THE UNIVERSITY OF MELBOURNE

Introduction to Deep Learning ——Recap of Neural network

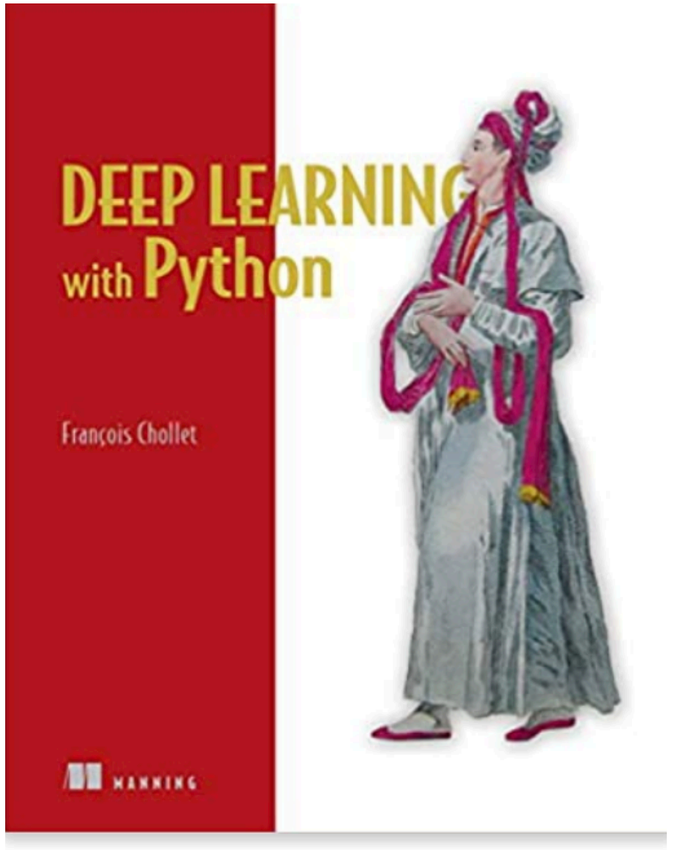
COMP90051 Statistical Machine Learning

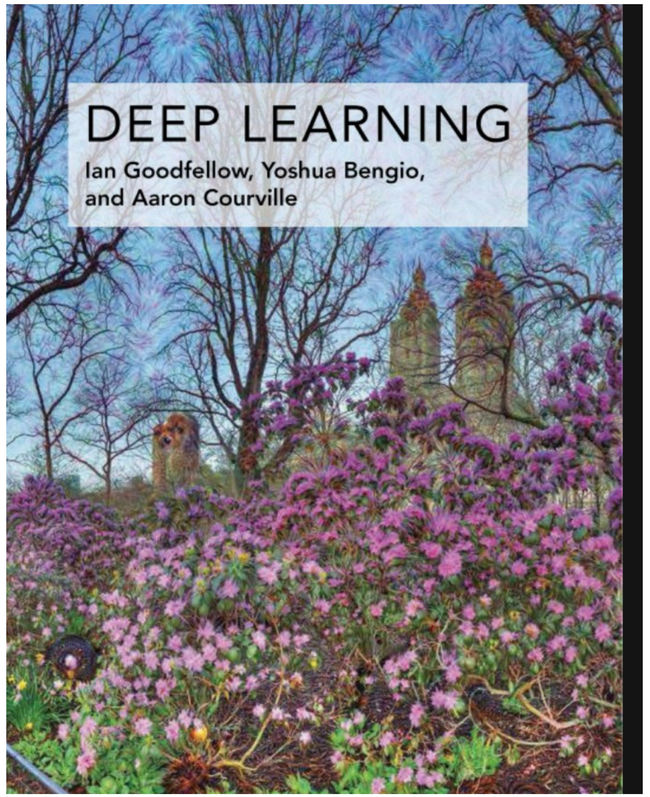
Qiuhong Ke

Before we start

Books & resources

- Deep Learning with Python, by Francois Chollet, available in unimelb library
- Deep Learning, by Ian Goodfellow and Yoshua Bengio and Aaron Courville, https://www.deeplearningbook.org/





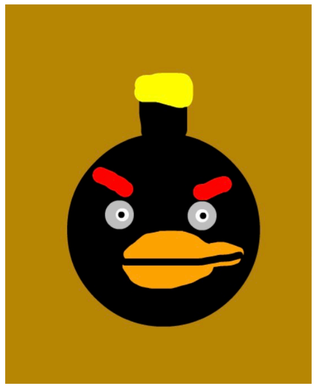
Angry birds

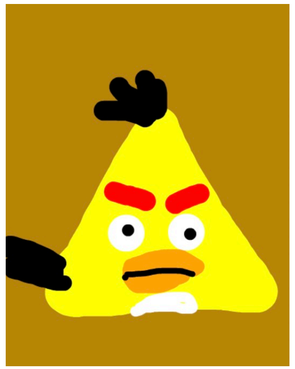
VS

Source: by Papachan (CC BY)
https://www.sketchport.com/drawing/4524248689278976/bomb-and-chuck

Bird classification

Input → Feature engineering → Classification → Output





Color or
Body shape or
…

SVM
…

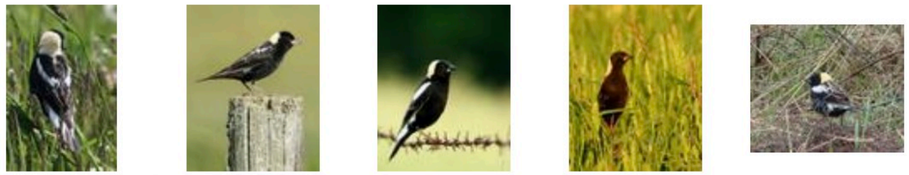
Bird class

Source: by Papachan (CC BY)
https://www.sketchport.com/drawing/4524248689278976/bomb-and-chuck

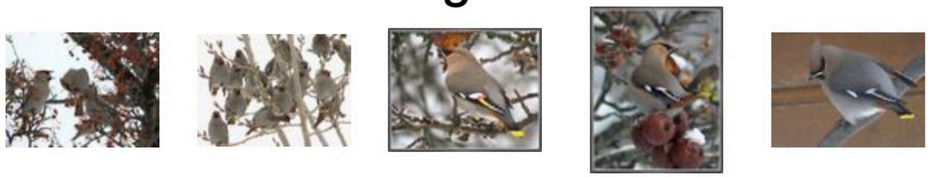
In real life…

Boat tailed Grackle

Bobolink

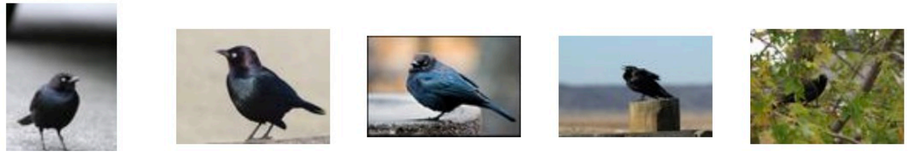


Bohemian Waxwing



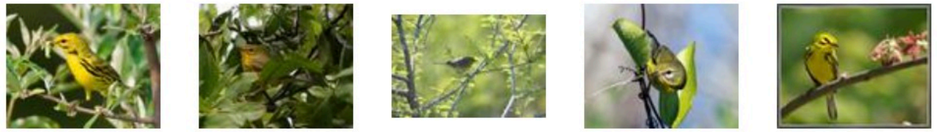
Brandt Cormorant

Brewer Blackbird

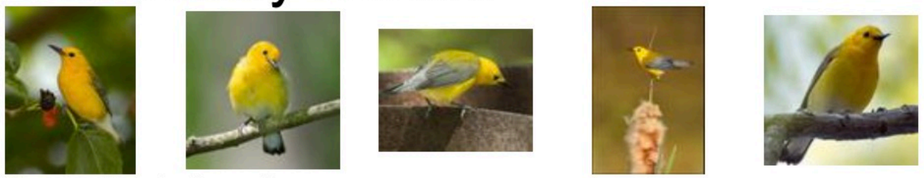


Pomarine Jaeger

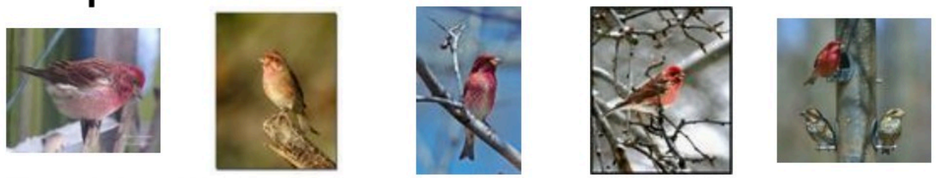
Prairie Warbler



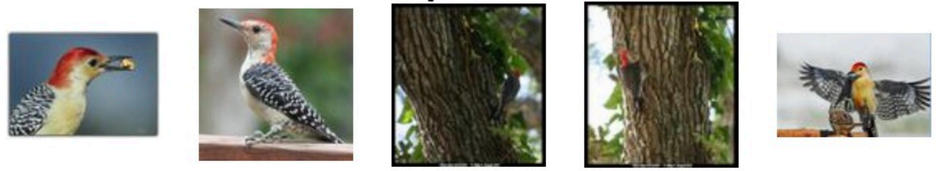
Prothonotary Warbler



Purple Finch

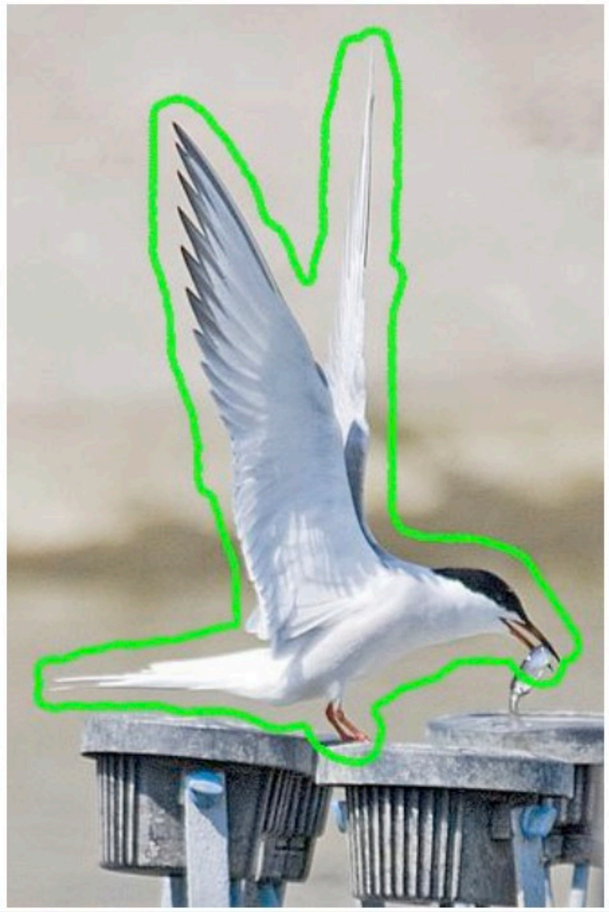


Red bellied Woodpecker



Source: Welinder, Peter, et al. "Caltech-UCSD birds 200." (2010).

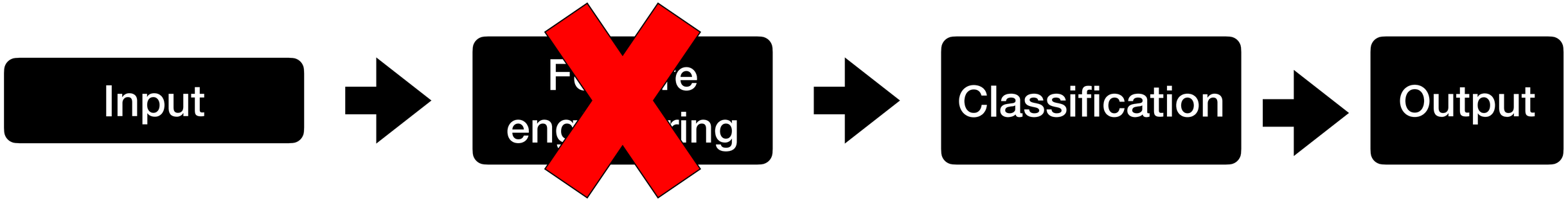
More features for real birds…



forehead_color	**black**	**black**	**black**
breast_pattern	**solid**	**solid**	**solid**
breast_color	**white**	**white**	**white**
head_pattern	**plain**	**capped**	plain
back_color	**white**	white	**black**
wing_color	grey/white	**grey**	**white**
leg_color	**orange**	orange	**orange**
size	**medium**	large	**medium**
bill_shape	needle	dagger	**dagger**
wing_shape	**pointed**	**tapered**	**long**
...	...	...	...
primary_color	white	white	white

Source: Welinder, Peter, et al. "Caltech-UCSD birds 200." (2010).

‘REAL Bird’ classification



Use deep learning models!

Bird class

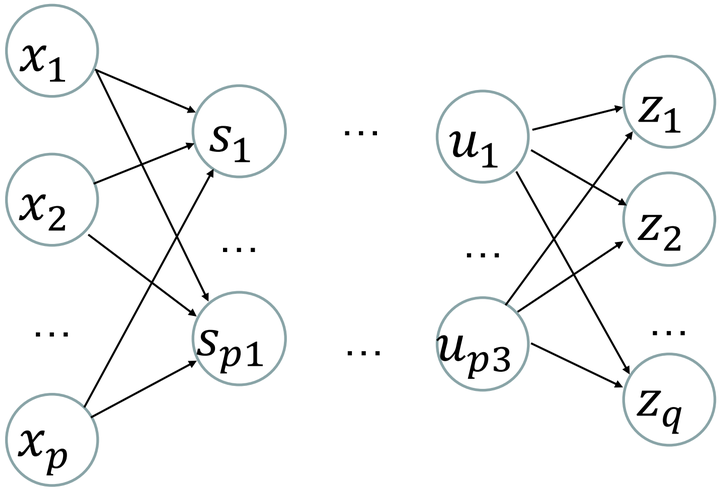
Outline

- Introduction
- Neural network (Perceptron and Multi-layer Perceptron)
- Gradient descent algorithm

Deep learning

Input → Feature learning + classification → Output

Deep neural network



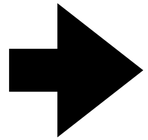
many hidden layers

Deep learning is everywhere

Near-human-level
- image classification
- speech recognition
- handwriting transcription
- autonomous driving
- ....

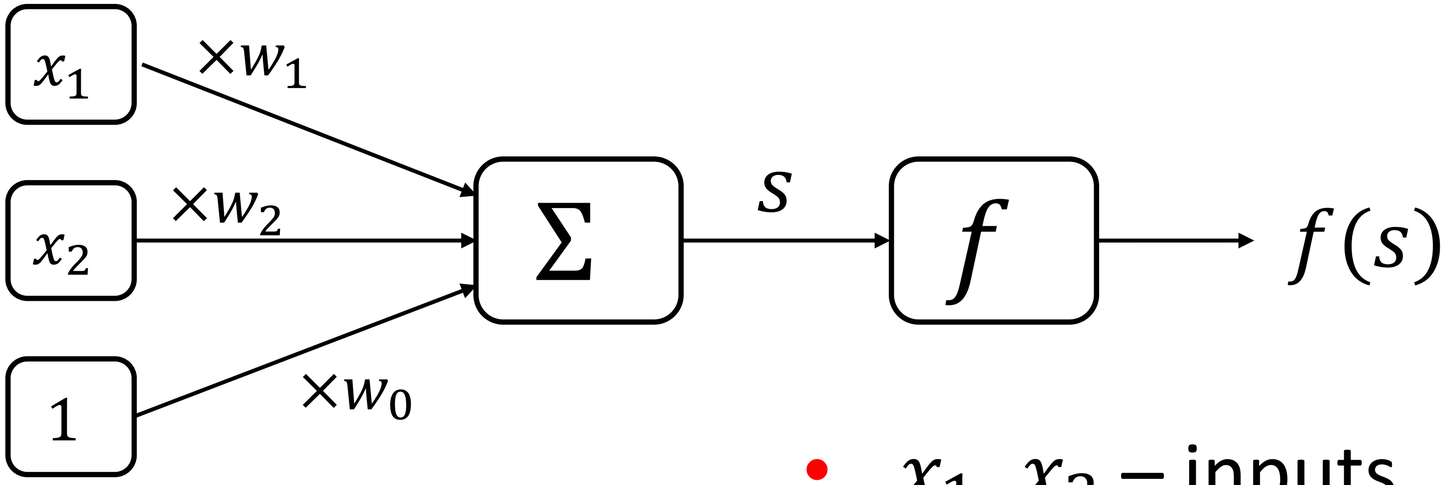
Why now deep learning is popular?

- Large dataset
- Powerful computing resources
- Better algorithms:
  - Weight-initialization schemes
  - Optimisation schemes
  - Activation functions
  - …



Network: deeper than deeper

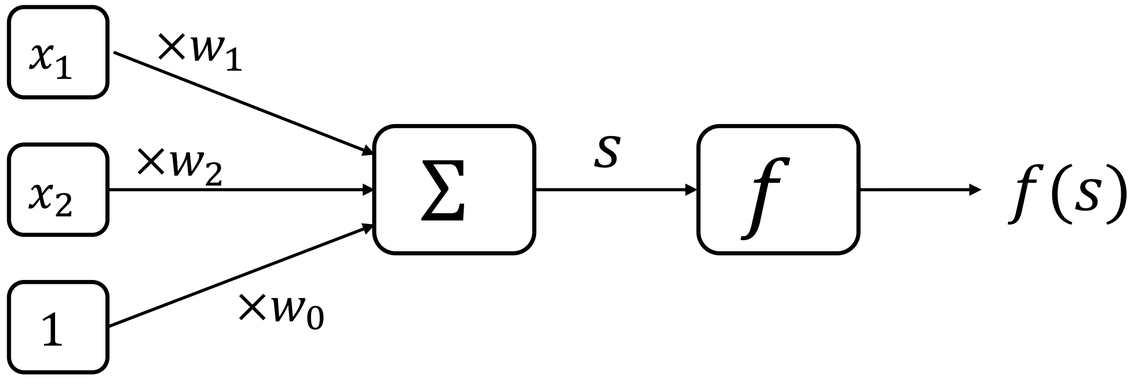
Perceptron



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

- x_1, x_2 – inputs
- w_1, w_2 – synaptic weights
- w_0 – bias weight
- f – activation function

Perceptron is a linear binary classifier



$$if\ s \geq 0 : f(s)\ is\ positive\ class$$

$$if\ s < 0 : f(s)\ is\ negative\ class$$

Step function

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

Sign function

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

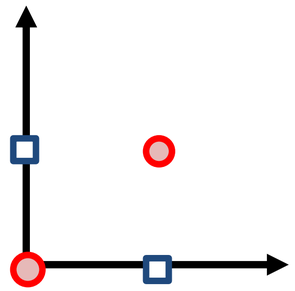
Simple example

Exercise: find weights of a perceptron capable of perfect classification of the following dataset

x_1	x_2	y
0	0	Class B
0	1	Class B
1	0	Class B
1	1	Class A

Limitations of perceptron learning

- If the data is linearly separable, the perceptron training algorithm will converge to a correct solution
  - It will converge to some solution (separating boundary), one of infinitely many possible ← bad!
- However, if the data is not linearly separable, the training will fail completely rather than give some approximate solution
  - Ugly ☹



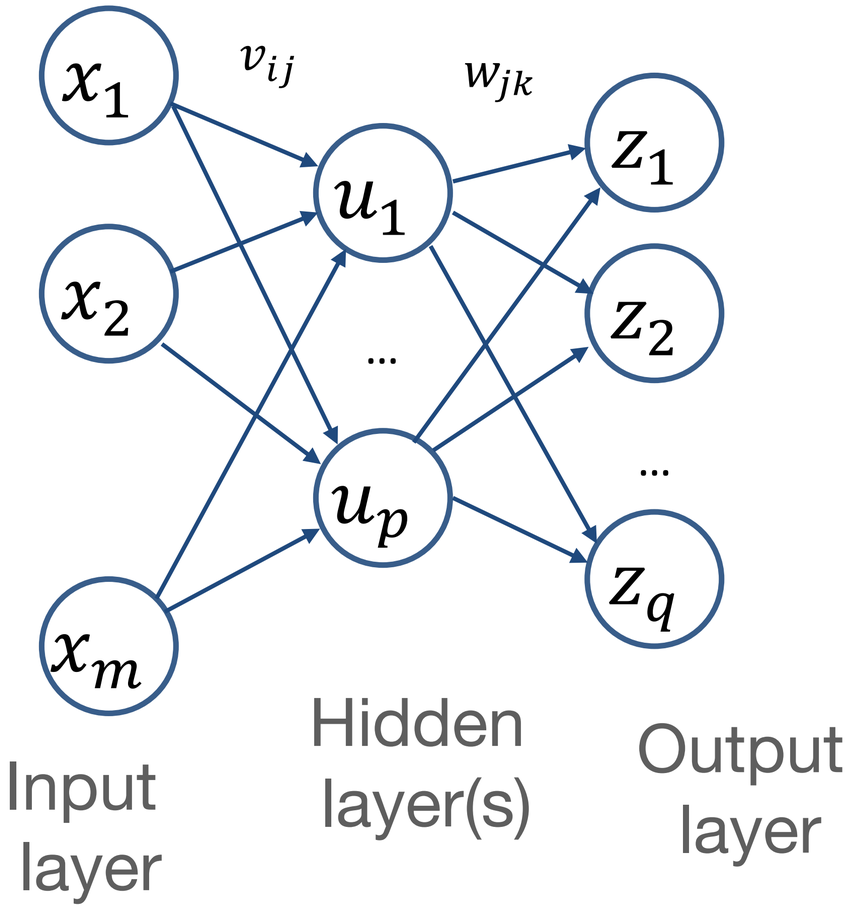
XOR

Multi-layer perceptron: function composition

$$u_j = g(r_j)$$

$$r_j = \sum_{i=0}^{m} x_i v_{ij}$$

g(x): activation function



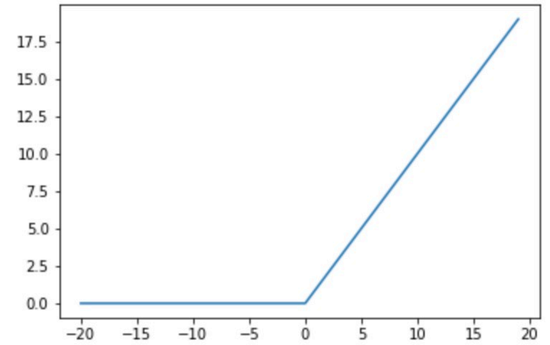
$$z_k = h(s_k)$$

$$s_k = \sum_{j=0}^{p} u_j w_{jk}$$

h(x): activation function

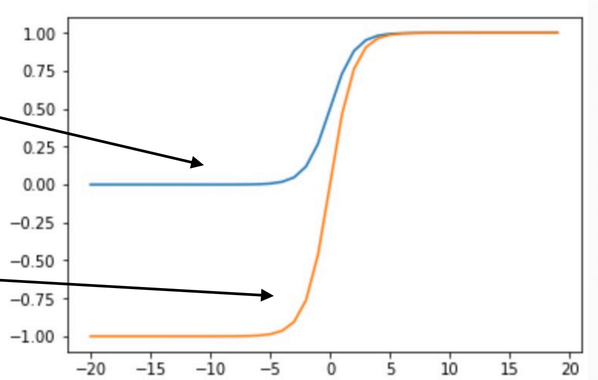
Common non-linear activation functions

Relu: $f(x) = \max(0,x)$



Sigmoid: $f(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1}$

TanH $f(x) = \frac{2}{1+e^{-2x}} - 1 = \frac{2e^{2x}}{e^{x2}+1} - 1$



Softmax: $f(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{C} e^{x_i}} \quad i = 1,\cdots,C$

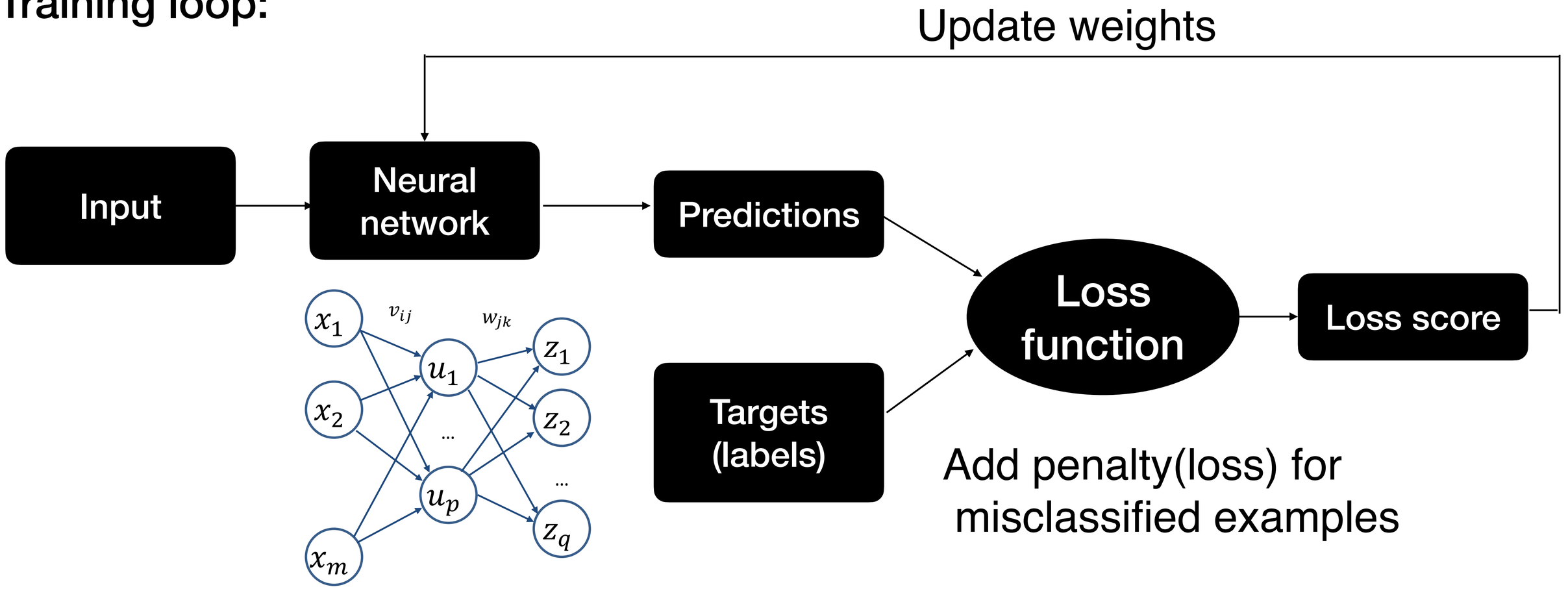
(Multi-class classification)

How to train your ~~dragon~~ network?

Adapted from Movie Poster from Flickr user jdxyw (CC BY-SA 2.0)

“Training”: adjust weights to minimise loss. **How?**

Training loop:



Derivative

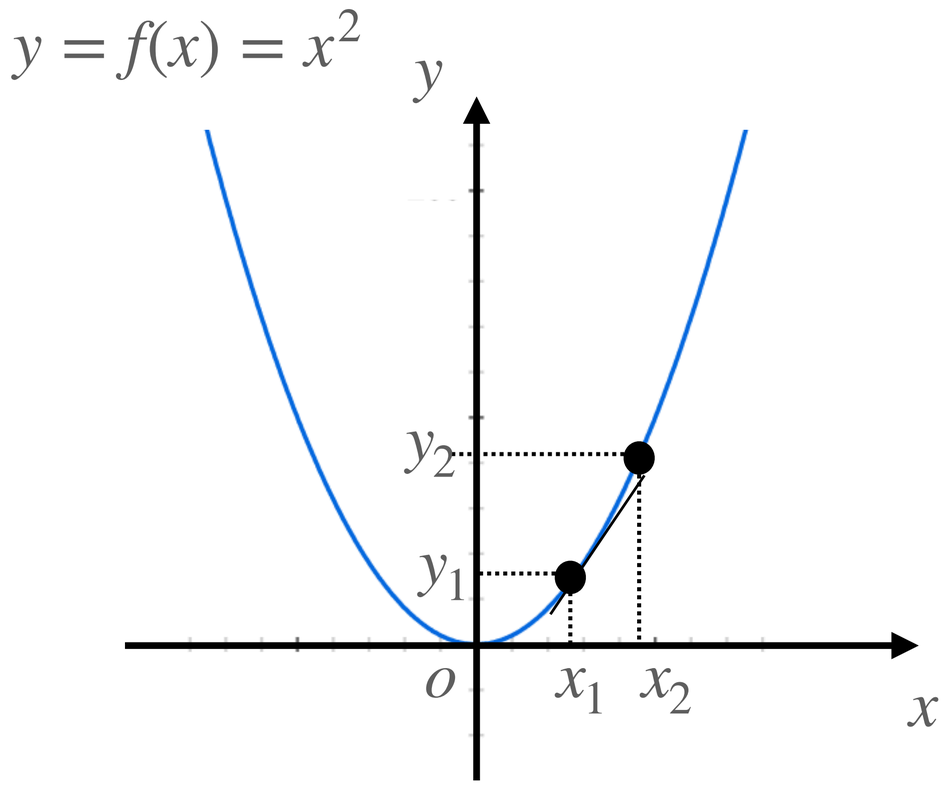
Input x, output y: $f(x)$

$$\Delta x \to 0 : f(x_1 + \Delta x) - f(x_1) = a\Delta x$$

$$a : \ rate\ of\ change\ (derivative)\ at\ x_1$$

derivative: a vector from origin.

Move x along OPPOSITE direction of the vector: decrease f(x)

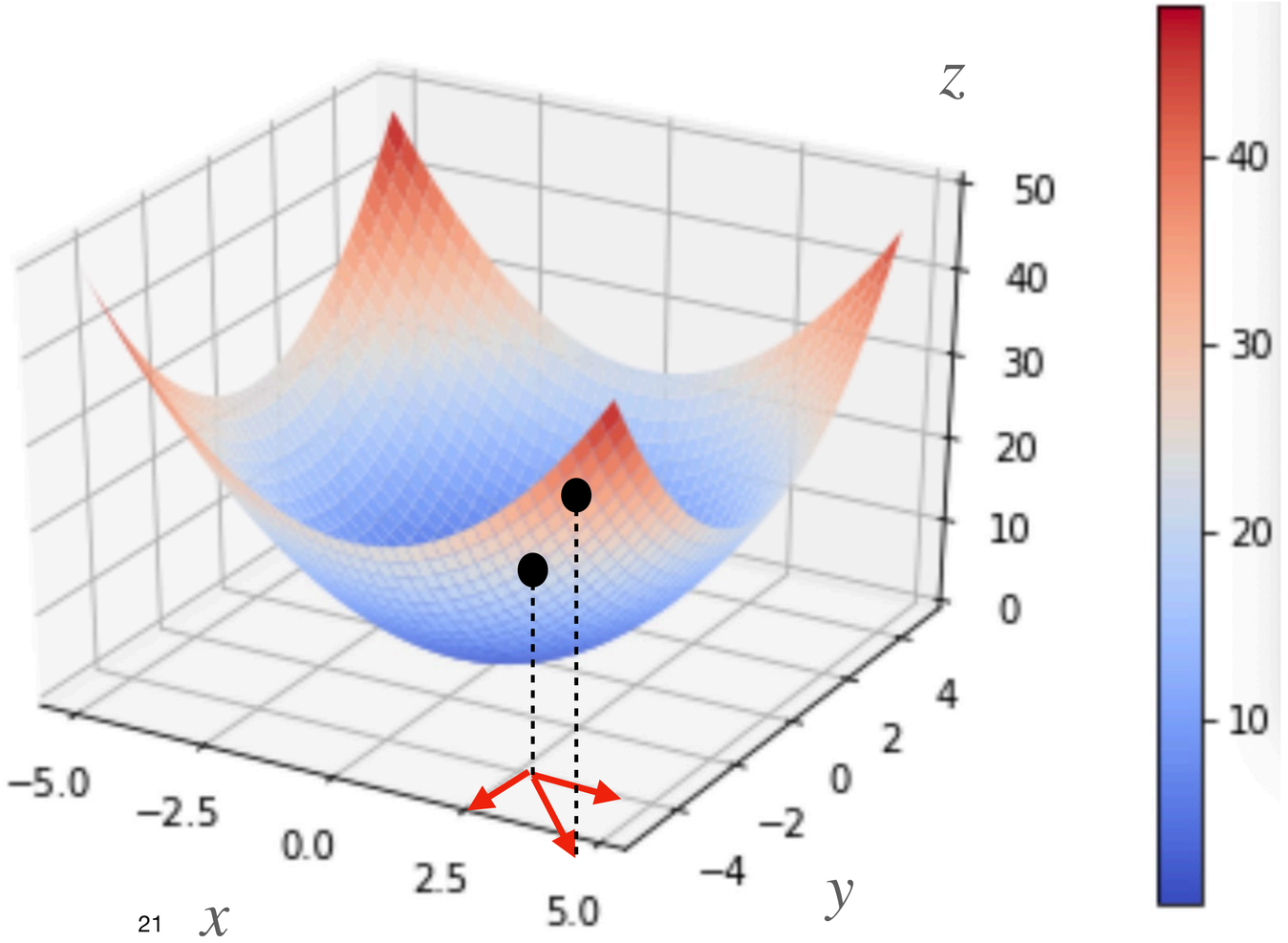


Gradient

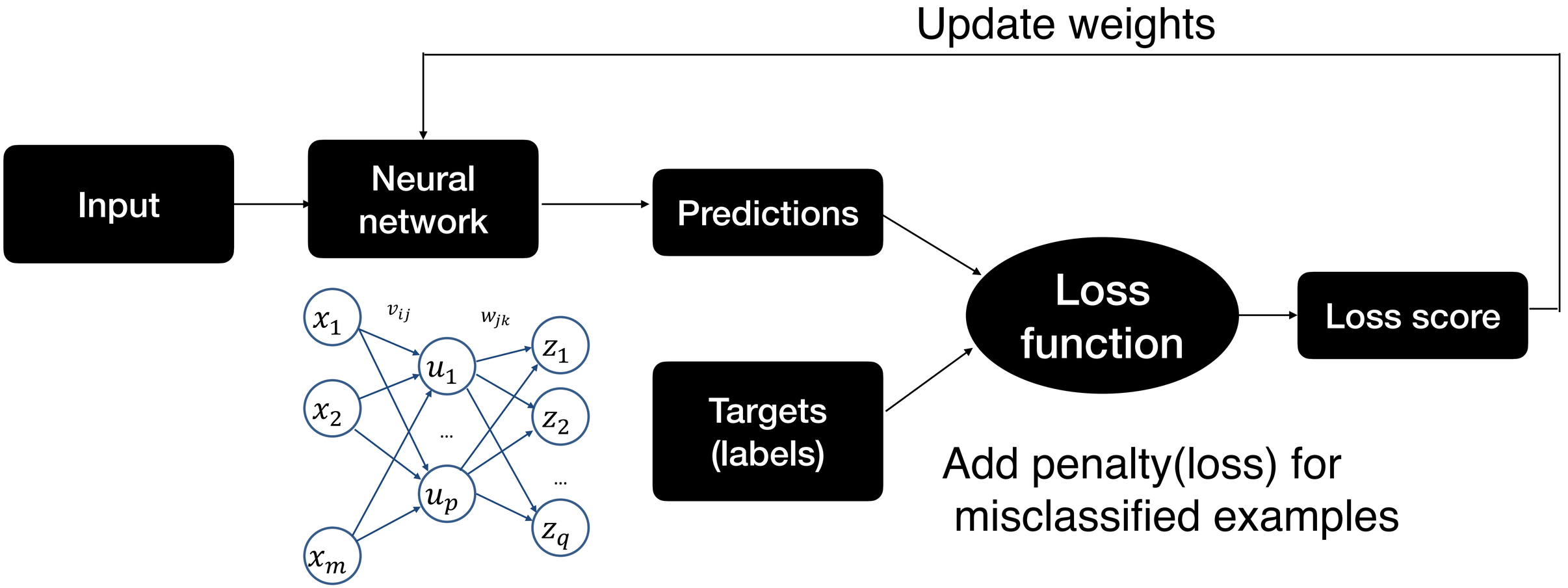
$$G = [\frac{\partial z}{\partial x}, \frac{\partial z}{\partial y}]$$

Move input (x,y) along OPPOSITE direction of gradient vector: decrease output f(x,y)

$$z = f(x, y) = x^2 + y^2$$



Loss = function (weights)



Loss = function (weights)

To reduce loss, update weights:

$$w_i^{new} = w_i^{old} - \eta * \Delta L(w_i)$$

$$\Delta L(w_i) = \frac{\partial L}{\partial w_i}$$

$$\eta : \ leanring \ rate$$

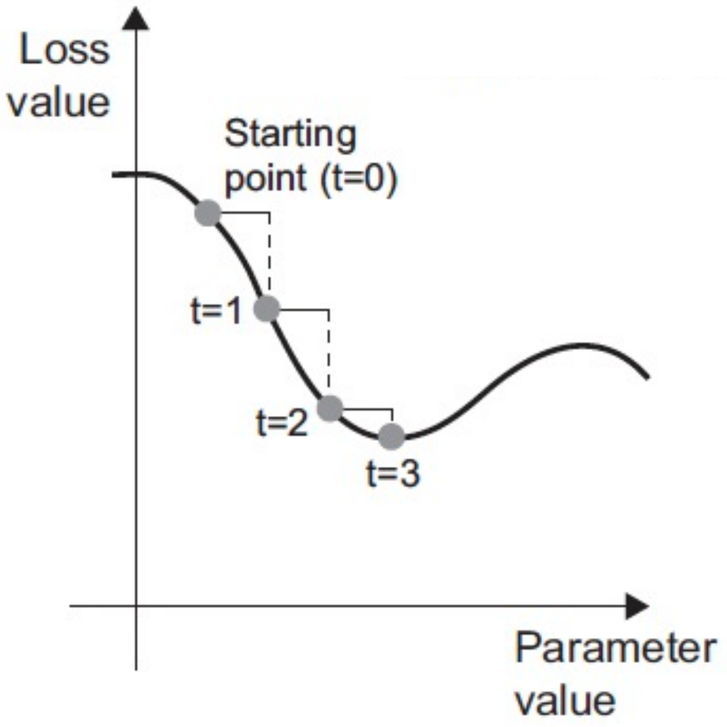


Figure 2.11 in Deep learning with python by Francois Chollet

Loss = function (weights)

η : *leanring rate*

Small η: local optimal value

Large η: random location

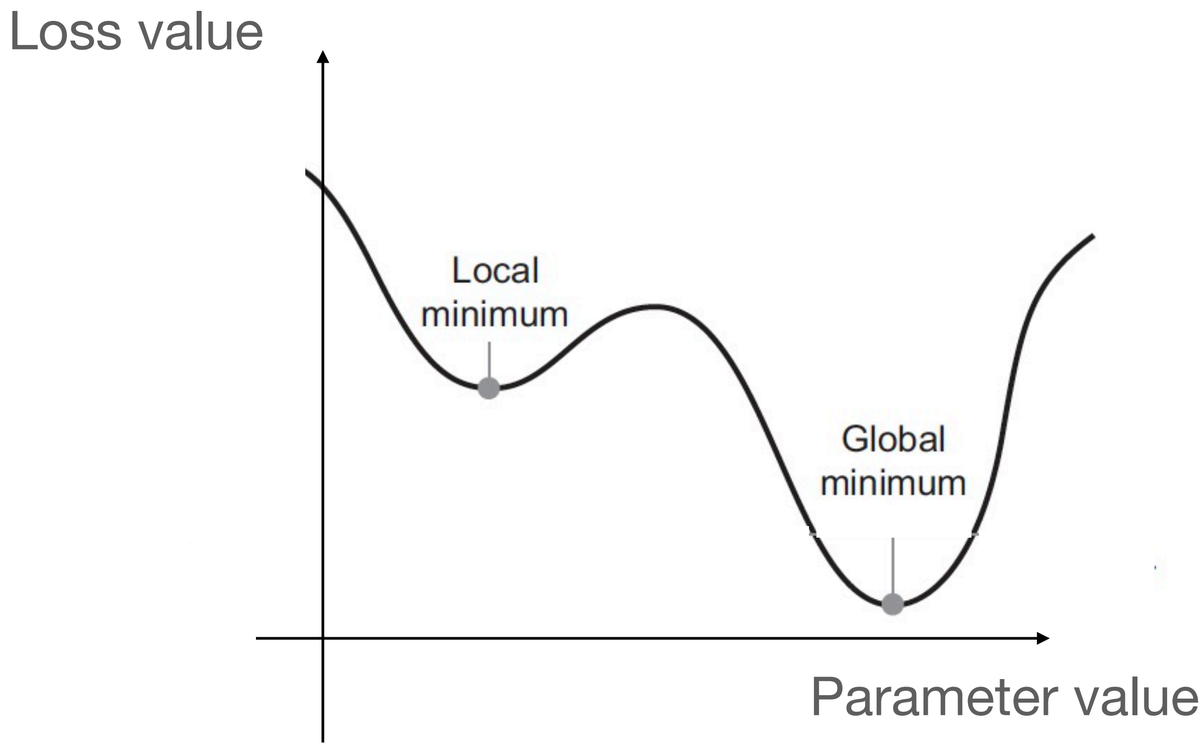


Figure 2.13 in Deep learning with python by Francois Chollet

Gradient descent Algorithm

- Randomly shuffle/split all training examples in B batches
- Choose initial $\boldsymbol{\theta}^{(1)}$
- For i from 1 to T
- For j from 1 to B
- Do gradient descent update using data from batch j

Iterations over the entire dataset are called *epochs*

- Advantage of such an approach: computational feasibility for large datasets

Stochastic gradient descent for perceptron

Choose initial guess $\boldsymbol{w}^{(0)}$, $k = 0$

For i from 1 to T (epochs)

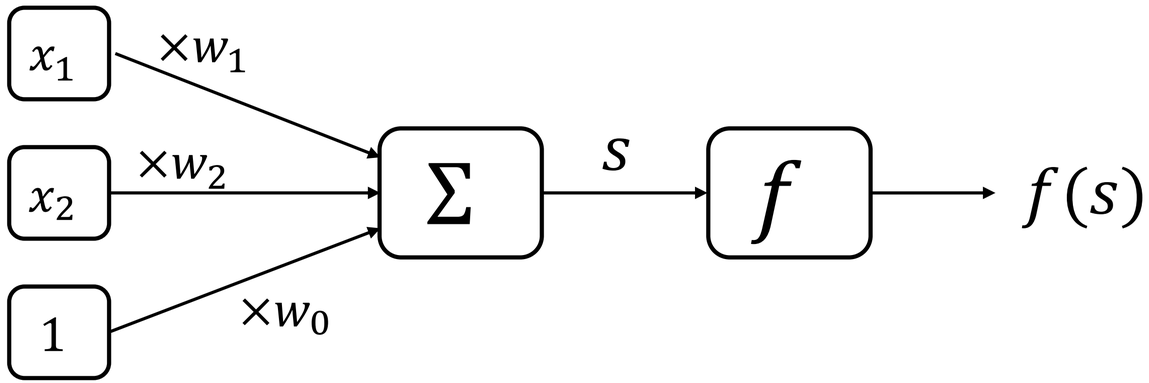
For j from 1 to N (training examples)

Consider example $\{\boldsymbol{x}_j, y_j\}$

Update*: $\boldsymbol{w}^{(k++)} = \boldsymbol{w}^{(k)} - \eta \boldsymbol{\nabla} L(\boldsymbol{w}^{(k)})$

Simple example: Perceptron Model

Encode class: A: +1, B:-1

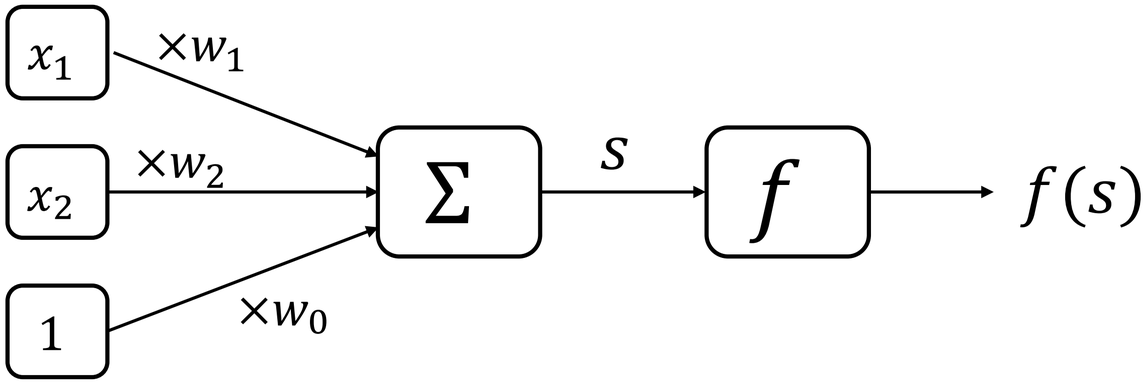


x_1	x_2	y
0	0	-1
0	1	-1
1	0	-1
1	1	1

$$if\ s \geq 0: \ \ prediction\ f(s) = 1$$
$$if\ s < 0: \ \ prediction\ f(s) = -1$$

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

Simple example: Loss



$$L(s, y) = \max(0, -ys)$$

$$= \max(0, -y \cdot \sum_{i=0}^{m} x_i w_i)$$

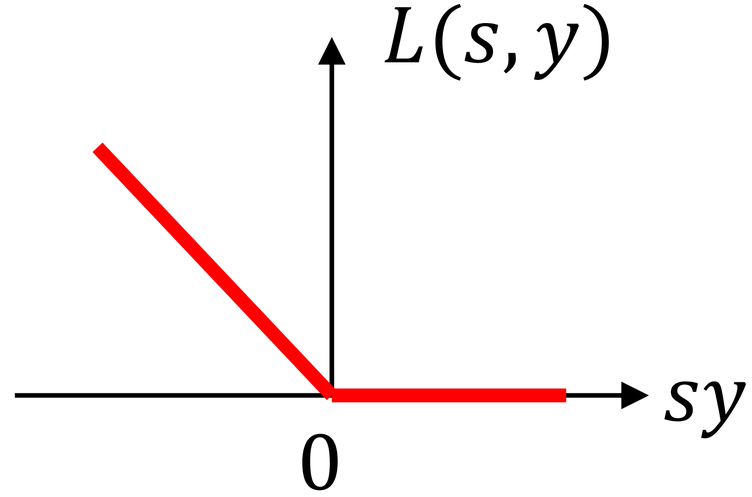
x_1	x_2	y
0	0	-1
0	1	-1
1	0	-1
1	1	1

Simple example: gradient

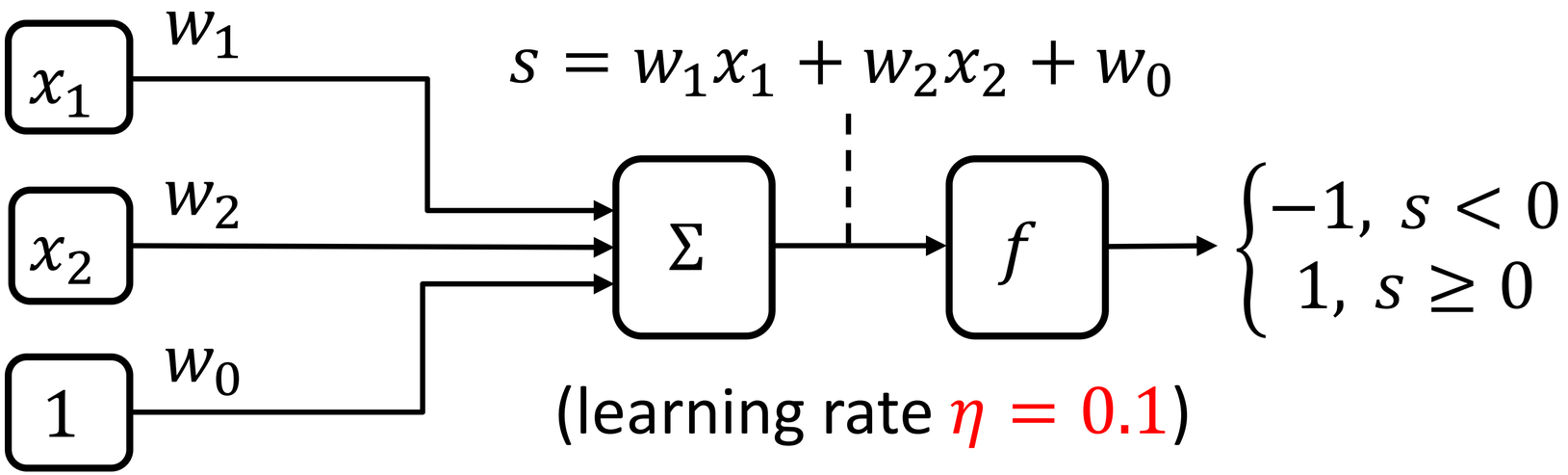
$$L(w) = \max(0, -ys) = \max(0, -y\sum_{i=0}^{m} x_i w_i)$$

$$\frac{\partial L}{\partial w_i} = -yx_i \ when \ sy < 0$$

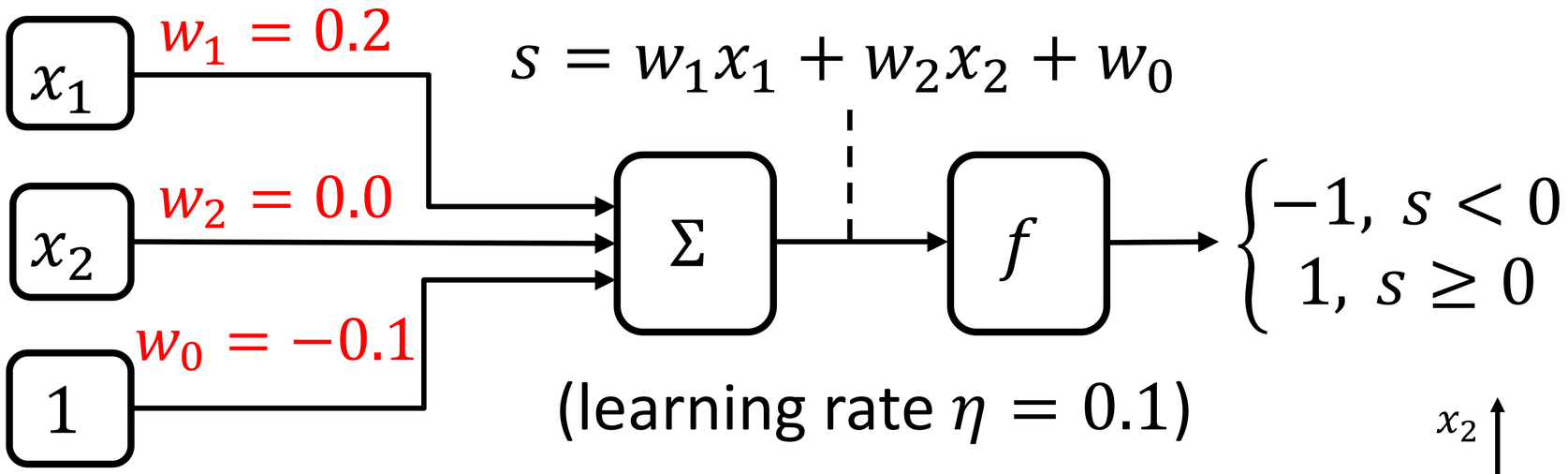
$$\frac{\partial L}{\partial w_i} = 0 \ when \ sy > 0$$

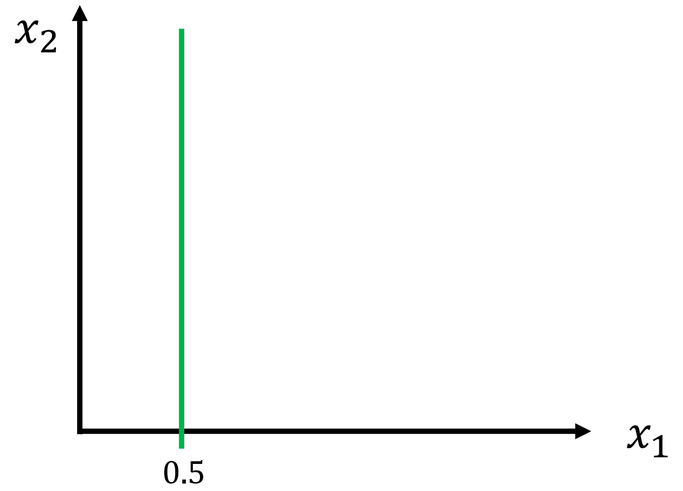


Learn perceptron on the simple example: basic setup

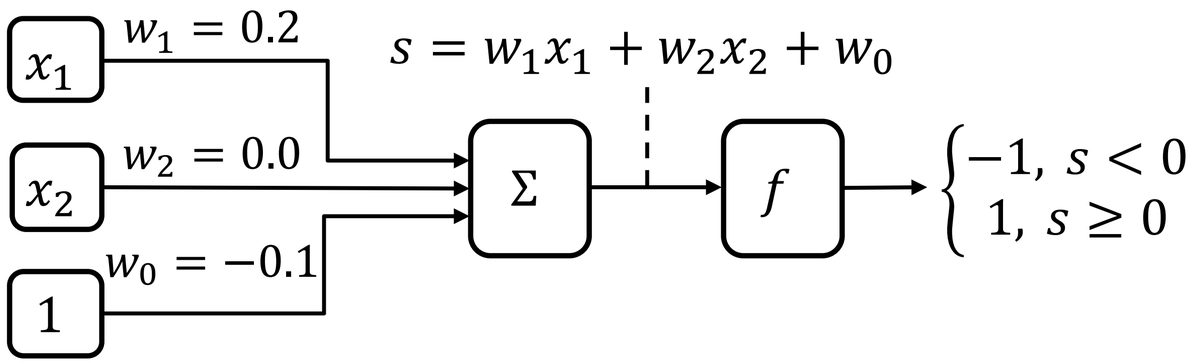


Learn perceptron on the simple example: Start with random weights





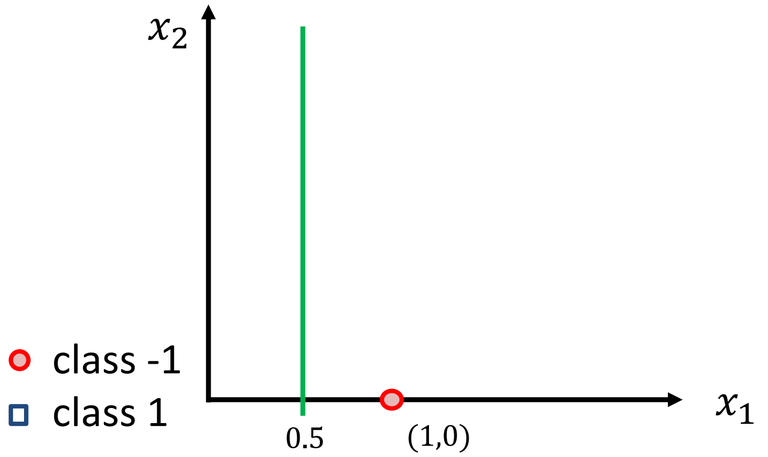
Learn perceptron on the simple example: epoch 1, data point 1



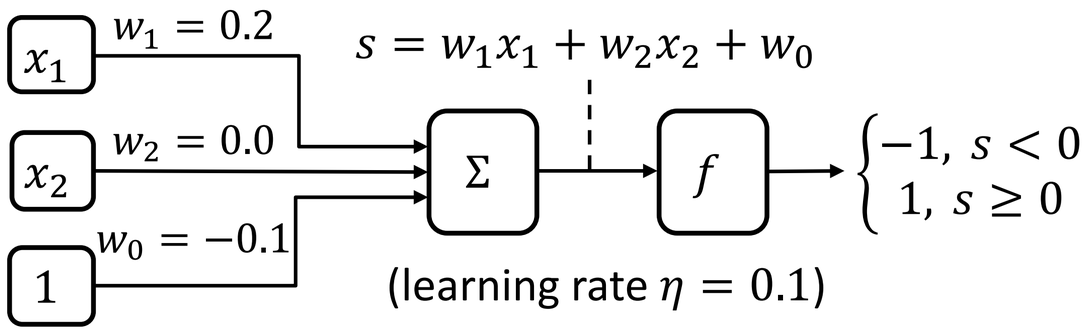
x_1	x_2	y
0	0	-1
0	1	-1
1	0	-1
1	1	1

Prediction s on (1,0):

$$w_1 \times 1 + w_2 \times 0 + w_0 \times 1 = 0.1 > 0$$



Learn perceptron on the simple example: epoch 1, data point 1

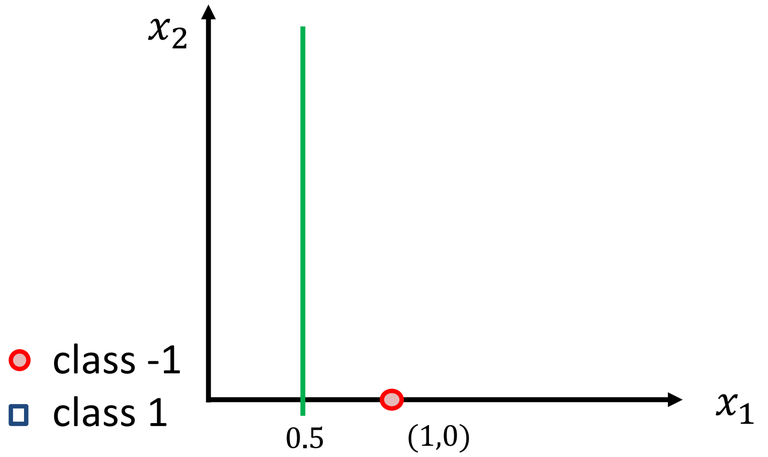


x_1	x_2	y
0	0	-1
0	1	-1
1	0	-1
1	1	1

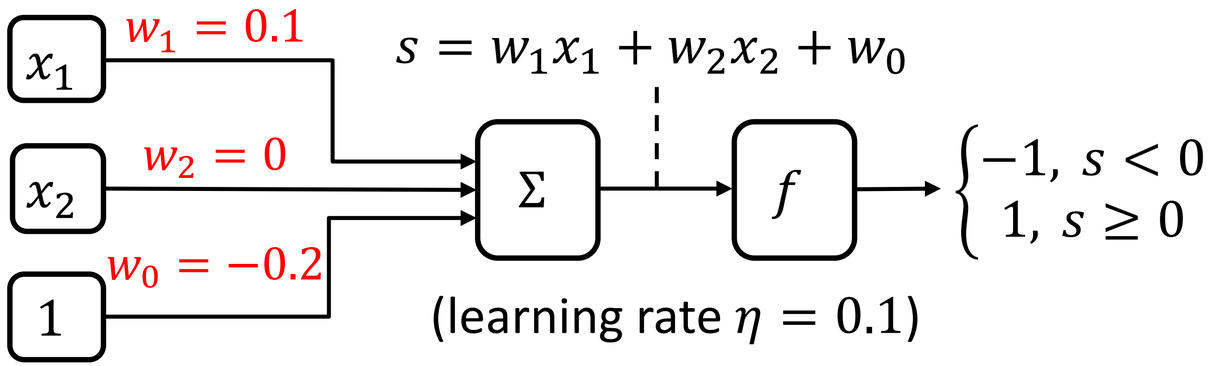
Gradient on (1,0):

s>0, y=-1, sy<0: $\dfrac{\partial L}{\partial w_i} = -yx_i$

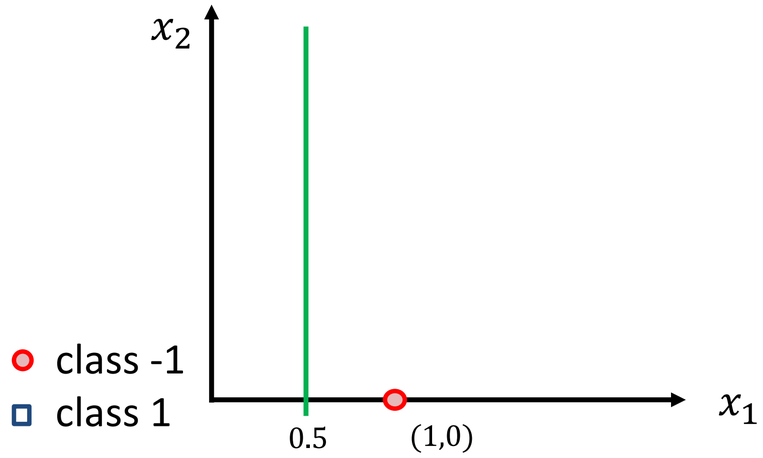
$$w_1 \leftarrow w_1 - \eta \cdot 1 = 0.1$$
$$w_2 \leftarrow w_2 - \eta \cdot 0 = 0$$
$$w_0 \leftarrow w_0 - \eta \cdot 1 = -0.2$$

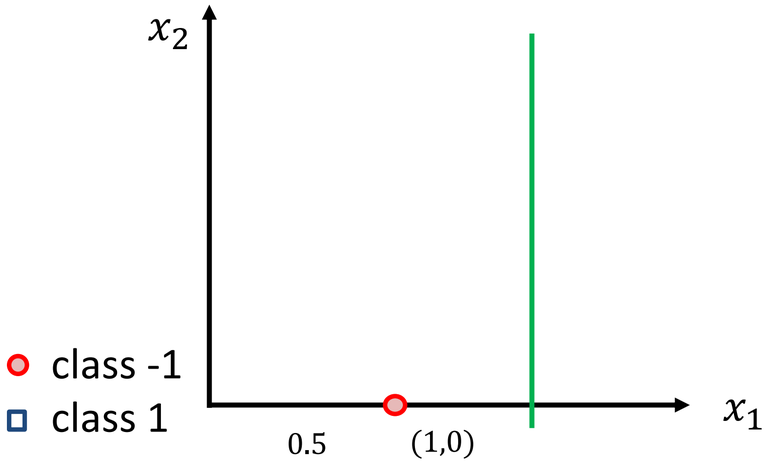


Learn perceptron on the simple example: update weights

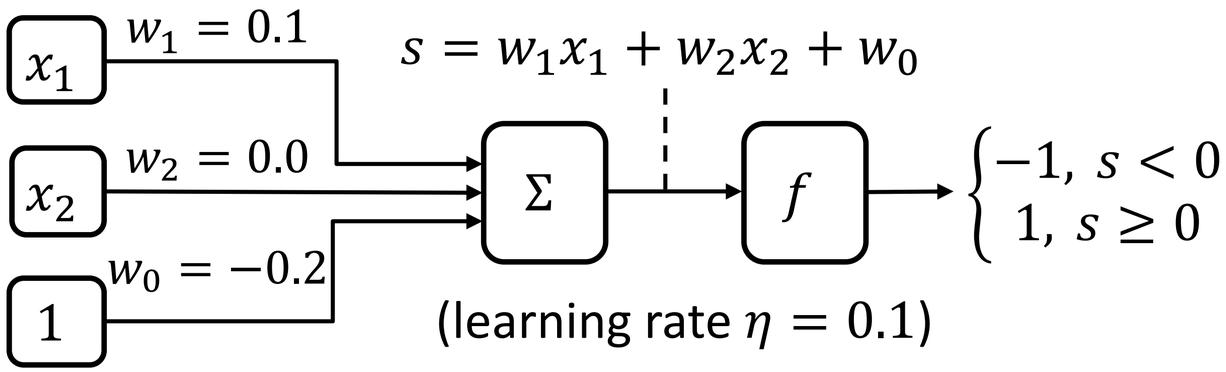


x_1	x_2	y
0	0	-1
0	1	-1
1	0	-1
1	1	1





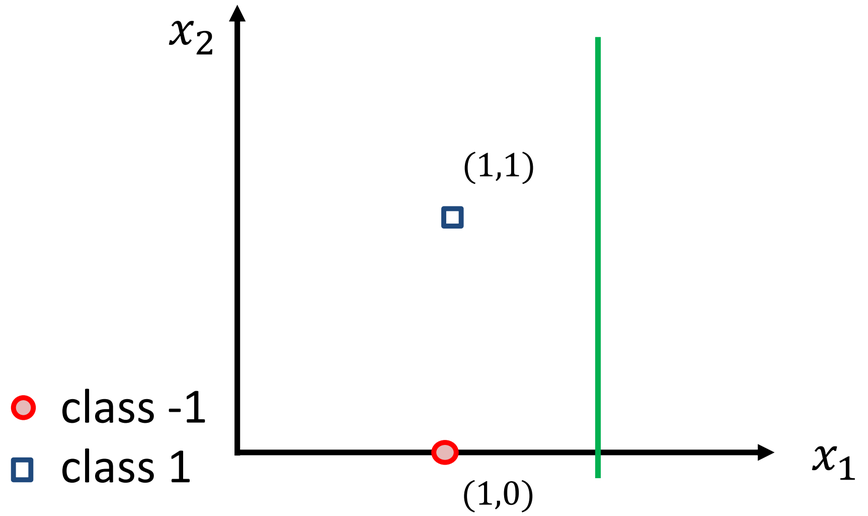
Learn perceptron on the simple example: epoch 1, data point 2



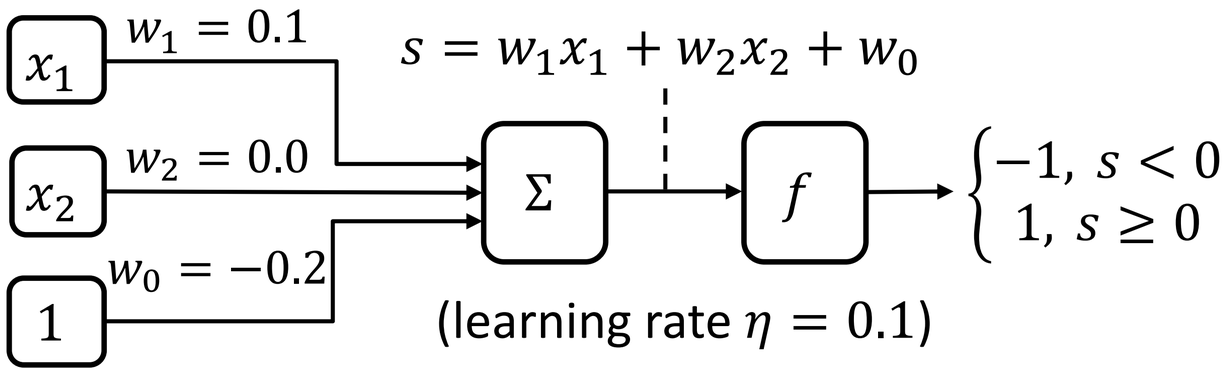
x_1	x_2	y
0	0	-1
0	1	-1
1	0	-1
1	1	1

Prediction s on (1,1):

$$w_1 \times 1 + w_2 \times 1 + w_0 \times 1 = -0.1 < 0$$



Learn perceptron on the simple example: epoch 1, data point 2

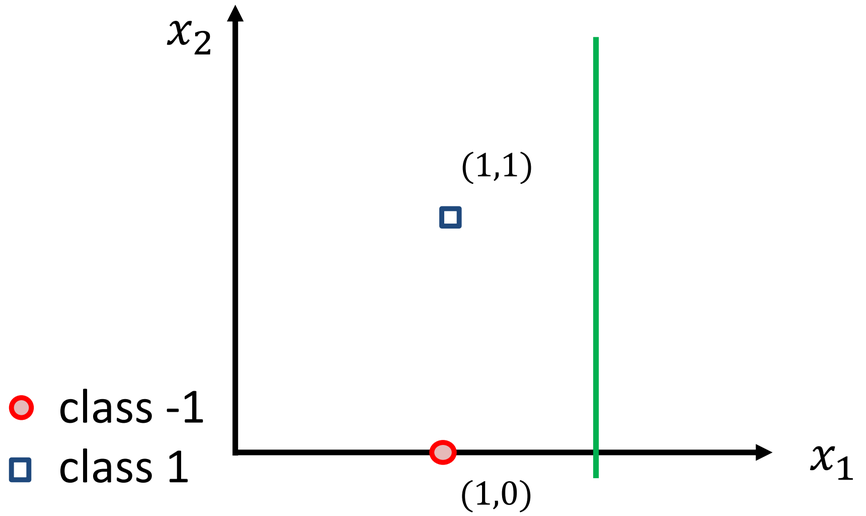


x_1	x_2	y
0	0	-1
0	1	-1
1	0	-1
1	1	1

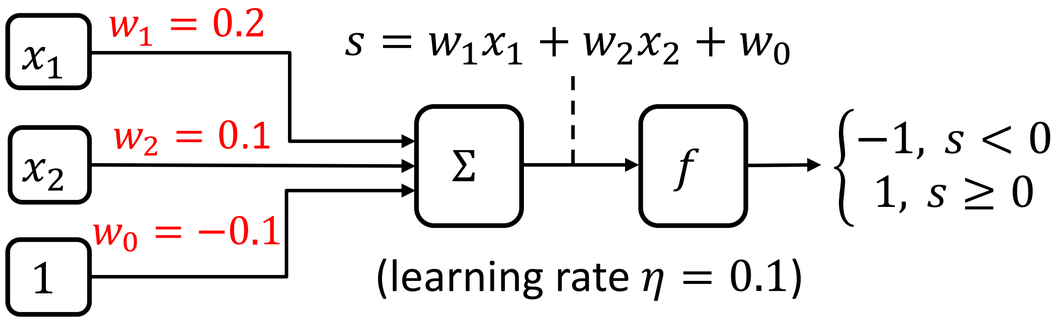
Gradient on (1,1):

s<0, y=1, sy<0: $\frac{\partial L}{\partial w_i} = -yx_i$

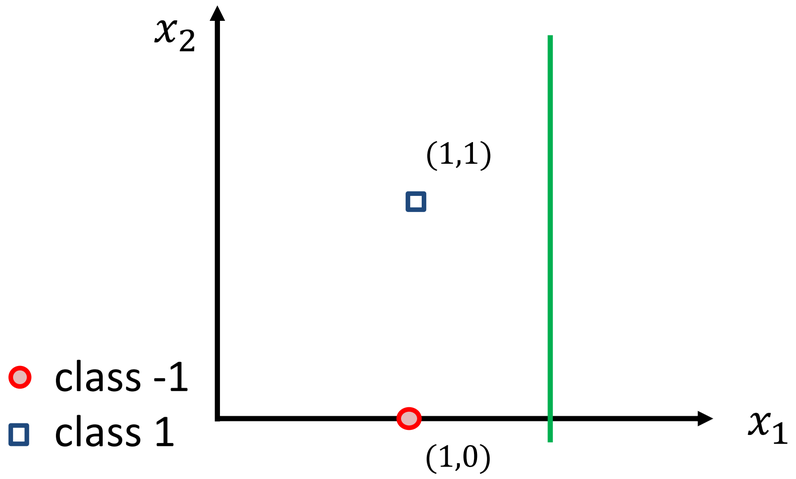
$$w_1 \leftarrow w_1 - \eta \cdot (-1) = 0.2$$
$$w_2 \leftarrow w_2 - \eta \cdot (-1) = 0.1$$
$$w_0 \leftarrow w_0 - \eta \cdot (-1) = -0.1$$

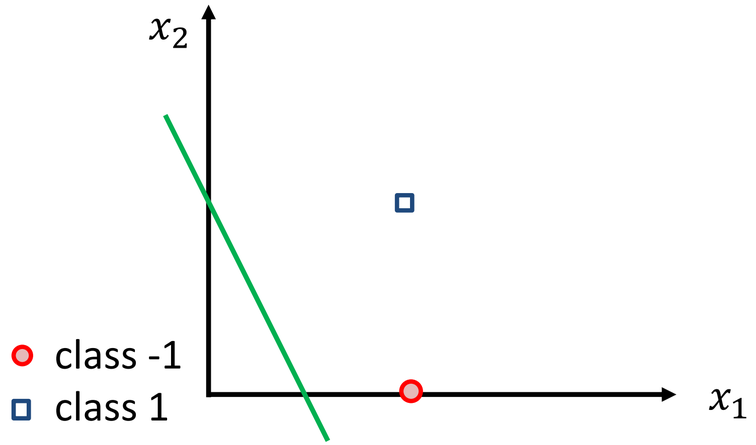


Learn perceptron on the simple example: Update weights

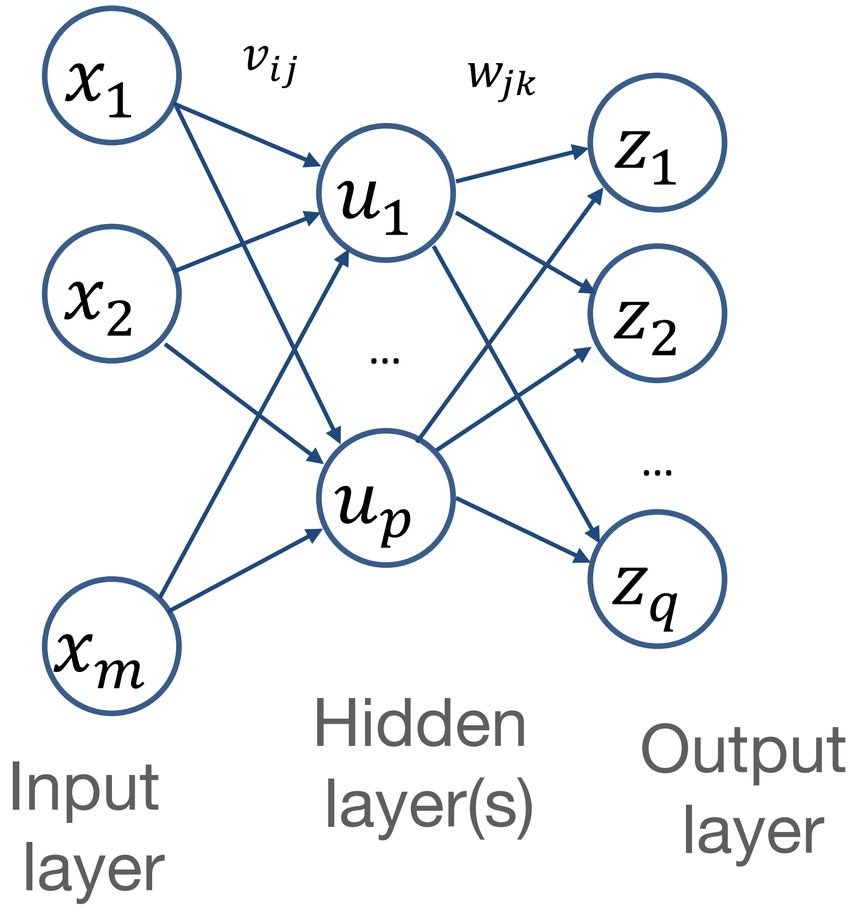


x_1	x_2	y
0	0	-1
0	1	-1
1	0	-1
1	1	1





Stochastic gradient descent for multi-layer perceptron



Stochastic gradient descent for multiple-layer perceptron

Choose initial guess $\boldsymbol{\theta}^{(0)}$, $k = 0$

Here $\boldsymbol{\theta}$ is a set of all weights form all layers

For i from 1 to T (epochs)

For j from 1 to N (training examples)

Consider example $\{\boldsymbol{x}_j, y_j\}$

Update: $\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} - \eta \nabla L(\boldsymbol{\theta}^{(k)})$; $k \leftarrow k+1$

Need to compute partial derivatives $\frac{\partial L}{\partial v_{ij}}$ and $\frac{\partial L}{\partial w_j}$

Chain rule

$$Given\ z = g(u) \quad u = f(x) \Rightarrow \frac{dz}{dx} = \frac{dz}{du}\frac{du}{dx}$$

$$Example : z = \sin(x^2)$$

$$z = sin(u) \qquad \frac{dz}{du} = \cos(u)$$

$$\uparrow \qquad\qquad \downarrow$$

$$u = x^2 \qquad \frac{dz}{dx} = \frac{dz}{du}\frac{du}{dx} = 2x\cos(u)$$

Multi-layer perceptron: Function composition

Forward prediction

$$z = f(s) = sigmoid(s) = \frac{1}{1 + e^{-s}}$$

$$\uparrow$$

$$s = \sum_{j=0}^{p} w_j u_j$$

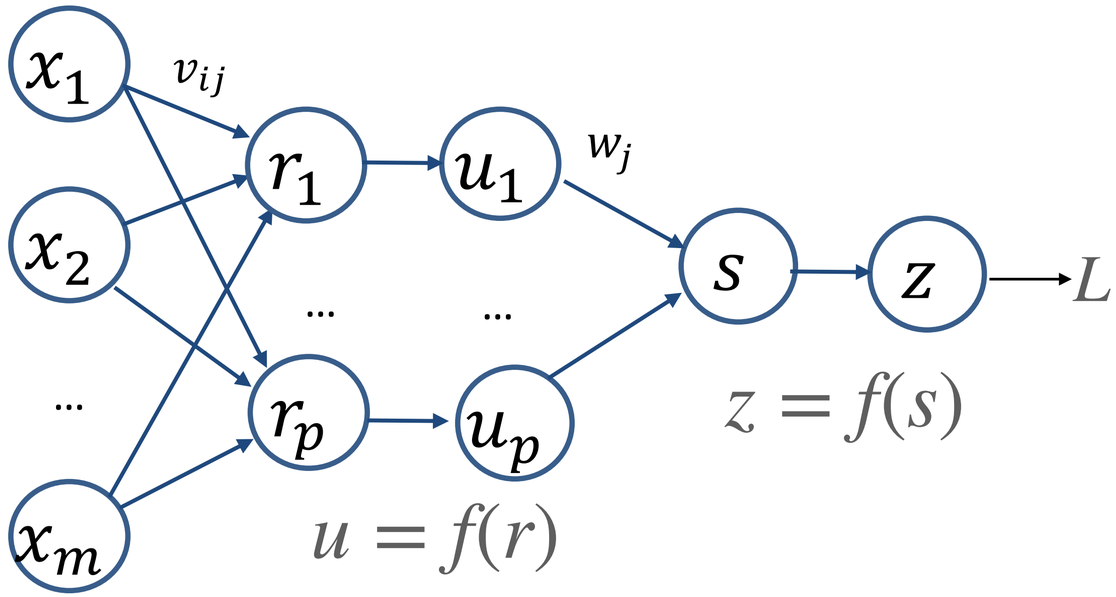
$$\uparrow$$

$$u_j = f(r_j) = sigmoid(r_j) = \frac{1}{1 + e^{-r_j}}$$

$$\uparrow$$

$$r_j = \sum_{i=0}^{m} x_i v_{ij}$$

$$x \rightarrow r \rightarrow u \rightarrow s \rightarrow z$$

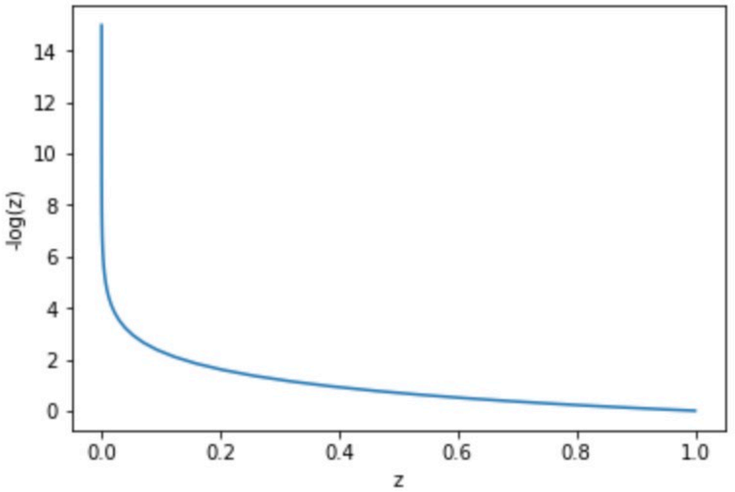


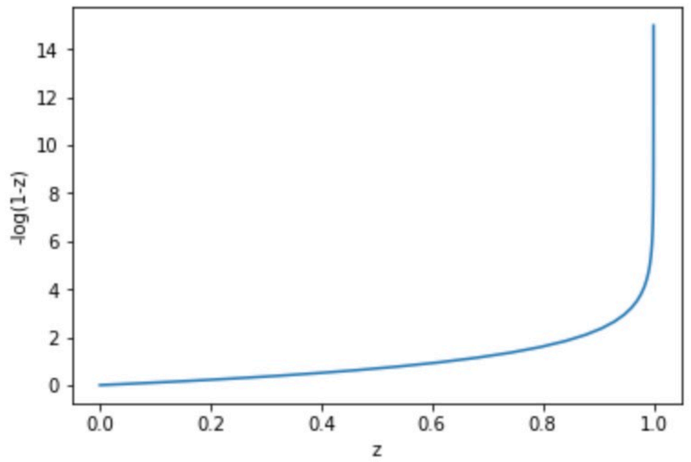
f : *activation functions*

Binary cross-entropy

$$L = -\left[y \log(z) + (1 - y)\log(1 - z))\right]$$

y: Labels of the data
y=1 for positive class or 0 for negative class





Multi-layer perceptron: Chain rule

Backward propagation:

$$\frac{\partial L}{\partial w_j} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial s}\frac{\partial s}{\partial w_j}$$

$$L = -\left[y\log(z) + (1-y)\log(1-z)\right]$$

$$\uparrow$$

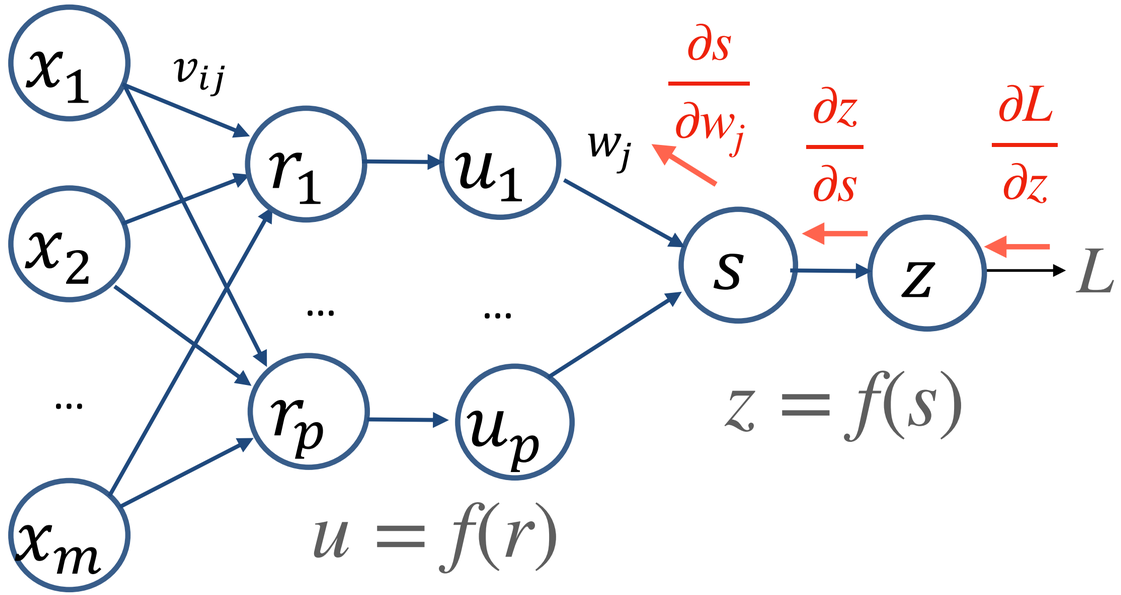
$$z = sigmoid(s) = \frac{1}{1+e^{-s}}$$

$$\uparrow$$

$$s = \sum_{j=0}^{p} u_j w_j$$

Forward

$$x \rightarrow r \rightarrow u \rightarrow s \rightarrow z$$



Multi-layer perceptron: Chain rule

Backward propagation:

$$\frac{\partial L}{\partial v_{ij}} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial s}\frac{\partial s}{\partial u_j}\frac{\partial u_j}{\partial r_j}\frac{\partial r_j}{\partial v_{ij}}$$

$$s = \sum_{j=0}^{p} u_j w_j$$

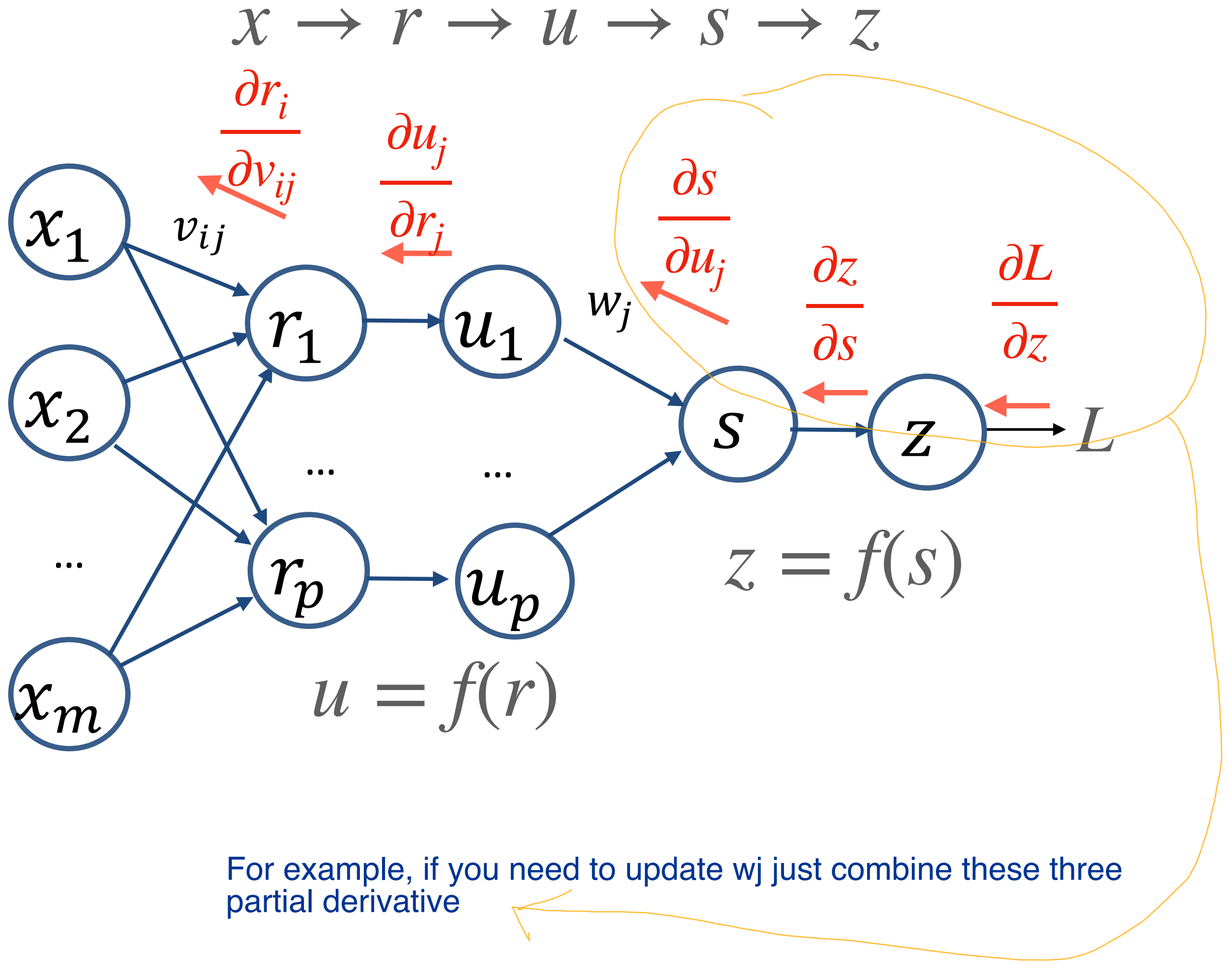
$$\uparrow$$

$$u_j = sigmoid(r_j) = \frac{1}{1+e^{-r_j}}$$

$$\uparrow$$

$$r_j = \sum_{i=0}^{m} x_i v_{ij}$$

Forward



For example, if you need to update wj just combine these three partial derivative

Forward prediction

$$L = -\left[y \log(z) + (1-y)\log(1-z)\right]$$

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

$$s = \sum_{j=0}^{p} u_j w_j$$

$$u_j = sigmoid(r_j) = \frac{1}{1+e^{-r_j}}$$

$$r_j = \sum_{i=0}^{m} x_i v_{ij}$$

Backward propagation

$$\frac{\partial L}{\partial z} = -\frac{y}{z} + \frac{1-y}{1-z}$$

$$\frac{\partial L}{\partial s} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial s} = z - y, \quad \frac{\partial z}{\partial s} = z(1-z)$$

$$\frac{\partial L}{\partial w_j} = \frac{\partial L}{\partial s}\frac{\partial s}{\partial w_j} = \frac{\partial L}{\partial s}u_j \qquad \frac{\partial L}{\partial u_j} = \frac{\partial L}{\partial s}\frac{\partial s}{\partial u_j} = \frac{\partial L}{\partial s}w_j$$

$$\frac{\partial L}{\partial r_j} = \frac{\partial L}{\partial u_j}\frac{\partial u_j}{\partial r_j} = \frac{\partial L}{\partial u_j}(u_j)(1-u_j)$$

$$\frac{\partial L}{\partial v_{ij}} = \frac{\partial L}{\partial r_j}\frac{\partial r_j}{\partial v_{ij}} = \frac{\partial L}{\partial r_j}x_i$$

Summary

- What is deep learning& difference with traditional machine learning?
- How to train neural network using gradient descent algorithm?
- How to train a perceptron using stochastic gradient descent?
- What is chain rule?
- How to perform forward prediction and back propagation in multilayer perceptron?

Next：Convolutional neural network

References

- Deep learning with python by Francois Chollet, chapter 1 & 2.4
- Pattern Recognition and Machine Learning by Chris Bishop, chapter 4.1.7, 5.1-5.3.