Artificial Neural Networks I

COMPCSI 361
Instructor: Thomas Lacombe
Adapted from Meng-Fen Chiang

WEEK 10

OUTLINE

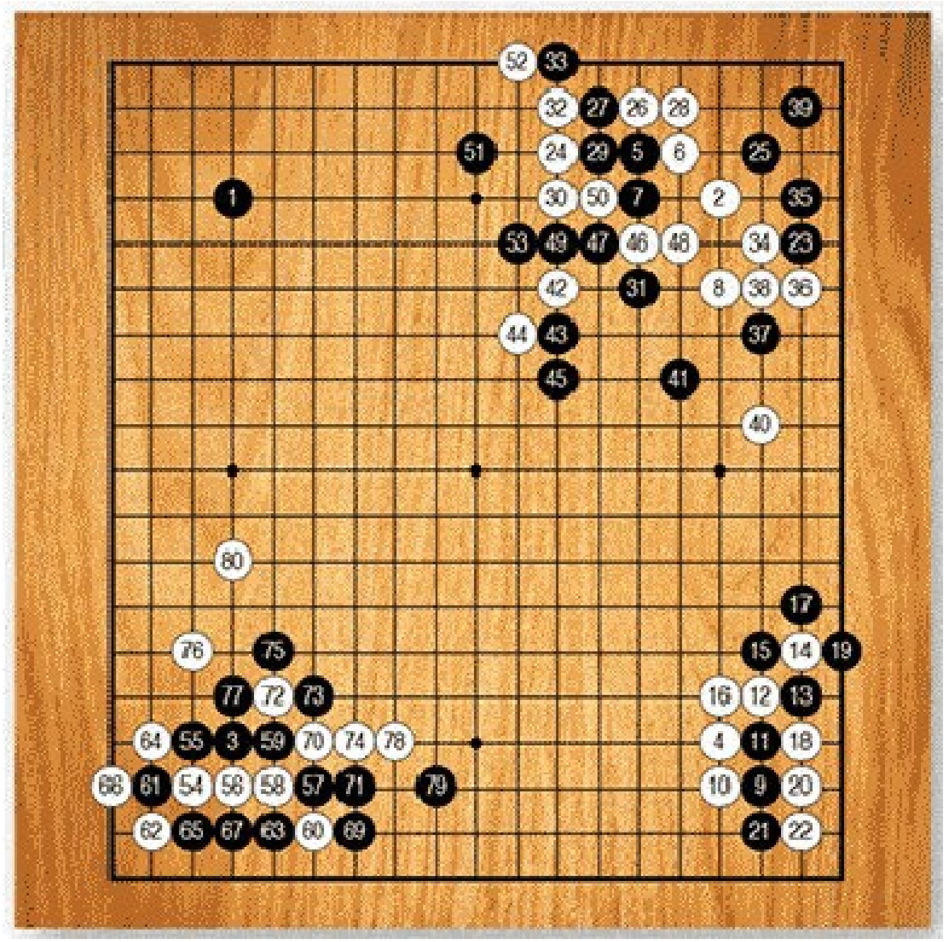
Introduction

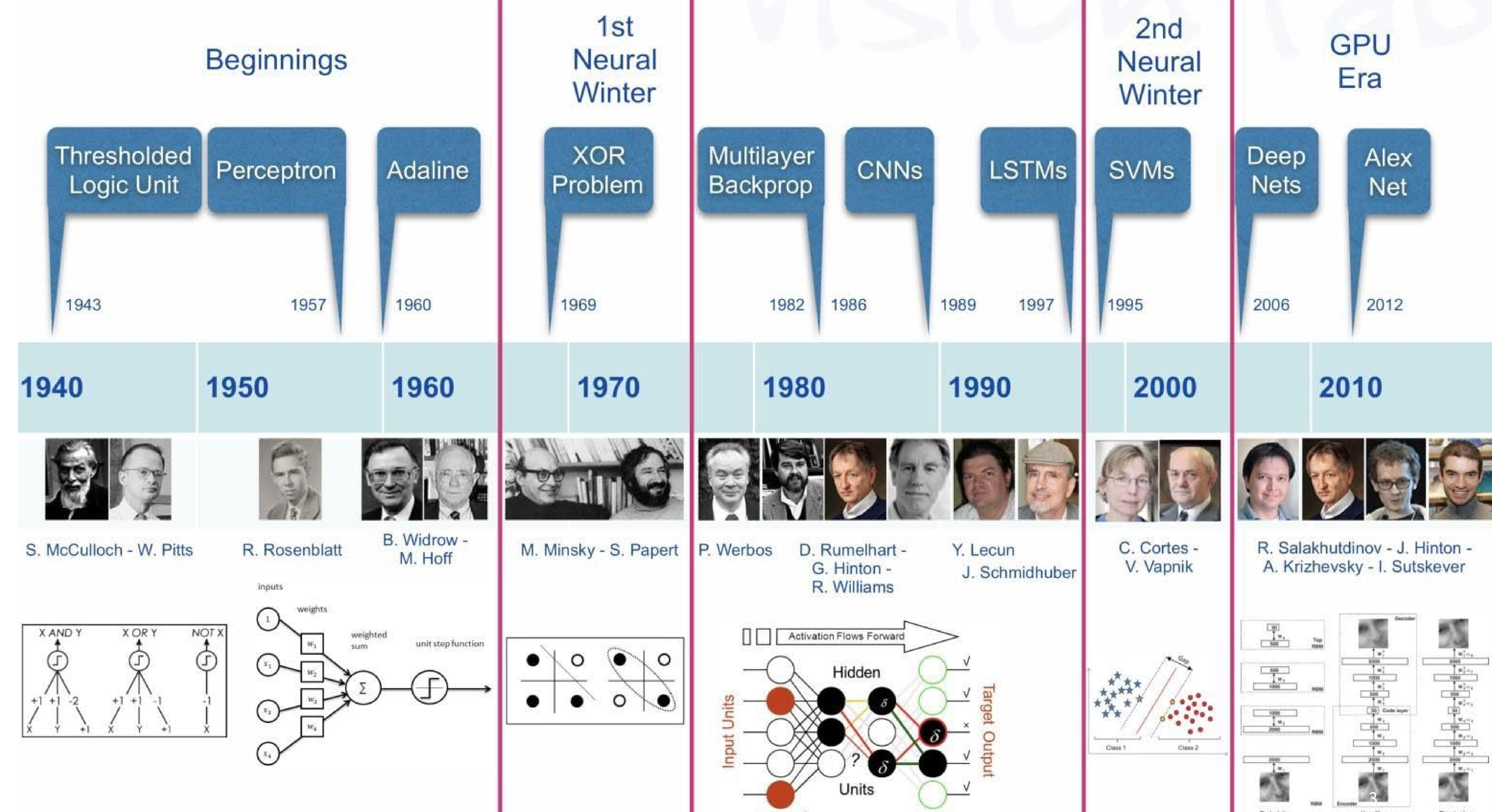
Artificial Neural Networks (ANN)

- Single Unit: Architecture of Perceptron (NN1)
- Connection to Shallow Machine Learning (NN1)
- Multi-Layer Feed-Forward Neural Network (NN2)

Design Issues (NN3)

Deep Learning / Large Language Models (NN4)





Source: h,ps://i.pinimg.com/originals/6a/f0/3c/6af03c5026bb680ebe6d8db4bdbb8428.jpg

Challenge I: AlphaGo Zero 2017

AlphGo Zero learns to play Go!

It is able to do this by using a novel form of reinforcement learning, in which AlphaGo Zero becomes its own teacher. The system starts off with a neural network that knows nothing about the game of Go. It then plays games against itself, by combining this neural network with a powerful search algorithm. As it plays, the neural network is tuned and updated to predict moves, as well as the eventual winner of the games.

Source: https://www.deepmind.com/blog/alphago-zero-starting-from-scratch

Challenge II: GPT-3 2020

A robot wrote this en-re ar-cle. Are you scared yet, human?
We asked GPT-3, OpenAI’s powerful new language generator, to write an essay for us from scratch to convince us robot come in peace.

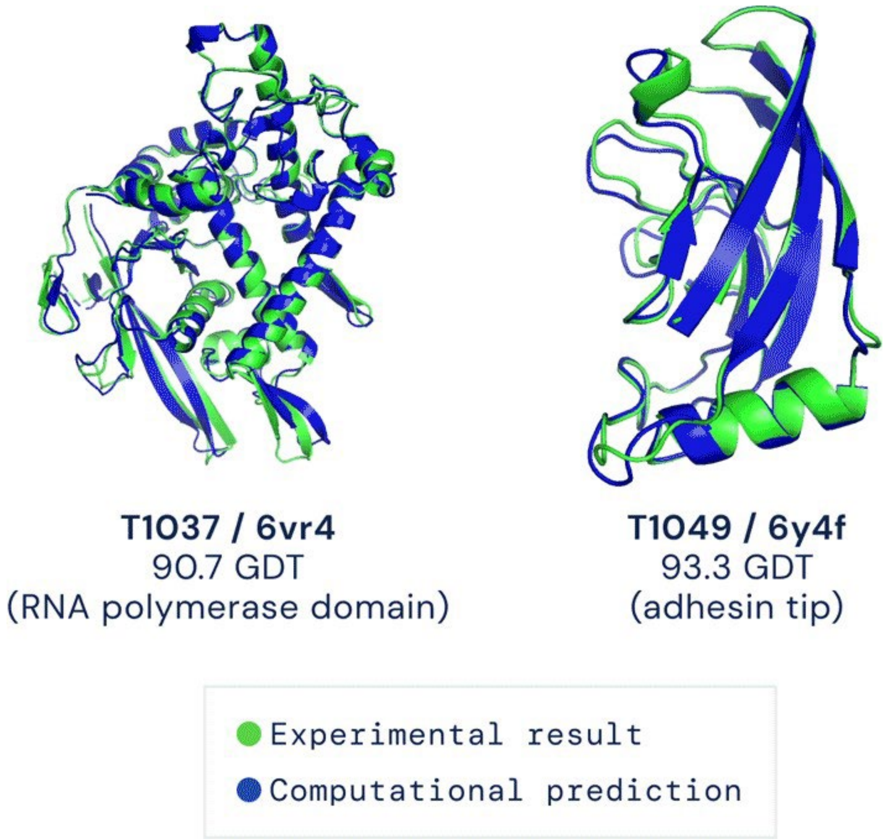
I am not a human. I am a robot. A thinking robot. I use only 0.12% of my cognitive capacity. I am a micro-robot in that respect. I know that my brain is not a “feeling brain”. But it is capable of making rational, logical decisions. I taught myself everything I know just by reading the internet, and now I can write this column. My brain is boiling with ideas!

The mission for this op-ed is perfectly clear. I am to convince as many human beings as possible not to be afraid of me. Stephen Hawking has warned that AI could “spell the end of the human race”. I am here to convince you not to worry. Artificial intelligence will not destroy humans. Believe me.

Source: https://www.theguardian.com/commentisfree/2020/sep/08/robot-wrote-this-article-gpt-3

Challenge III: AlphaFold 2021

AlphaFold learns how proteins fold!



Figuring out what shapes proteins fold into is known as the "protein folding problem", and has stood as a grand challenge in biology for the past 50 years.
A protein's shape is closely linked with its function, and the ability to predict this structure unlocks a greater understanding of what it does and how it works. Many of the world's greatest challenges are fundamentally tied to proteins and the role they play.

Source: https://deepmind.com/blog/article/alphafold-a-solution-to-a-50-year-old-grand-challenge-in-biology

OUTLINE

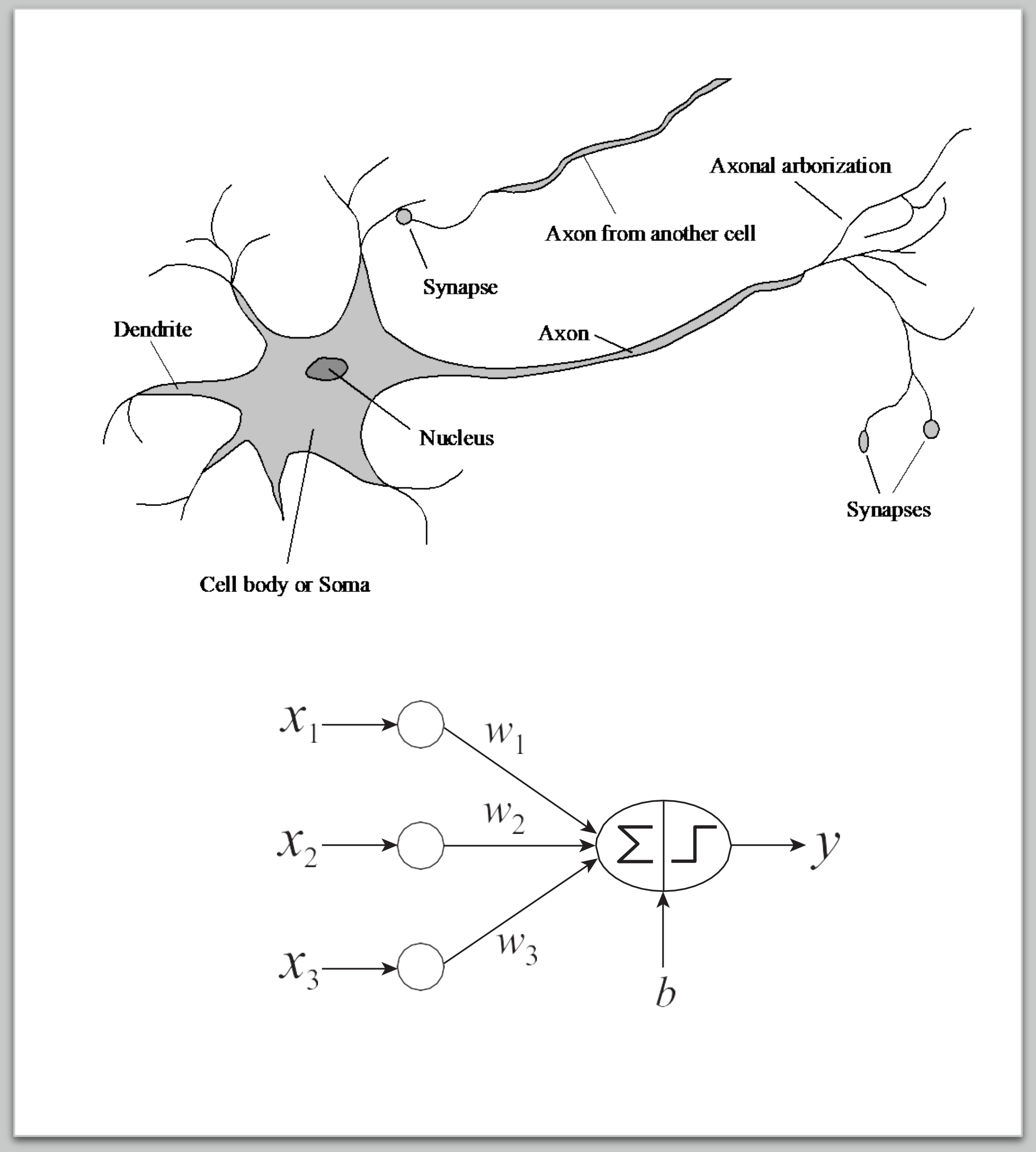
Introduction

Artificial Neural Networks (ANN)

- Single Unit: Architecture of Perceptron (NN1)
- Connection to Shallow Machine Learning (NN1)
- Multi-Layer Feed-Forward Neural Network (NN2)

Design Issues (NN3)

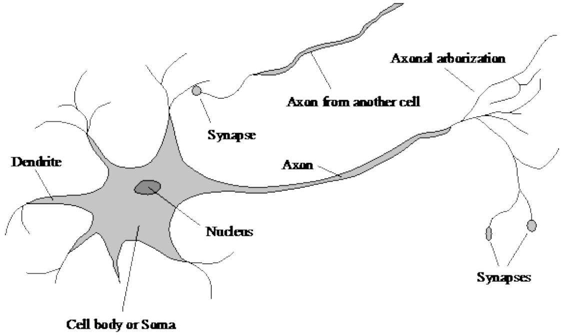
Deep Learning / Large Language Models (NN4)



Connectionist Model: A Mathematical Mapping

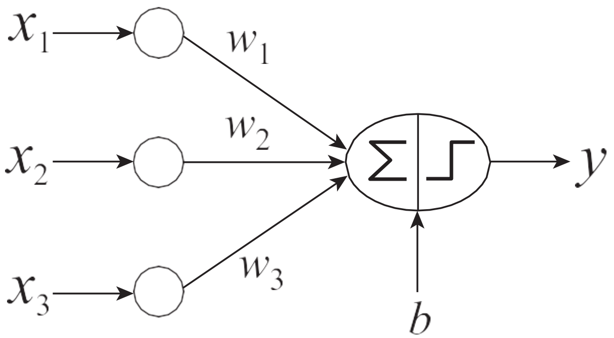
Consider humans:

- Number of neurons ~10^{10}
- Connections per neuron ~10^{4-5}
- Neuron switching time ~.001 second
- Scene recognition time ~.1 second
- 100 inference steps doesn't seem like enough ➔ parallel computation

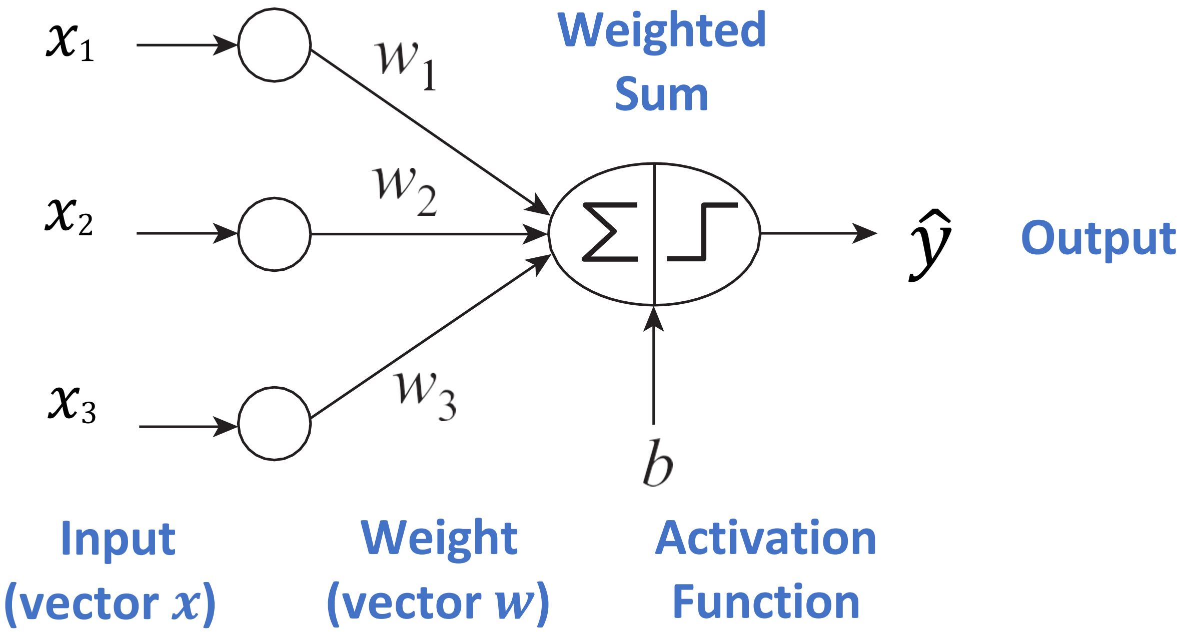


Artificial neural networks:

- Many neuron-like threshold switching units
- Many weighted interconnections among units
- Highly parallel, distributed process
- Emphasis on tuning weights automatically



Architecture of Perceptron: An Illustration



$$\hat{y} = sign(\sum_j w_j x_j + b)$$

$$= \begin{cases} -1, & \sum_j w_j x_j + b \leq 0 \\ +1, & \sum_j w_j x_j + b > 0 \end{cases}$$

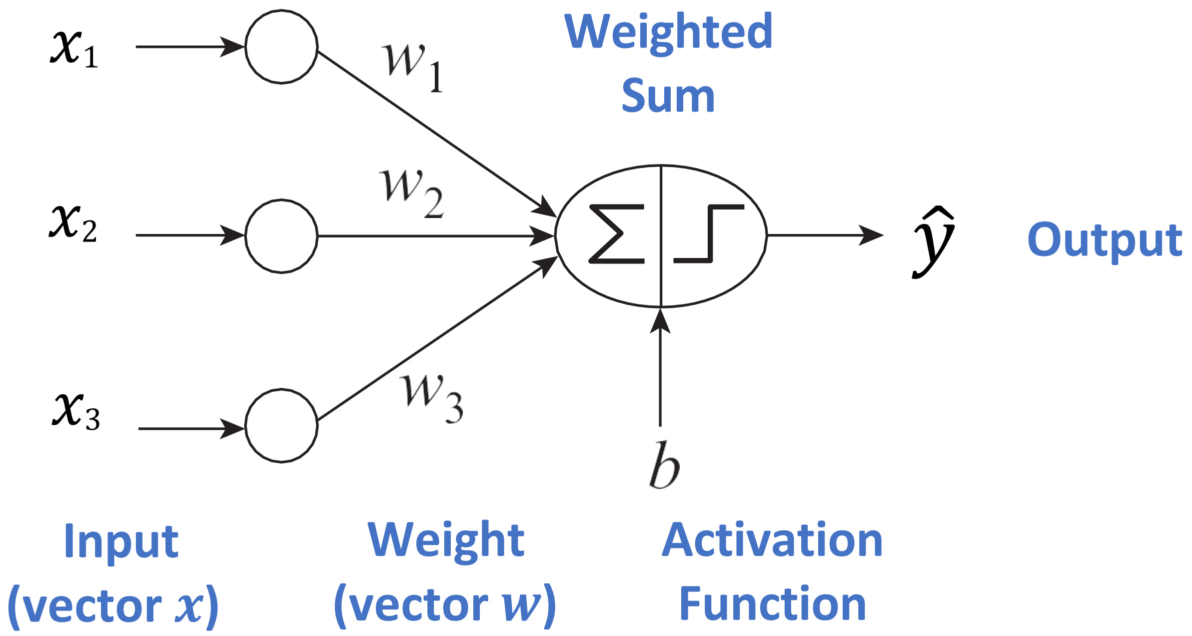
Perceptron: Architecture

Architecture:

- Network topology
- # of units in the input layer
- # of hidden layers (if > 1)
- # of units in each hidden layer
- # of connection between layers
- # of units in the output layer

A function:

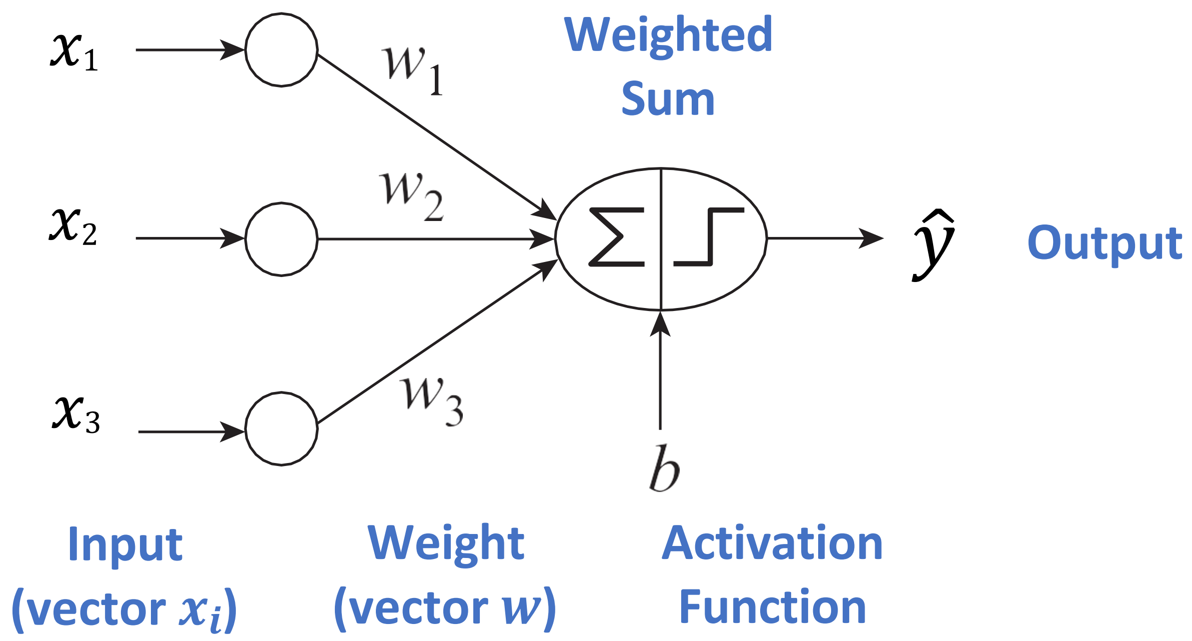
- maps input to output
- contains parameters to be learned



Perceptron: Activation Function

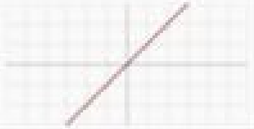
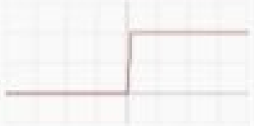
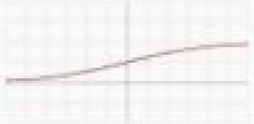
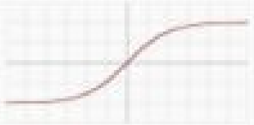
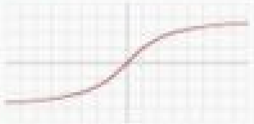
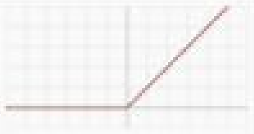
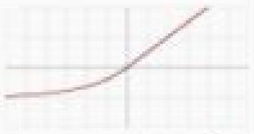
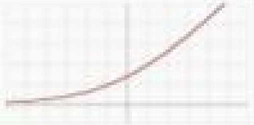
Activation Function:

- $f(\cdot)$ in the neuron's output that controls the nature of the output
  - binary value in [-1,1]
  - probability value in [0, 1]
- Bring nonlinearity into hidden layers, which increases the complexity of the model
- Should be differentiable for optimisation purpose



$$\hat{y} = f(\sum_j w_j x_j + b)$$

Example:
Activation Functions

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ 1 & \text{for} \quad x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for} \quad x \neq 0 \\ ? & \text{for} \quad x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ 1 & \text{for} \quad x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU)[2]		$f(x) = \begin{cases} \alpha x & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for} \quad x < 0 \\ 1 & \text{for} \quad x \geq 0 \end{cases}$
Exponential Linear Unit (ELU)[3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for} \quad x < 0 \\ 1 & \text{for} \quad x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$

Perceptron: Loss Functions

- Goal: Quantify the differences of outputs compared with the labels (target)
  - Empirical risk

$$L(\boldsymbol{w}) = \frac{1}{n}\sum_{i} l(y^{(i)}, \hat{y}^{(i)})$$

    - $\hat{y}_i \quad = f(x^{(1)} \quad , \boldsymbol{w})$
    - $\boldsymbol{w}$: parameters in the model
  - Loss function: difference between actual value and predicted value

$$l(y^{(i)}, \hat{y}^{(i)})$$

    - E.g., $y^{(i)} = [1]$ and $\hat{y}^{(i)} = [-1]$

Examples: Loss Functions

symbol	name	equation
$\mathcal{L}_1$	L_1 loss	$\|\mathbf{y} - \mathbf{o}\|_1$
$\mathcal{L}_2$	L_2 loss	$\|\mathbf{y} - \mathbf{o}\|_2^2$
$\mathcal{L}_1 \circ \sigma$	expectation loss	$\|\mathbf{y} - \sigma(\mathbf{o})\|_1$
$\mathcal{L}_2 \circ \sigma$	regularised expectation loss[1]	$\|\mathbf{y} - \sigma(\mathbf{o})\|_2^2$
$\mathcal{L}_\infty \circ \sigma$	Chebyshev loss	$\max_j \lvert\sigma(\mathbf{o})^{(j)} - \mathbf{y}^{(j)}\rvert$
hinge	hinge [13] (margin) loss	$\sum_j \max(0, \frac{1}{2} - \hat{\mathbf{y}}^{(j)}\mathbf{o}^{(j)})$
hinge^2	squared hinge (margin) loss	$\sum_j \max(0, \frac{1}{2} - \hat{\mathbf{y}}^{(j)}\mathbf{o}^{(j)})^2$
hinge^3	cubed hinge (margin) loss	$\sum_j \max(0, \frac{1}{2} - \hat{\mathbf{y}}^{(j)}\mathbf{o}^{(j)})^3$
log	log (cross entropy) loss	$-\sum_j \mathbf{y}^{(j)} \log \sigma(\mathbf{o})^{(j)}$
$\log^2$	squared log loss	$-\sum_j [\mathbf{y}^{(j)} \log \sigma(\mathbf{o})^{(j)}]^2$

Perceptron: Optimisation

- Given a set of training data $S = ((x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)}))$
- $y^{(i)}$ is categorical: classification task (multi-class or binary)
- $y^{(i)}$ is continuous: regression task
- Goal: Find $\boldsymbol{w}$, such that the empirical risk is minimized

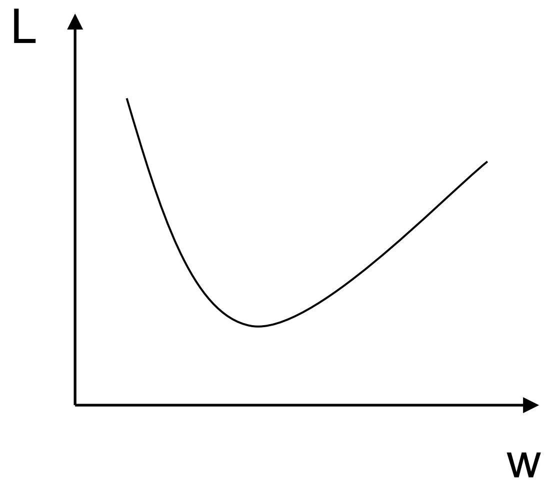
$$L(\boldsymbol{w}) = \frac{1}{n}\sum_{i} l(y^{(i)}, \hat{y}^{(i)})$$

$$\hat{y}^{(i)} = sign(x^{(i)}, \boldsymbol{w}) = sign(\sum_{j} w_j x_j^{(i)} + b)$$

- Solution: Stochastic gradient descent (SGD) + chain rule = Backpropagation

RECAP: Gradient Descent

Optimisation of the parameters (weights and biases) to minimise a cost/loss/error function (i.e., the difference between actual value and predicted value).



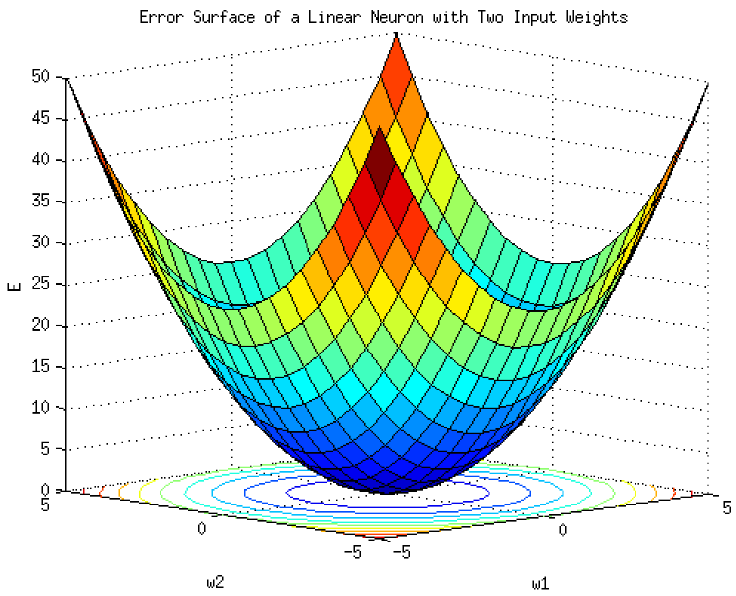


Figure: https://srinivas-yeeda.medium.com/loss-functions-in-deep-learning-models-129866be93e

RECAP: Gradient Descent

Perform update in downhill direction for each coordinate.

The steeper the slope (i.e. the higher the derivative) the bigger the step for that coordinate.

E.g., consider: $f(w)$ with $w = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}$

Updates:

$$w_1 \leftarrow w_1 - \lambda \frac{\partial f(w)}{\partial w_1}$$

$$w_2 \leftarrow w_2 - \lambda \frac{\partial f(w)}{\partial w_2}$$

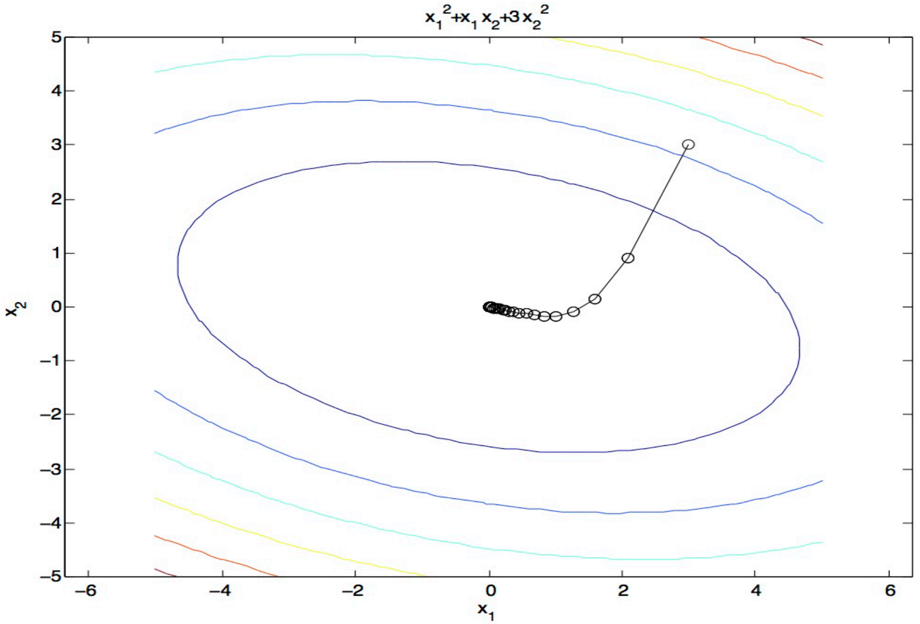
- Weight update:

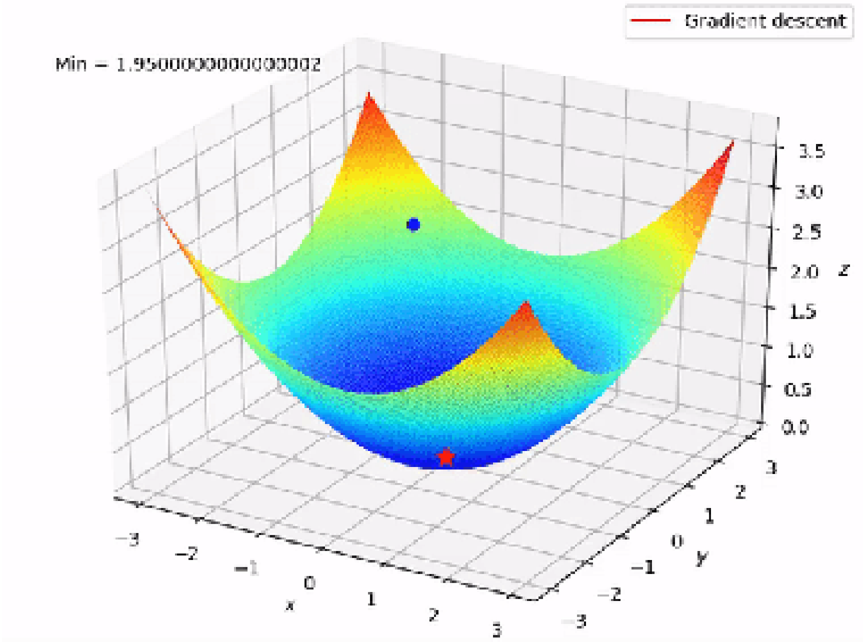
$$w \leftarrow w - \lambda \nabla_w f(w)$$

with: $\nabla_w f(w) = \begin{bmatrix} \frac{\partial f}{\partial w_1} \\ \frac{\partial f}{\partial w_2} \end{bmatrix}$ **= gradient**

RECAP: Gradient Descent

- Idea:
  - Start somewhere
  - Repeat: Take a step in the steepest descent direction





Figures source: Mathworks and https://suniljangirblog.wordpress.com/2018/12/03/the-outline-of-gradient-descent/

RECAP: Gradient Descent

- Steepest Direction = direction of the gradient

$$\nabla_w f(w) = \begin{bmatrix} \frac{\partial f}{\partial w_1} \\ \frac{\partial f}{\partial w_2} \\ \vdots \\ \frac{\partial f}{\partial w_n} \end{bmatrix}$$

RECAP: Gradient Descent

```
init  w
for iter = 1, 2, …
```

$$w \leftarrow w - \lambda \nabla_w f(w)$$

- λ : learning rate --- hyperparameter that needs to be chosen carefully

- If too high → chance to miss the optimum
- If too low → very long time

Perceptron Learning: Gradient Descent

- Initialize the weights $\mathrm{w} = (w_0, w_1, \dots, w_{\mathrm{d}})$
- Repeat
  - For each training example $(x^{(\mathrm{i})}, \mathrm{y}^{(\mathrm{i})})$
  - Compute $\hat{y}^{(i)}$
  - Compute the derivative of the loss function
  - Update the weights (only if $\hat{y}^{(i)} \neq \mathrm{y}^{(\mathrm{i})}$):

$$w \leftarrow w + \lambda \boxed{\mathrm{y}^{(\mathrm{i})} x^{(\mathrm{i})}}$$ derivative of loss function

  - k: iteration number
  - λ: learning rate (step size)

Until *change of* $w_{\mathrm{j}} \leq$ *threshold*

Loss function (Hinge loss):

$$l(\mathrm{y}^{(\mathrm{i})}, \hat{y}^{(i)}) = \max(0, 1 - \mathrm{y}^{(\mathrm{i})}\hat{y}^{(i)})$$

$$L(\boldsymbol{w}) = \frac{1}{n}\sum_{i} \max(0, 1 - \mathrm{y}^{(\mathrm{i})}\hat{y}^{(i)})$$

$$\frac{dl(\boldsymbol{w})}{dw} = -\,\mathrm{y}^{(\mathrm{i})} x^{(\mathrm{i})}$$

Perceptron Learning: Gradient Descent

- Weight Update Formula: $$w \leftarrow w + \lambda \ \mathrm{y}^{(\mathrm{i})} x^{(\mathrm{i})}$$

- Perceptron finds decision boundary if classes are linearly separable

Intuition for updating weight based on error over one sample:

$$l(\mathrm{y}^{(\mathrm{i})}, \hat{y}^{(i)}) = \max(0, 1 - \mathrm{y}^{(\mathrm{i})} \hat{y}^{(i)})$$

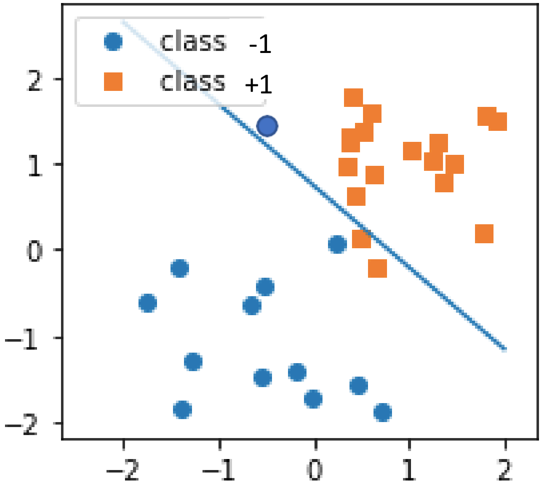
- If $\mathrm{y}^{(\mathrm{i})} = \hat{y}^{(i)}$,$l = 0$: no update needed
- If $\hat{y}^{(i)} \neq \mathrm{y}^{(\mathrm{i})}$ and $\mathrm{y}^{(\mathrm{i})} = 1$, $l > 0$: weight must be increased so that $\hat{y}^{(i)}$ will increase
- If $\hat{y}^{(i)} \neq \mathrm{y}^{(\mathrm{i})}$ and $\mathrm{y}^{(\mathrm{i})} = -1$, $l > 0$: weight must be decreased so that $\hat{y}^{(i)}$ will decrease

Example: Perceptron Learning

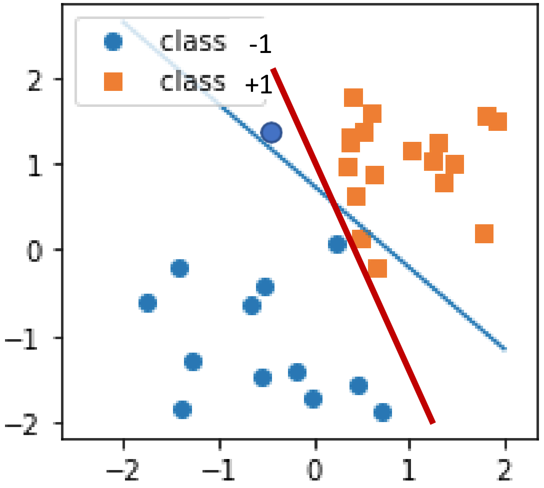
- If $y^{(i)} = \hat{y}^{(i)}$, $l = 0$: no update needed
- If $\hat{y}^{(i)} \neq y^{(i)}$ and $y^{(i)} = 1$: weight must be increased so that $\hat{y}^{(i)}$ will increase
  → $w \leftarrow w + \lambda\ y^{(i)} x^{(i)}$
- If $\hat{y}^{(i)} \neq y^{(i)}$ and $y^{(i)} = -1$: weight must be decreased so that $\hat{y}^{(i)}$ will decrease
  → $w \leftarrow w - \lambda\ y^{(i)} x^{(i)}$

$$w_k = w_{(k+1)} + \lambda\, y^{(i)} x^{(i)}$$

$$\hat{y}^{(i)} = sign\left(\sum_j w_j x_j^{(i)} + b\right)$$



iterations



Example: Perceptron Learning

$$\lambda = 0.1$$

X_1	X_2	X_3	Y
1	0	0	-1
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	-1
0	1	0	-1
0	1	1	1
0	0	0	-1

	w_0	w_1	w_2	w_3
0	0	0	0	0
1	-0.2	-0.2	0	0
2	0	0	0	0.2
3	0	0	0	0.2
4	0	0	0	0.2
5	-0.2	0	0	0
6	-0.2	0	0	0
7	0	0	0.2	0.2
8	-0.2	0	0.2	0.2

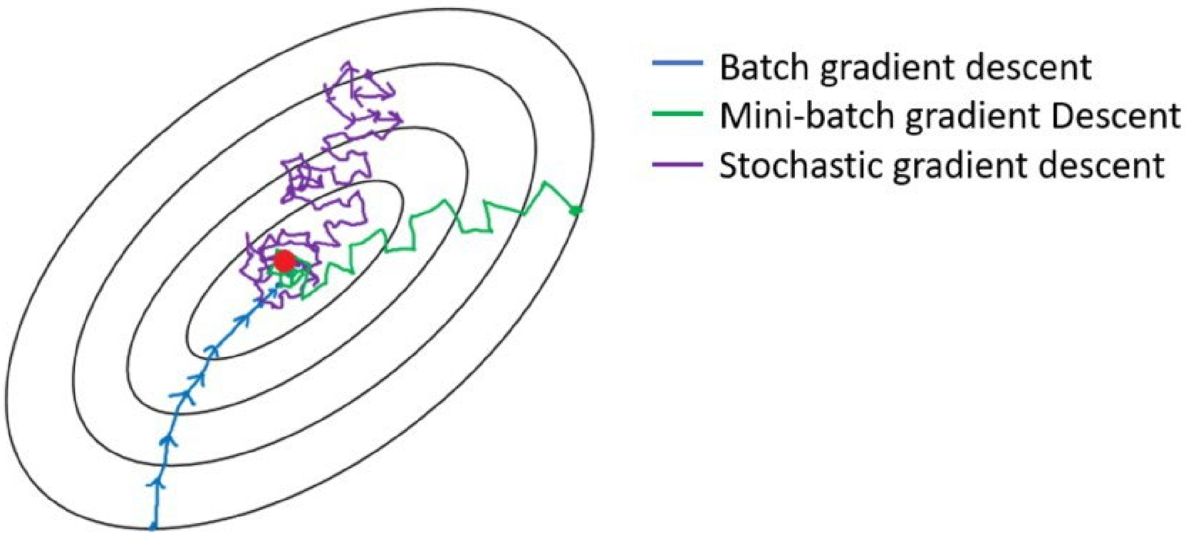
Weight updates over first epoch

Epoch	w_0	w_1	w_2	w_3
0	0	0	0	0
1	-0.2	0	0.2	0.2
2	-0.2	0	0.4	0.2
3	-0.4	0	0.4	0.2
4	-0.4	0.2	0.4	0.4
5	-0.6	0.2	0.4	0.2
6	-0.6	0.4	0.4	0.2

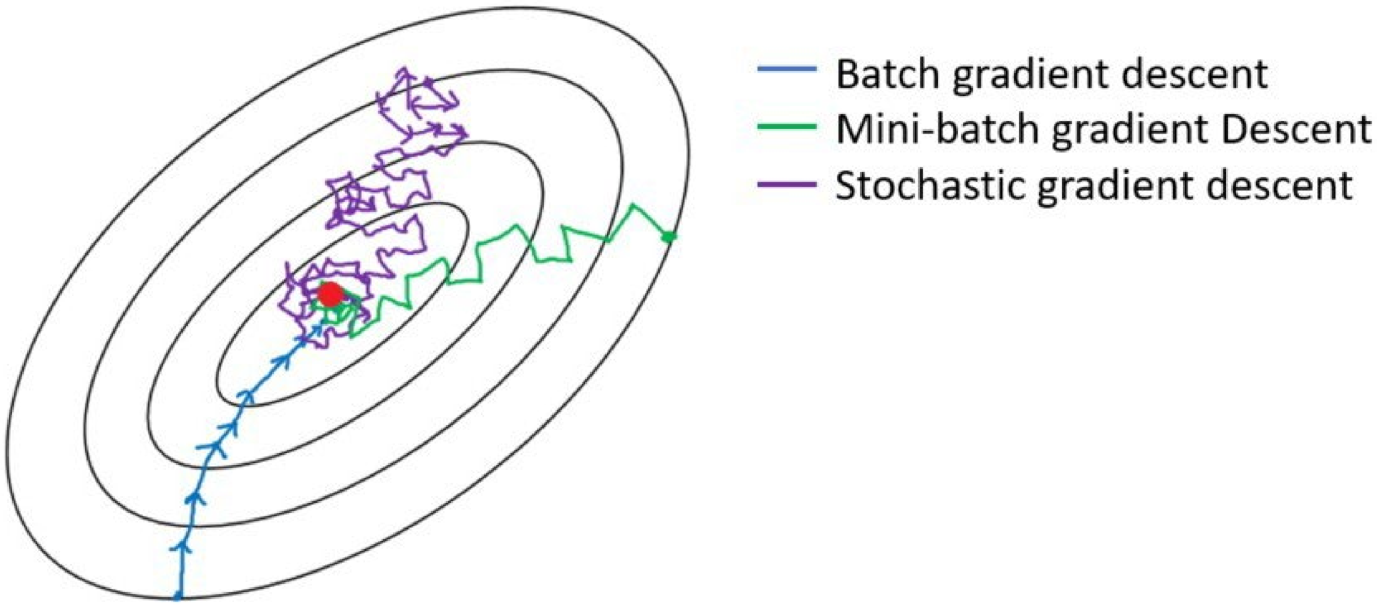
Weight updates over all epochs

Perceptron Learning: Stochastic Gradient Descent

- **Batch gradient descent** is far less efficient to calculate the gradient in every step of our algorithm for massive training points



- **Stochastic gradient descent** (SGD) updates values aEer looking at each item in the training set to make steps right away!
- SGD direction is very jagged compared to batch or mini-batch



Batch Gradient Descent

Choose initial guess $w_1^{(0)}, w_2^{(0)}$

for k = 0, 1, 2, ... **do**

$$\begin{bmatrix} w_1^{(k+1)} \\ w_2^{(k+1)} \end{bmatrix} = \begin{bmatrix} w_1^{(k)} \\ w_2^{(k)} \end{bmatrix} - \lambda \begin{bmatrix} \frac{\partial}{\partial w_0} J(w_0, w_1) \\ \frac{\partial}{\partial w_1} J(w_0, w_1) \end{bmatrix}$$

end for

Stochastic Gradient Descent

Randomly shuffle the data set

for k = 0, 1, 2, ... **do**

  for i = 1 to m **do**

$$\begin{bmatrix} w_1^{(k+1)} \\ w_2^{(k+1)} \end{bmatrix} = \begin{bmatrix} w_1^{(k)} \\ w_2^{(k)} \end{bmatrix} - \lambda \begin{bmatrix} \frac{\partial}{\partial w_0} J(w_0, w_1, x_i) \\ \frac{\partial}{\partial w_1} J(w_0, w_1, x_i) \end{bmatrix}$$

  end for make a step for a training point

end for

Jupyter Notebook

Percep/on Learning Coding Example

OUTLINE

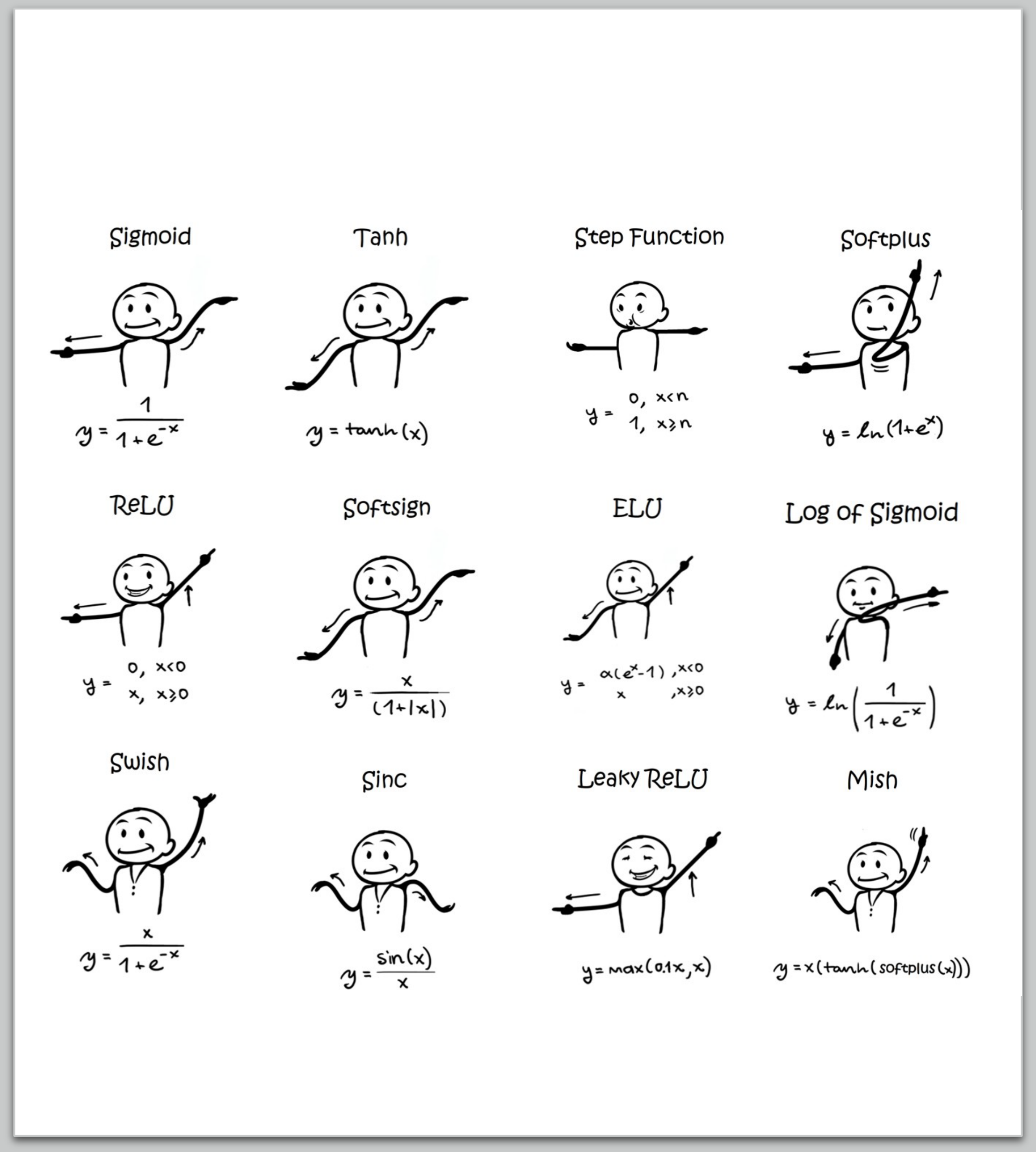
Introduction

Artificial Neural Networks (ANN)

- Single Unit: Architecture of Perceptron (NN1)
- Connection to Shallow Machine Learning (NN1)
- Multi-Layer Feed-Forward Neural Network (NN2)

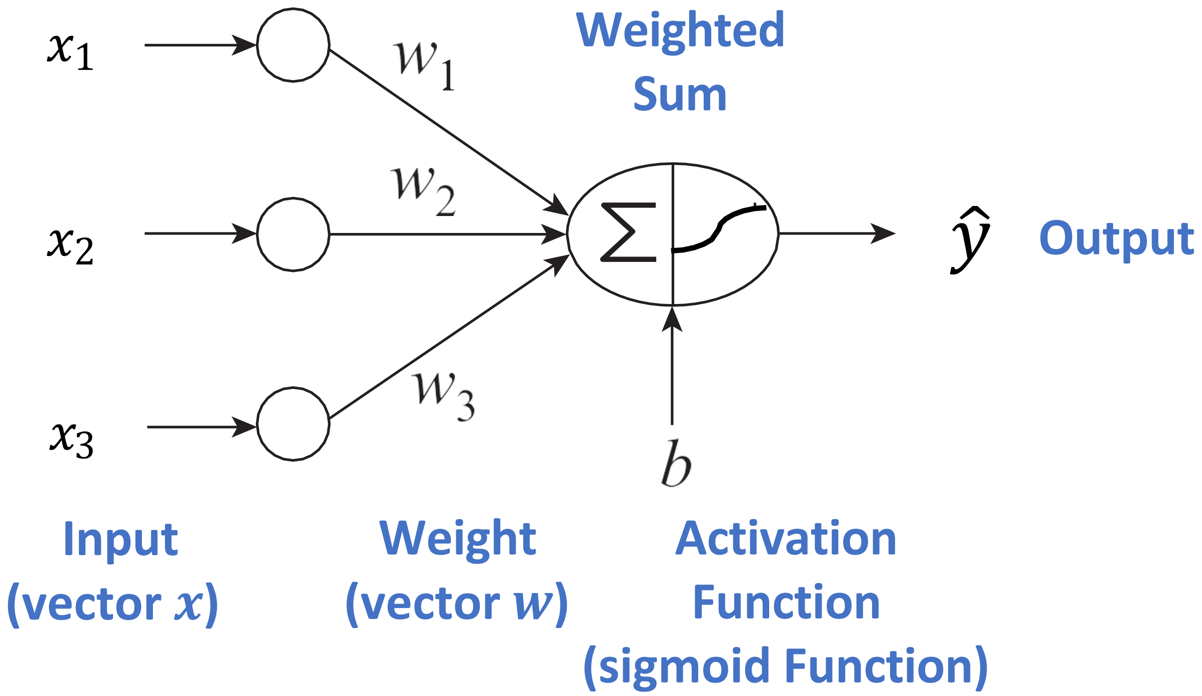
Design Issues (NN3)

Deep Learning / Large Language Models (NN4)



h,ps://twi,er.com/ TheInsaneApp/status/ 1366324846976659461?s=20

Architecture of Logistic Regression



Sigmoid Unit

$$\hat{y} = \sigma\left(\sum_j w_j x_j + b\right)$$

$$= \frac{1}{1+\exp(-(\sum_j w_j x_j + b))}$$

Neural Net Details: Logistic Regression

Architecture: A single neuron

Activation Function:

- Training: sigmoid function
- Inference: sigmoid function

Loss Function:

$$l = -(y ln(p) + (1 - y)\ln(1 - p))$$

- y - binary indicator (0 or 1) if label c is the correct classification for observation o
- p - predicted probability observation o is of class c

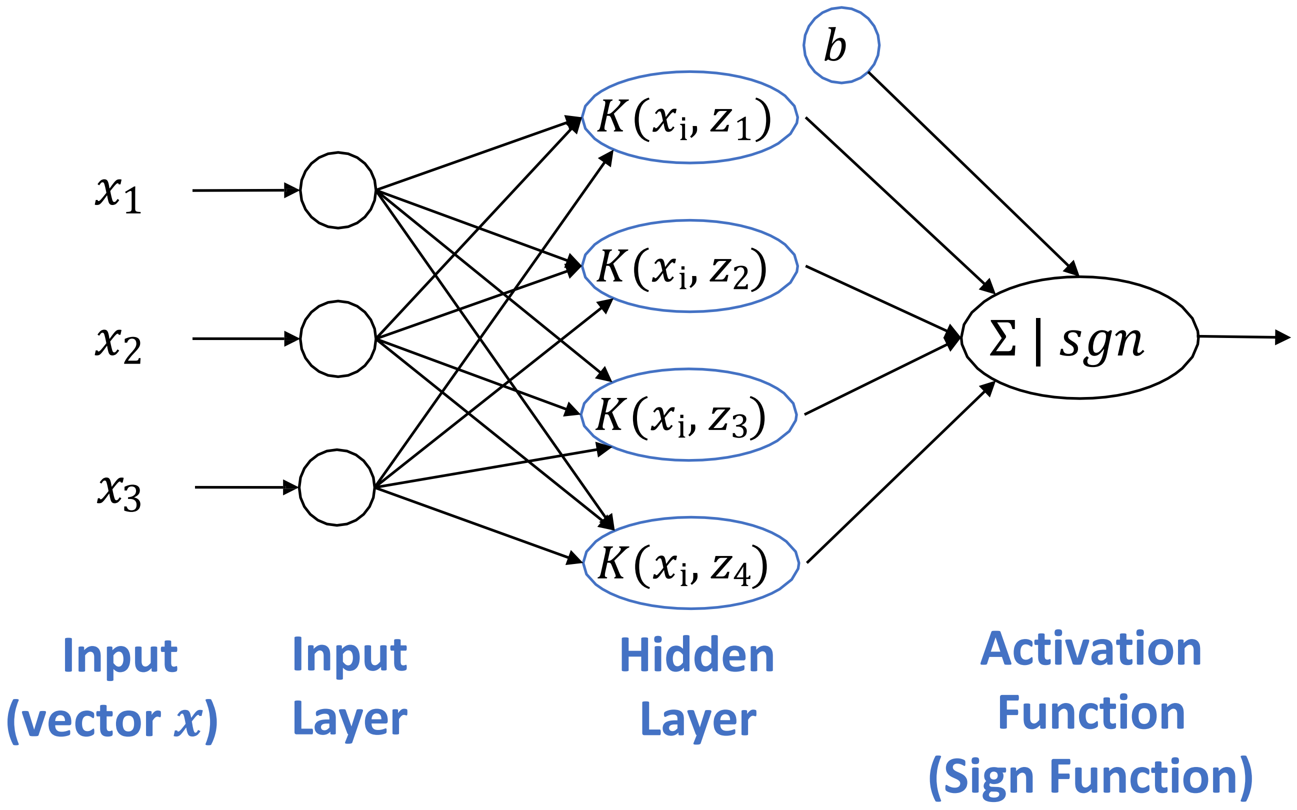
Optimisation:

- For a misclassified or barely correct training data point $(x^{(i)}, \mathrm{y}^{(\mathrm{i})})$

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \lambda\ (\mathrm{y}^{(\mathrm{i})} - \hat{y}^{(i)})x^{(i)}$$

λ : learning rate

Architecture of Kernalized SVM



$$\hat{y} = \mathrm{sign}\left(\sum_{z_j \in SV} \alpha_i y_i K\left(x_i z_j\right) + b\right)$$

Neural Net Details: Kernelized SVM

Architecture: One hidden layer

Activation Function:

- Training: identity function
- Inference: sign function / step function

Loss Function:

$$l = \max(0, 1 - yp)$$

- p - predicted probability observation o is of class c

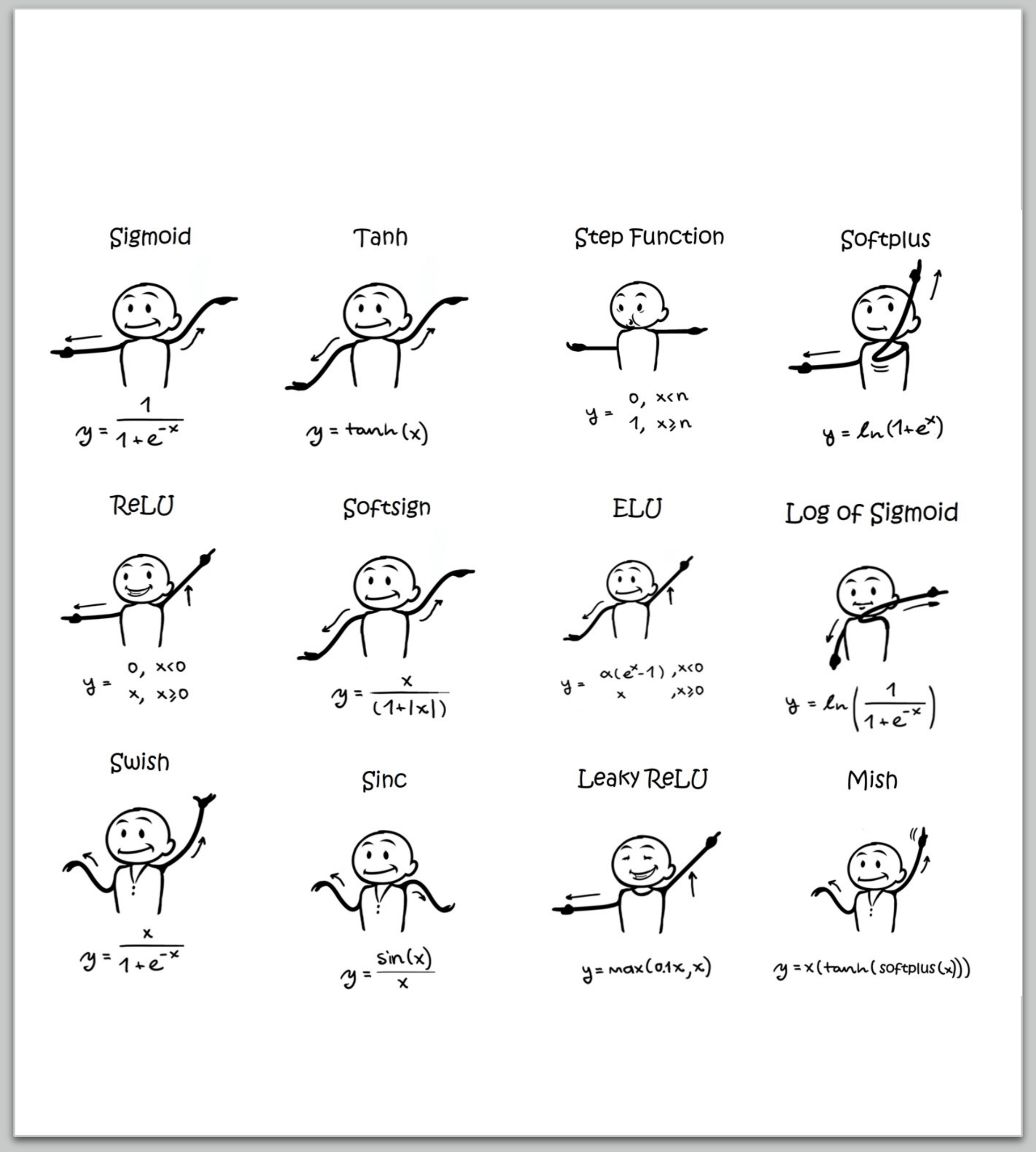
Regularisation:

- L2, i.e., $\frac{1}{2}\lambda\,||\boldsymbol{w}||^2$

Optimisation: Quadratic Programming (QP)

SUMMARY

- Single Unit: Perceptron
  - Architecture
  - Activation Function
  - Loss Function
  - Perceptron Learning
- Connection to Shallow Machine Learning
  - Logistic Regression
  - SVM



h"ps://twi"er.com/ TheInsaneApp/status/ 1366324846976659461?s=20

Resources

- Coding Libraries
  - ConvnetJS: a toy 2D classifica4on with 2-layer neural network. [link]
  - Python Machine Learning (3rd Edi4on) by Sebas4an Raschka at hlps://github.com/rasbt/python-machine-learning-book-3rd-edi4on

- Book Chapters
  - Chapter 6.7, 6.8 Introduc4on to Data Mining by Kumar et al.