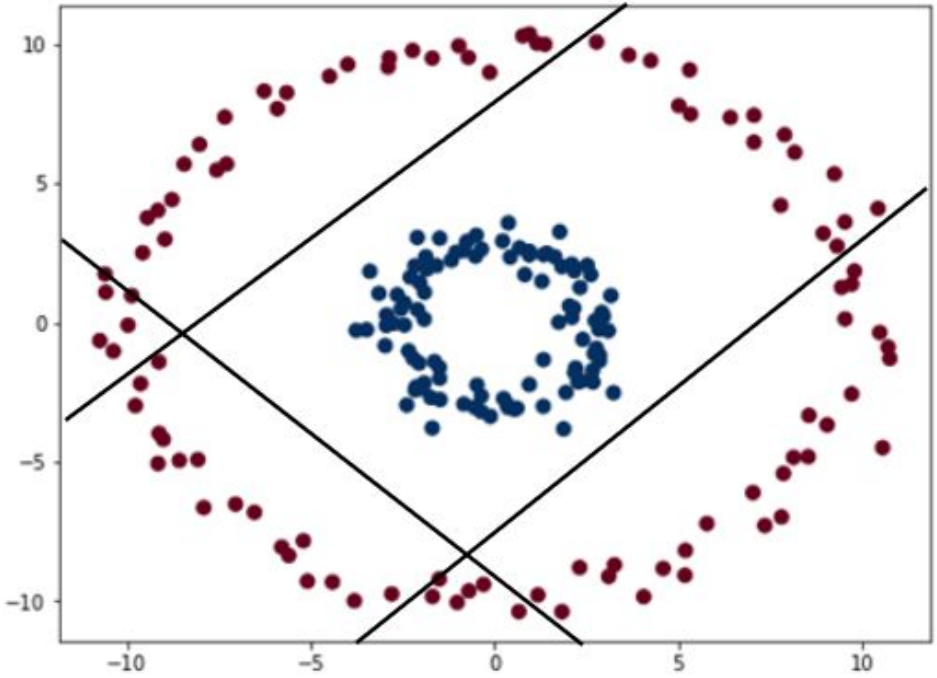
First Neuron



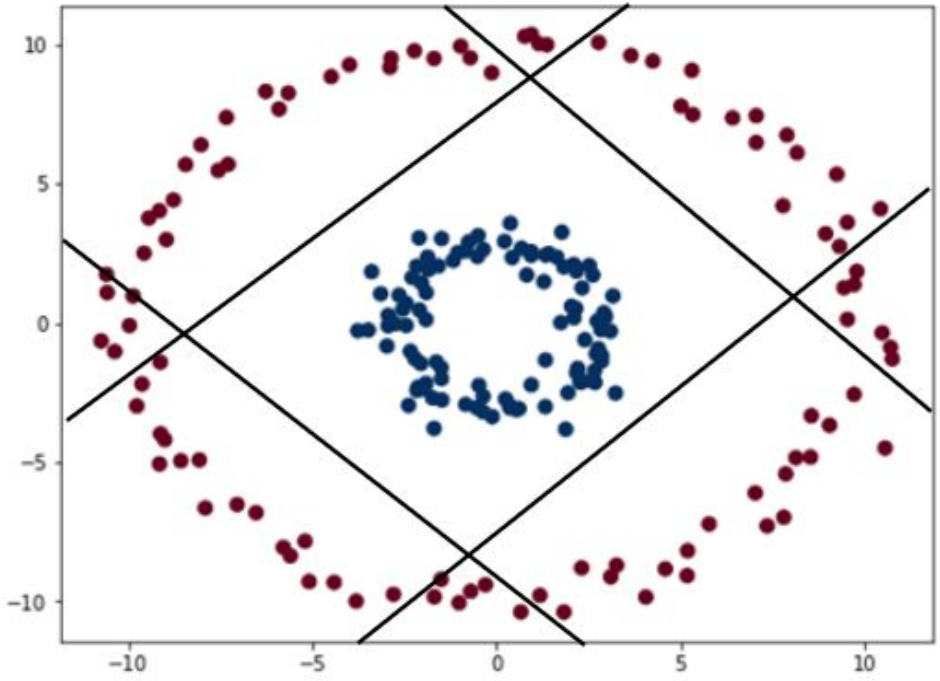
Second Neuron



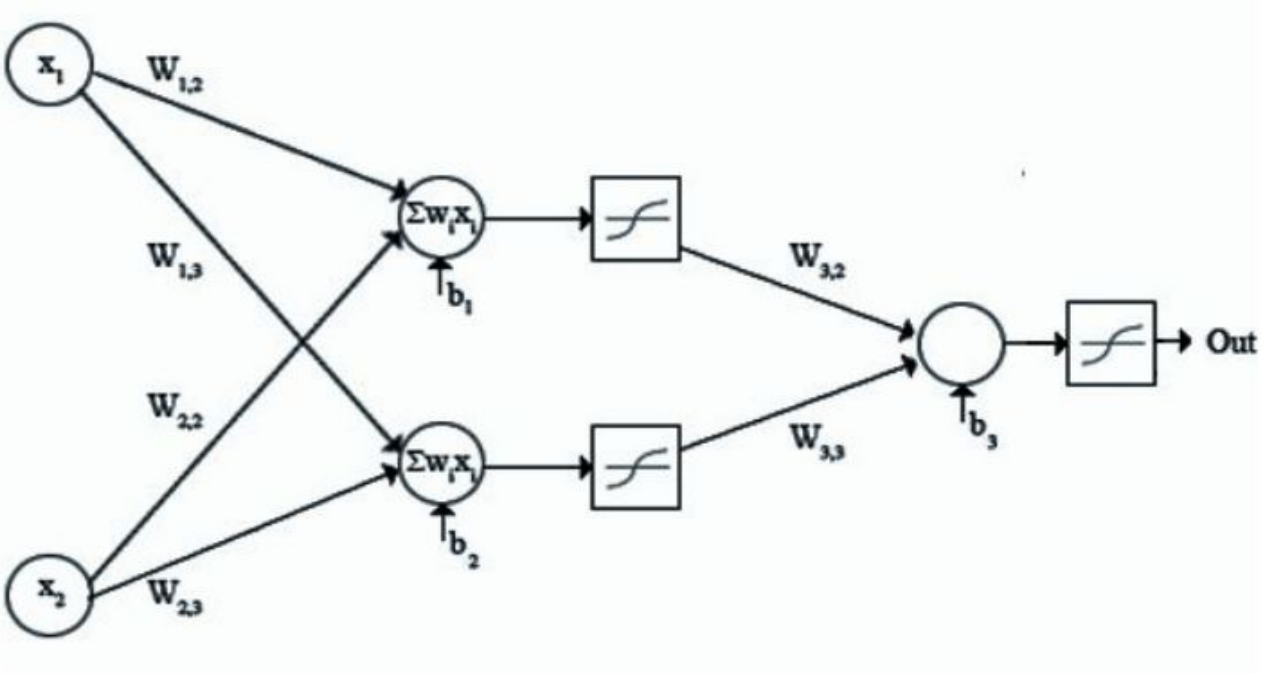
Third Neuron



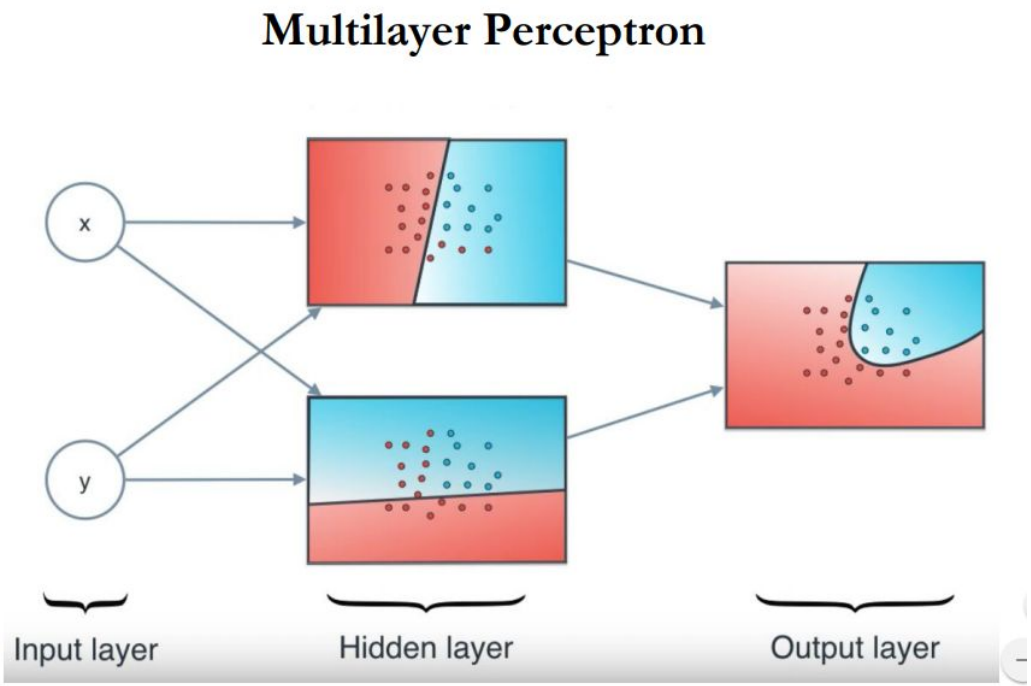
Fourth Neuron



Multilayer Perceptron



Multilayer Perceptron



Neural Network for classification

Vector function with tunable parameters θ

$$\mathbf{f}(\cdot;\theta) : \mathbb{R}^N \to (0,1)^K$$

Sample s in dataset S:

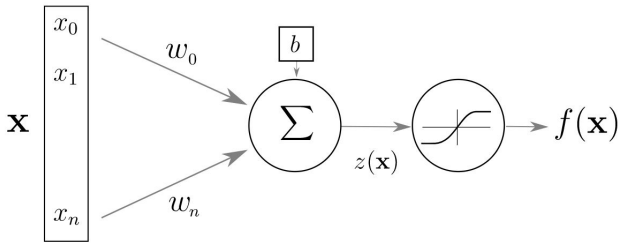
- input: $\mathbf{x}^s \in \mathbb{R}^N$
- expected output: $y^s \in [0, K-1]$

Output is a conditional probability distribution:

$$\mathbf{f}(\mathbf{x}^s;\theta)_c = P(Y = c | X = \mathbf{x}^s)$$

Slide Credit: ip-paris, deep

Artificial Neuron



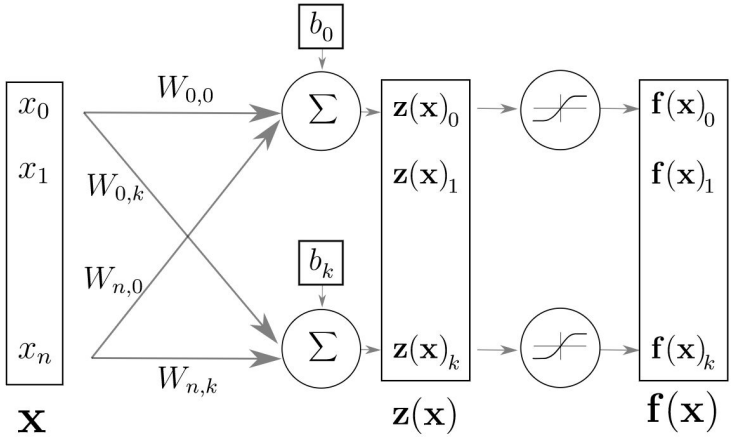
$$z(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$$

$$f(\mathbf{x}) = g(\mathbf{w}^T\mathbf{x} + b)$$

- $\mathbf{x}$, $f(\mathbf{x})$ input and output
- $z(\mathbf{x})$ pre-activation
- $\mathbf{w}$, b weights and bias
- g activation function

Slide Credit: ip-paris, deep learning

Layer of Neurons

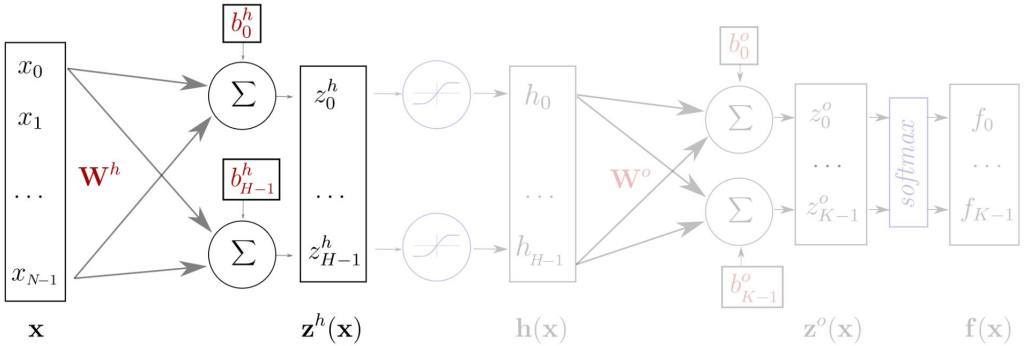


$$\mathbf{f}(\mathbf{x}) = g(\mathbf{z}(\mathbf{x})) = g(\mathbf{W}\mathbf{x} + \mathbf{b})$$

- $\mathbf{W}, \mathbf{b}$ now matrix and vector

Slide Credit: ip-paris, deep

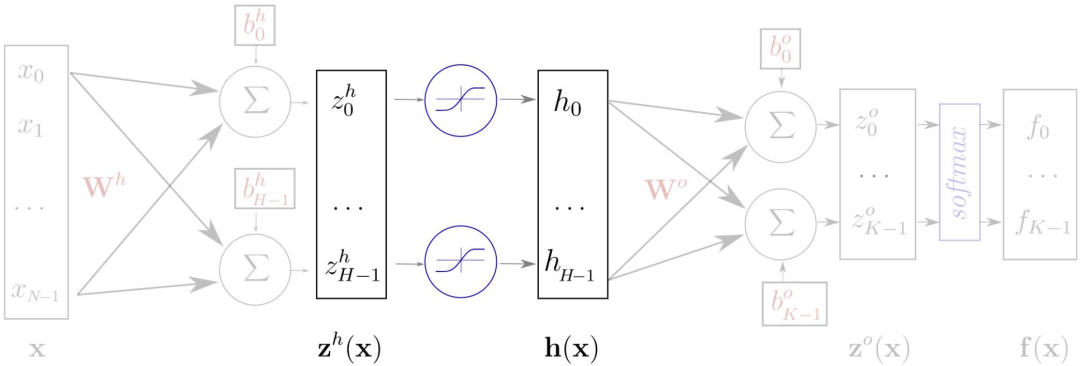
One Hidden Layer Network



- $\mathbf{z}^h(\mathbf{x}) = \mathbf{W}^h\mathbf{x} + \mathbf{b}^h$
- $\mathbf{h}(\mathbf{x}) = g(\mathbf{z}^h(\mathbf{x})) = g(\mathbf{W}^h\mathbf{x} + \mathbf{b}^h)$
- $\mathbf{z}^o(\mathbf{x}) = \mathbf{W}^o\mathbf{h}(\mathbf{x}) + \mathbf{b}^o$
- $\mathbf{f}(\mathbf{x}) = softmax(\mathbf{z}^o) = softmax(\mathbf{W}^o\mathbf{h}(\mathbf{x}) + \mathbf{b}^o)$

Slide Credit: ip-paris, deep learning

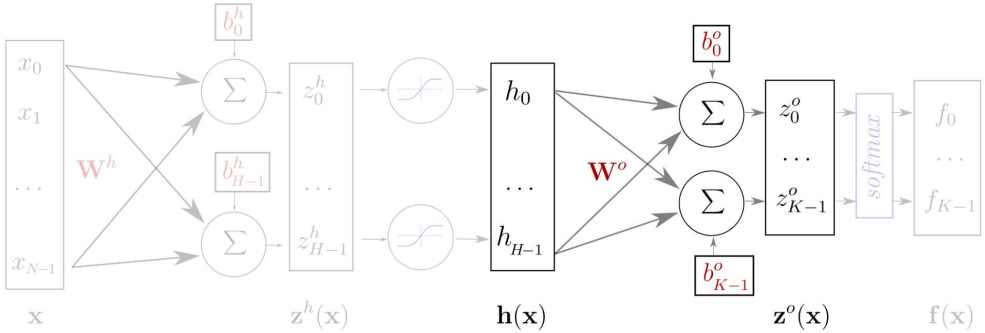
One Hidden Layer Network



- $\mathbf{z}^h(\mathbf{x}) = \mathbf{W}^h\mathbf{x} + \mathbf{b}^h$
- $\mathbf{h}(\mathbf{x}) = g(\mathbf{z}^h(\mathbf{x})) = g(\mathbf{W}^h\mathbf{x} + \mathbf{b}^h)$
- $\mathbf{z}^o(\mathbf{x}) = \mathbf{W}^o\mathbf{h}(\mathbf{x}) + \mathbf{b}^o$
- $\mathbf{f}(\mathbf{x}) = softmax(\mathbf{z}^o) = softmax(\mathbf{W}^o\mathbf{h}(\mathbf{x}) + \mathbf{b}^o)$

Slide Credit: ip-paris, deep lear

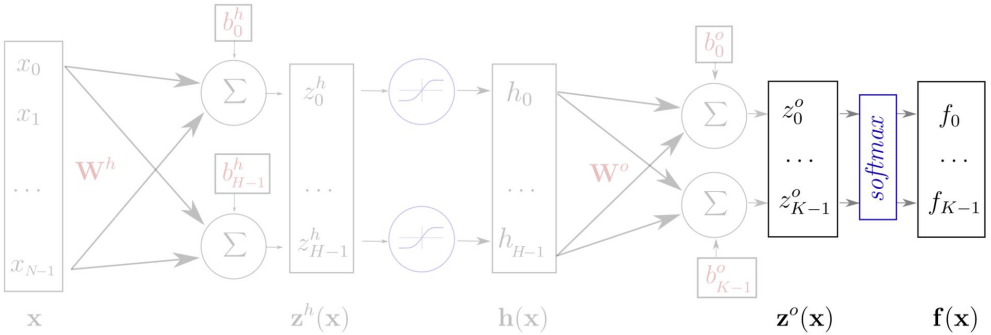
One Hidden Layer Network



- $\mathbf{z}^h(\mathbf{x}) = \mathbf{W}^h\mathbf{x} + \mathbf{b}^h$
- $\mathbf{h}(\mathbf{x}) = g(\mathbf{z}^h(\mathbf{x})) = g(\mathbf{W}^h\mathbf{x} + \mathbf{b}^h)$
- $\mathbf{z}^o(\mathbf{x}) = \mathbf{W}^o\mathbf{h}(\mathbf{x}) + \mathbf{b}^o$
- $\mathbf{f}(\mathbf{x}) = softmax(\mathbf{z}^o) = softmax(\mathbf{W}^o\mathbf{h}(\mathbf{x}) + \mathbf{b}^o)$

Slide Credit: ip-paris, deep learning

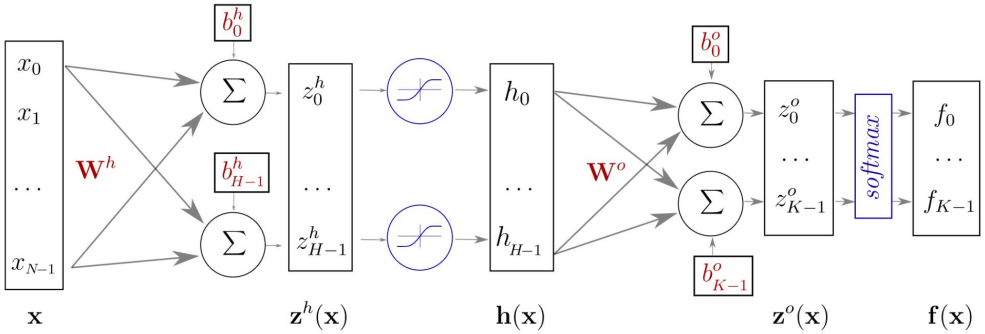
One Hidden Layer Network



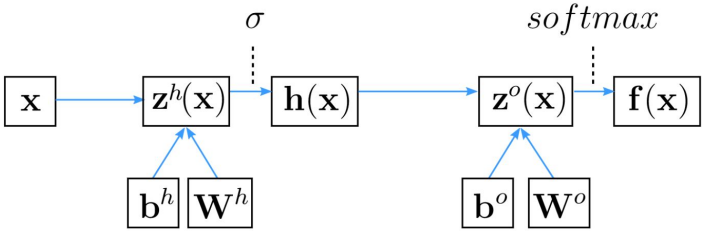
- $\mathbf{z}^h(\mathbf{x}) = \mathbf{W}^h\mathbf{x} + \mathbf{b}^h$
- $\mathbf{h}(\mathbf{x}) = g(\mathbf{z}^h(\mathbf{x})) = g(\mathbf{W}^h\mathbf{x} + \mathbf{b}^h)$
- $\mathbf{z}^o(\mathbf{x}) = \mathbf{W}^o\mathbf{h}(\mathbf{x}) + \mathbf{b}^o$
- $\mathbf{f}(\mathbf{x}) = softmax(\mathbf{z}^o) = softmax(\mathbf{W}^o\mathbf{h}(\mathbf{x}) + \mathbf{b}^o)$

Slide Credit: ip-paris, deep learning

One Hidden Layer Network

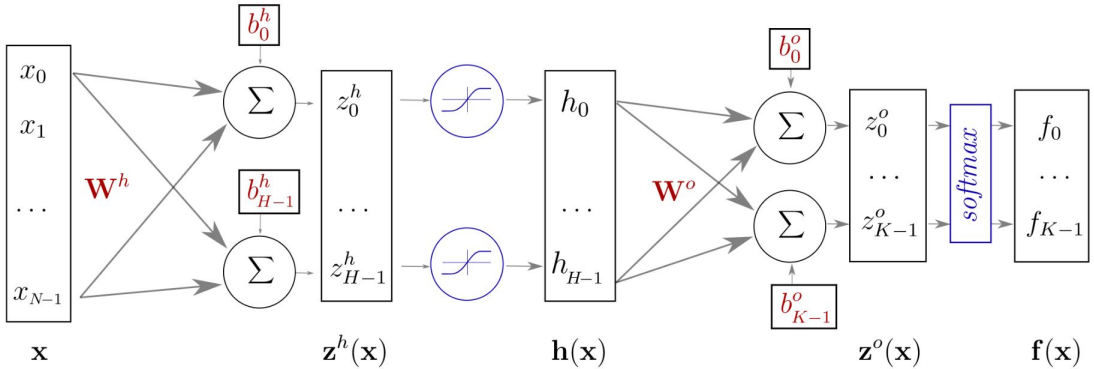


Alternate representation



Slide Credit: ip-paris, deep learning

One Hidden Layer Network



Keras implementation

```
model = Sequential()
model.add(Dense(H, input_dim=N))  # weight matrix dim [N * H]
model.add(Activation("tanh"))
model.add(Dense(K))               # weight matrix dim [H x K]
model.add(Activation("softmax"))
```

Slide Credit: ip-paris, deep learning

Softmax Function

$$softmax(\mathbf{x}) = \frac{1}{\sum_{i=1}^{n} e^{x_i}} \cdot \begin{bmatrix} e^{x_1} \\ e^{x_2} \\ \vdots \\ e^{x_n} \end{bmatrix}$$

$$\frac{\partial softmax(\mathbf{x})_i}{\partial x_j} = \begin{cases} softmax(\mathbf{x})_i \cdot (1 - softmax(\mathbf{x})_i) & i = j \\ -softmax(\mathbf{x})_i \cdot softmax(\mathbf{x})_j & i \neq j \end{cases}$$

- vector of values in (0, 1) that add up to 1
- $p(Y = c|X = \mathbf{x}) = \text{softmax}(\mathbf{z}(\mathbf{x}))_c$
- the pre-activation vector $\mathbf{z}(\mathbf{x})$ is often called "the logits"

Slide Credit: ip-paris, deep learning

Training the Network

Find parameters $\theta = (\mathbf{W}^h; \mathbf{b}^h; \mathbf{W}^o; \mathbf{b}^o)$ that minimize the **negative log likelihood** (or cross entropy)

The loss function for a given sample $s \in S$:

$$l(\mathbf{f}(\mathbf{x}^s; \theta), y^s) = nll(\mathbf{x}^s, y^s; \theta) = -\log \mathbf{f}(\mathbf{x}^s; \theta)_{y^s}$$

example $y^s = 3$

$$l(\mathbf{f}(\mathbf{x}^s; \theta), y^s) = l\left(\begin{bmatrix} f_0 \\ \dots \\ f_3 \\ \dots \\ f_{K-1} \end{bmatrix}, \begin{bmatrix} 0 \\ \dots \\ 1 \\ \dots \\ 0 \end{bmatrix}\right) = -\log\ f_3$$

Slide Credit: ip-paris, deep learning

Training the Network

Find parameters $\theta = (\mathbf{W}^h; \mathbf{b}^h; \mathbf{W}^o; \mathbf{b}^o)$ that minimize the **negative log likelihood** (or cross entropy)

The loss function for a given sample $s \in S$:

$$l(\mathbf{f}(\mathbf{x}^s; \theta), y^s) = nll(\mathbf{x}^s, y^s; \theta) = -\log \mathbf{f}(\mathbf{x}^s; \theta)_{y^s}$$

The cost function is the negative likelihood of the model computed on the full training set (for i.i.d. samples):

$$L_S(\theta) = -\frac{1}{|S|} \sum_{s \in S} \log \mathbf{f}(\mathbf{x}^s; \theta)_{y^s} + \lambda \Omega(\theta)$$

$\lambda \Omega(\theta) = \lambda(||W^h||^2 + ||W^o||^2)$ is an optional regularization term.

Slide Credit: ip-paris, deep learning

Stochastic Gradient Descent

Initialize θ randomly

For E epochs perform:

- Randomly select a small batch of samples $(B \subset S)$
  - Compute gradients: $\Delta = \nabla_\theta L_B(\theta)$
  - Update parameters: $\theta \leftarrow \theta - \eta\Delta$
  - $\eta > 0$ is called the learning rate
- Repeat until the epoch is completed (all of S is covered)

Stop when reaching criterion:

- nll stops decreasing when computed on validation set

Slide Credit: ip-paris, deep learning

Computing Gradients

Output Weights: $\frac{\partial l(\mathbf{f}(\mathbf{x}),y)}{\partial W^o_{i,j}}$

Output bias: $\frac{\partial l(\mathbf{f}(\mathbf{x}),y)}{\partial b^o_i}$

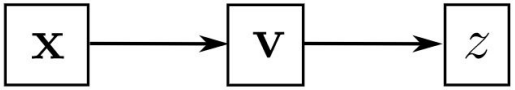
Hidden Weights: $\frac{\partial l(\mathbf{f}(\mathbf{x}),y)}{\partial W^h_{i,j}}$

Hidden bias: $\frac{\partial l(\mathbf{f}(\mathbf{x}),y)}{\partial b^h_i}$

- The network is a composition of differentiable modules
- We can apply the "chain rule"

Slide Credit: ip-paris, deep learning

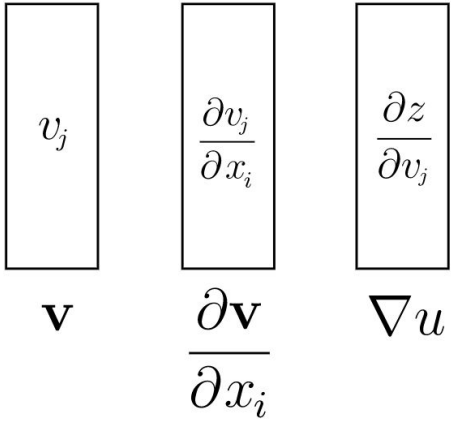
Chain rule



$$z = u(\mathbf{v}(\mathbf{x}))$$

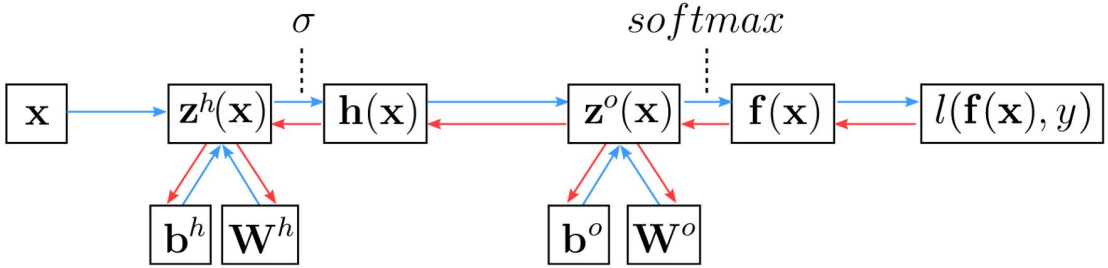
chain-rule

$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial v_j} \frac{\partial v_j}{\partial x_i} = \nabla u . \frac{\partial \mathbf{v}}{\partial x_i}$$



Slide Credit: ip-paris, deep learning

Backpropagation



Compute partial derivatives of the loss

- $\frac{\partial l(\mathbf{f}(\mathbf{x}),y)}{\partial \mathbf{f}(\mathbf{x})_i} = \frac{\partial -\log \mathbf{f}(\mathbf{x})_y}{\partial \mathbf{f}(\mathbf{x})_i} = \frac{-1_{y=i}}{\mathbf{f}(\mathbf{x})_y} = \frac{\partial \boldsymbol{l}}{\partial \mathbf{f}(\mathbf{x})_i}$
- $\frac{\partial \boldsymbol{l}}{\partial \mathbf{z}^o(\mathbf{x})_i} = ?$

Slide Credit: ip-paris, deep learning

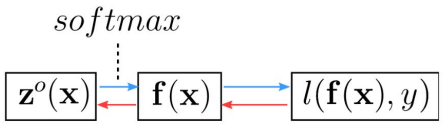
Backpropagation

$$\frac{\partial l}{\partial \mathbf{z}^o(\mathbf{x})_i} = \sum_j \frac{\partial l}{\partial \mathbf{f}(\mathbf{x})_j} \frac{\partial \mathbf{f}(\mathbf{x})_j}{\partial \mathbf{z}^o(\mathbf{x})_i}$$

$$= \sum_j \frac{-1_{y=j}}{\mathbf{f}(\mathbf{x})_y} \frac{\partial softmax(\mathbf{z}^o(\mathbf{x}))_j}{\partial \mathbf{z}^o(\mathbf{x})_i}$$

$$\frac{\partial l}{\partial \mathbf{f}(\mathbf{x})_i} = \frac{-1_{y=i}}{\mathbf{f}(\mathbf{x})_y}$$

$$\mathbf{f}(\mathbf{x}) = softmax(\mathbf{z}^o(\mathbf{x}))$$



Slide Credit: ip-paris, deep learning

Backpropagation

$$\frac{\partial l}{\partial \mathbf{z}^o(\mathbf{x})_i} = \sum_j \frac{\partial l}{\partial \mathbf{f}(\mathbf{x})_j} \frac{\partial \mathbf{f}(\mathbf{x})_j}{\partial \mathbf{z}^o(\mathbf{x})_i}$$

$$= \sum_j \frac{-1_{y=j}}{\mathbf{f}(\mathbf{x})_y} \frac{\partial softmax(\mathbf{z}^o(\mathbf{x}))_j}{\partial \mathbf{z}^o(\mathbf{x})_i}$$

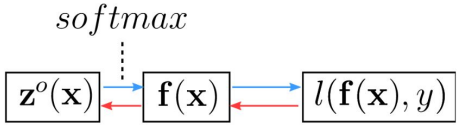
$$= -\frac{1}{\mathbf{f}(\mathbf{x})_y} \frac{\partial softmax(\mathbf{z}^o(\mathbf{x}))_y}{\partial \mathbf{z}^o(\mathbf{x})_i}$$

$$= \begin{cases} -\frac{1}{\mathbf{f}(\mathbf{x})_y} softmax(\mathbf{z}^o(\mathbf{x}))_y (1 - softmax(\mathbf{z}^o(\mathbf{x}))_y) & \text{if } i = y \\ \frac{1}{\mathbf{f}(\mathbf{x})_y} softmax(\mathbf{z}^o(\mathbf{x}))_y softmax(\mathbf{z}^o(\mathbf{x}))_i & \text{if } i \neq y \end{cases}$$

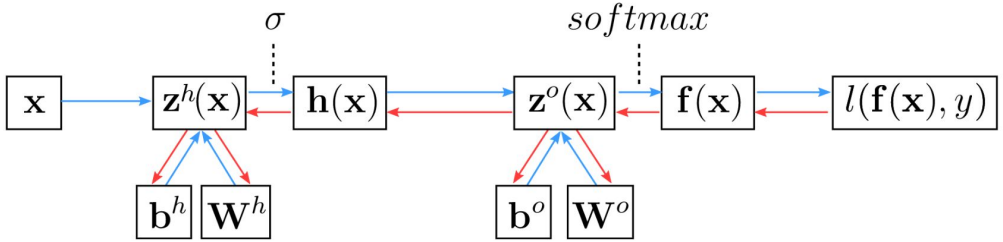
$$= \begin{cases} -1 + \mathbf{f}(\mathbf{x})_y & \text{if } i = y \\ \mathbf{f}(\mathbf{x})_i & \text{if } i \neq y \end{cases}$$

$$\nabla_{\mathbf{z}^o(\mathbf{x})} l(\mathbf{f}(\mathbf{x}), y) = \mathbf{f}(\mathbf{x}) - \mathbf{e}(y)$$

$\mathbf{e}(y)$: one-hot encoding of y



Backpropagation



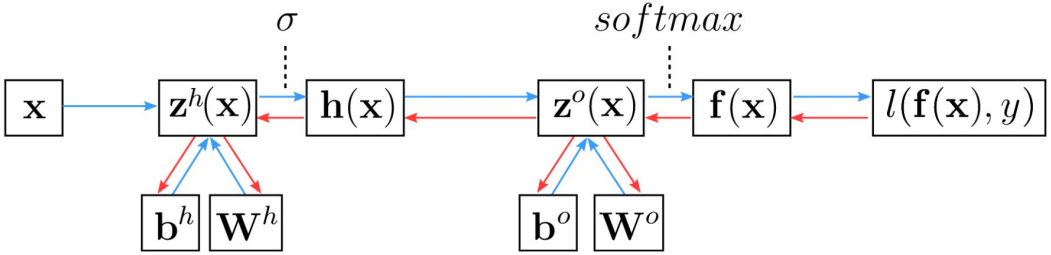
Gradients

- $\nabla_{\mathbf{z}^o(\mathbf{x})} \boldsymbol{l} = \mathbf{f}(\mathbf{x}) - \mathbf{e}(y)$
- $\nabla_{\mathbf{b}^o} \boldsymbol{l} = \mathbf{f}(\mathbf{x}) - \mathbf{e}(y)$

because $\mathbf{z}^o(\mathbf{x}) = \mathbf{W}^o \mathbf{h}(\mathbf{x}) + \mathbf{b}^o$ and then $\frac{\partial \mathbf{z}^o(\mathbf{x})_i}{\partial \mathbf{b}^o_j} = 1_{i=j}$

Slide Credit: ip-paris, deep learning

Backpropagation



Partial derivatives related to $\mathbf{W}^o$

- $\frac{\partial \boldsymbol{l}}{\partial W^o_{i,j}} = \sum_k \frac{\partial \boldsymbol{l}}{\partial \mathbf{z}^o(\mathbf{x})_k} \frac{\partial \mathbf{z}^o(\mathbf{x})_k}{\partial W^o_{i,j}}$
- $\nabla_{\mathbf{W}^o} \boldsymbol{l} = (\mathbf{f}(\mathbf{x}) - \mathbf{e}(y)).\mathbf{h}(\mathbf{x})^\top$

Slide Credit: ip-paris, deep learning

Backprop gradients

Compute activation gradients

- $\nabla_{\mathbf{z}^o(\mathbf{x})} \boldsymbol{l} = \mathbf{f}(\mathbf{x}) - \mathbf{e}(y)$

Compute layer params gradients

- $\nabla_{\mathbf{W}^o} \boldsymbol{l} = \nabla_{\mathbf{z}^o(\mathbf{x})} \boldsymbol{l} \cdot \mathbf{h}(\mathbf{x})^\top$
- $\nabla_{\mathbf{b}^o} \boldsymbol{l} = \nabla_{\mathbf{z}^o(\mathbf{x})} \boldsymbol{l}$

Compute prev layer activation gradients

- $\nabla_{\mathbf{h}(\mathbf{x})} \boldsymbol{l} = \mathbf{W}^{o\top} \nabla_{\mathbf{z}^o(\mathbf{x})} \boldsymbol{l}$
- $\nabla_{\mathbf{z}^h(\mathbf{x})} \boldsymbol{l} = \nabla_{\mathbf{h}(\mathbf{x})} \boldsymbol{l} \odot \sigma'(\mathbf{z}^{\mathbf{h}}(\mathbf{x}))$

Slide Credit: ip-paris, deep learning

Initialization and normalization

- Input data should be normalized to have approx. same range:
  - standardization or quantile normalization
- Initializing W^h and W^o:
  - Zero is a saddle point: no gradient, no learning
  - Constant init: hidden units collapse by symmetry
  - Solution: random init, ex: $w \sim \mathcal{N}(0, 0.01)$
  - Better inits: Xavier Glorot and Kaming He & orthogonal
- Biases can (should) be initialized to zero

Slide Credit: ip-paris, deep learning

SGD Learning Rate

- Very sensitive:
  - Too high $\rightarrow$ early plateau or even divergence
  - Too low $\rightarrow$ slow convergence
  - Try a large value first: $\eta = 0.1$ or even $\eta = 1$
  - Divide by 10 and retry in case of divergence
- Large constant LR prevents final convergence
  - multiply η_t by $\beta < 1$ after each update
  - or monitor validation loss and divide η_t by 2 or 10 when no progress
  - See ReduceLROnPlateau in Keras

Slide Credit: ip-paris, deep learning

Alternative optimizers

- SGD (with Nesterov momentum)
  - Simple to implement
  - Very sensitive to initial value of η
  - Need learning rate scheduling
- Adam: adaptive learning rate scale for each param
  - Global η set to 3e-4 often works well enough
  - Good default choice of optimizer (often)
- But well-tuned SGD with LR scheduling can generalize better than Adam (with naive l2 reg)...
- Promising stochastic second order methods: K-FAC and Shampoo can be used to accelerate training of very large models.

Slide Credit: ip-paris, deep learning

Next Class:

Support Vector Machines