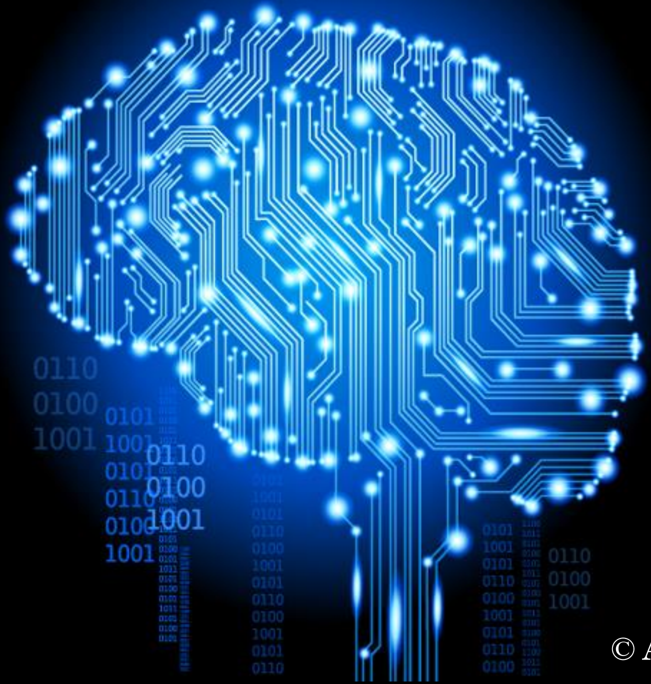


© AlchemyAPI

CENG 501
Deep Learning:

A GAME OF NEURONS

Sinan Kalkan

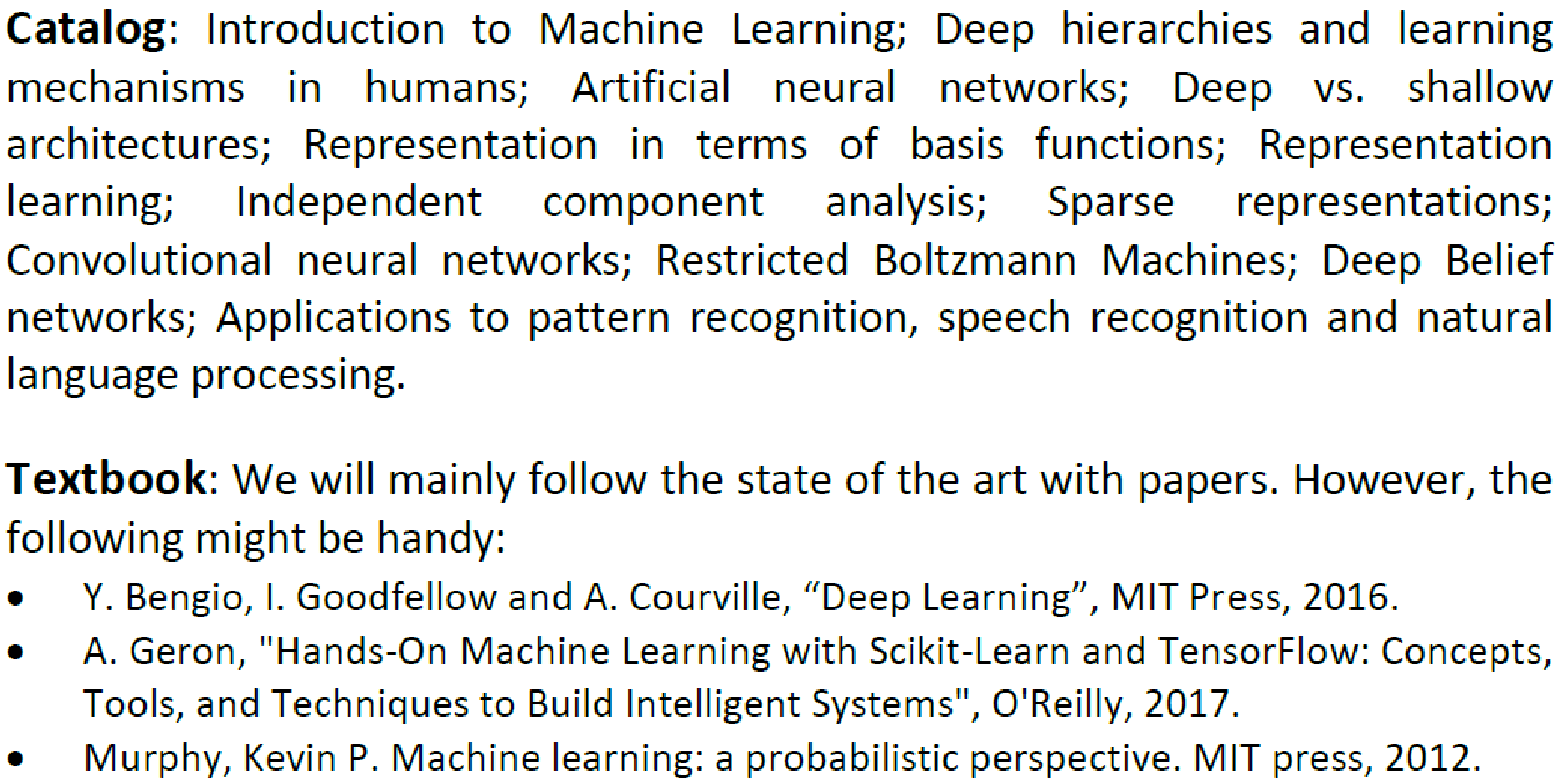
Today

- Overview of the course
- A review of deep learning fundamentals

About the course

Syllabus

http://ceng.metu.edu.tr/~skalkan/DL/syllabus.pdf



Catalog: Introduction to Machine Learning; Deep hierarchies and learning mechanisms in humans; Artificial neural networks; Deep vs. shallow architectures; Representation in terms of basis functions; Representation learning; Independent component analysis; Sparse representations; Convolutional neural networks; Restricted Boltzmann Machines; Deep Belief networks; Applications to pattern recognition, speech recognition and natural language processing.

Textbook: We will mainly follow the state of the art with papers. However, the following might be handy:

- Y. Bengio, I. Goodfellow and A. Courville, “Deep Learning”, MIT Press, 2016.
- A. Geron, "Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems", O'Reilly, 2017.
- Murphy, Kevin P. Machine learning: a probabilistic perspective. MIT press, 2012.

Syllabus

Web: https://user.ceng.metu.edu.tr/~skalkan/DL/

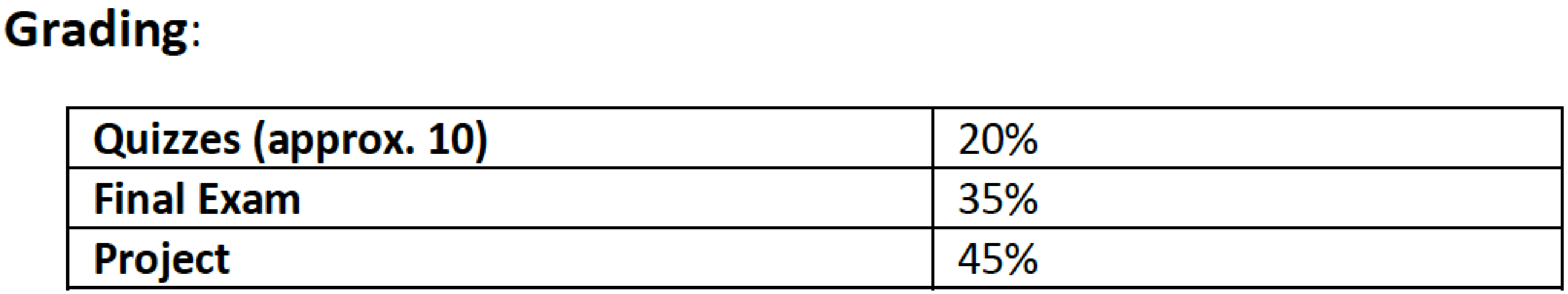
Emailing List: METU Class page of the course.

Instructor: Sinan Kalkan, skalkan@metu.edu.tr (Office hours: by appointment)

Lectures: Wednesdays, 9:40-11:30, BMB2

Credits: METU: 3 Theoretical, 0 Laboratory; ECTS: 8.0

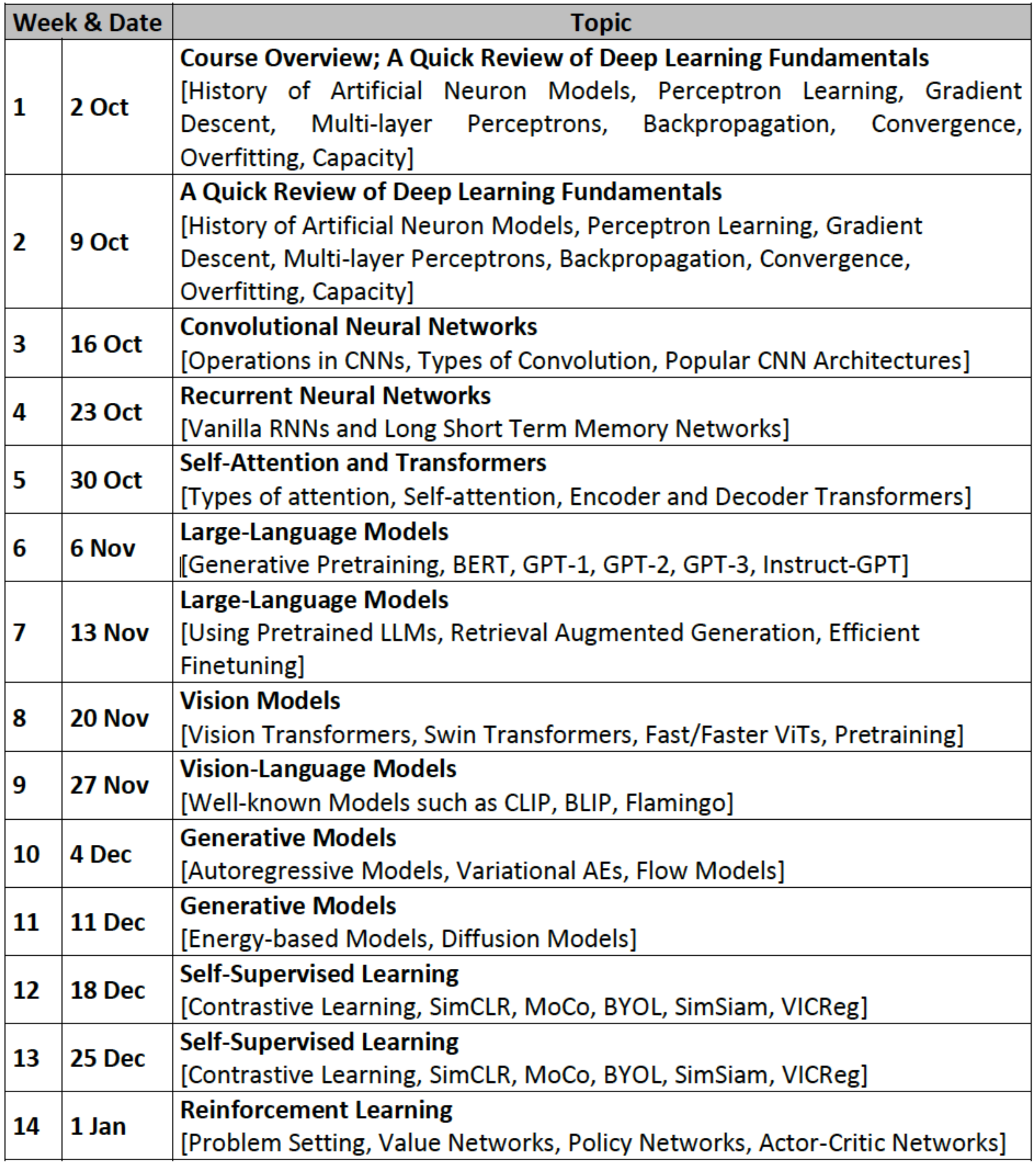
Syllabus



Grading:

Quizzes (approx. 10)	20%
Final Exam	35%
Project	45%

Weekly outline

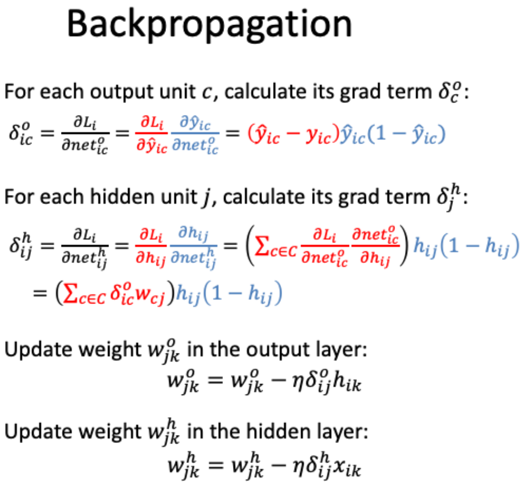


Week & Date		Topic
1	2 Oct	**Course Overview; A Quick Review of Deep Learning Fundamentals** [History of Artificial Neuron Models, Perceptron Learning, Gradient Descent, Multi-layer Perceptrons, Backpropagation, Convergence, Overfitting, Capacity]
2	9 Oct	**A Quick Review of Deep Learning Fundamentals** [History of Artificial Neuron Models, Perceptron Learning, Gradient Descent, Multi-layer Perceptrons, Backpropagation, Convergence, Overfitting, Capacity]
3	16 Oct	**Convolutional Neural Networks** [Operations in CNNs, Types of Convolution, Popular CNN Architectures]
4	23 Oct	**Recurrent Neural Networks** [Vanilla RNNs and Long Short Term Memory Networks]
5	30 Oct	**Self-Attention and Transformers** [Types of attention, Self-attention, Encoder and Decoder Transformers]
6	6 Nov	**Large-Language Models** [Generative Pretraining, BERT, GPT-1, GPT-2, GPT-3, Instruct-GPT]
7	13 Nov	**Large-Language Models** [Using Pretrained LLMs, Retrieval Augmented Generation, Efficient Finetuning]
8	20 Nov	**Vision Models** [Vision Transformers, Swin Transformers, Fast/Faster ViTs, Pretraining]
9	27 Nov	**Vision-Language Models** [Well-known Models such as CLIP, BLIP, Flamingo]
10	4 Dec	**Generative Models** [Autoregressive Models, Variational AEs, Flow Models]
11	11 Dec	**Generative Models** [Energy-based Models, Diffusion Models]
12	18 Dec	**Self-Supervised Learning** [Contrastive Learning, SimCLR, MoCo, BYOL, SimSiam, VICReg]
13	25 Dec	**Self-Supervised Learning** [Contrastive Learning, SimCLR, MoCo, BYOL, SimSiam, VICReg]
14	1 Jan	**Reinforcement Learning** [Problem Setting, Value Networks, Policy Networks, Actor-Critic Networks]

Project

- Implement a conference paper without any implementation
  - Reproduce their results and provide a critical analysis
  - Produce a Github Repo with a detailed Readme file
- Papers can **only** be from the following top conferences:
  - ICLR, AAAI, NeurIPS, ICML, CVPR, ECCV or similar
- You can work in groups of two.
- Deadline for picking papers: 13 October.

Taking the course



Backpropagation

For each output unit c, calculate its grad term δ_c^o:

$$\delta_{ic}^o = \frac{\partial L_i}{\partial net_{ic}^o} = \frac{\partial L_i}{\partial \hat{y}_{ic}} \frac{\partial \hat{y}_{ic}}{\partial net_{ic}^o} = (\hat{y}_{ic} - y_{ic})\hat{y}_{ic}(1 - \hat{y}_{ic})$$

For each hidden unit j, calculate its grad term δ_j^h:

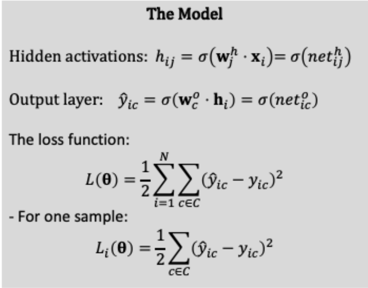
$$\delta_{ij}^h = \frac{\partial L_i}{\partial net_{ij}^h} = \frac{\partial L_i}{\partial h_{ij}} \frac{\partial h_{ij}}{\partial net_{ij}^h} = \left(\sum_{c \in C} \frac{\partial L_i}{\partial net_{ic}^o} \frac{\partial net_{ic}^o}{\partial h_{ij}}\right) h_{ij}(1 - h_{ij})$$
$$= \left(\sum_{c \in C} \delta_{ic}^o w_{cj}\right) h_{ij}(1 - h_{ij})$$

Update weight w_{jk}^o in the output layer:

$$w_{jk}^o = w_{jk}^o - \eta \delta_{ij}^o h_{ik}$$

Update weight w_{jk}^h in the hidden layer:

$$w_{jk}^h = w_{jk}^h - \eta \delta_{ij}^h x_{ik}$$



The Model

Hidden activations: $h_{ij} = \sigma(\mathbf{w}_j^h \cdot \mathbf{x}_i) = \sigma(net_{ij}^h)$

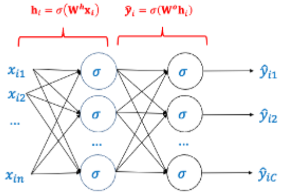
Output layer: $\hat{y}_{ic} = \sigma(\mathbf{w}_c^o \cdot \mathbf{h}_i) = \sigma(net_{ic}^o)$

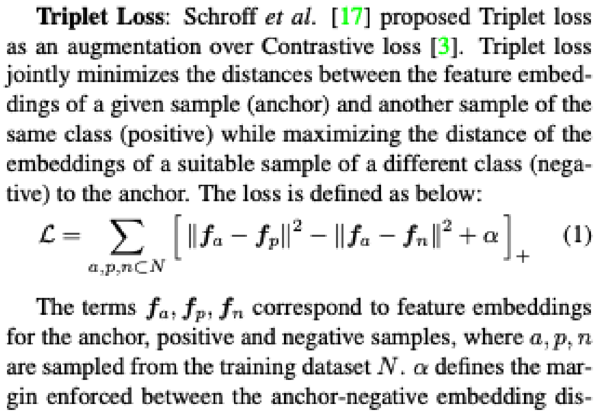
The loss function:

$$L(\boldsymbol{\theta}) = \frac{1}{2}\sum_{i=1}^{N}\sum_{c \in C}(\hat{y}_{ic} - y_{ic})^2$$

- For one sample:

$$L_i(\boldsymbol{\theta}) = \frac{1}{2}\sum_{c \in C}(\hat{y}_{ic} - y_{ic})^2$$

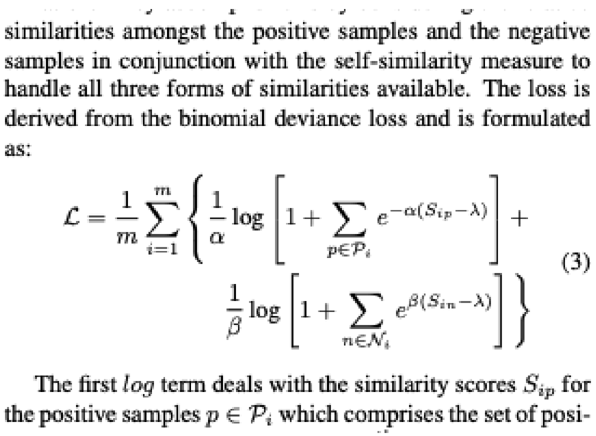

$\mathbf{h}_i = \sigma(\mathbf{W}^h \mathbf{x}_i)$
$\hat{\mathbf{y}}_i = \sigma(\mathbf{W}^o \mathbf{h}_i)$
x_{i1}
x_{i2}
...
x_{in}
σ
...
$\hat{y}_{i1}$
$\hat{y}_{i2}$
$\hat{y}_{iC}$



Triplet Loss: Schroff *et al.* [17] proposed Triplet loss as an augmentation over Contrastive loss [3]. Triplet loss jointly minimizes the distances between the feature embeddings of a given sample (anchor) and another sample of the same class (positive) while maximizing the distance of the embeddings of a suitable sample of a different class (negative) to the anchor. The loss is defined as below:

$$\mathcal{L} = \sum_{a,p,n \subset N} \left[\|\boldsymbol{f}_a - \boldsymbol{f}_p\|^2 - \|\boldsymbol{f}_a - \boldsymbol{f}_n\|^2 + \alpha \right]_+ \quad (1)$$

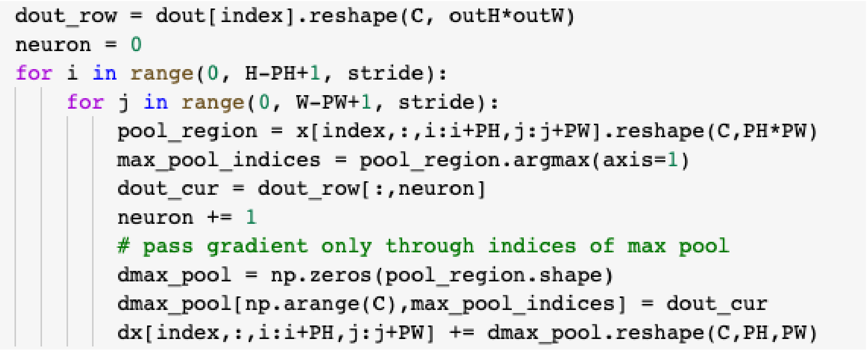
The terms $\boldsymbol{f}_a, \boldsymbol{f}_p, \boldsymbol{f}_n$ correspond to feature embeddings for the anchor, positive and negative samples, where a, p, n are sampled from the training dataset N. α defines the margin enforced between the anchor-negative embedding dis-



similarities amongst the positive samples and the negative samples in conjunction with the self-similarity measure to handle all three forms of similarities available. The loss is derived from the binomial deviance loss and is formulated as:

$$\mathcal{L} = \frac{1}{m}\sum_{i=1}^{m}\left\{\frac{1}{\alpha}\log\left[1 + \sum_{p \in \mathcal{P}_i} e^{-\alpha(S_{ip} - \lambda)}\right] + \frac{1}{\beta}\log\left[1 + \sum_{n \in \mathcal{N}_i} e^{\beta(S_{in} - \lambda)}\right]\right\} \quad (3)$$

The first *log* term deals with the similarity scores S_{ip} for the positive samples $p \in \mathcal{P}_i$ which comprises the set of posi-



```
dout_row = dout[index].reshape(C, outH*outW)
neuron = 0
for i in range(0, H-PH+1, stride):
    for j in range(0, W-PW+1, stride):
        pool_region = x[index,:,i:i+PH,j:j+PW].reshape(C,PH*PW)
        max_pool_indices = pool_region.argmax(axis=1)
        dout_cur = dout_row[:,neuron]
        neuron += 1
        # pass gradient only through indices of max pool
        dmax_pool = np.zeros(pool_region.shape)
        dmax_pool[np.arange(C),max_pool_indices] = dout_cur
        dx[index,:,i:i+PH,j:j+PW] += dmax_pool.reshape(C,PH,PW)
```

Basic DL Concepts

Ensure that you know the following:

- Why does DL work now?
- End-to-end learning
- Distributed representations
- Advantages and disadvantages of DL

Basic ML Concepts

Ensure that you know the following:

- Supervised vs. unsupervised learning
- Discriminative vs. generative learning
- Model selection, cross validation
- Overfitting, memorization, bias-variance trade-off

Taking the course

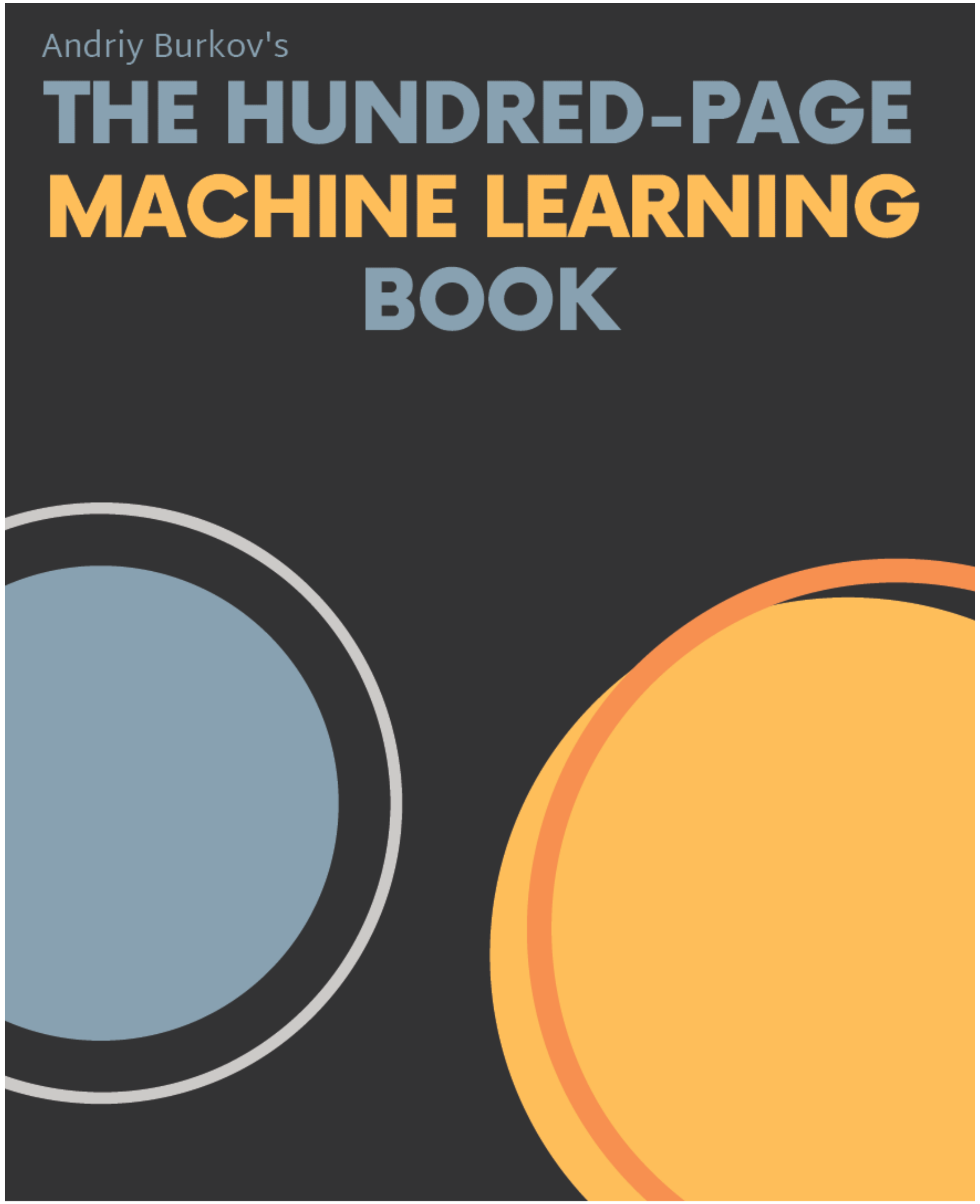
- Background
  - Programming, Python
  - Data structures and algorithms
  - Linear algebra, Calculus, Statistics
  - Fundamental deep learning models: MLP, CNN, RNN

- Fill out the following form until October 3 (Wednesday), midnight:

https://forms.gle/zyu7wUAFabVbxSGR6

- Read CH1-7 (pages 4-96) of the book
- Quiz next week

- Alternative:
  - Machine Learning - A First Course for Engineers and Scientists
  - https://smlbook.org/



Andriy Burkov's

THE HUNDRED-PAGE MACHINE LEARNING BOOK

BEFORE Deep Learning

History of deep learning

Biological neuron

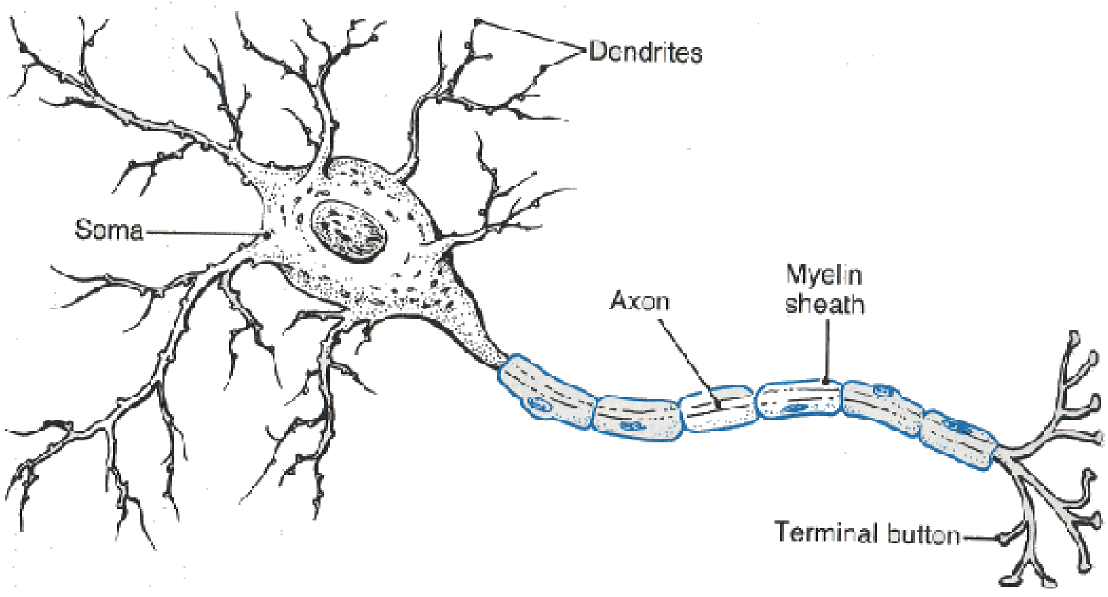
Artificial neuron

Perceptron learning

Neuron

A neuron

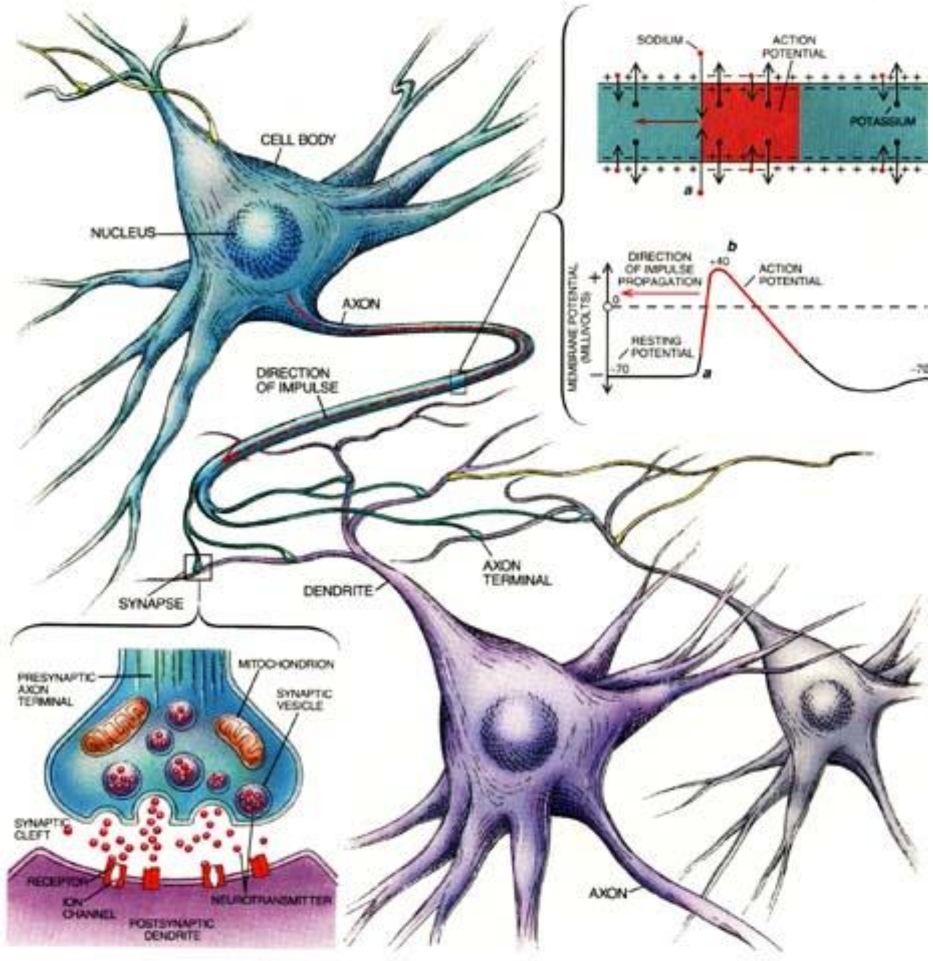
- receives input signals generated by other neurons through its dendrites,
- integrates these signals in its body,
- then generates its own signal (a series of electric pulses) that travel along the axon which in turn makes contacts with dendrites of other neurons.
- The points of contact between neurons are called synapses.



http://animatlab.com/Help/Documentation/Neural-Network-Editor/Neural-Simulation-Plug-ins/Firing-Rate-Neural-Plug-in/Neuron-Basics

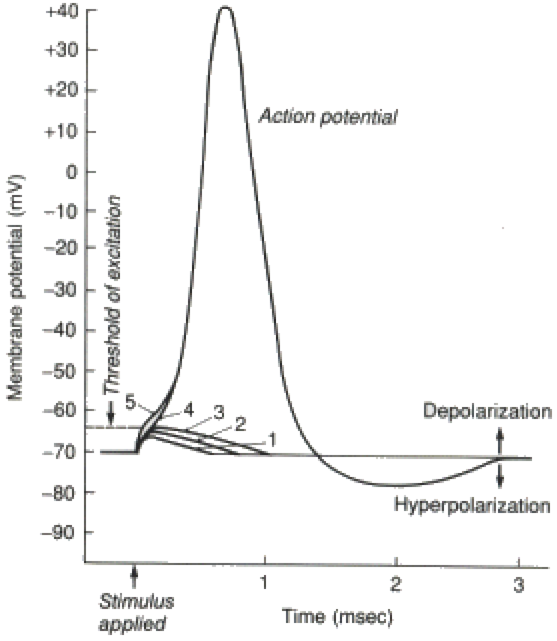
Neuron

- The pulses generated by the neuron travels along the axon as an electrical wave.
- Once these pulses reach the synapses at the end of the axon open up chemical vesicles exciting the other neuron.

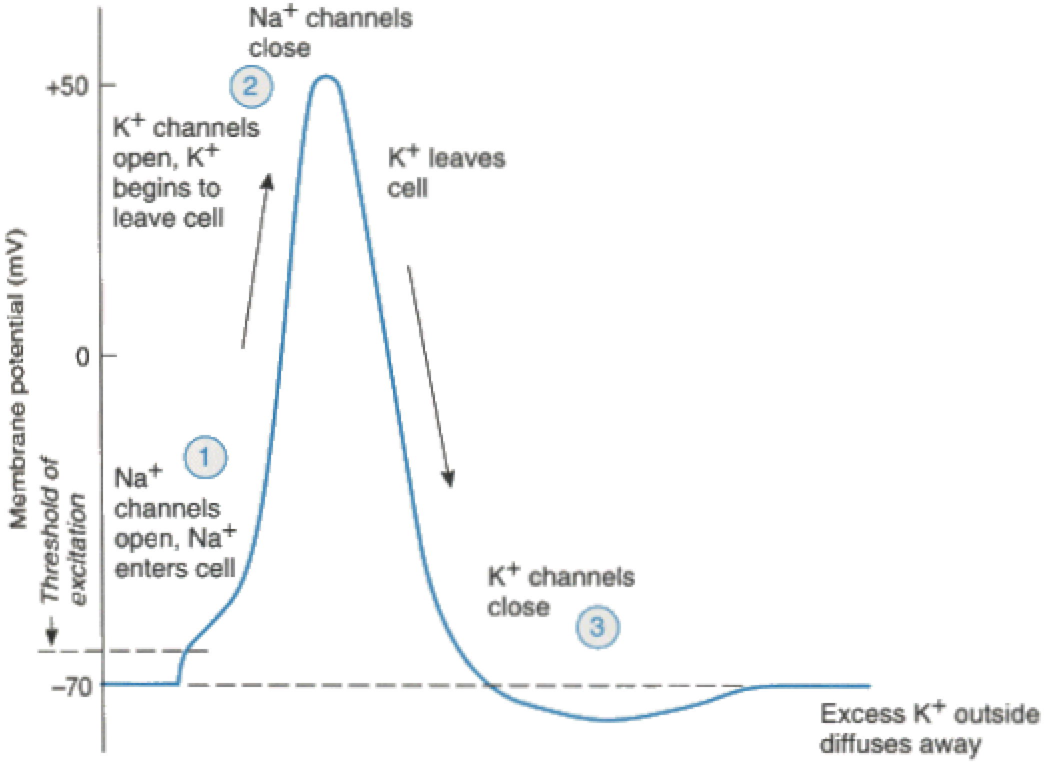


Slide credit: Erol Sahin

Neuron



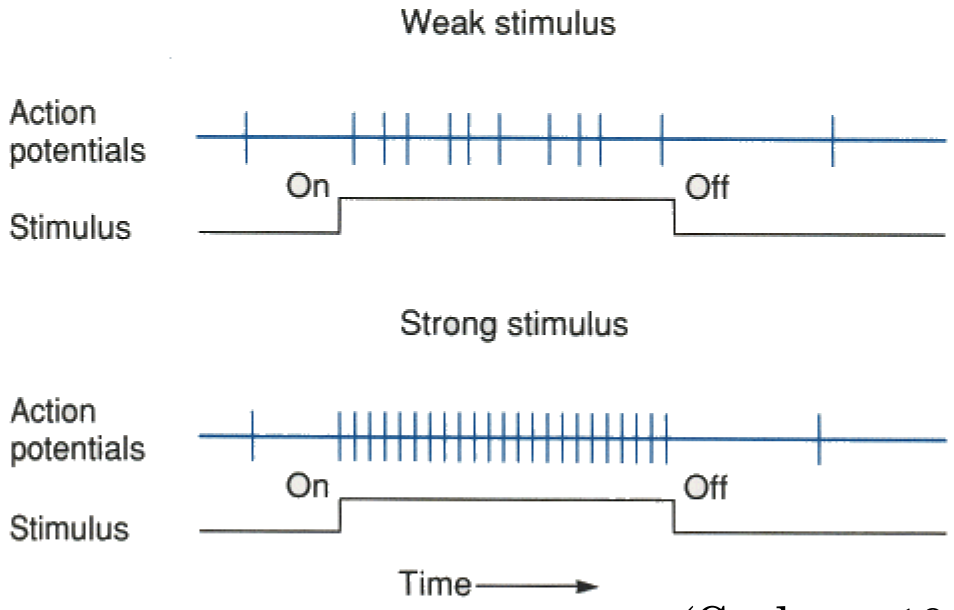
(Carlson, 1992)



(Carlson, 1992)

http://animatlab.com/Help/Documentation/Neural-Network-Editor/Neural-Simulation-Plug-ins/Firing-Rate-Neural-Plug-in/Neuron-Basics

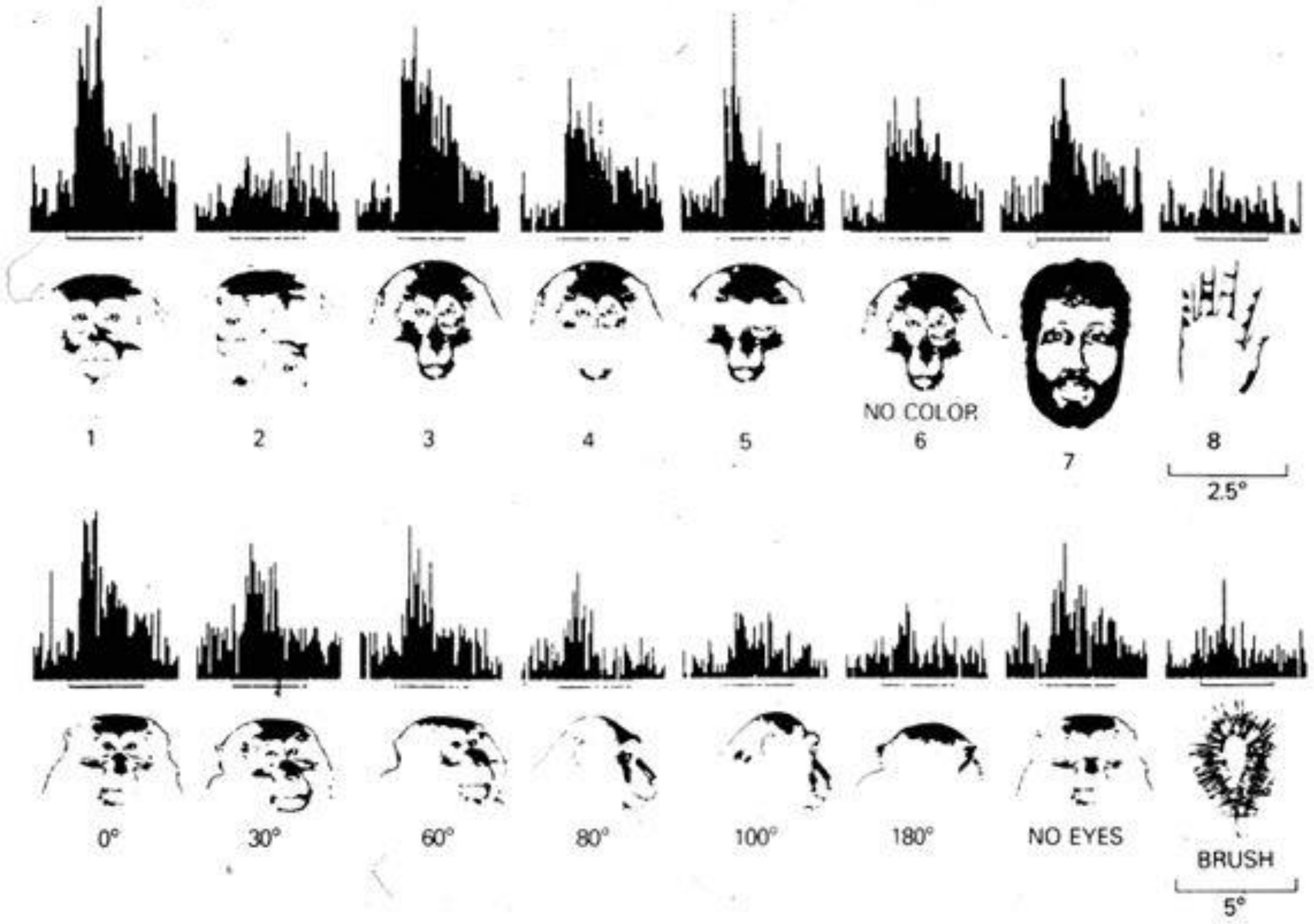
The biological neuron - 2



(Carlson, 1992)

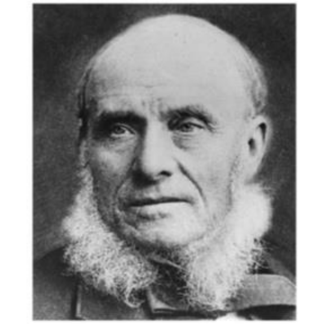
http://animatlab.com/Help/Documentation/Neural-Network-Editor/Neural-Simulation-Plug-ins/Firing-Rate-Neural-Plug-in/Neuron-Basics

Face selectivity in IT

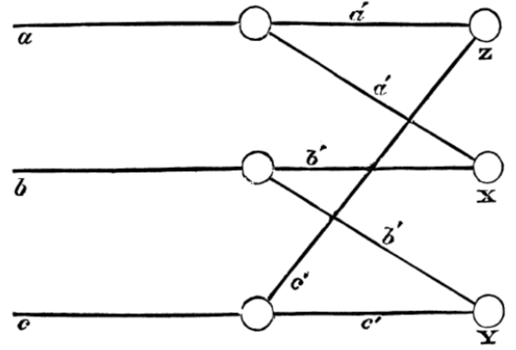


http://www.billconnelly.net/?p=291

Artificial neuron

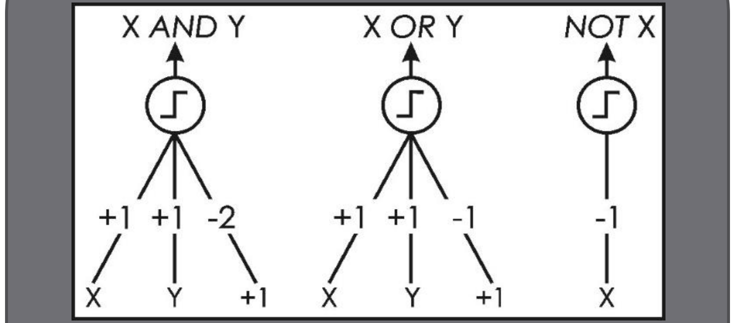


Alexander Bain
(1818 –1903)



“Bain on Neural Networks”, Wilkes & Wade, 1997.

McCulloch-Pitts Neuron



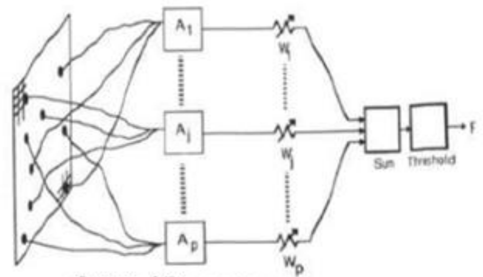
http://www.webpages.ttu.edu/dleverin/neural_network/neural_networks.html

$$net = \sum_i w_i x_i + b$$

$$f(net) = \begin{cases} 0, & net < 0 \\ 1, & net \geq 0 \end{cases}$$

Perceptron Learning

Frank Rosenblatt
(1928-1971)

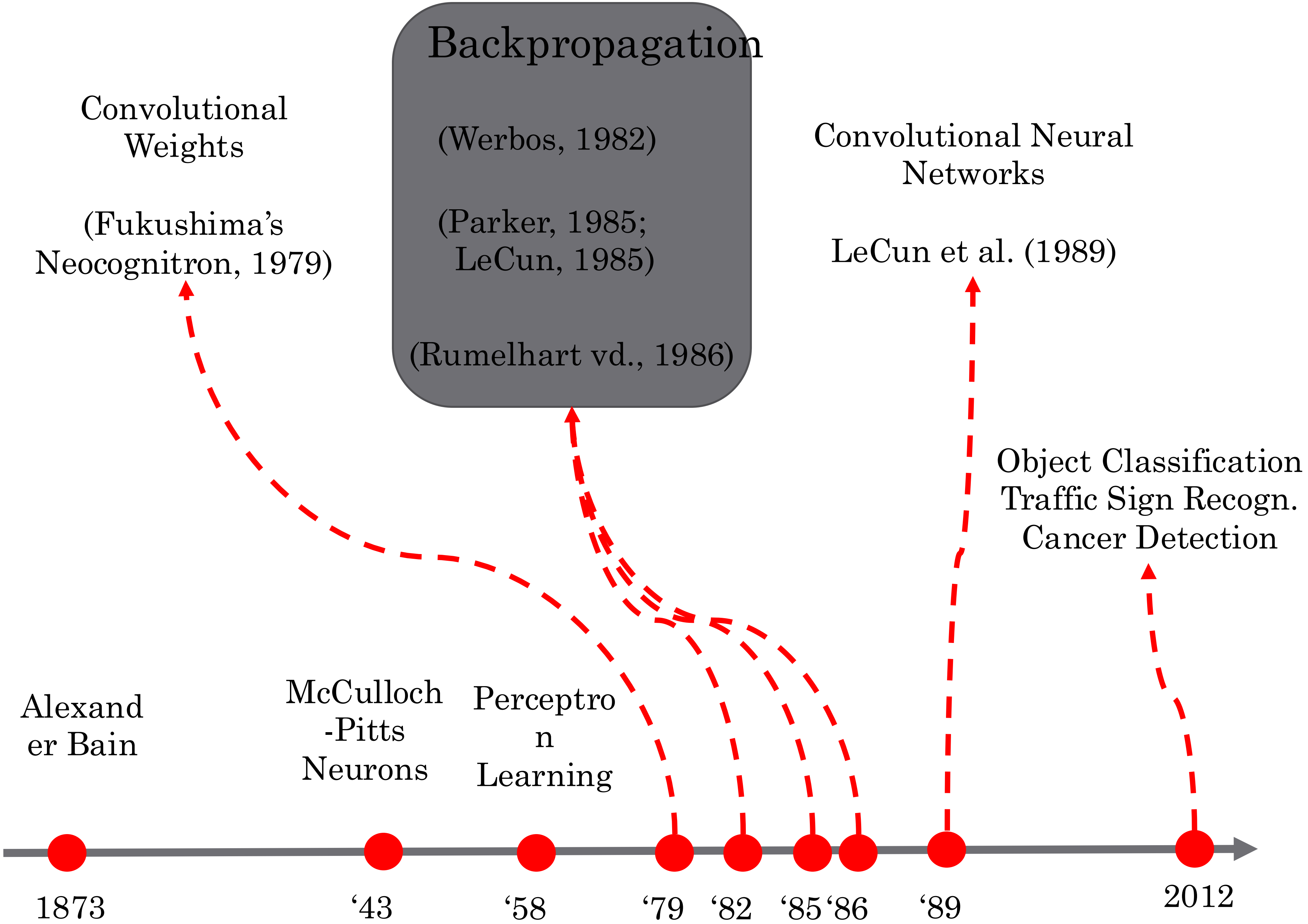


Original Perceptron
(From Perceptrons by M. L. Minsky and S. Papert, 1969, Cambridge, MA: MIT Press. Copyright 1969 by MIT Press.

1873

1943

1958



Backpropagation

(Werbos, 1982)

(Parker, 1985;
LeCun, 1985)

(Rumelhart vd., 1986)

Convolutional
Weights

(Fukushima's
Neocognitron, 1979)

Convolutional Neural
Networks

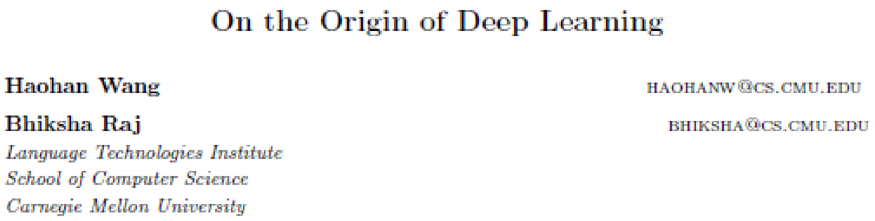
LeCun et al. (1989)

Object Classification
Traffic Sign Recogn.
Cancer Detection

Alexand
er Bain

McCulloch
-Pitts
Neurons

Perceptro
n
Learning



On the Origin of Deep Learning

Haohan Wang haohanw@cs.cmu.edu
Bhiksha Raj bhiksha@cs.cmu.edu
Language Technologies Institute
School of Computer Science
Carnegie Mellon University

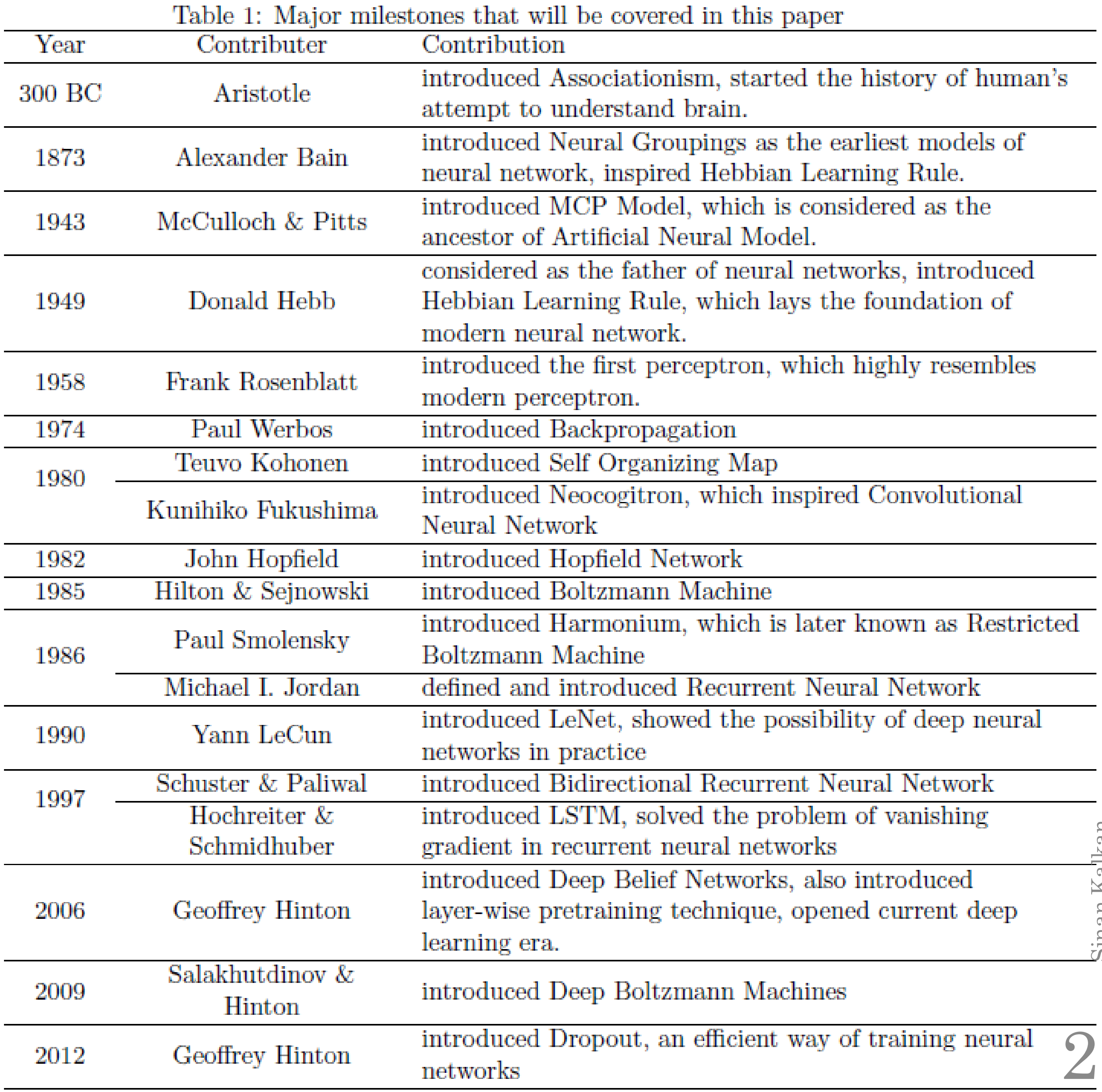
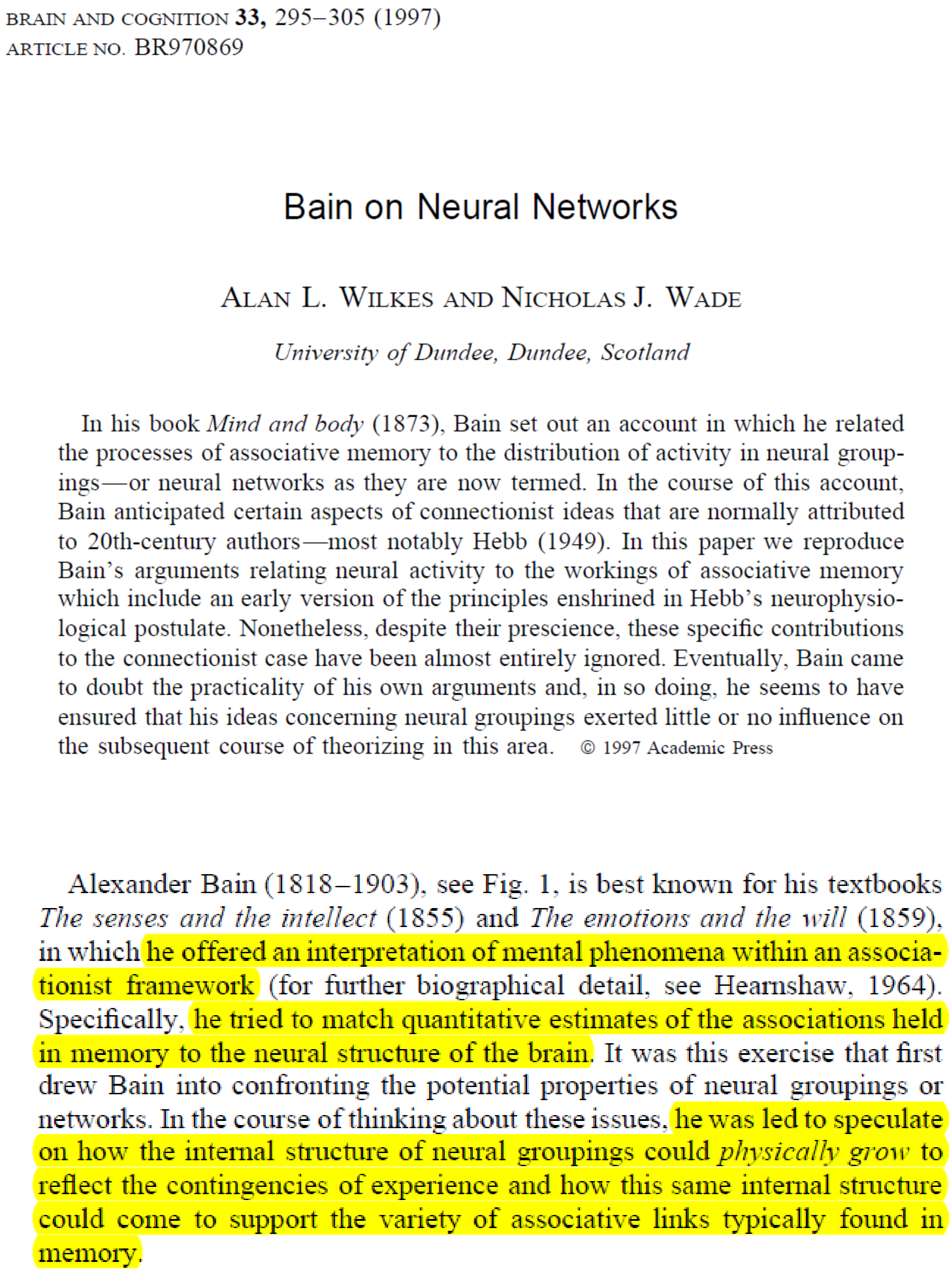


Table 1: Major milestones that will be covered in this paper

Year	Contributer	Contribution
300 BC	Aristotle	introduced Associationism, started the history of human's attempt to understand brain.
1873	Alexander Bain	introduced Neural Groupings as the earliest models of neural network, inspired Hebbian Learning Rule.
1943	McCulloch & Pitts	introduced MCP Model, which is considered as the ancestor of Artificial Neural Model.
1949	Donald Hebb	considered as the father of neural networks, introduced Hebbian Learning Rule, which lays the foundation of modern neural network.
1958	Frank Rosenblatt	introduced the first perceptron, which highly resembles modern perceptron.
1974	Paul Werbos	introduced Backpropagation
1980	Teuvo Kohonen	introduced Self Organizing Map
	Kunihiko Fukushima	introduced Neocogitron, which inspired Convolutional Neural Network
1982	John Hopfield	introduced Hopfield Network
1985	Hilton & Sejnowski	introduced Boltzmann Machine
1986	Paul Smolensky	introduced Harmonium, which is later known as Restricted Boltzmann Machine
	Michael I. Jordan	defined and introduced Recurrent Neural Network
1990	Yann LeCun	introduced LeNet, showed the possibility of deep neural networks in practice
1997	Schuster & Paliwal	introduced Bidirectional Recurrent Neural Network
	Hochreiter & Schmidhuber	introduced LSTM, solved the problem of vanishing gradient in recurrent neural networks
2006	Geoffrey Hinton	introduced Deep Belief Networks, also introduced layer-wise pretraining technique, opened current deep learning era.
2009	Salakhutdinov & Hinton	introduced Deep Boltzmann Machines
2012	Geoffrey Hinton	introduced Dropout, an efficient way of training neural networks



BRAIN AND COGNITION **33,** 295–305 (1997)
ARTICLE NO. BR970869

Bain on Neural Networks

ALAN L. WILKES AND NICHOLAS J. WADE

University of Dundee, Dundee, Scotland

In his book *Mind and body* (1873), Bain set out an account in which he related the processes of associative memory to the distribution of activity in neural groupings—or neural networks as they are now termed. In the course of this account, Bain anticipated certain aspects of connectionist ideas that are normally attributed to 20th-century authors—most notably Hebb (1949). In this paper we reproduce Bain's arguments relating neural activity to the workings of associative memory which include an early version of the principles enshrined in Hebb's neurophysiological postulate. Nonetheless, despite their prescience, these specific contributions to the connectionist case have been almost entirely ignored. Eventually, Bain came to doubt the practicality of his own arguments and, in so doing, he seems to have ensured that his ideas concerning neural groupings exerted little or no influence on the subsequent course of theorizing in this area.

Alexander Bain (1818–1903), see Fig. 1, is best known for his textbooks *The senses and the intellect* (1855) and *The emotions and the will* (1859), in which he offered an interpretation of mental phenomena within an associationist framework (for further biographical detail, see Hearnshaw, 1964). Specifically, he tried to match quantitative estimates of the associations held in memory to the neural structure of the brain. It was this exercise that first drew Bain into confronting the potential properties of neural groupings or networks. In the course of thinking about these issues, he was led to speculate on how the internal structure of neural groupings could *physically grow* to reflect the contingencies of experience and how this same internal structure could come to support the variety of associative links typically found in memory.

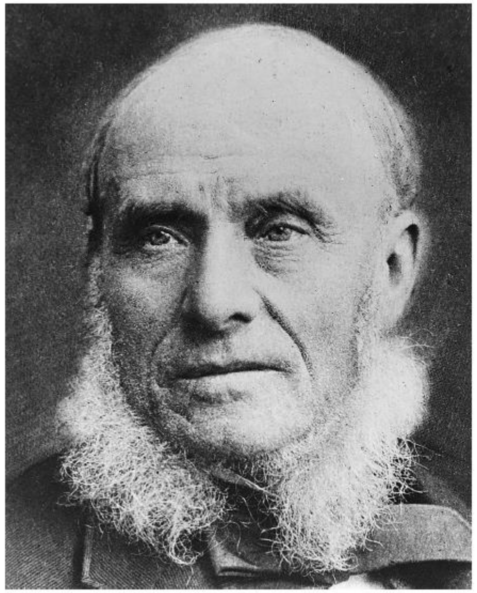


FIG. 1. Alexander Bain in 1892 from a photograph in his *Autobiography* (1904).

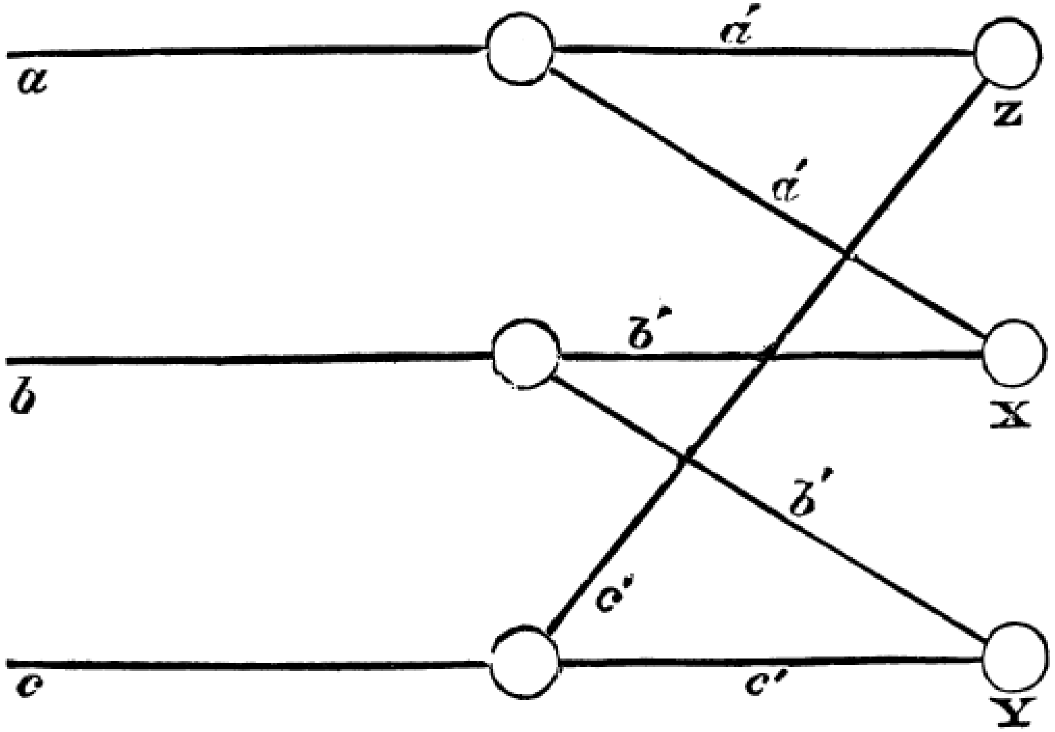


FIG. 2. Bain's diagram illustrating the way in which the connections in a neural network can channel activation in different directions:

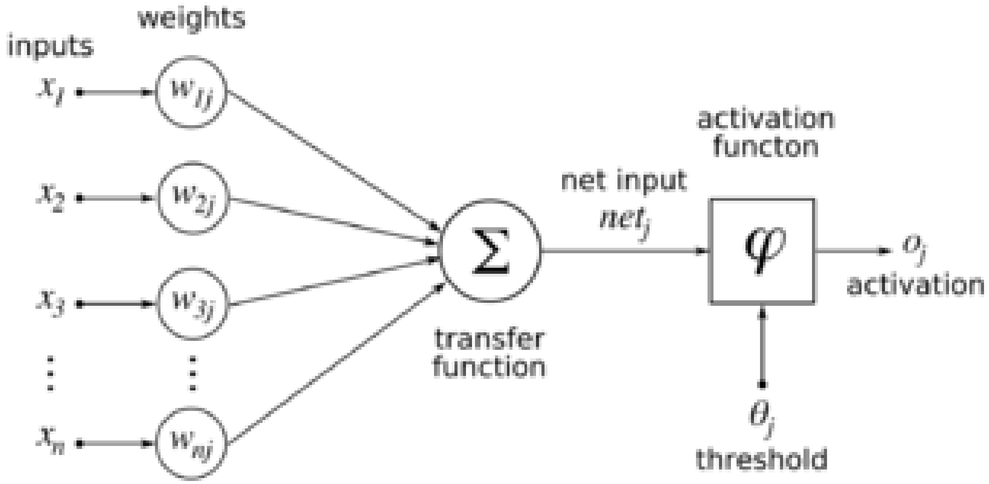
> It requires us to assume, not merely fibres multiplying by ramification through the cell junctions, but also an extensive arrangement of cross connections. We can typify it in this way. Suppose *a* enters a cell junction, and is replaced by several branches, a', a' etc; *b* in like manner, is multiplied into b', b' etc. Let one of the branches of *a* or a', pass into some second cell, and a branch of *b*, or b', pass into the same, and let one of the emerging branches be *X*, we have then a means of connecting united *a* and *b* with *X;* and in some other crossing, a branch of *b* may unite with a branch of *c*, from which crossing *Y* emerges and so on. . . . By this plan we comply with the primary condition of assigning a separate outcome to every different combination of sensory impressions.
>
> The diagram shows the arrangement. The fibre *a* branches into two a', a'; the fibre *b* into b', b'; c', c'. One of the branches of *a* unites with one of the branches of *b*, or a', b' in a cell *X;* b', c' unite in *Y;* a', c' in *Z*. (1873, pp. 110, 111)

McCulloch-Pitts Neuron (McCulloch & Pitts, 1943)

- Binary input-output
- Can represent Boolean functions.
- No training.

$$net = \sum_i (w_{y,x_i} x_i) + w_{y,x_b}$$

$$f(net) = \begin{cases} 0, & net < 0 \\ 1, & net \geq 0 \end{cases}$$

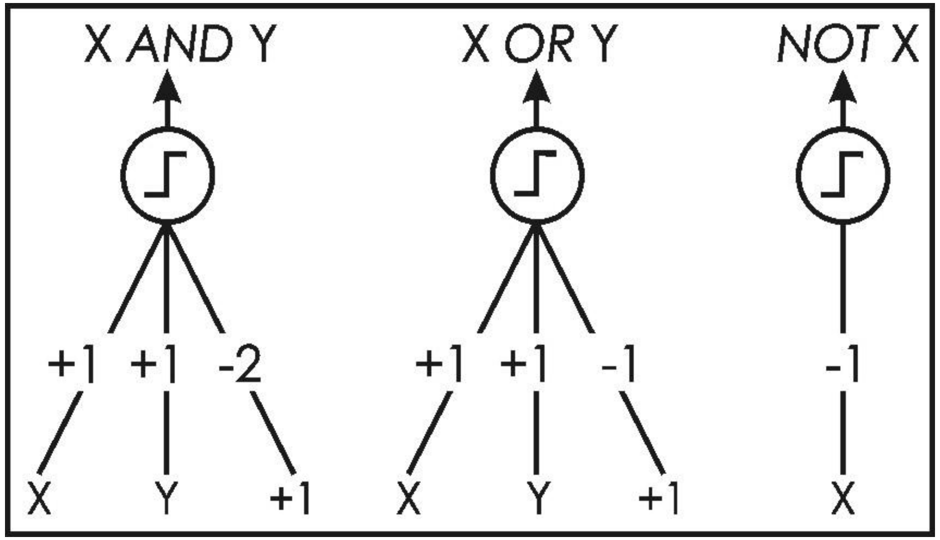


http://www.tau.ac.il/~tsirel/dump/Static/knowino.org/wiki/Artificial_neural_network.html

McCulloch-Pitts Neuron

- Implement AND(x, y):
  - Let w_x and w_y to be 1, and w_{+1} to be -2.
- When input is 1 & 1; net is 0.
- When one input is 0; net is -1.
- When input is 0 & 0; net is -2.

$$f(net) = \begin{cases} 0, & net < 0 \\ 1, & net \geq 0 \end{cases}$$



http://www.webpages.ttu.edu/dleverin/neural_network/neural_networks.html

http://www.tau.ac.il/~tsirel/dump/Static/knowino.org/wiki/Artificial_neural_network.html

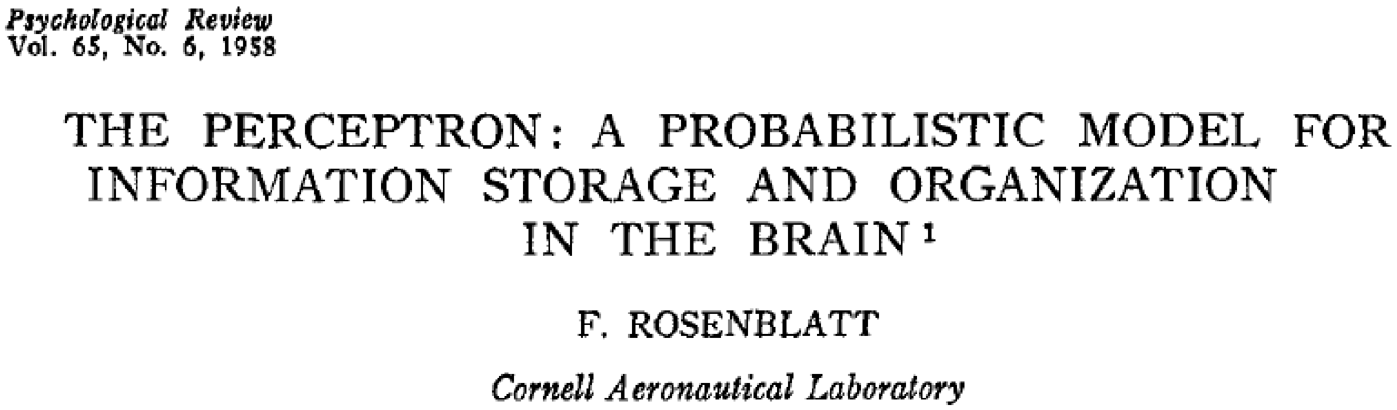
McCulloch-Pitts Neuron

- Binary input-output is a big limitation
- Also called

  “[…] *caricature models since they are intended to reflect one or more neurophysiological observations, but without regard to realism* […]”

  -- Wikipedia

- No training! No learning!
- They were useful in inspiring research into connectionist models



Psychological Review
Vol. 65, No. 6, 1958

THE PERCEPTRON: A PROBABILISTIC MODEL FOR INFORMATION STORAGE AND ORGANIZATION IN THE BRAIN[1]

F. ROSENBLATT

Cornell Aeronautical Laboratory

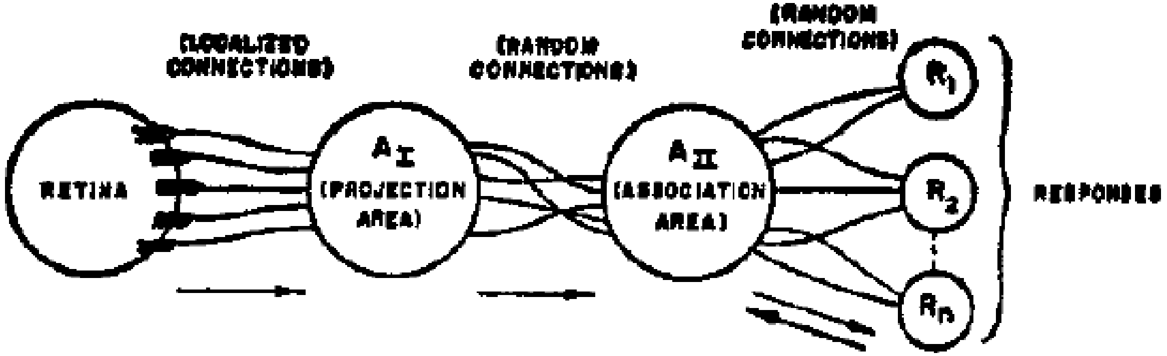
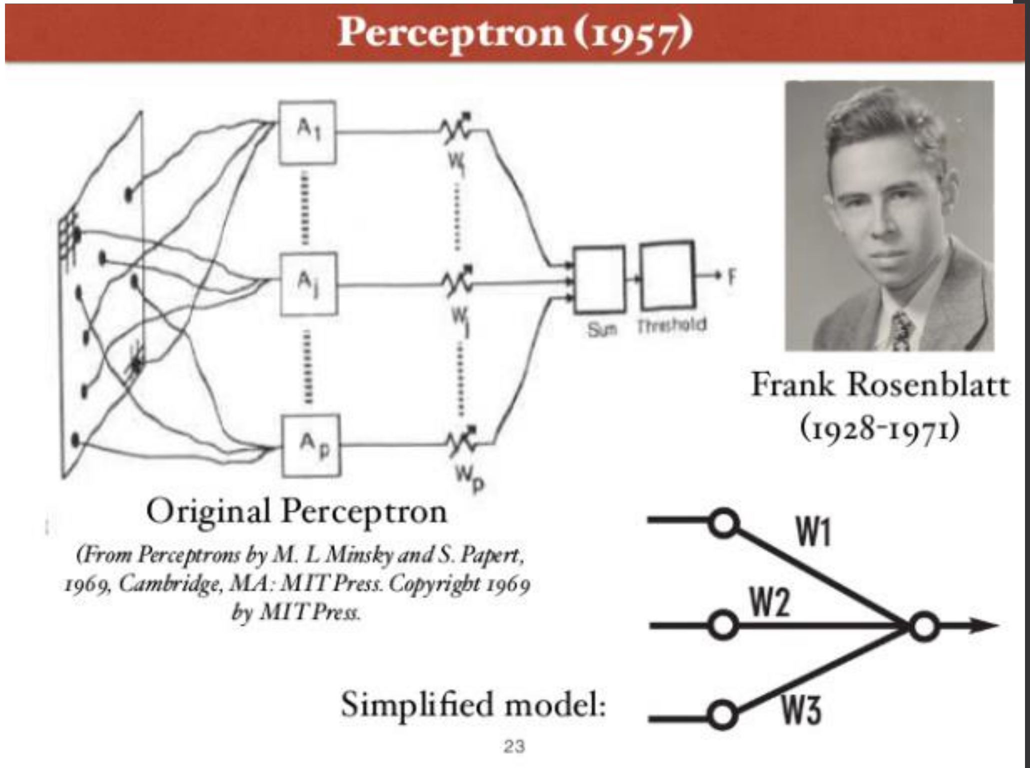


FIG. 1. Organization of a perceptron.



Perceptron (1957)

Frank Rosenblatt
(1928-1971)

Original Perceptron

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

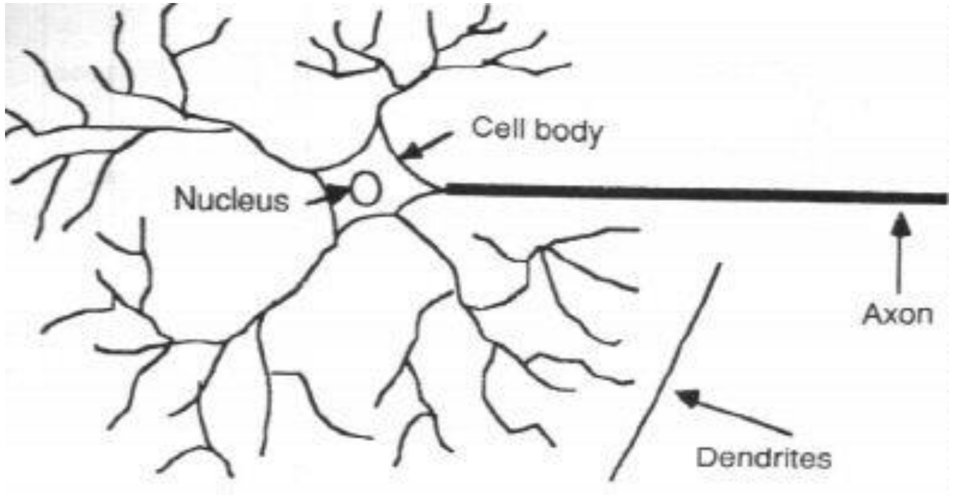
Simplified model:

23

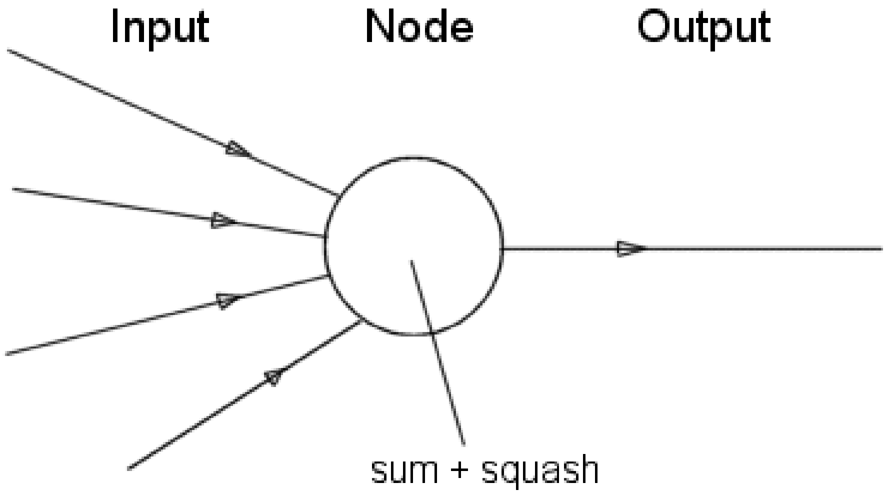
https://www.youtube.com/watch?v=cNxadbrN_aI

Let’s go back to a biological neuron

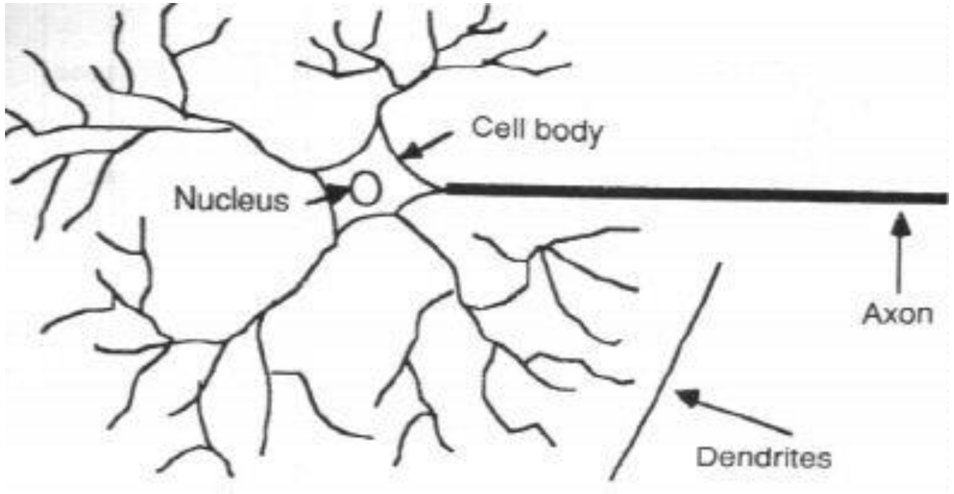
- A biological neuron has:
  - Dendrites
  - Soma
  - Axon

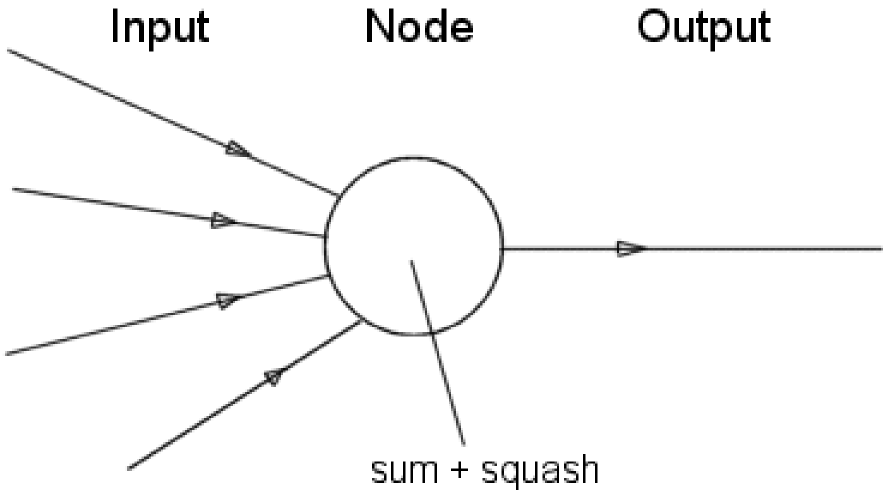


- Firing is continuous, unlike most artificial neurons
- Rather than the response value, the firing rate is critical



- Neurone vs. Node





- Very crude abstraction
- Many details overseen

“Spherical cow” problem!

Spherical cow

Q: How does a physicist milk a cow?
A: Well, first let us consider a spherical cow...

Or

https://en.wikipedia.org/wiki/Spherical_cow

“Milk production at a dairy farm was low, so the farmer wrote to the local university, asking for help from academia. A multidisciplinary team of professors was assembled, headed by a theoretical physicist, and two weeks of intensive on-site investigation took place. The scholars then returned to the university, notebooks crammed with data, where the task of writing the report was left to the team leader. Shortly thereafter the physicist returned to the farm, saying to the farmer, "I have the solution, but it only works in the case of spherical cows in a vacuum".”

https://www.washingtonpost.com/news/wonk/wp/2013/09/04/the-coase-theorem-is-widely-cited-in-economics-ronald-coase-hated-it/

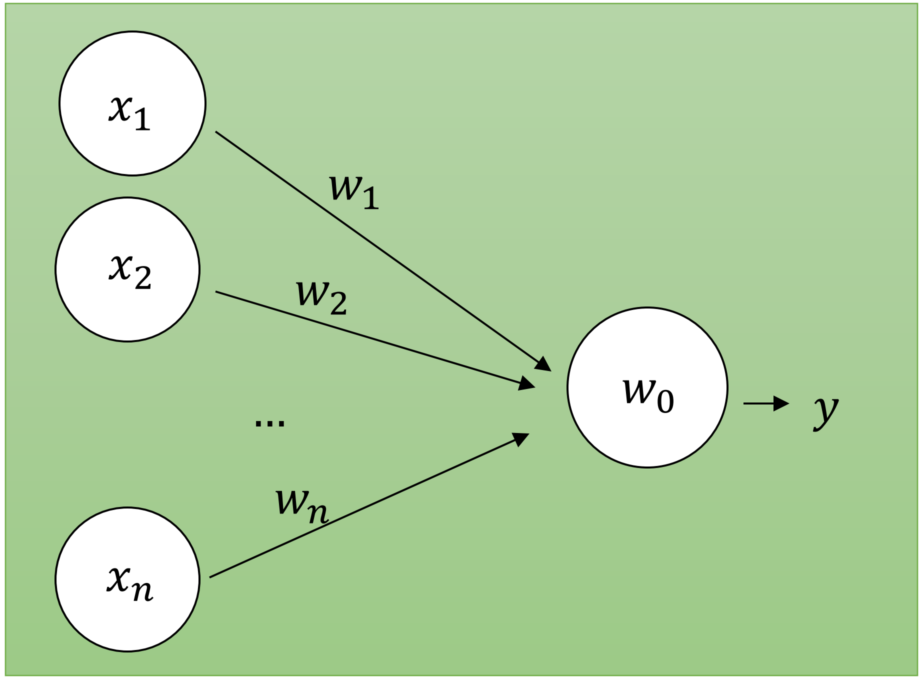
More on this

- https://medium.com/intuitionmachine/neurons-are-more-complex-than-what-we-have-imagined-b3dd00a1dcd3

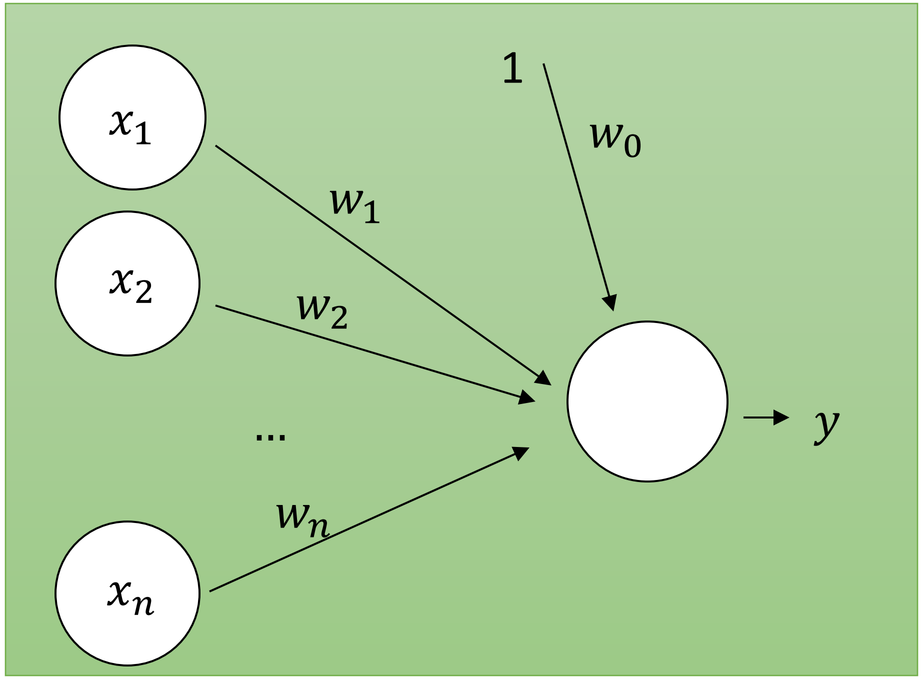
Perceptron Learning

Let us take a closer look at perceptrons

- Initial proposal of connectionist networks
- Rosenblatt, 50’s and 60’s
- Essentially a linear model composed of nodes and weights, followed by a non-linear thresholding operation



or



Activation Function

$$y(\mathbf{x}) = \begin{cases} 1, & w_0 + w_1 x_1 + .. w_n x_n > 0 \\ 0, & \text{otherwise} \end{cases}$$

Or, simply

$$y(\mathbf{x}) = \operatorname{sgn}(\mathbf{w} \cdot \mathbf{x})$$

where $\operatorname{sgn}(x) = \begin{cases} 0, & x \leq 0 \\ 1, & x > 0 \end{cases}$

Motivation for perceptron learning

(No gradient descent yet)

- We have estimated an output $\hat{y} = \mathrm{sgn}(\mathbf{w} \cdot \mathbf{x})$
  - But the target was y
- Error (simply): $y - \hat{y}$
- Let us update each weight such that we “learn” from the error:
  - $w_i \leftarrow w_i + \Delta w_i$
  - where $\Delta w_i \propto (y - \hat{y})$
- We somehow need to distribute the error to the weights. How?
  - Distribute the error according to how much they contributed to the error: Bigger input contributes more to the error.
  - Therefore: $\Delta w_i \propto (y - \hat{y}) x_i$

An example

- Consider $x_i = 0.8, y = 1, \hat{y} = -1$
  - Then, $(y - \hat{y})x_i = 1.6$
  - Which will increase weight w_i by 1.6.
  - Which makes sense considering the output and the target

Perceptron training rule

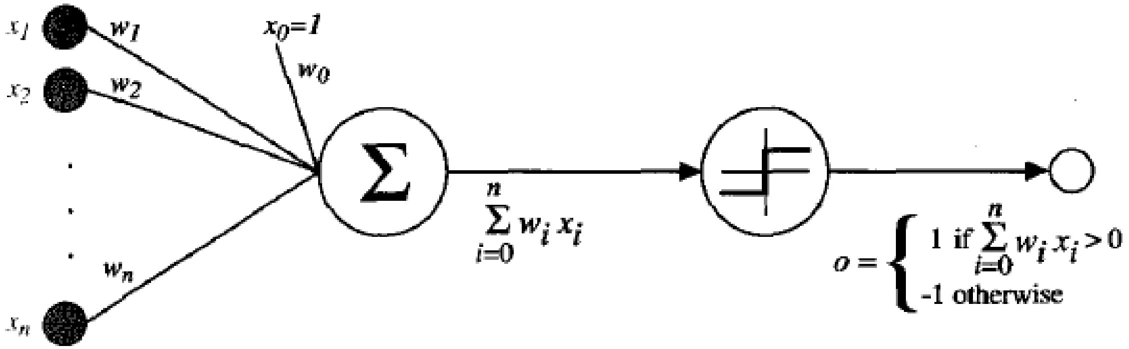
- Update weights

$$w_i \leftarrow w_i + \Delta w_i$$

- How to determine Δw_i?

$$\Delta w_i \leftarrow \eta(y - \hat{y})x_i$$

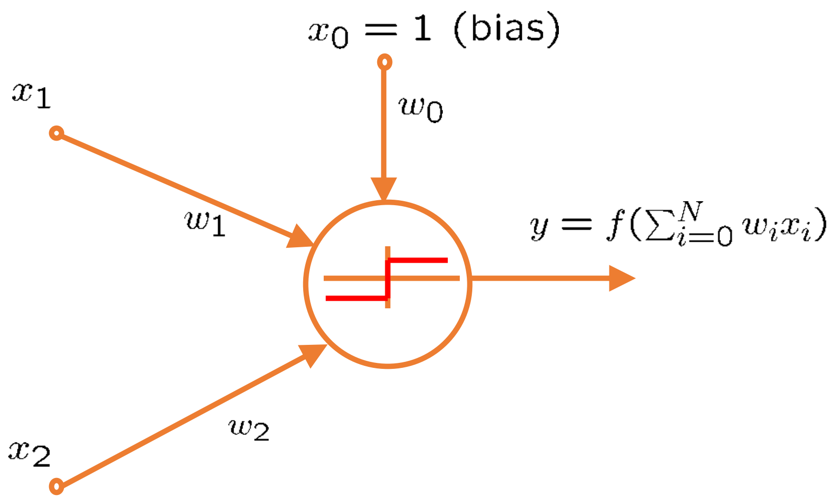
  - η: learning rate – can be slowly decreased
  - y: target/desired output
  - $\hat{y}$: current output, prediction

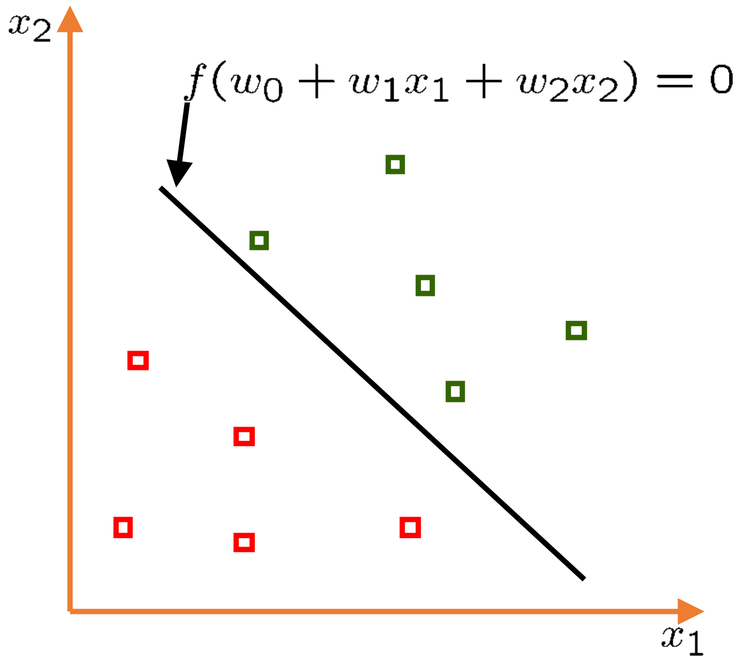


x_1
x_2
x_n
w_1
w_2
w_n
$x_0=1$
w_0
Σ
$\sum_{i=0}^{n} w_i x_i$
$o = \begin{cases} 1 \text{ if } \sum_{i=0}^{n} w_i x_i > 0 \\ -1 \text{ otherwise} \end{cases}$

Perceptron - intuition

- A perceptron defines a hyperplane in N-1 space: a line in 2-D (two inputs), a plane in 3-D (three inputs),….
- The perceptron is a linear classifier: It’s output is -1 on one side of the plane, and 1 for the other.
- Given a linearly separable problem, the perceptron learning rule guarantees convergence.





Sinan Kalkan

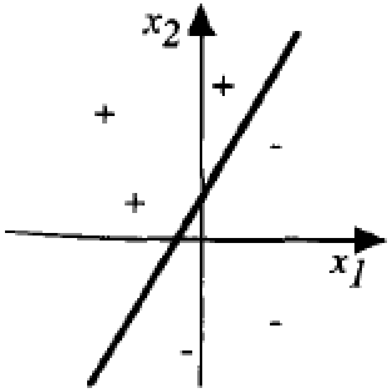
Slide credit: Erol Sahin

Problems with perceptron

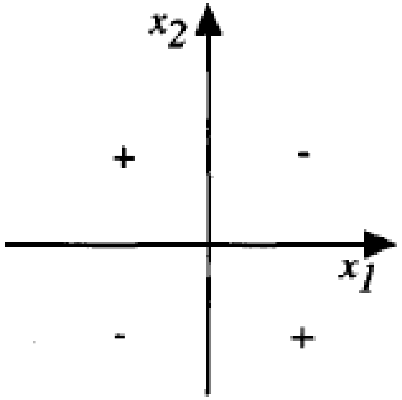
- Perceptron unit is non-linear. However, it provides zero gradient (due to thresholding function), which makes it unsuitable to gradient descent in multi-layer networks.

Problems with perceptron learning

- Can only learn linearly separable classification.



linearly separable



not linearly separable

Towards deep learning

Linear classification/regression

Non-linear classification/regression

Multi-layer perceptrons

Linear classification and regression

Linear Classification

- Goal: Find the following mapping, given the training set $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{N}$:

$$\mathbf{y} = f(\mathbf{x})$$

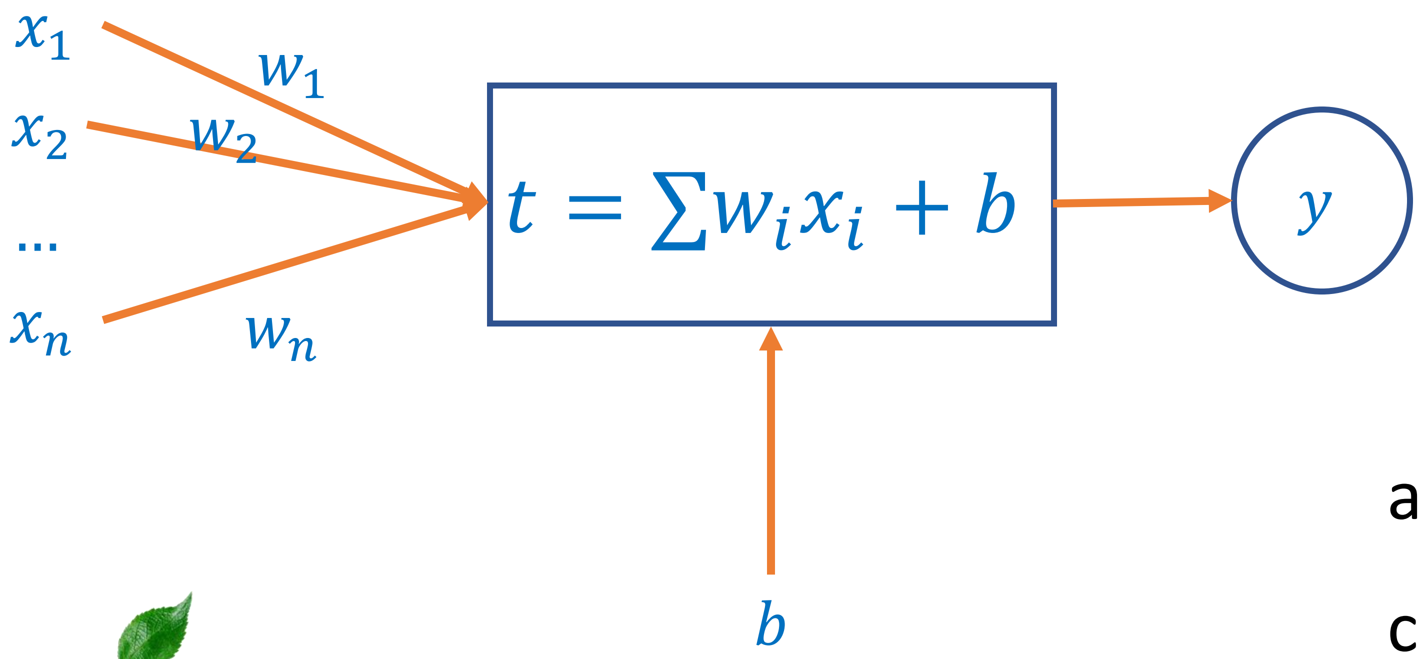
$$f: \mathbb{X} \rightarrow \mathbb{Y}.$$

- Linear model:

$$\mathbf{y} \approx f(\mathbf{x}) \equiv f(\mathbf{x}; W, b) = W\mathbf{x} + b = \sum_{i=1}^{N} w_i x_i + b,$$

where w_i $(i = 1, \ldots, N)$ and b are parameters to be learned.

Linear Classification with Neurons



apple

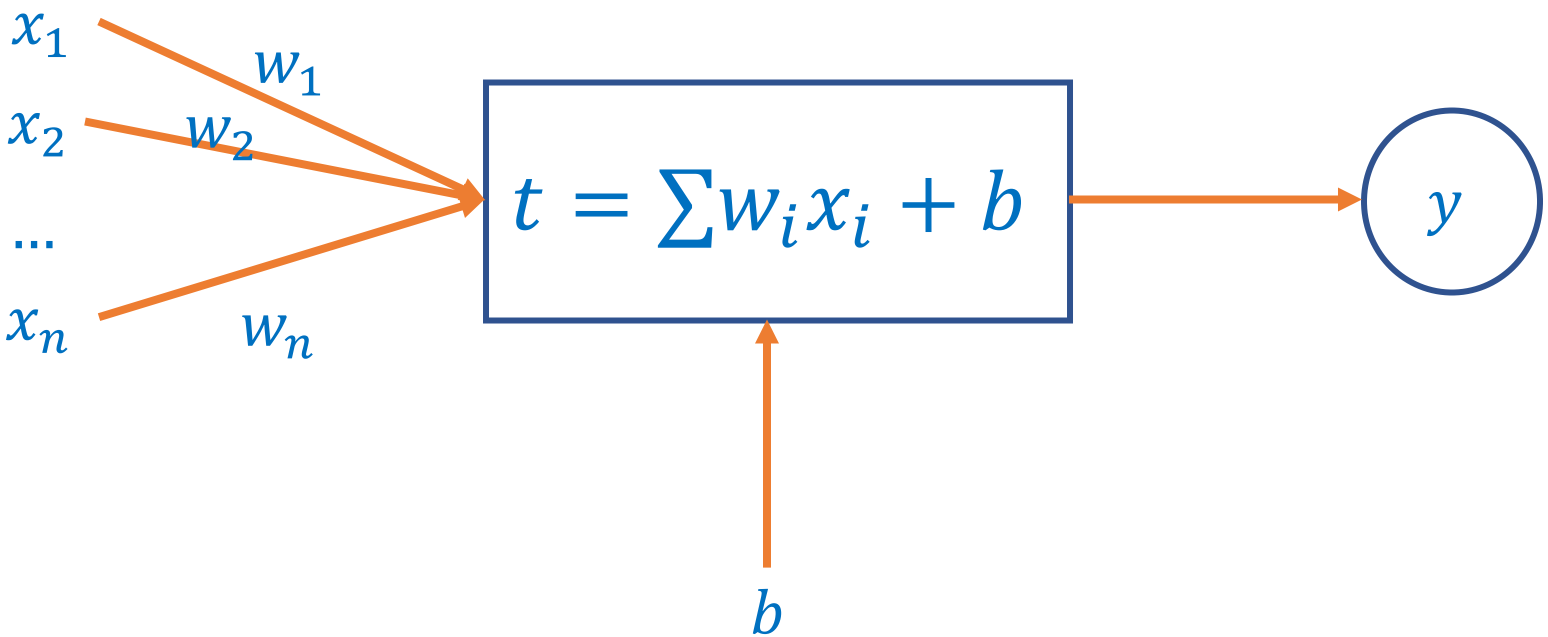
car

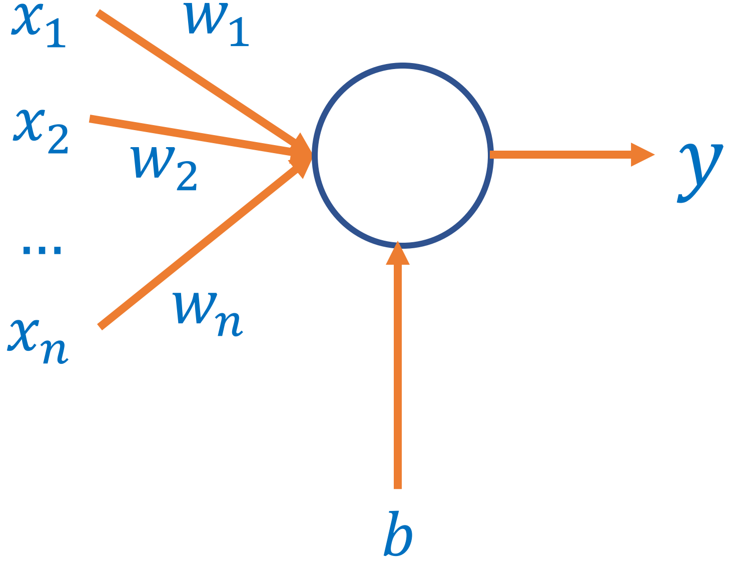
house

person

...

Linear Classification with Neurons



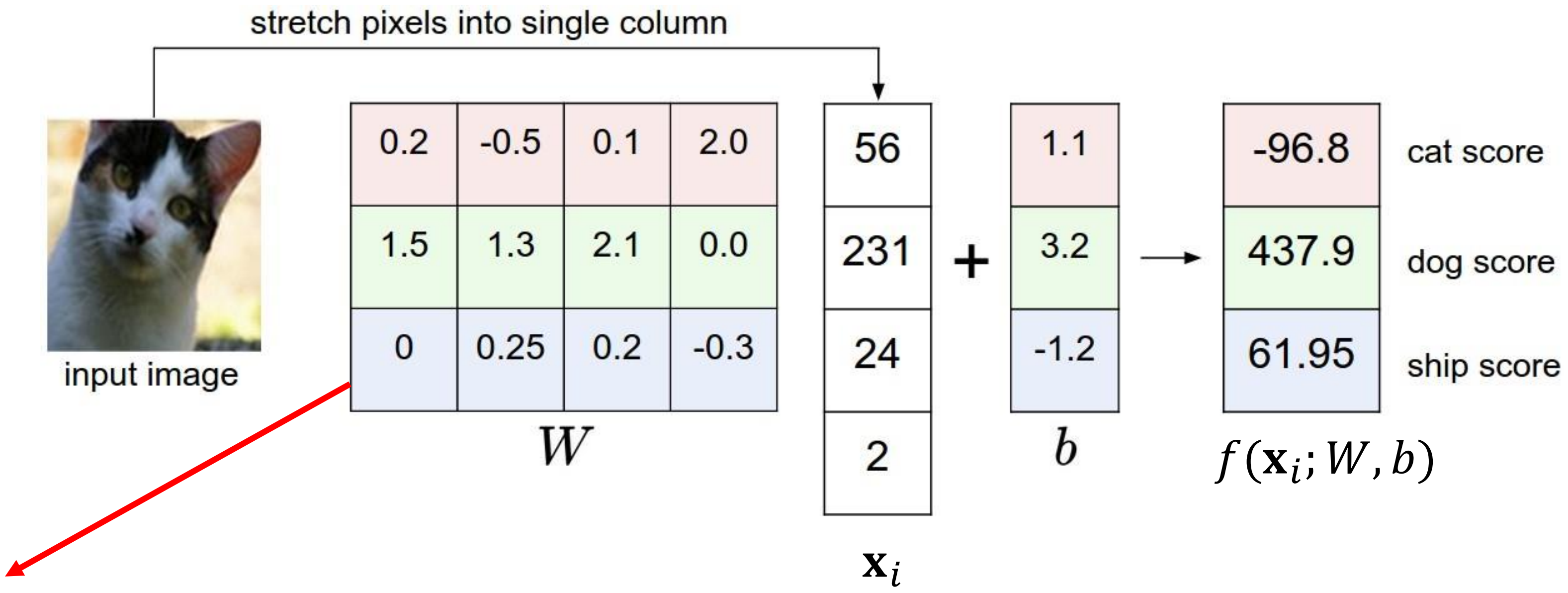


$$\boldsymbol{y} \approx f(\mathbf{x}) \equiv f(\mathbf{x}; \boldsymbol{\theta})$$
$$= \sum_i w_i x_i + b = \mathbf{w} \cdot \mathbf{x}$$

Linear classification

For class j: $f_j = f(\mathbf{x}; W, b)_{\boldsymbol{j}} = \boldsymbol{w}_j \cdot \mathbf{x} + b_j = \sum_{i=1}^{N} w_{ji} x_i + b_j$

For all classes: $f(\mathbf{x}; W, b) = W\mathbf{x} + b$



$$\underbrace{\begin{bmatrix} 0.2 & -0.5 & 0.1 & 2.0 \\ 1.5 & 1.3 & 2.1 & 0.0 \\ 0 & 0.25 & 0.2 & -0.3 \end{bmatrix}}_{W} \underbrace{\begin{bmatrix} 56 \\ 231 \\ 24 \\ 2 \end{bmatrix}}_{\mathbf{x}_i} + \underbrace{\begin{bmatrix} 1.1 \\ 3.2 \\ -1.2 \end{bmatrix}}_{b} \longrightarrow \underbrace{\begin{bmatrix} -96.8 \\ 437.9 \\ 61.95 \end{bmatrix}}_{f(\mathbf{x}_i; W, b)}$$

One row per class

Figure: http://cs231n.github.io/linear-classify/

Linear classification: One interpretation

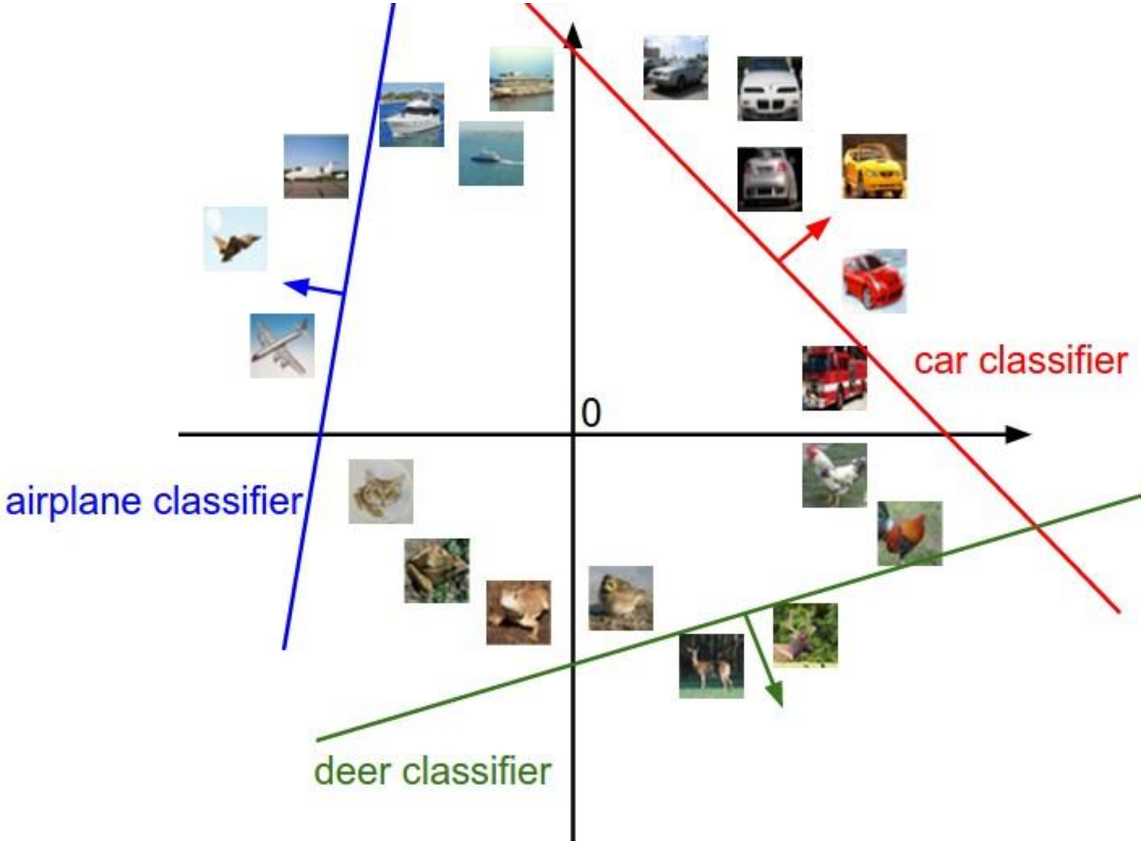


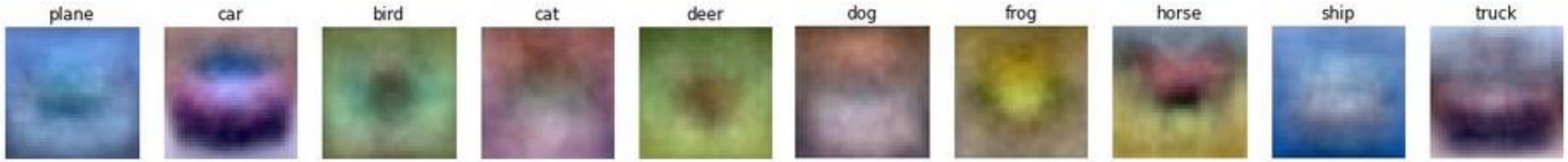
Figure: http://cs231n.github.io/linear-classify/

$$f(\mathbf{x}_i; W, b) = W\mathbf{x}_i + b$$

Interpretation: Each row of W and b describes a line for a class

Linear classification: Another interpretation

- Each row in W can be interpreted as a template of that class.
  - $f(\mathbf{x}_i; W, b) = W\mathbf{x}_i + b$ calculates the inner product to find which template best fits $\mathbf{x}_i$.
  - Effectively, we are doing Nearest Neighbor with the "prototype" images of each class.

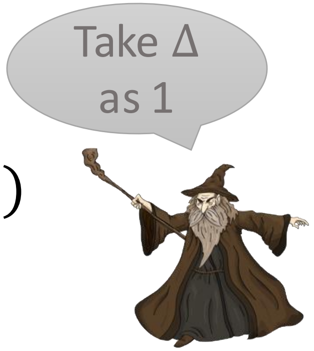


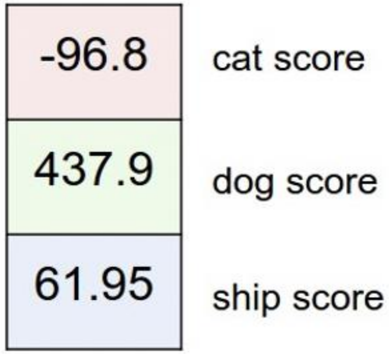
http://cs231n.github.io/linear-classify/

Loss function

- A function which measures how good our parameters (weights) are.
  - Other names: cost function, objective function
- Let $s_j = f(\mathbf{x}_i; W)_j$
- An example loss function:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + \Delta)$$





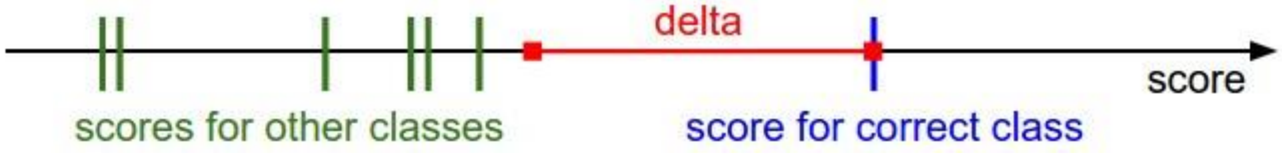
-96.8	cat score
437.9	dog score
61.95	ship score

$f(\mathbf{x}_i; W, b)$

Or equivalently:

$$L_i = \sum_{j \neq y_i} \max(0, \mathbf{w}_j^T \mathbf{x}_i - \mathbf{w}_{y_i}^T \mathbf{x}_i + \Delta)$$

- This forces the distances to other classes to be more than Δ (the margin)



http://cs231n.github.io/linear-classify/

Example

- Consider our scores for $\mathbf{x}_i$ to be $s = [13, -7, 11]$ and assume Δ as 10.
- Then,

$$L_i = \max(0, -7 - 13 + 10) + \max(0, 11 - 13 + 10)$$

http://cs231n.github.io/linear-classify/

Regularization

- In practice, there are many possible solutions leading to the same loss value.
  - Based on the requirements of the problem, we might want to penalize certain solutions.
- E.g.,

$$R(W) = \sum_i \sum_j W_{i,j}^2$$

  - which penalizes large weights.
    - Why do we want to do that?

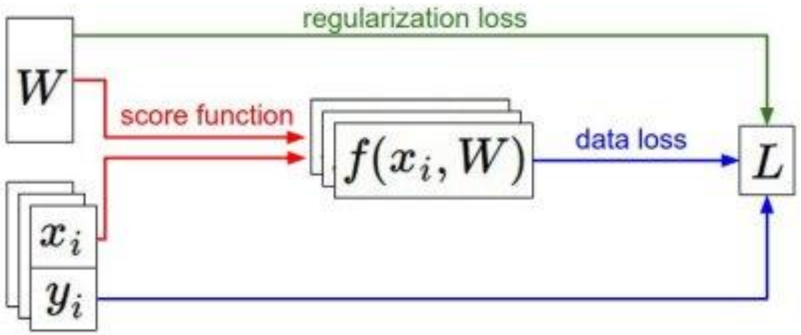
http://cs231n.github.io/linear-classify/

Why penalize large weights?

$$R(W) = \sum_i \sum_j W_{i,j}^2$$

- The solution is not unique: W is a solution, so is $\propto W$.
- Large W has large variance:
    - Large and small weights can lead to abrupt changes in the boundary.
    - I.e. overfitting.
- Robustness to small changes in the input.

Combined Loss Function



- The loss function becomes:

$$L = \underbrace{\frac{1}{N}\sum_i L_i}_{\text{data loss}} + \underbrace{\lambda R(W)}_{\text{regularization loss}}$$

- If you expand it:

$$L = \frac{1}{N}\sum_i \sum_{j\neq y_i}\left[\max\left(0, f(\mathbf{x}_i, W)_j - f(\mathbf{x}_i, W)_{y_i} + \Delta\right)\right] + \lambda \sum_i \sum_j W_{i,j}^2$$

Hyper parameters
(estimated using validation set)

http://cs231n.github.io/linear-classify/

Hinge Loss, or Max-Margin Loss

$$L = \frac{1}{N}\sum_{i}\sum_{j\neq y_i}\left[\max\left(0, f(\mathbf{x}_i, W)_j - f(\mathbf{x}_i, W)_{y_i} + \Delta\right)\right] + \lambda \sum_{i}\sum_{j} W_{i,j}^2$$

http://cs231n.github.io/linear-classify/

Regression loss

$$L = \frac{1}{N}\sum_i \sum_j \left(s_{ij} - y_{ij}\right)^2 + \lambda \sum_i \sum_j w_{i,j}^2$$

In general:

$$\sum_j \left|s_j - y_j\right|^q$$

- $q = 1$: Absolute Value Loss
- $q = 2$: Square Error Loss.

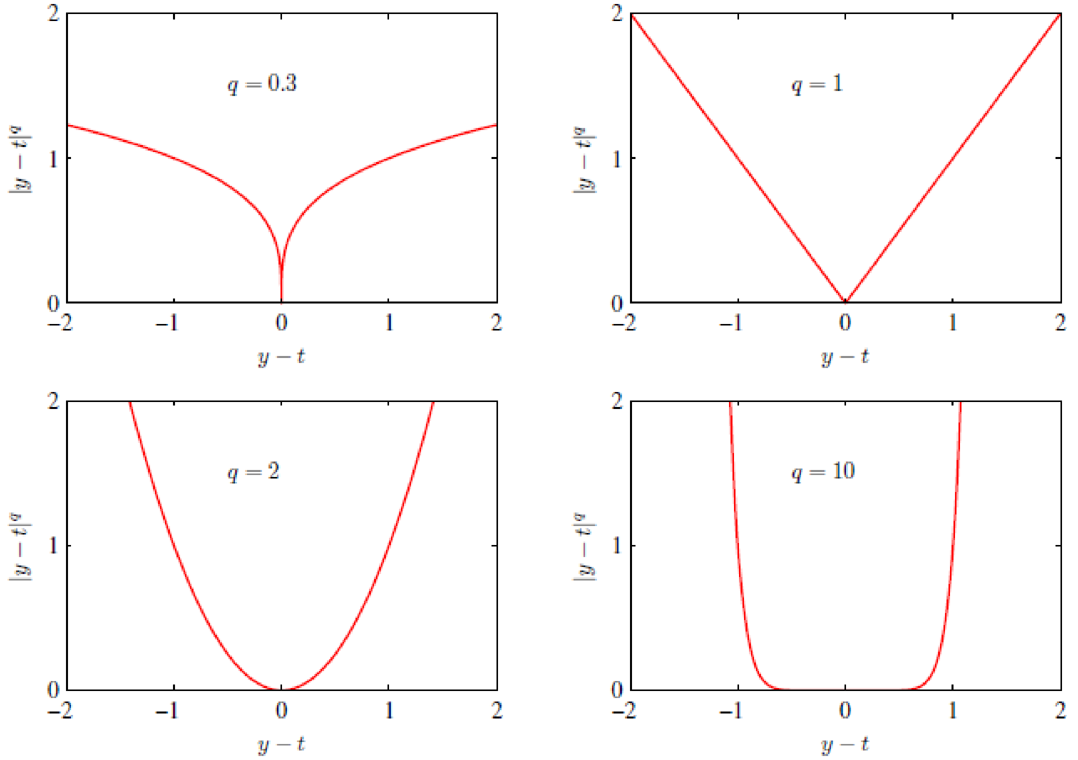
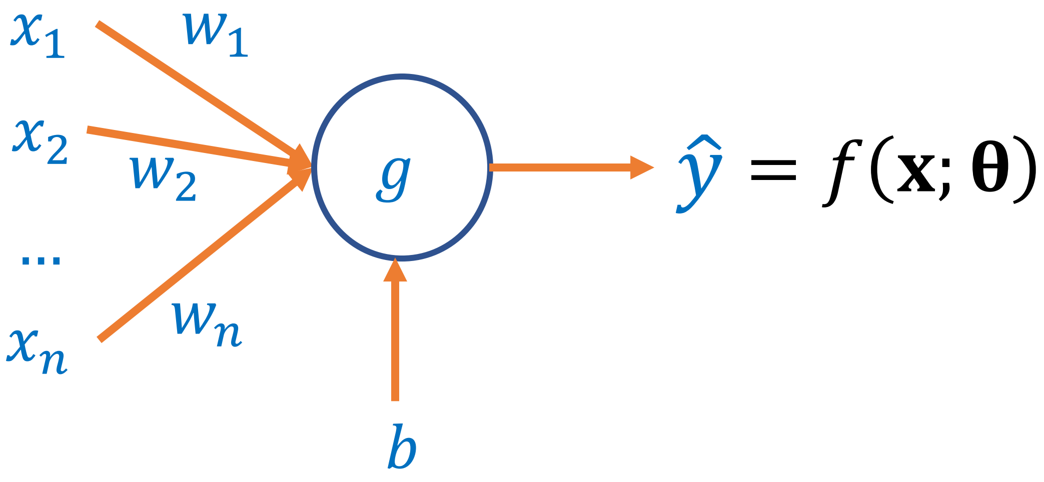


Figure 1.29 Plots of the quantity $L_q = |y - t|^q$ for various values of q.

Bishop

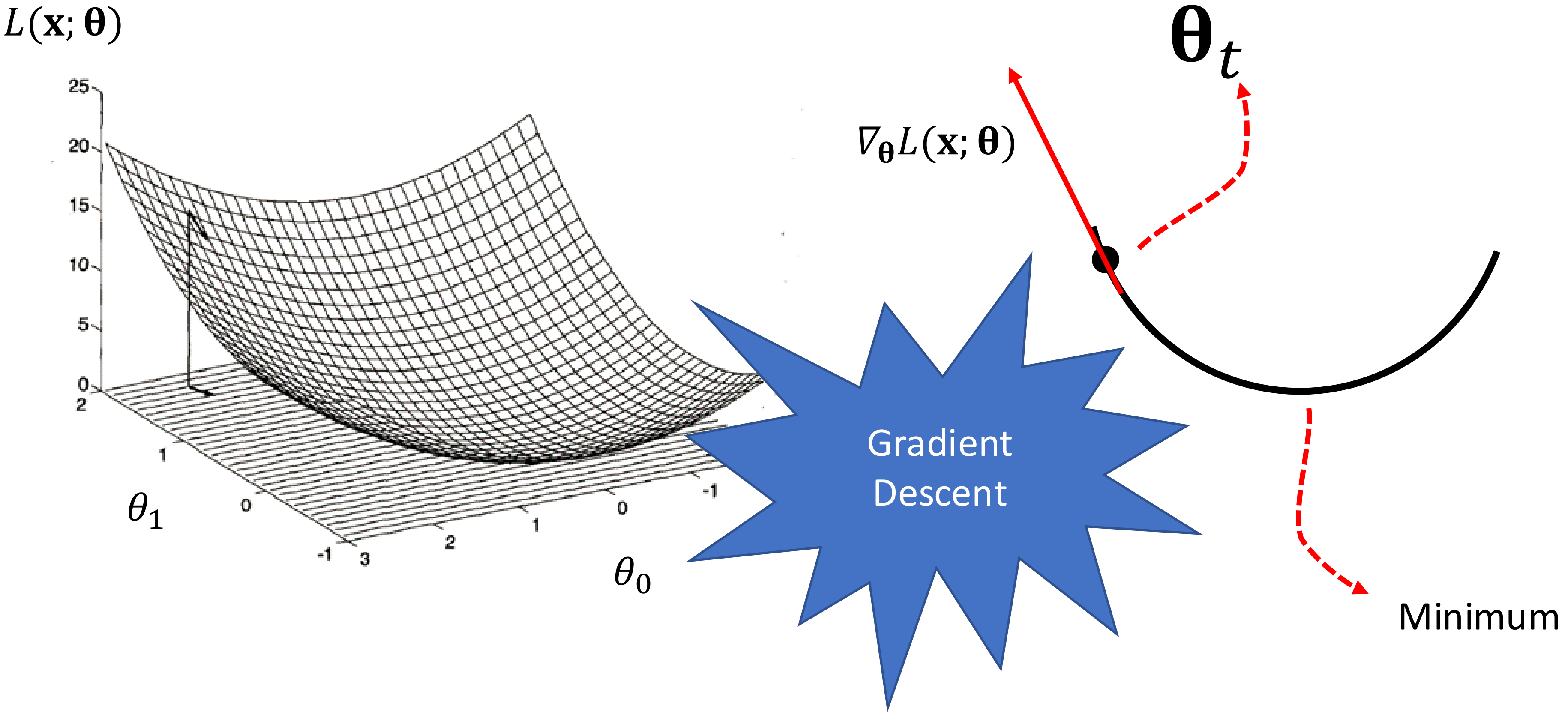


$$L(\mathbf{x}; \boldsymbol{\theta}) = \mathrm{d}\big(y, f(\mathbf{x}; \boldsymbol{\theta})\big)$$

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} L(\mathbf{x}; \boldsymbol{\theta})$$

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} L(\mathbf{x};\boldsymbol{\theta})$$

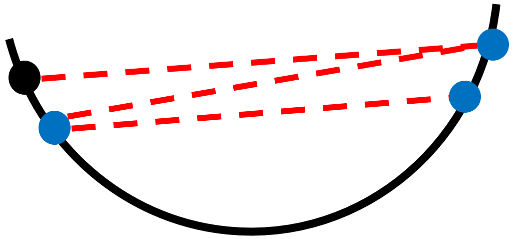
$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \eta\, \nabla_{\boldsymbol{\theta}} L(\mathbf{x};\boldsymbol{\theta})$$


$L(\mathbf{x};\boldsymbol{\theta})$
25
20
15
10
5
0
2
1
0
-1
θ_1
3
2
1
0
-1
θ_0
$\boldsymbol{\theta}_t$
$\nabla_{\boldsymbol{\theta}} L(\mathbf{x};\boldsymbol{\theta})$
Minimum

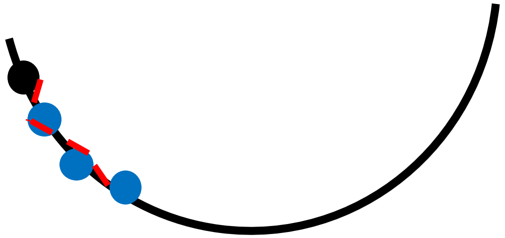
Gradient Descent

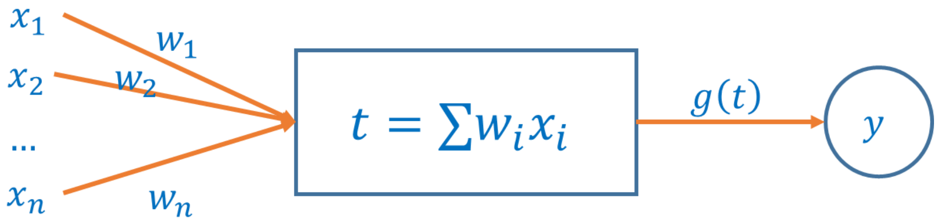
$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \eta \, \nabla_{\boldsymbol{\theta}} L(\mathbf{x}; \boldsymbol{\theta})$$

Learning Rate
(Step Size)



$$\nabla_{\boldsymbol{\theta}} L(\mathbf{x}; \boldsymbol{\theta}) = \frac{\partial L(\mathbf{x}; \boldsymbol{\theta})}{\partial \boldsymbol{\theta}}$$

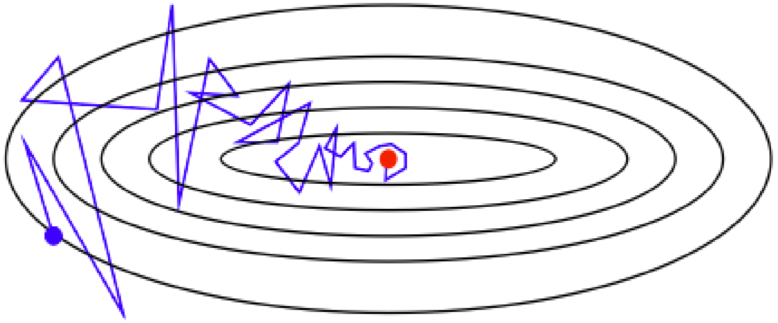




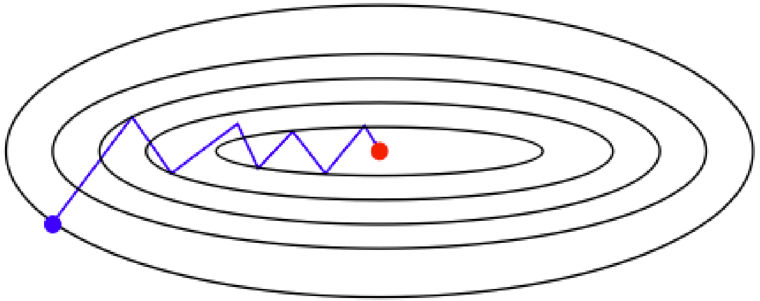
Gradient Descent

- True Gradient Descent
  - Calculate the loss & the gradient on the whole dataset
  - Then make the update
- Stochastic Gradient Descent
  - Calculate the loss & the gradient on examples one at a time
  - Update the weights after each example
- Batch Gradient Descent
  - Calculate the loss & the gradient on a set of examples (batch)
  - Update the weights after each bath

Stochastic Gradient Descent



Batch Gradient Descent



F. Bach

Gradient Descent

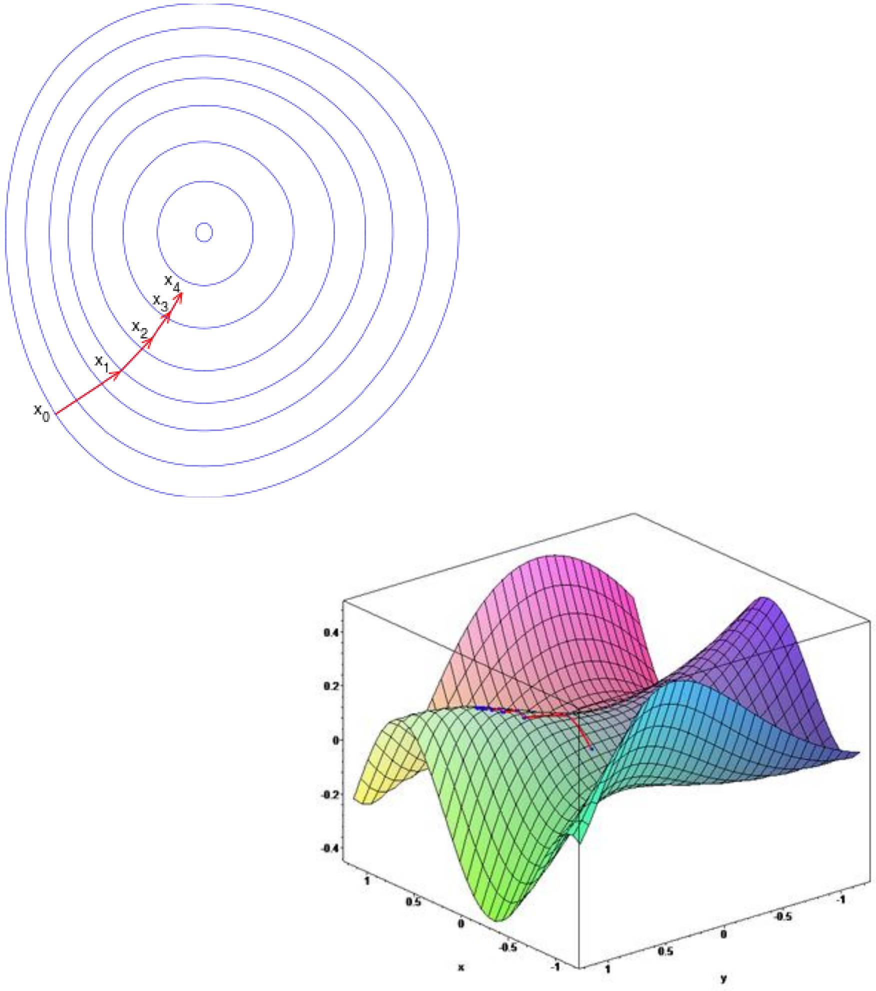
Input: Training set: $\{(\mathbf{x}_i, y_i)\}, i = 1, \ldots, N$

The network architecture.

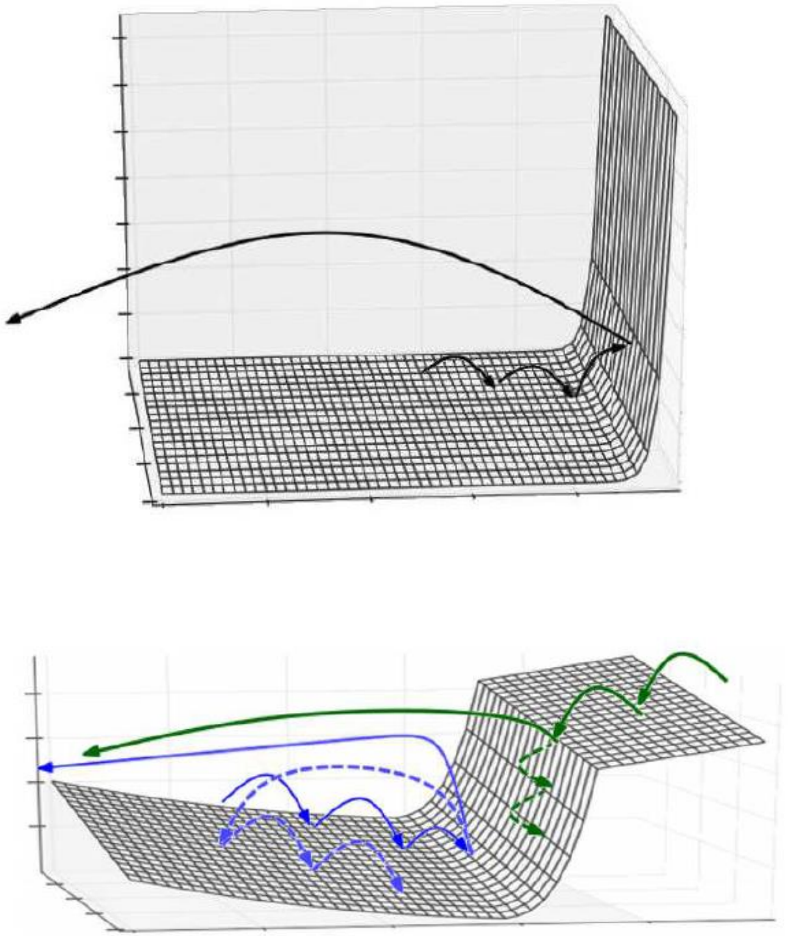
Output: Network parameters, $\boldsymbol{\theta}$

```
1. $\boldsymbol{\theta}_0 \leftarrow$ Random initial values
2. Until convergence:
   i.   Take $m$ samples from the dataset randomly
   ii.  Calculate predictions, $\hat{y}$, on $m$ samples using the current parameters $\boldsymbol{\theta}_t$
   iii. Calculate loss $L()$ and take the gradient $\nabla_{\boldsymbol{\theta}}\mathrm{L}$
   iv.  Update the weights
            $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \eta\nabla_{\boldsymbol{\theta}}\mathrm{L}$
```

Gradient descent



https://en.wikipedia.org/wiki/Gradient_descent



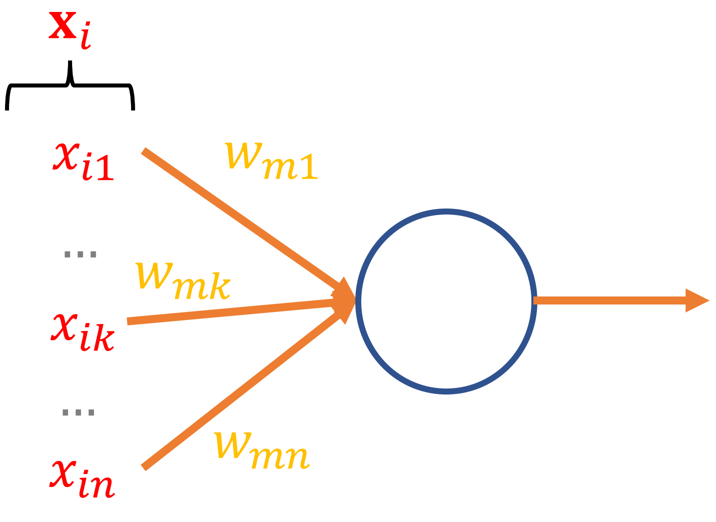
(Goodfellow vd., 2016)

Derive the gradients of hinge loss

$$\frac{\partial L_i}{\partial w_{mk}} = ?$$

$$\frac{\partial L_i}{\partial w_{mk}} = \underbrace{\frac{\partial L_i}{\partial e_m}}_{1}\;\underbrace{\frac{\partial e_m}{\partial s_m}}_{\mathbb{I}(s_m - s_{y_i} + \Delta > 0)}\;\underbrace{\frac{\partial s_m}{\partial w_{mk}}}_{x_{ik}}$$

$$\frac{\partial L_i}{\partial w_{mk}} = \mathbb{I}\left(s_m - s_{y_i} + \Delta > 0\right)x_{ik}$$



$$s_m = \mathbf{w}_m \mathbf{x}_i = \sum_k w_{mk}\, x_{ik}$$

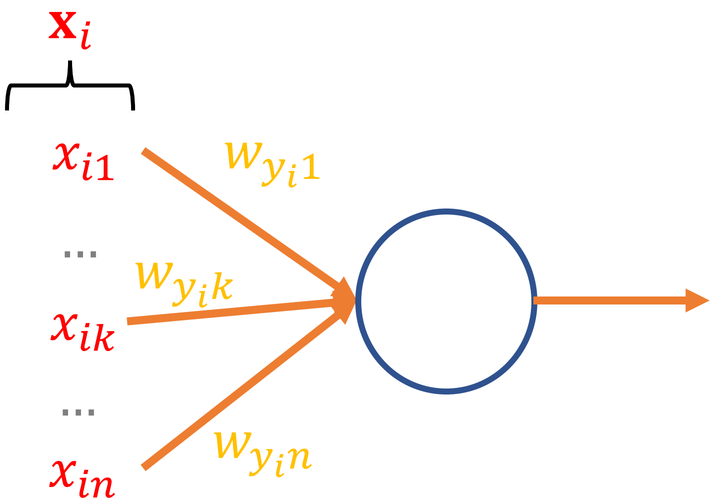
$$L_i = \sum_{j \neq y_i} \underbrace{\max\left(0, s_j - s_{y_i} + \Delta\right)}_{e_j}$$

This assumed that $m \neq y_i$.
What happens if that's not the case?
See the next page.

Derive the gradients of hinge loss

$$\frac{\partial L_i}{\partial w_{y_i k}} = ?$$

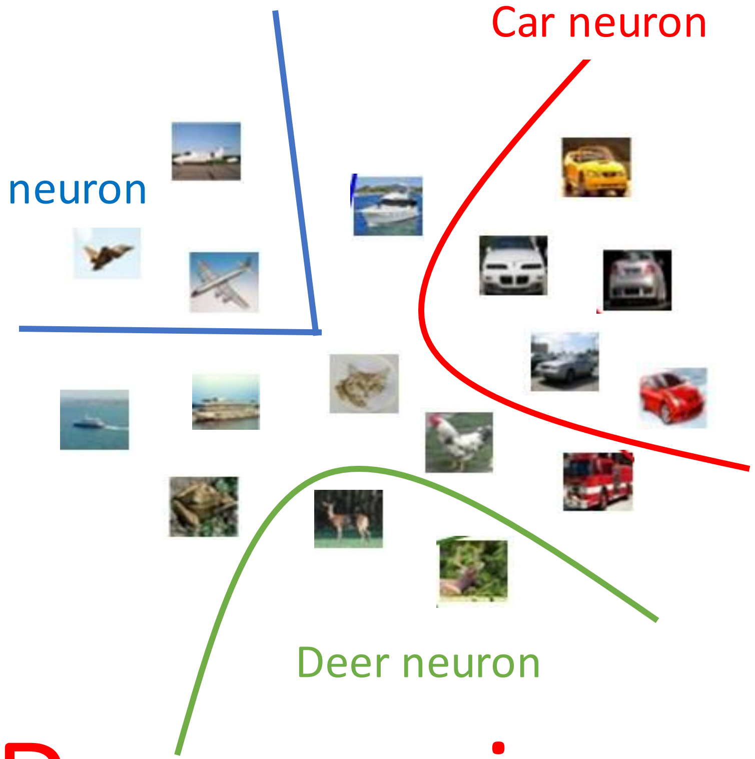
$$\frac{\partial L_i}{\partial w_{y_i k}} = \sum\nolimits_{j \neq y_i} \underbrace{\frac{\partial L_i}{\partial e_j}}_{1} \underbrace{\frac{\partial e_j}{\partial s_{y_i}}}_{\mathbb{I}(s_j - s_{y_i} + \Delta > 0)(-1)} \underbrace{\frac{\partial s_{y_i}}{\partial w_{y_i k}}}_{x_{ik}}$$



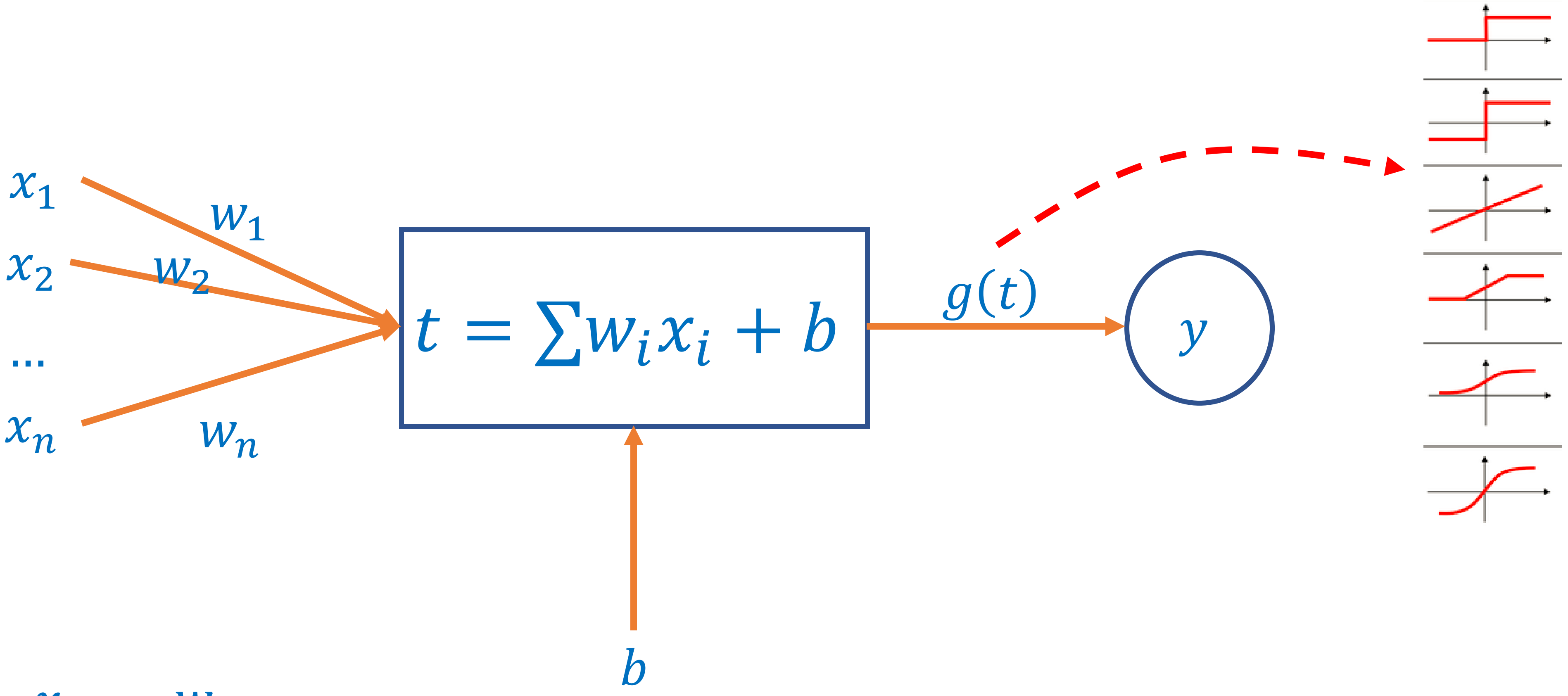
$$s_{y_i} = \mathbf{w}_{y_i} \mathbf{x}_i = \sum_k w_{y_i k}\, x_{ik}$$

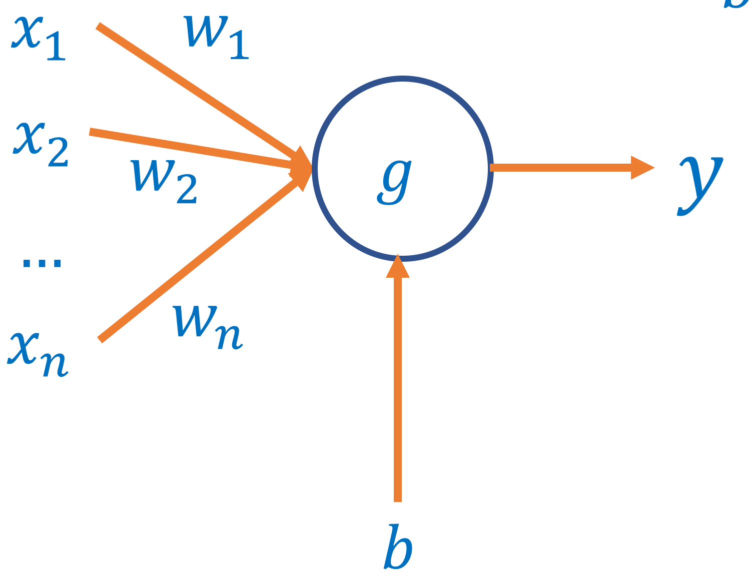
$$L_i = \sum_{j \neq y_i} \underbrace{\max(0, s_j - s_{y_i} + \Delta)}_{e_j}$$

$$\frac{\partial L_i}{\partial w_{y_i k}} = \sum_{j \neq y_i} \mathbb{I}(s_j - s_{y_i} + \Delta > 0)(-1) x_{ik}$$



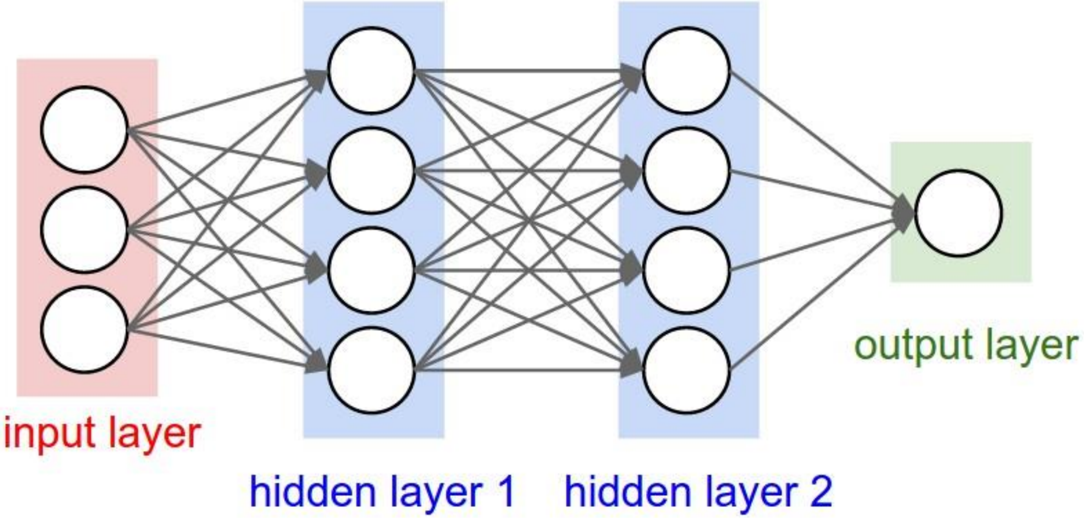
Non-linear Classification/Regression





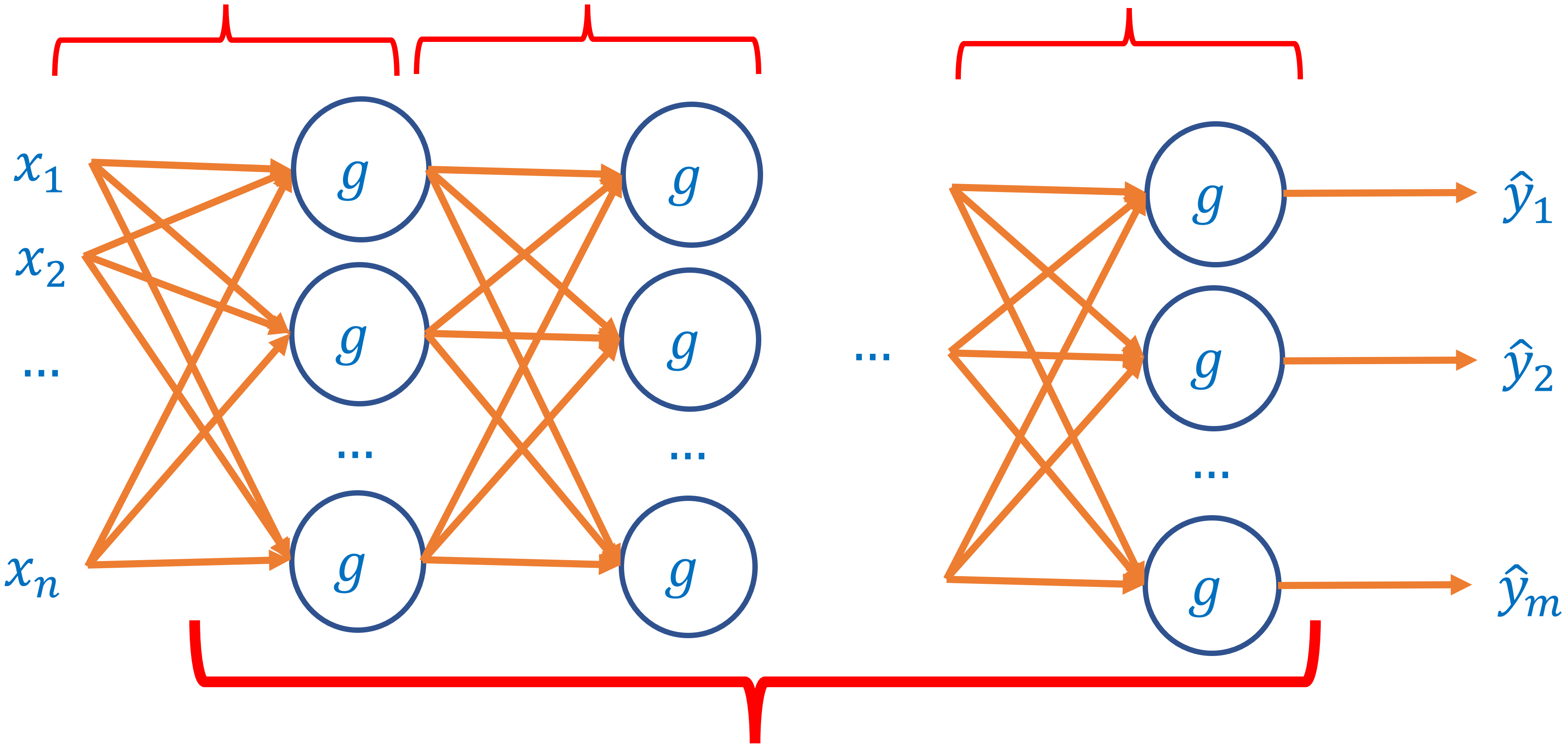
$$\boldsymbol{y} = f(\mathbf{x}) = f(\mathbf{x}; \boldsymbol{\theta})$$

$$= g\left(\sum_i w_i x_i + b\right) = g(\mathbf{w} \cdot \mathbf{x})$$

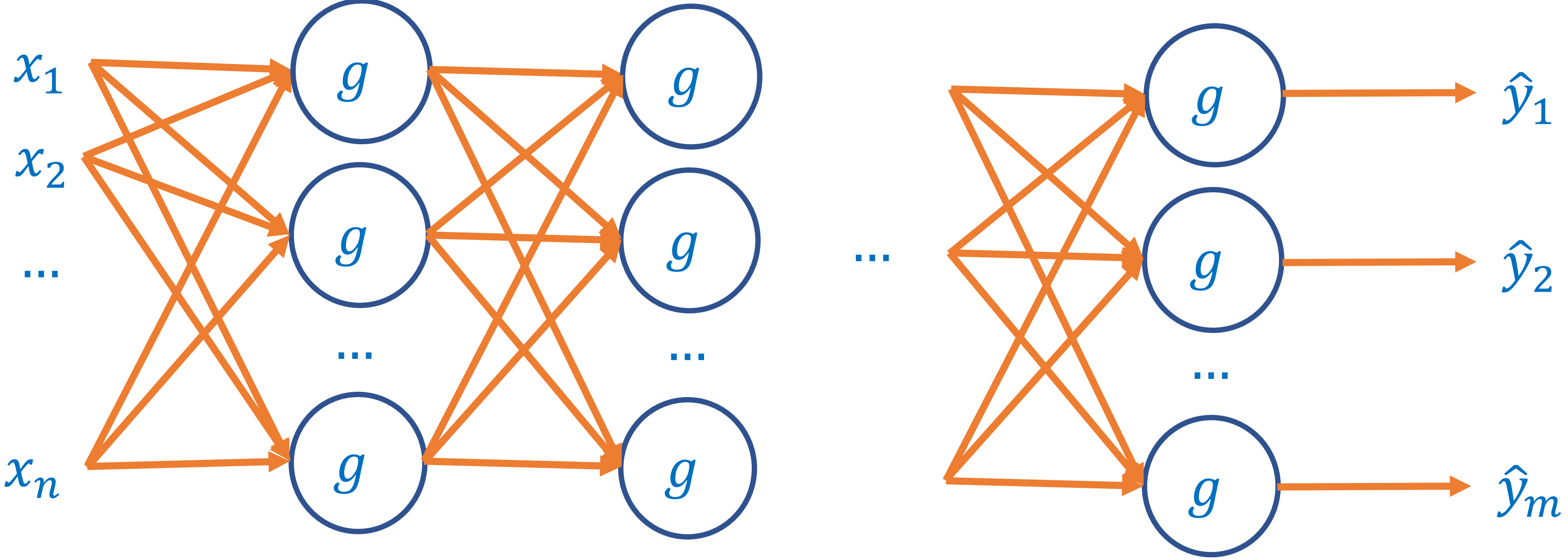


Multi-layer Perceptrons

$\mathbf{a}_1 = \mathrm{g}\left(\mathbf{W}_0\mathbf{x}^\mathrm{T}\right)$ $\mathbf{a}_2 = \mathrm{g}\left(\mathbf{W}_1\mathbf{a}_1^\mathrm{T}\right)$ $\mathbf{a}_l = \mathrm{g}\left(\mathbf{W}_{\mathbf{l-1}}\mathbf{a}_{l-1}^\mathrm{T}\right)$



$$f() = g\left(g\left(g\left(g(\dots.)\right)\right)\right)$$

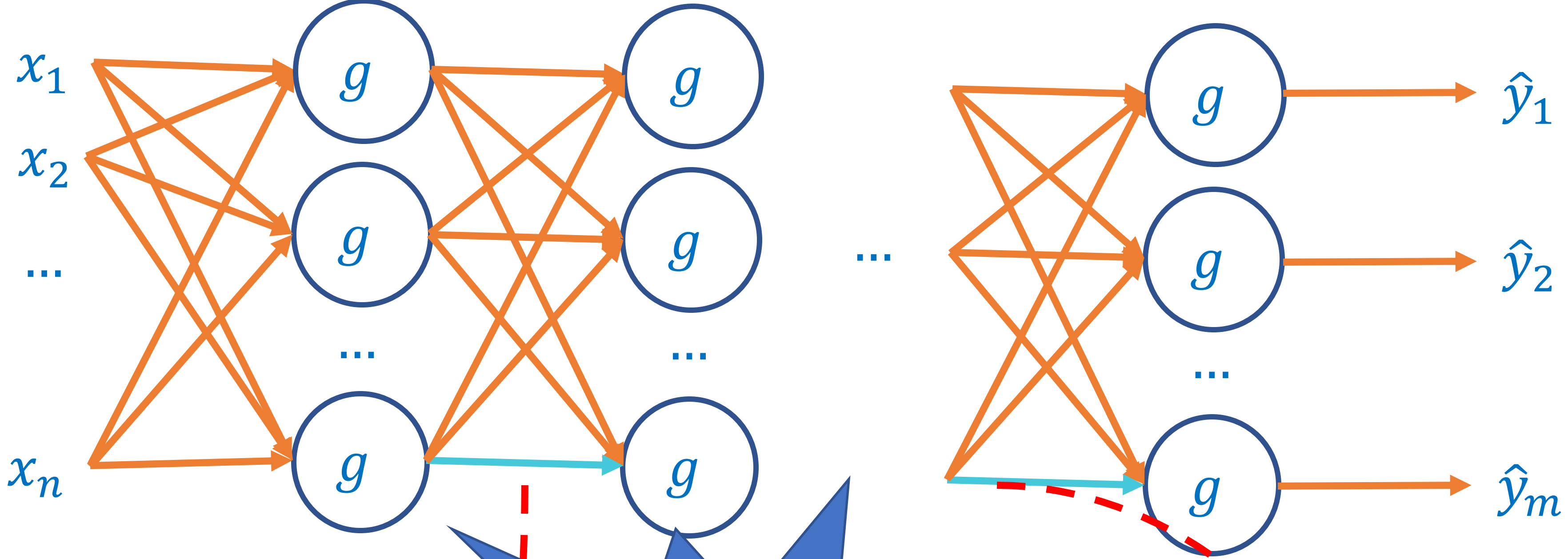


$$L(\mathbf{x};\boldsymbol{\theta}) = \mathrm{d}\big(\mathbf{y}, f(\mathbf{x};\boldsymbol{\theta})\big)$$

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \eta\, \nabla_{\boldsymbol{\theta}} L(\mathbf{x};\boldsymbol{\theta})$$

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} L(\mathbf{x};\boldsymbol{\theta})$$

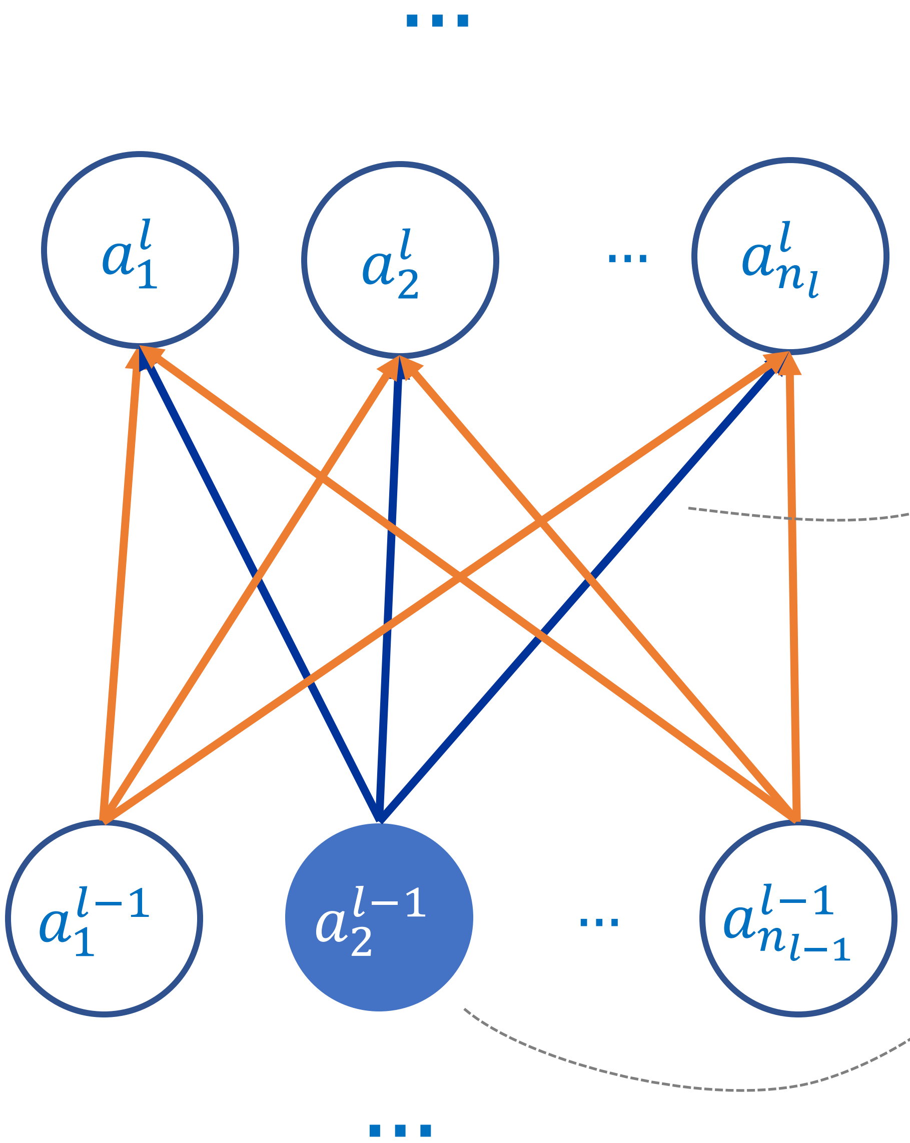
$$\nabla_{\boldsymbol{\theta}} L(\mathbf{x};\boldsymbol{\theta}) = \frac{\partial L(\mathbf{x};\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}$$



Backpropagation

$$\frac{\partial L(\mathbf{x};\boldsymbol{\theta})}{\partial w_{mi}} = \frac{\partial L}{\partial \hat{y}_m}\frac{\partial \hat{y}_m}{\partial g_m^l}\frac{\partial g_m^l}{\partial t_m}\frac{\partial t_m}{\partial w_{mi}}$$

$$\frac{\partial L(\mathbf{x};\boldsymbol{\theta})}{\partial w_j} = \frac{\partial L}{\partial \hat{y}_m}\frac{\partial \hat{y}_m}{\partial g_m^l}\frac{\partial g_m^l}{\partial t_m}\frac{\partial t_m}{\partial g_k^{l-1}}\frac{\partial g_k^{l-1}}{\partial t_k}\dots$$



$$\frac{\partial L(\mathbf{x};\boldsymbol{\theta})}{\partial w_{ij}} = \frac{\partial L(\mathbf{x};\boldsymbol{\theta})}{\partial a_i^l} \frac{\partial a_i^l}{\partial w_{ij}}$$

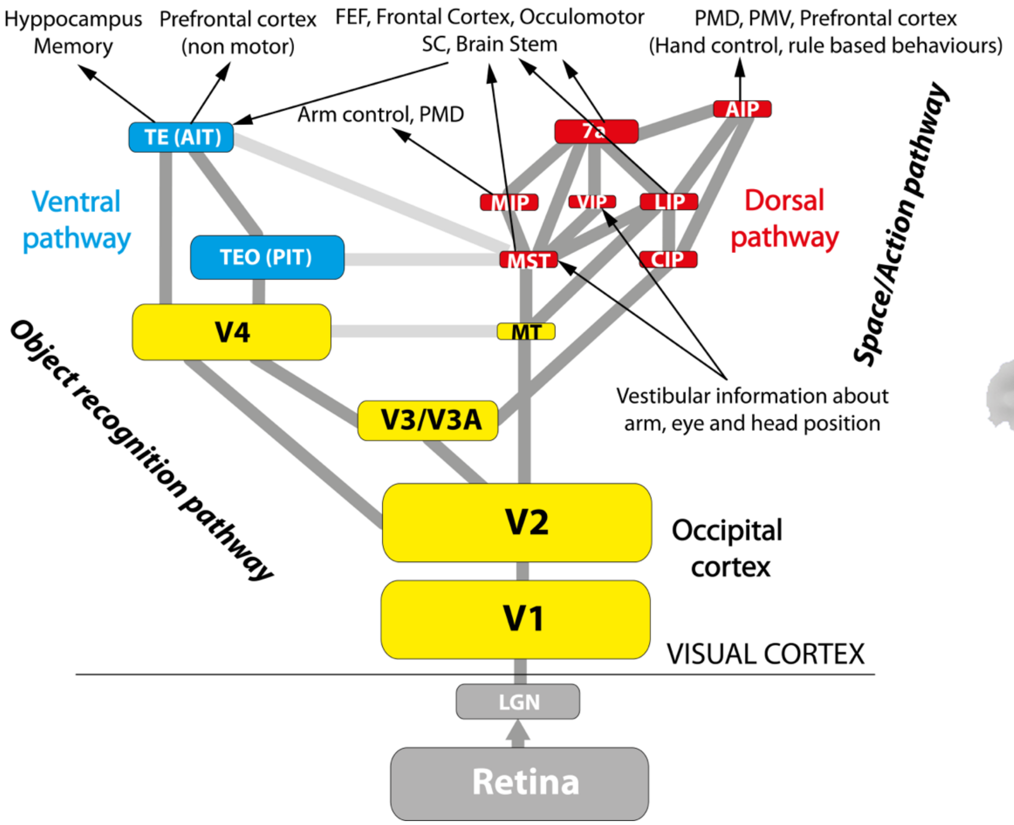
$$\frac{\partial L(\mathbf{x};\boldsymbol{\theta})}{\partial a_i^{l-1}} = \sum_j \frac{\partial L(\mathbf{x};\boldsymbol{\theta})}{\partial a_j^l} \frac{\partial a_j^l}{\partial a_i^{l-1}}$$

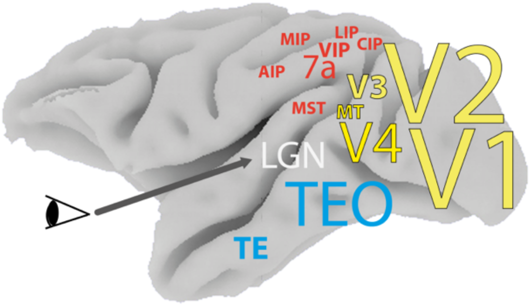
Importance of increasing layers

- Continuous functions:
  - Every bounded continuous function can be approximated with small error with two layers
- Arbitrary functions:
  - Three layers can approximate any arbitrary function
- Why do we need deep layers then?
  - If the problem has a hierarchical nature, more layers yield better performance
  - Lin vd., “Why does deep and cheap learning work so well?”, 2017.

Cybenko, G. (1989) "Approximations by superpositions of sigmoidal functions", Mathematics of Control, Signals, and Systems, 2 (4), 303-314
Kurt Hornik (1991) "Approximation Capabilities of Multilayer Feedforward Networks", Neural Networks, 4(2), 251257.
Hornik, Kurt, Maxwell Stinchcombe, and Halbert White. "Multilayer feedforward networks are universal approximators." Neural networks 2.5 (1989): 359-366.

Importance of increasing layers





Krueger, Jannsen, Kalkan, Lappe, .., “Deep Hierarchies in the Primate Visual Cortex: What Can We Learn For Computer Vision”, IEEE PAMI, 2013.

Multi-layer Perceptrons

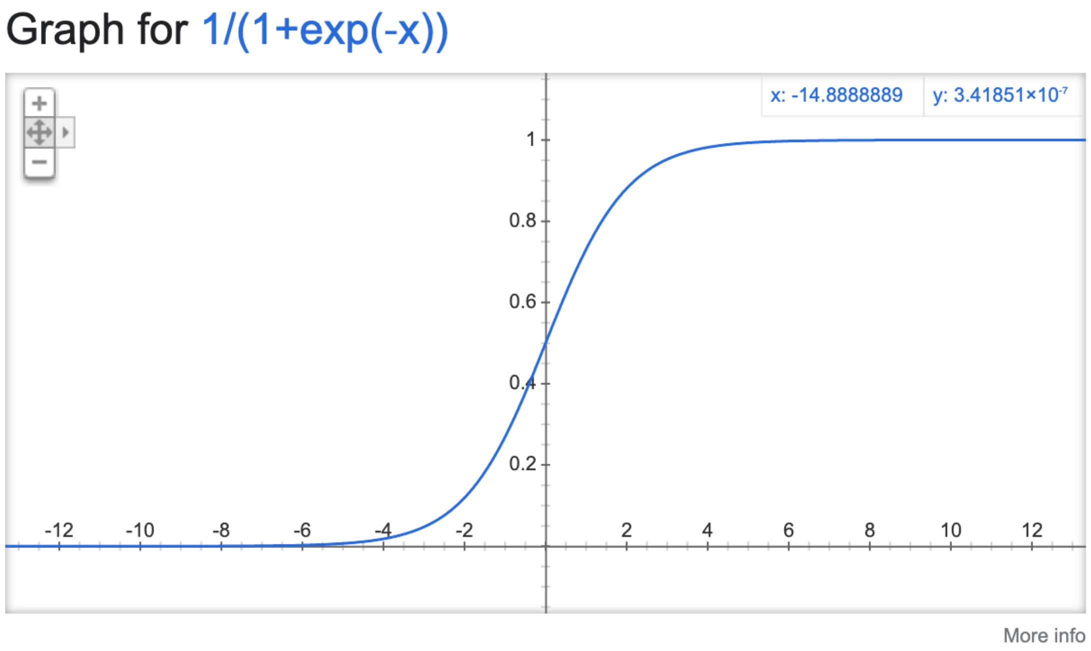
- To be able to have solutions for linearly non-separable cases, we need a non-linear and differentiable unit, e.g.:

$$\hat{y} = \sigma(\mathbf{w} \cdot \mathbf{x})$$

where

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

- Sigmoid (logistic) function
- Output is in range (0,1)
- Since it maps a large domain to (0,1) it is also called squashing function
- Alternatives: *tanh*



Graph for 1/(1+exp(-x))

Multi-layer Perceptrons

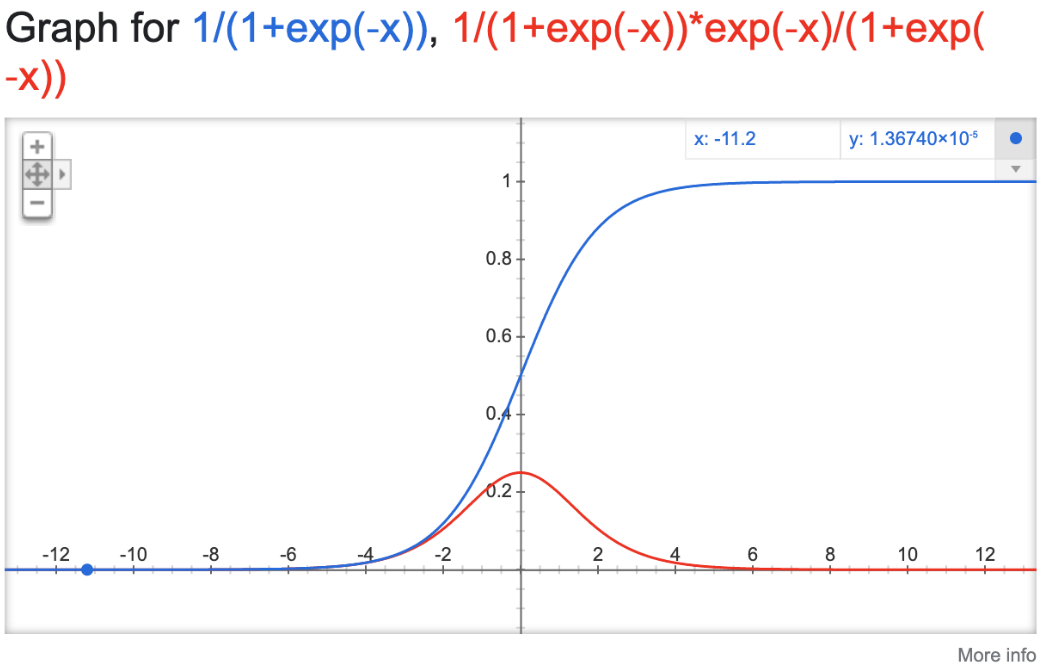
Derivative of the sigmoid:

$$\frac{d\sigma(x)}{dx} = \frac{d}{dx}\left(\frac{1}{1+e^{-x}}\right) = \frac{0\cdot(1+e^{-x}) - 1\cdot(-e^{-x})}{(1+e^{-x})^2}$$

$$= \frac{e^{-x}}{(1+e^{-x})^2} = \frac{1}{1+e^{-x}} \cdot \frac{e^{-x}}{1+e^{-x}}$$

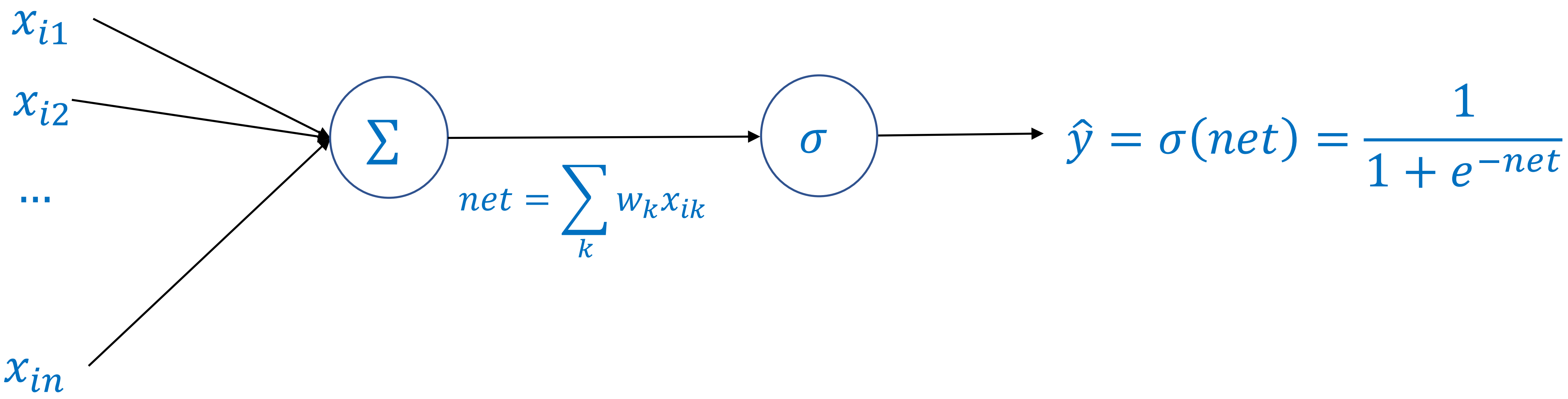
$$= \frac{1}{1+e^{-x}} \cdot \left(1 - \frac{1}{1+e^{-x}}\right)$$

$$= \sigma(x)\cdot(1-\sigma(x))$$



Graph for 1/(1+exp(-x)), 1/(1+exp(-x))*exp(-x)/(1+exp(-x))

A neuron with sigmoid function

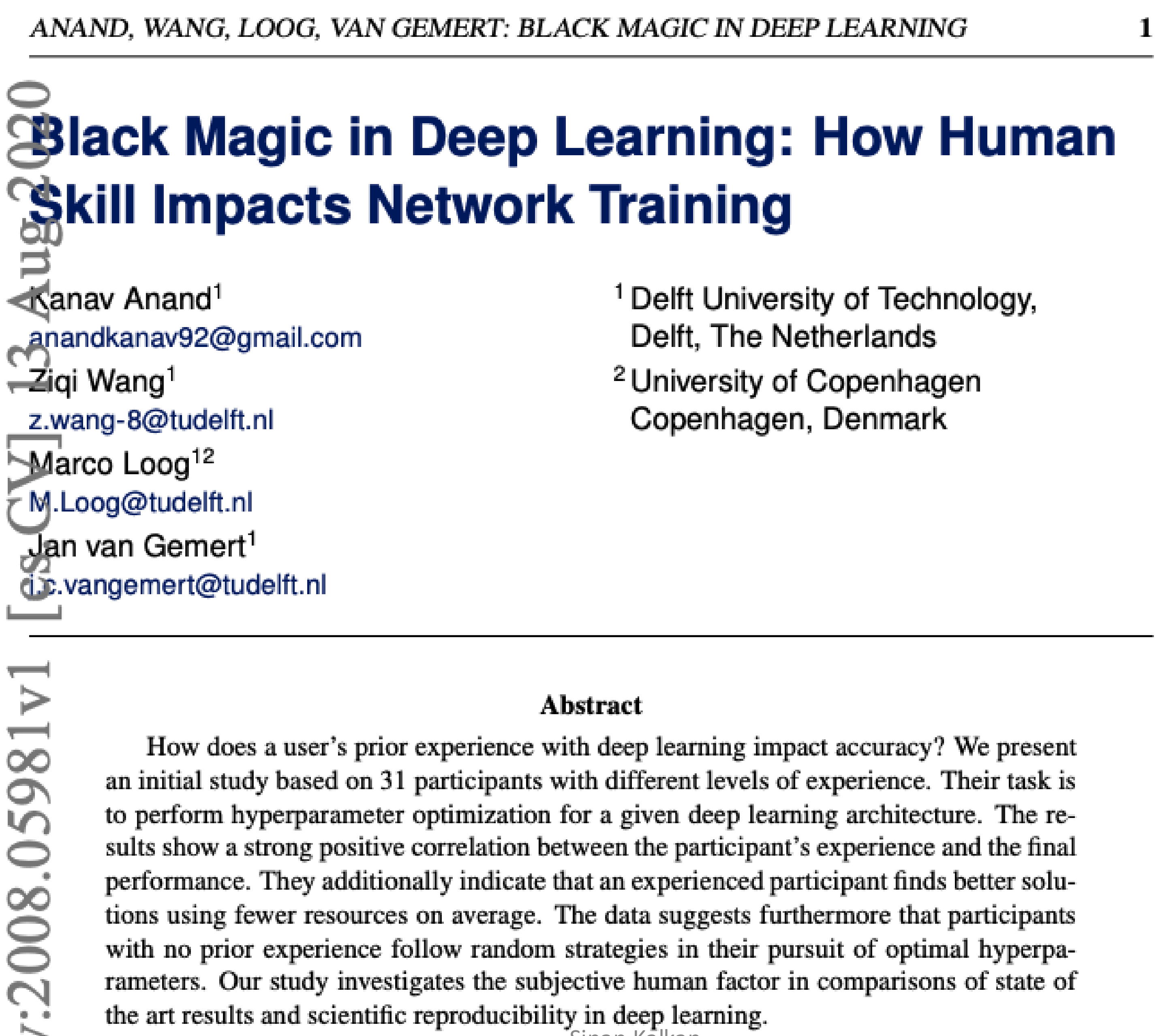

x_{i1}
x_{i2}
…
x_{in}
Σ
$net = \sum_k w_k x_{ik}$
σ
$\hat{y} = \sigma(net) = \frac{1}{1 + e^{-net}}$

Backpropagation

Why do we need to learn backpropagation?

- “Many frameworks implement backpropagation for us, why do we need to learn?”
  - This is not a blackbox. There are many problems/issues involved. You can only deal with them if you have a good understanding of backpropagation.

https://medium.com/@karpathy/yes-you-should-understand-backprop-e2f06eab496b#.7zawffou2



Black Magic in Deep Learning: How Human Skill Impacts Network Training

Kanav Anand[1]
anandkanav92@gmail.com
Ziqi Wang[1]
z.wang-8@tudelft.nl
Marco Loog[12]
M.Loog@tudelft.nl
Jan van Gemert[1]
j.c.vangemert@tudelft.nl

[1] Delft University of Technology,
Delft, The Netherlands
[2] University of Copenhagen
Copenhagen, Denmark

Abstract

How does a user's prior experience with deep learning impact accuracy? We present an initial study based on 31 participants with different levels of experience. Their task is to perform hyperparameter optimization for a given deep learning architecture. The results show a strong positive correlation between the participant's experience and the final performance. They additionally indicate that an experienced participant finds better solutions using fewer resources on average. The data suggests furthermore that participants with no prior experience follow random strategies in their pursuit of optimal hyperparameters. Our study investigates the subjective human factor in comparisons of state of the art results and scientific reproducibility in deep learning.

Backpropagation

The Model

Hidden activations:

$$h_{ij} = \sigma\left(\mathbf{w}_j^h \cdot \mathbf{x}_i\right) = \sigma\left(net_{ij}^h\right)$$

Output layer:

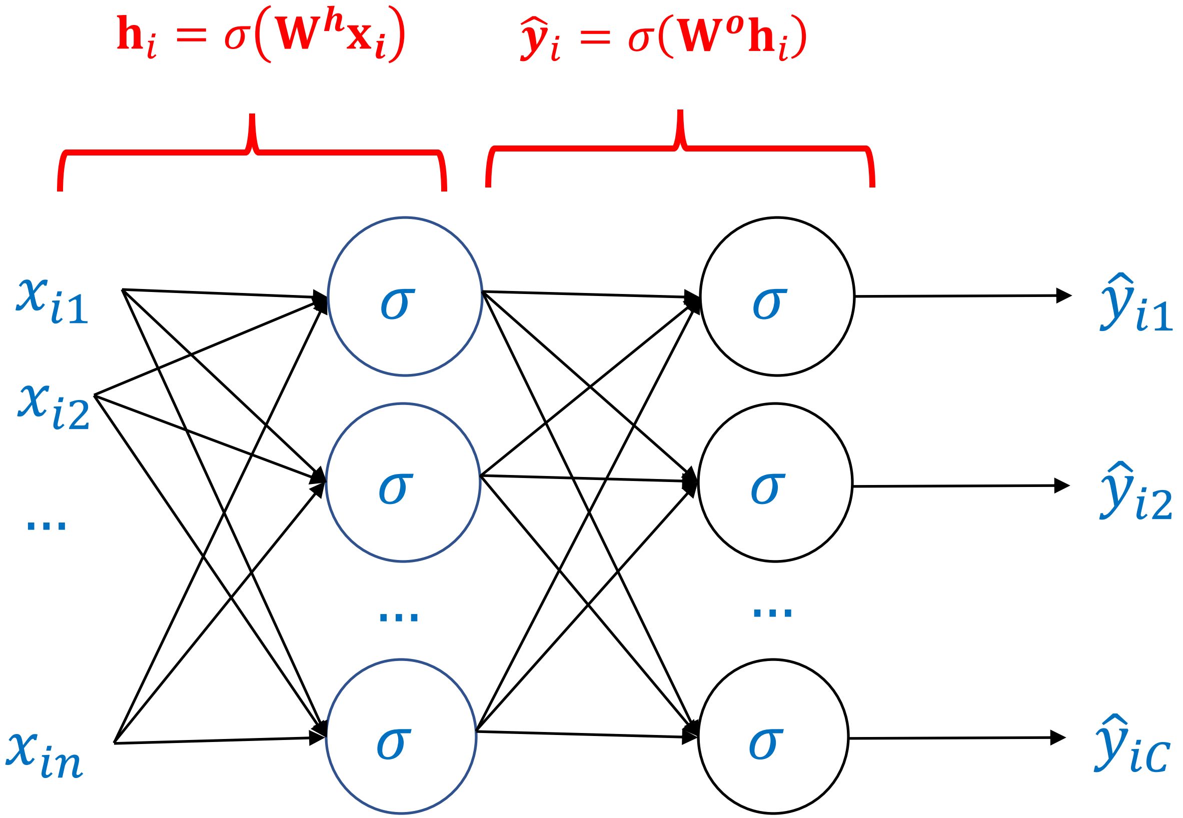
$$\hat{y}_{ic} = \sigma(\mathbf{w}_c^o \cdot \mathbf{h}_i) = \sigma(net_{ic}^o)$$

The loss function:

$$L(\boldsymbol{\theta}) = \frac{1}{2}\sum_{i=1}^{N}\sum_{c\in C}(\hat{y}_{ic} - y_{ic})^2$$

- For one sample:

$$L_i(\boldsymbol{\theta}) = \frac{1}{2}\sum_{c\in C}(\hat{y}_{ic} - y_{ic})^2$$



$\mathbf{h}_i = \sigma\left(\mathbf{W}^h\mathbf{x}_i\right)$
$\hat{\mathbf{y}}_i = \sigma(\mathbf{W}^o\mathbf{h}_i)$
x_{i1}
x_{i2}
…
x_{in}
σ
…
$\hat{y}_{i1}$
$\hat{y}_{i2}$
$\hat{y}_{iC}$

Backpropagation

For each output unit c, calculate its grad term δ_c^o:

$$\delta_{ic}^o = \frac{\partial L_i}{\partial net_{ic}^o} = \frac{\partial L_i}{\partial \hat{y}_{ic}} \frac{\partial \hat{y}_{ic}}{\partial net_{ic}^o} = (\hat{y}_{ic} - y_{ic})\hat{y}_{ic}(1 - \hat{y}_{ic})$$

For each hidden unit j, calculate its grad term δ_j^h:

$$\delta_{ij}^h = \frac{\partial L_i}{\partial net_{ij}^h} = \frac{\partial L_i}{\partial h_{ij}} \frac{\partial h_{ij}}{\partial net_{ij}^h} = \left(\textstyle\sum_{c\in C} \frac{\partial L_i}{\partial net_{ic}^o} \frac{\partial net_{ic}^o}{\partial h_{ij}}\right) h_{ij}(1 - h_{ij})$$

$$= \left(\textstyle\sum_{c\in C} \delta_{ic}^o w_{cj}\right) h_{ij}(1 - h_{ij})$$

Update weight w_{jk}^o in the output layer:

$$w_{jk}^o = w_{jk}^o - \eta\delta_{ij}^o h_{ik}$$

Update weight w_{jk}^h in the hidden layer:

$$w_{jk}^h = w_{jk}^h - \eta\delta_{ij}^h x_{ik}$$

The Model

Hidden activations: $h_{ij} = \sigma(\mathbf{w}_j^h \cdot \mathbf{x}_i) = \sigma(net_{ij}^h)$

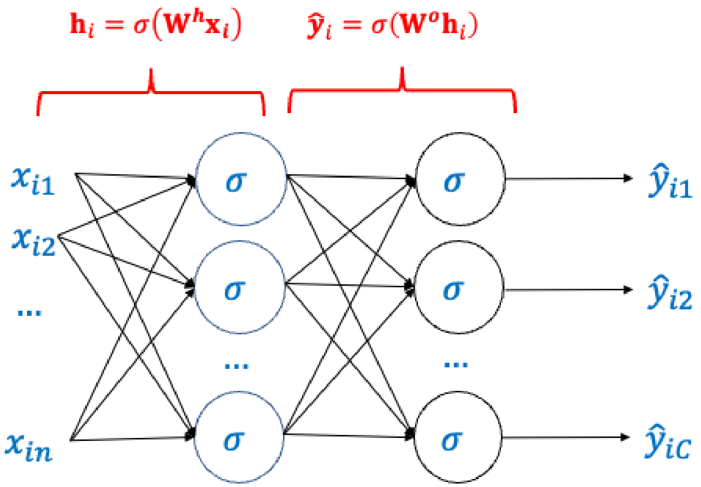
Output layer: $\hat{y}_{ic} = \sigma(\mathbf{w}_c^o \cdot \mathbf{h}_i) = \sigma(net_{ic}^o)$

The loss function:

$$L(\boldsymbol{\theta}) = \frac{1}{2}\sum_{i=1}^{N}\sum_{c\in C}(\hat{y}_{ic} - y_{ic})^2$$

- For one sample:

$$L_i(\boldsymbol{\theta}) = \frac{1}{2}\sum_{c\in C}(\hat{y}_{ic} - y_{ic})^2$$



Derivation of backpropagation

Derivation of the output unit weights

$$\Delta w^o_{ck} = -\eta \frac{\partial L_i}{\partial w^o_{ck}}$$

$$\frac{\partial L_i}{\partial w^o_{ck}} = \frac{\partial L_i}{\partial net^o_{ic}} \boxed{\frac{\partial net^o_{ic}}{\partial w^o_{ck}}} \longrightarrow h_{ik} \quad 1$$

$$\frac{\partial L_i}{\partial net^o_{ic}} = \frac{\partial L_i}{\partial \hat{y}_{ic}} \boxed{\frac{\partial \hat{y}_{ic}}{\partial net^o_{ic}}} \longrightarrow$$

Derivative of sigmoid: $\hat{y}_{ic}(1-\hat{y}_{ic})$ 2

$$\frac{\partial L_i}{\partial \hat{y}_{ic}} = \frac{\partial}{\partial \hat{y}_{ic}} \frac{1}{2} \Sigma_{c\in C} (\hat{y}_{ic} - y_{ic})^2 = \frac{\partial}{\partial \hat{y}_{ic}} \frac{1}{2} (\hat{y}_{ic} - y_{ic})^2 = \underbrace{(\hat{y}_{ic} - y_{ic})}_{3}$$

$$\Delta w^o_{ck} = -\eta \frac{\partial L_i}{\partial w^o_{ck}} = -\eta \underbrace{(\hat{y}_{ic} - y_{ic})}_{3} \underbrace{\hat{y}_{ic}(1-\hat{y}_{ic})}_{2} \underbrace{h_{ik}}_{1} = -\eta \delta^o_{ic} h_{ik}$$

The Model

Hidden activations: $h_{ij} = \sigma(\mathbf{w}^h_j \cdot \mathbf{x}_i) = \sigma(net^h_{ij})$

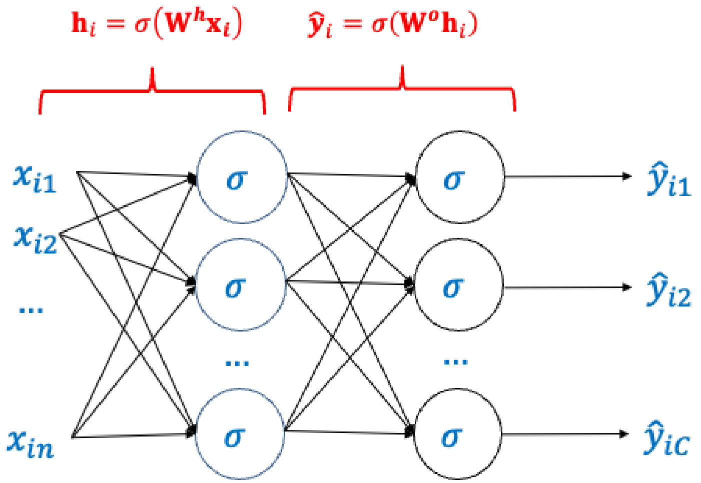
Output layer: $\hat{y}_{ic} = \sigma(\mathbf{w}^o_c \cdot \mathbf{h}_i) = \sigma(net^o_{ic})$

The loss function:

$$L(\boldsymbol{\theta}) = \frac{1}{2} \sum_{i=1}^{N} \sum_{c\in C} (\hat{y}_{ic} - y_{ic})^2$$

- For one sample:

$$L_i(\boldsymbol{\theta}) = \frac{1}{2} \sum_{c\in C} (\hat{y}_{ic} - y_{ic})^2$$



$\mathbf{h}_i = \sigma(\mathbf{W}^h \mathbf{x}_i)$
$\hat{\mathbf{y}}_i = \sigma(\mathbf{W}^o \mathbf{h}_i)$
x_{i1}
x_{i2}
...
x_{in}
σ
...
σ
...
$\hat{y}_{i1}$
$\hat{y}_{i2}$
$\hat{y}_{ic}$

Derivation of backpropagation

Derivation of the hidden unit weights

$$\Delta w_{jk}^h = -\eta \frac{\partial L_i}{\partial w_{jk}^h}$$

$$\frac{\partial L_i}{\partial w_{jk}^h} = \frac{\partial L_i}{\partial net_{ij}^h} \boxed{\frac{\partial net_{ij}^h}{\partial w_{jk}^h}} \longrightarrow x_{ik}$$

$$\frac{\partial L_i}{\partial net_{ij}^h} = \left(\sum\nolimits_c \frac{\partial L_i}{\partial \hat{y}_{ic}} \frac{\partial \hat{y}_{ic}}{\partial net_{ic}^o} \frac{\partial net_{ic}^o}{\partial h_{ij}}\right) \frac{\partial h_{ij}}{\partial net_{ij}^h}$$

$$= \left(\sum\nolimits_c \ \delta_{ic}^o \ w_{cj}\right) \ h_{ij}(1 - h_{ij})$$

$$\Delta w_{jk}^h = -\eta \frac{\partial L_i}{\partial w_{jk}^h} = -\eta \left(\sum_c \delta_{ic}^o w_{cj}\right) h_{ij}(1 - h_{ij}) x_{ik} = -\eta \delta_{ij}^h x_{ik}$$

The Model

Hidden activations: $h_{ij} = \sigma(\mathbf{w}_j^h \cdot \mathbf{x}_i) = \sigma(net_{ij}^h)$

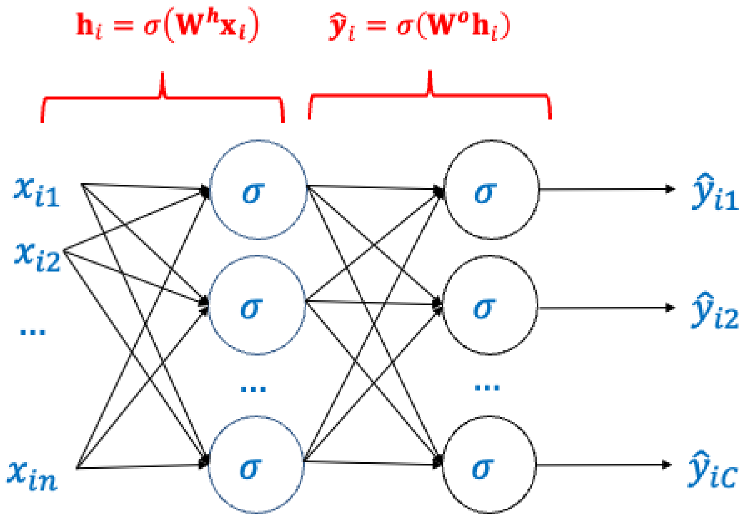
Output layer: $\hat{y}_{ic} = \sigma(\mathbf{w}_c^o \cdot \mathbf{h}_i) = \sigma(net_{ic}^o)$

The loss function:

$$L(\boldsymbol{\theta}) = \frac{1}{2} \sum_{i=1}^{N} \sum_{c \in C} (\hat{y}_{ic} - y_{ic})^2$$

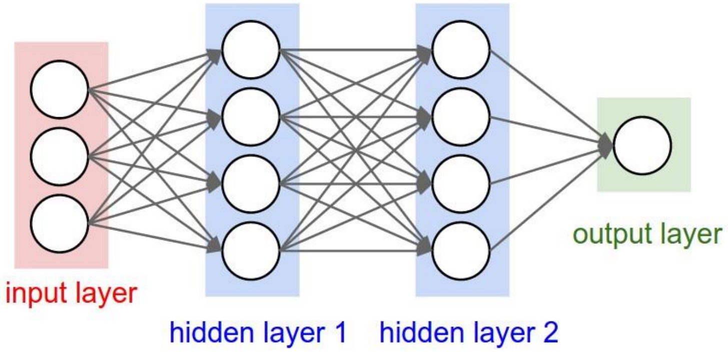
- For one sample:

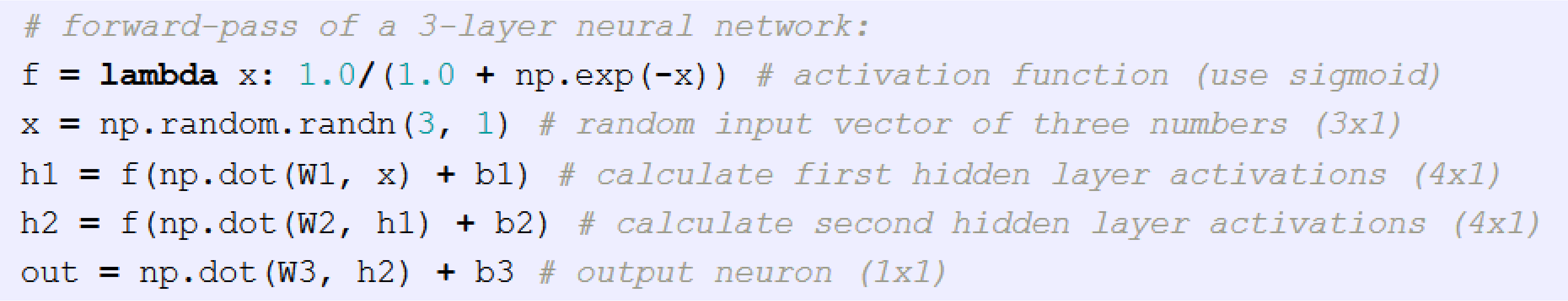
$$L_i(\boldsymbol{\theta}) = \frac{1}{2} \sum_{c \in C} (\hat{y}_{ic} - y_{ic})^2$$



$\mathbf{h}_i = \sigma(\mathbf{W}^h \mathbf{x}_i)$
$\hat{\mathbf{y}}_i = \sigma(\mathbf{W}^o \mathbf{h}_i)$
x_{i1}
x_{i2}
...
x_{in}
σ
...
$\hat{y}_{i1}$
$\hat{y}_{i2}$
$\hat{y}_{iC}$

Forward pass



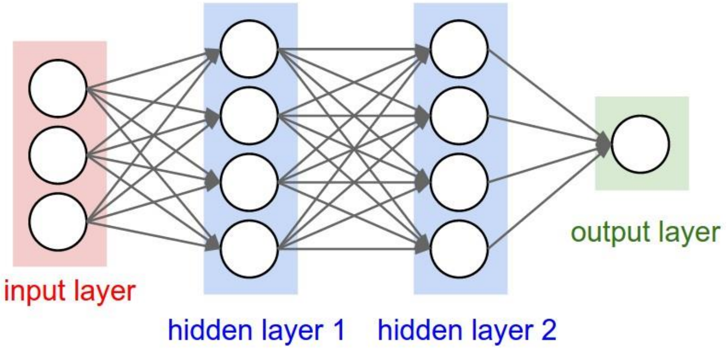


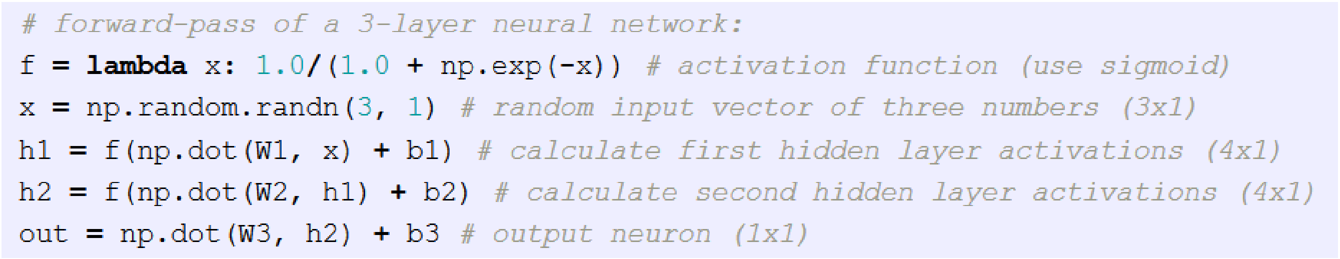
```
# forward-pass of a 3-layer neural network:
f = lambda x: 1.0/(1.0 + np.exp(-x)) # activation function (use sigmoid)
x = np.random.randn(3, 1) # random input vector of three numbers (3x1)
h1 = f(np.dot(W1, x) + b1) # calculate first hidden layer activations (4x1)
h2 = f(np.dot(W2, h1) + b2) # calculate second hidden layer activations (4x1)
out = np.dot(W3, h2) + b3 # output neuron (1x1)
```

https://cs231n.github.io/

Backward pass

```
loss = 0.5*np.sum((out-y)**2)
dout = (out-y)
dW3 = np.dot(dout, h2.T)
…
```





```
# forward-pass of a 3-layer neural network:
f = lambda x: 1.0/(1.0 + np.exp(-x)) # activation function (use sigmoid)
x = np.random.randn(3, 1) # random input vector of three numbers (3x1)
h1 = f(np.dot(W1, x) + b1) # calculate first hidden layer activations (4x1)
h2 = f(np.dot(W2, h1) + b2) # calculate second hidden layer activations (4x1)
out = np.dot(W3, h2) + b3 # output neuron (1x1)
```

https://cs231n.github.io/

Backpropagation vs. numerical differentiation

What are their complexities?

- Backpropagation:
  - $O(|\theta|)$

- Numerical differentiation
  - $O(|\theta|^2)$

Neural Engineering

- Loss functions
- On optimization
- Activation functions
- Capacity, convergence
- Preprocessing
- …