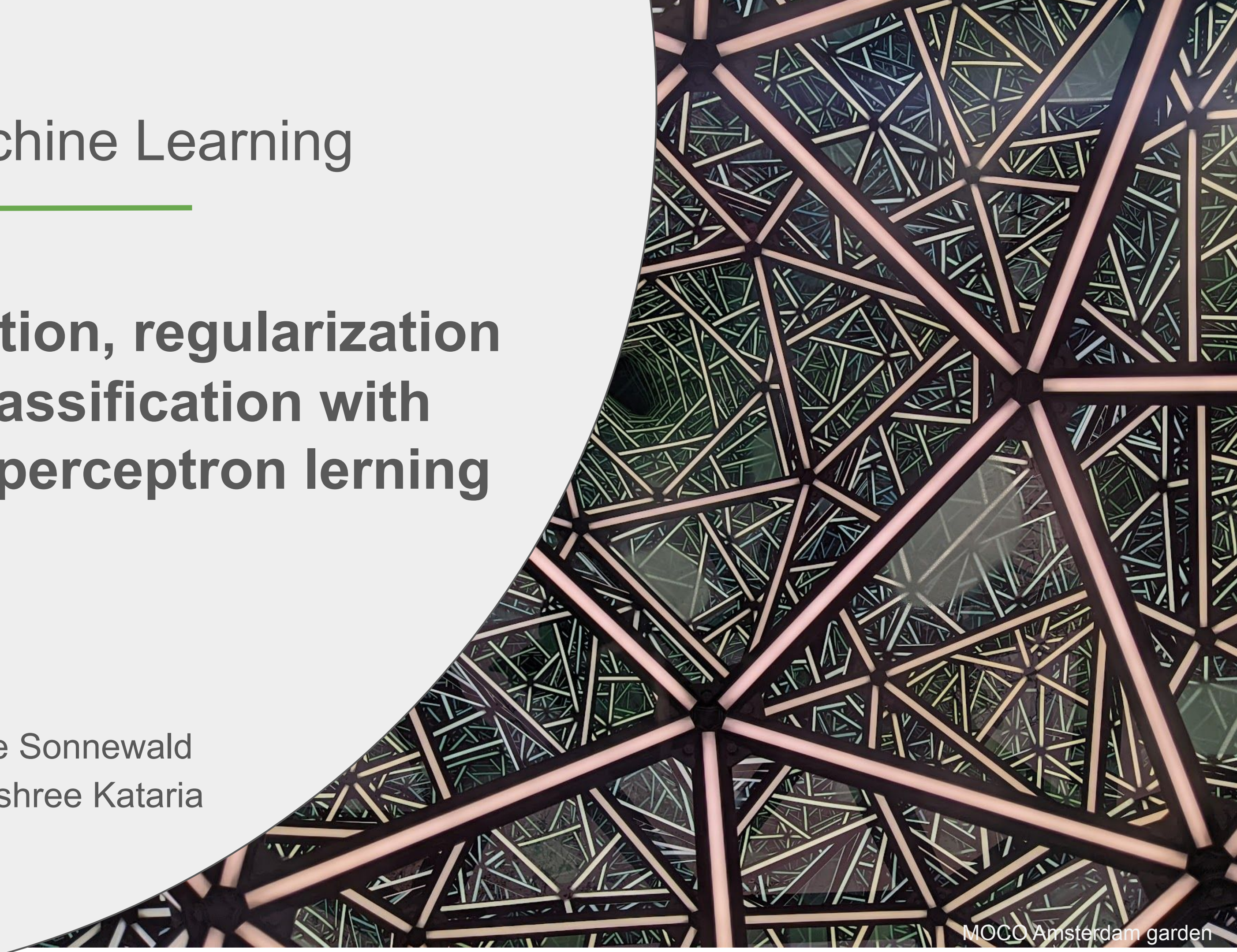
ECS171: Machine Learning

L5 Optimization, regularization and linear classification with logistic and perceptron lerning

Instructor: Prof. Maike Sonnewald

TAs: Pu Sun & Devashree Kataria



MOCO Amsterdam garden

Intended Learning Outcomes

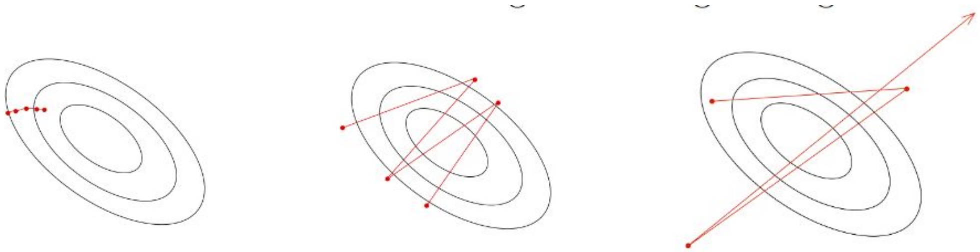
- Describe stochastic and batch gradient descent, compare and contrast to Newton’s method and implications of convexity in f(x)
- Evaluate and describe under and overfitting using error metrics
- Describe and apply momentum, as well as L1 and L2 regularization
- Describe and demonstrate how linear classification is related to linear regression
- Describe and apply logistic regression as a basis function expansion
- Describe perceptron learning for classification, and implications of using different cost functions
- In the context of what we previously learned, describe, compare and contrast the neural network layers and activation function concepts

Recommended reading:
Note on linear classifiers by R. Grosse for L5:
Extra note on linear regression by R. Grosse for L2: notes_on_linear_regression.pdf

Optimisation

Good values are typically between 0.001 and 0.1. You should do a grid search if you want good performance (i.e. try 0.1, 0.03, 0.01, . . .).

In gradient descent, the learning rate λ is a hyperparameter we need to tune. Here are some things that can go wrong:



λ too small: slow progress

λ too large: oscillations

λ much too large: instability

Terminology: Epoch and batch

- 1 epoch: All instances in training set are processed once to update weights of model
- Batch: Dataset partitioned into ‘batches’ that are multiple groups of equal size. Number and size are important ‘hyper parameters’
- If all observations in the training dataset is equal to the batches there is only one observation per batch (batch-size = 1)
    - Batch Gradient Descent is same as Stochastic Gradient Descent
    - Batch Gradient Descent always converges
- There are performance trade-offs between batch size and number

Why not always use Batch Gradient Descent?

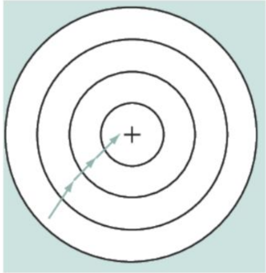
- Example: We have 10,000 data points and 10 features. SSE is as many terms as datpoints so here 10,000
- We must compute the derivative and do 100,000 computations (10,000*10) per iteration
- Usually one takes around 1000 iterations; 100,000,000 computations (100,000*1000)
- The overhead is very large and thus convergence is slow and expensive
- Stochastic Gradient Descent to the rescue:
  - When selecting data points at each step to calculate the derivatives, randomly pick one data set at each iteration
  - Reduces the computations enormously

Modified from: towardsdatascience.com

GD Update Rule for m observations (iteration)

There are 2 ways to deal with *m observations* and *n attributes*:

Batch Gradient Descent



```
Repeat until convergence:
{
  for j=1 to n
      $w_j = w_j + a \sum_{i=1}^{m}(y^{(i)} - wx^{(i)})x_j^{(i)}$
}
```

For 1 batch

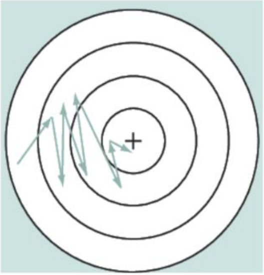
Always converges

Assuming there are m observations in one batch

Stochastic Gradient Descent

```
Repeat until convergence:
{
  for i=1 to m
      for j=1 to n
          $w_j = w_j + a((y^{(i)} - wx^{(i)})x_j^{(i)})$
}
```

For 1 epoch

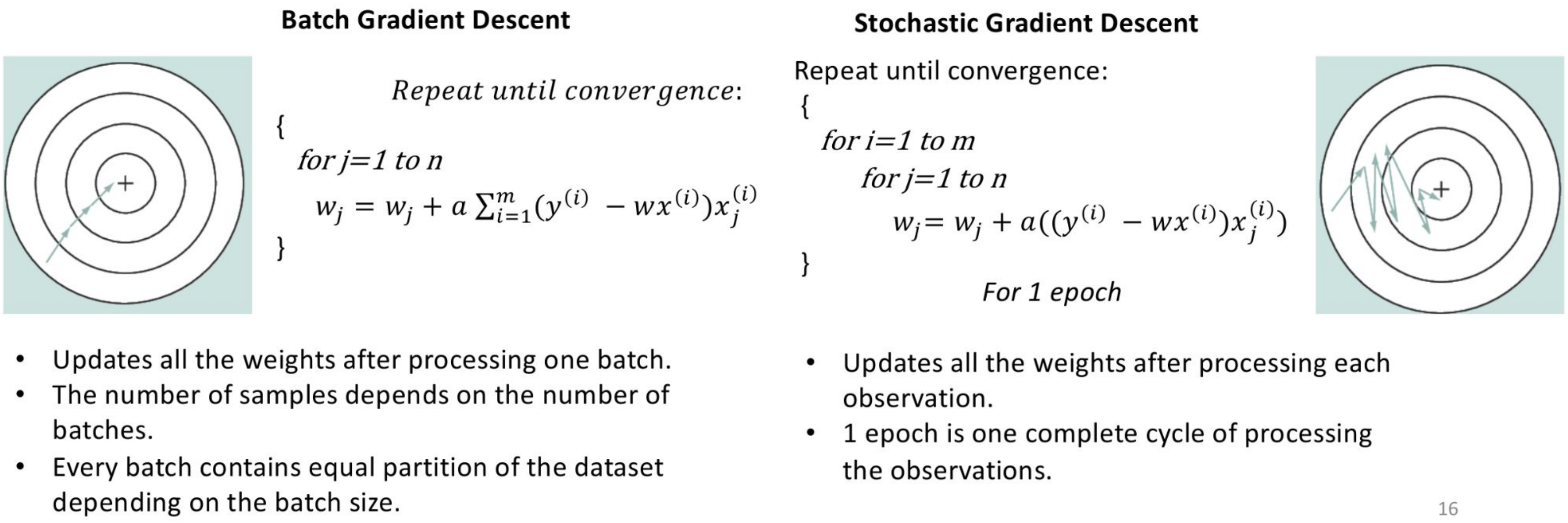


Can take many epochs to converge, or never converge.

1 epoch means after one complete round (cycle) of processing the observations in the dataset.

GD Update Rule for m observations (iteration)

There are 2 ways to deal with *m observations* and *n attributes*:



Batch Gradient Descent

```
Repeat until convergence:
{
  for j=1 to n
    $w_j = w_j + a\sum_{i=1}^{m}(y^{(i)} - wx^{(i)})x_j^{(i)}$
}
```

Stochastic Gradient Descent

```
Repeat until convergence:
{
  for i=1 to m
     for j=1 to n
        $w_j = w_j + a((y^{(i)} - wx^{(i)})x_j^{(i)})$
}
              For 1 epoch
```

- Updates all the weights after processing one batch.
- The number of samples depends on the number of batches.
- Every batch contains equal partition of the dataset depending on the batch size.

- Updates all the weights after processing each observation.
- 1 epoch is one complete cycle of processing the observations.

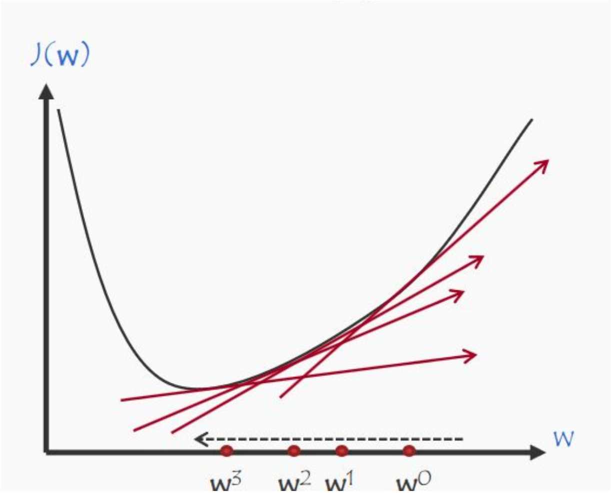
Activity-Stochastic GD Example-1.ipynb

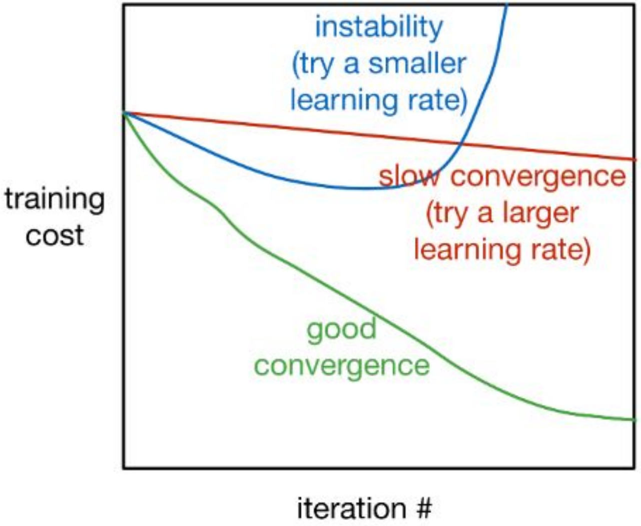
S. Rafatirad

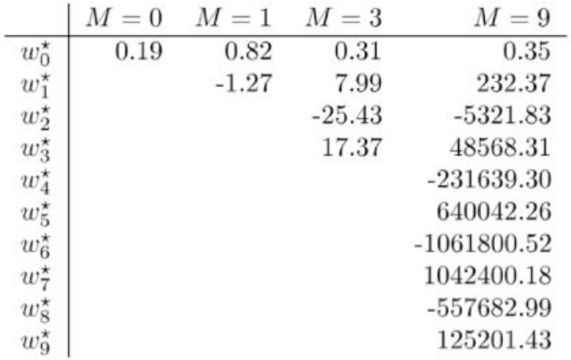
Diagnosing optimisation problems

To diagnose optimization problems, it's useful to look at training curves: plot the training cost as a function of iteration.

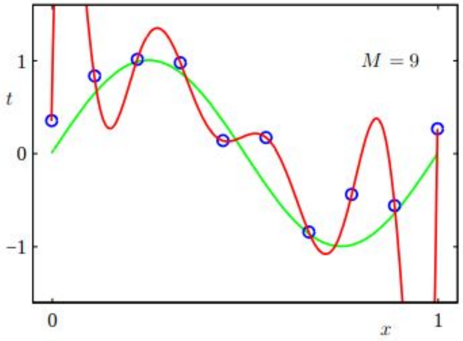
Warning: in general, it's very hard to tell from the training curves whether an optimizer has converged. They can reveal major problems, but they can't guarantee convergence.







	$M = 0$	$M = 1$	$M = 3$	$M = 9$
$w_0^\star$	0.19	0.82	0.31	0.35
$w_1^\star$		-1.27	7.99	232.37
$w_2^\star$			-25.43	-5321.83
$w_3^\star$			17.37	48568.31
$w_4^\star$				-231639.30
$w_5^\star$				640042.26
$w_6^\star$				-1061800.52
$w_7^\star$				1042400.18
$w_8^\star$				-557682.99
$w_9^\star$				125201.43



As M increases, the magnitude of coefficients gets larger.

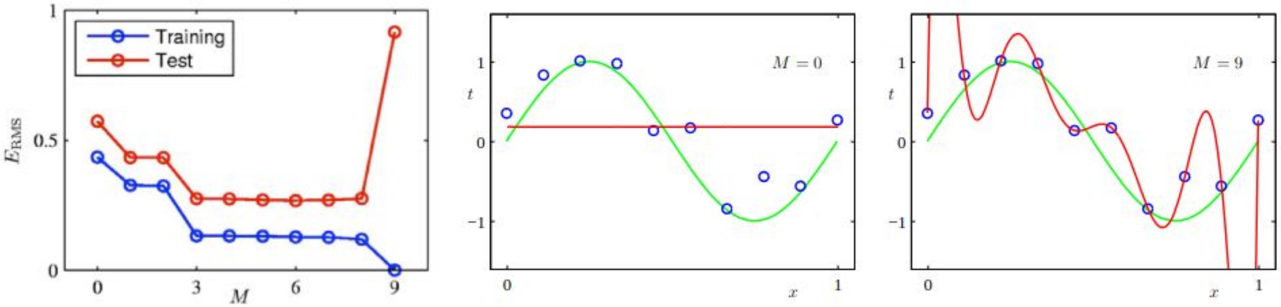
For M = 9, the coefficients have become finely tuned to the data.

Between data points, the function exhibits large oscillations.

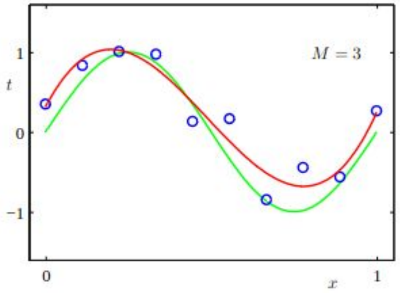
Reminder: Generalisation

Underfitting (M=0): model is too simple — does not fit the data.

Overfitting (M=9): model is too complex — fits perfectly.

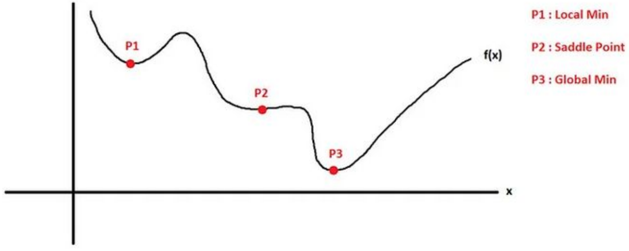


Good model (M=3): Achieves small test error (generalizes well).

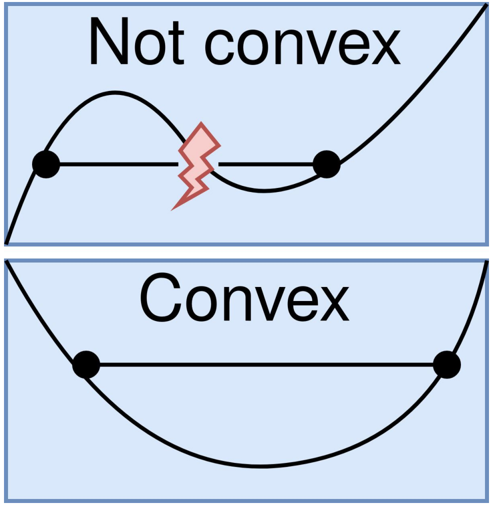


Convex functions

- When selecting an optimization algorithm, it's important to consider if the cost function is convex or non-convex
- Convex:
  - only one minima
- Non-convex:
  - Several minima and e.g. saddle points



f(x) a non-convex function with Local minima, Global minima and a Saddle point

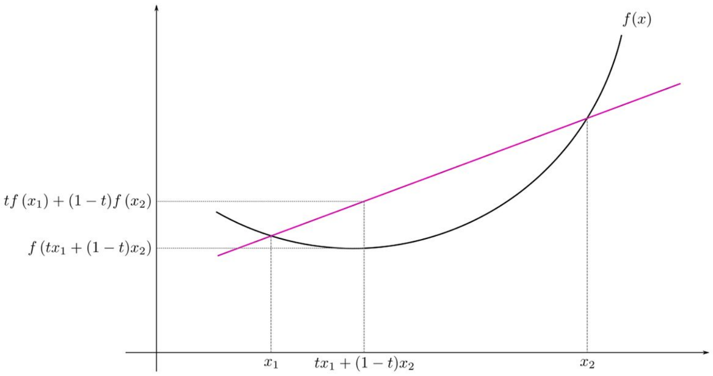


Convex functions continued

- How can we tell if a function f(x) is convex or non-convex?
- For t in range [0,1] then f(x) is convex in this range if:

$$f(tx_1 + (t-1)x_2) \geq tf(x_1) + (1-t)f(x_2)$$

- Line segment between any two points on the graph of the function lies above or on the graph of the function, and not below it



Graph of a convex function : https://en.wikipedia.org/wiki/Convex_function

$$f(a)+\frac{f'(a)}{1!}(x-a)+\frac{f''(a)}{2!}(x-a)^2+\frac{f'''(a)}{3!}(x-a)^3+\cdots=\sum_{n=0}^{\infty}\frac{f^{(n)}(a)}{n!}(x-a)^n.$$

Newton’s Method for gradient descent (Newton-Raphson)

- Taylor Series is an expansion around a function f(x) into an **infinite sum of terms.** Each term has a larger exponent like x, x^2, x^3… increasingly approximating f(x)

- Taylor expanding around minima x*:

$$f(x)=f(x*)+\overbrace{(x-x*)f'(x*)}^{\text{0 at minima}}+\frac{1}{2}(x-x*)f''(x*)+\frac{1}{3!}(x-x*)f'''(x*)...$$

$$=f(x*)+\frac{1}{2}(x-x*)^2f''(x*)+\frac{1}{3!}(x-x*)f'''(x*)...$$

- If x-x* is small the higher order terms are negligible

-

$$f(x)=a+\frac{b}{2}(x-x*)^2$$

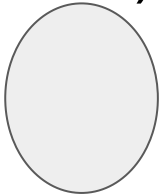
If the cost function is quadratic

$$f'(x)=b(x-x*),f''(x)=b$$

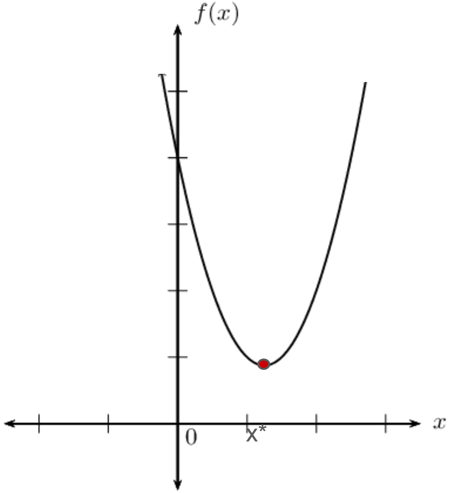
$$x-x*=\frac{f'(x)}{b}=\frac{f'(x)}{f''(x)}$$

$$x*=x-\frac{f'(x)}{f''(x)}$$

Isaac Newton
(English 1600s)



Joseph Raphson
(English 1600s)



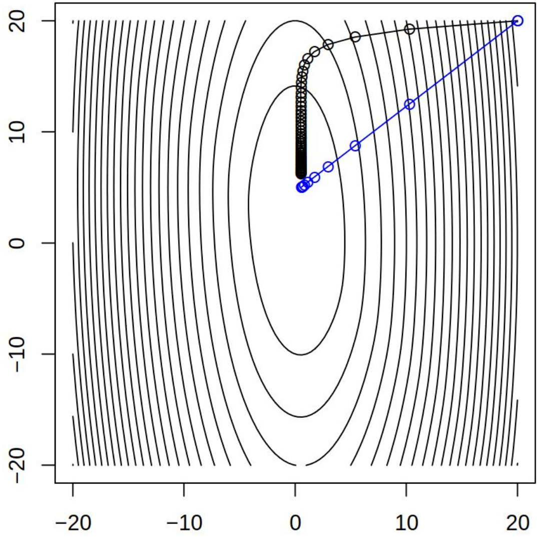
Adapted from siyavula.com

Newton’s Method for gradient descent (Newton-Raphson)

- Newton’s method, als called ‘convex optimisation):

$$w_j^{t+1} = w_j^t - \lambda \frac{\left.\frac{\partial J(w)}{\partial w_j}\right\}\text{First derivative}}{\left.\frac{\partial^2 J(w)}{(\partial w_j)^2}\right\}\text{Second derivative}}$$

- First derivative: Slope of the tangent line
- Second derivative: Instantaneous rate of change of first derivative. Sign of second derivative indicates if the slope of the tangent line is increasing or decreasing
- Pros:
  - Fast: Quadratic convergence
  - Generalises well
- Cons:
  - Varying robustness: Sometimes fails
  - Some smoothness requirements



Black: Gradient Descent
Blue: Newton’s method

Adding ‘momentum’ to gradient descent

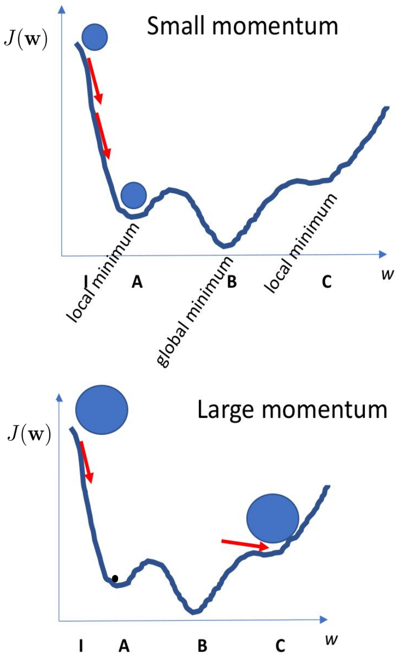
In (stochastic) gradient descent with momentum, the update rule at each iteration is given by:

$$b_i = \mu * b_{i-1} + g_i$$

$$\theta_i = \theta_{i-1} - \gamma * b_i$$

Where:

- θ denotes the parameters to the cost function
- g_i is the gradient indicating which direction to decrease the cost function by
- γ is the hyperparameter representing the learning rate
- b_i is the modified step direction term (as opposed to just using g_i) that incorporates momentum
- μ is a new hyperparameter that denotes the momentum constant



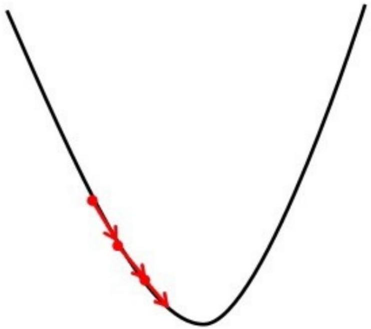
Source: Discovering Knowledge in Data, D. Larose

Adding ‘momentum’ to gradient descent

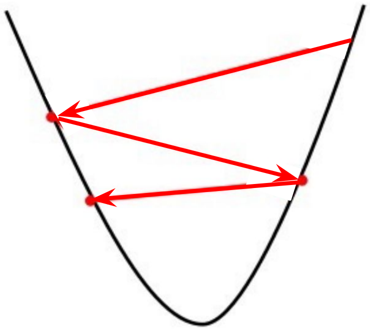
- Small μ values:
  - Reduce interia effect and impact of recent adjustments
  - Starting with low μ allows algorithm to explore ‘optimisation landscape’ more fully

- Larger μ values:
  - Allows algorithm to move in same direction as previous adjustment
  - Starting with high μ helps speed-up convergence

- Commonly, momentum μ is initialised (e.g. 0.9) and tuned similar to learning rate λ



Non-oscillating descent



Oscillating descent

Generalisation: Regularization

Regularization: Regularization reduces the variance at the cost of increasing the bias.

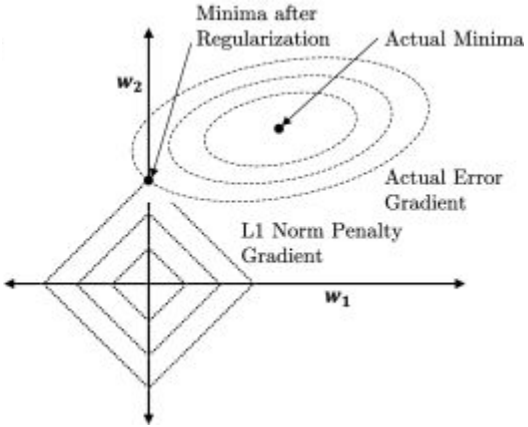
- The degree M of the polynomial controls the model's complexity.
- The value of M is a hyperparameter for polynomial expansion. We can tune it using a validation set.
- Restricting the number of parameters (e.g. M in polynomial example from before) is a crude approach to controlling the model complexity.
- Another approach: keep the model large, but **regularize** it
  - Regularizer: a function that quantifies how much we prefer one hypothesis vs. another
- A lot of common loss and regularization functions are **convex** functions, for example:
  - L1 'lasso' regularization
  - L2 'ridge regression' regularization

L1 and L2 regularization are convex functions

L1 Lasso: Sum of absolute values of the coefficient

$$J(\mathbf{w}) = \underbrace{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}_{\text{MSE}} + \underbrace{\lambda\sum_{i=1}^{n}|w_i|}_{\text{Penalty term}}$$

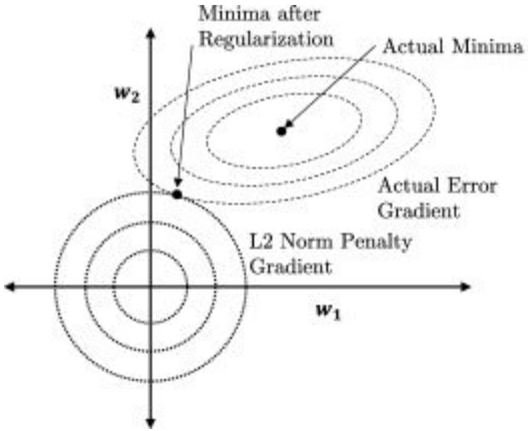
Encourages weights to be **exactly** zero



L2 'Ridge': Squared magnitude of the coefficient

$$J(\mathbf{w}) = \underbrace{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}_{\text{MSE}} + \underbrace{\lambda\sum_{i=1}^{n}w_i^2}_{\text{Penalty term}}$$

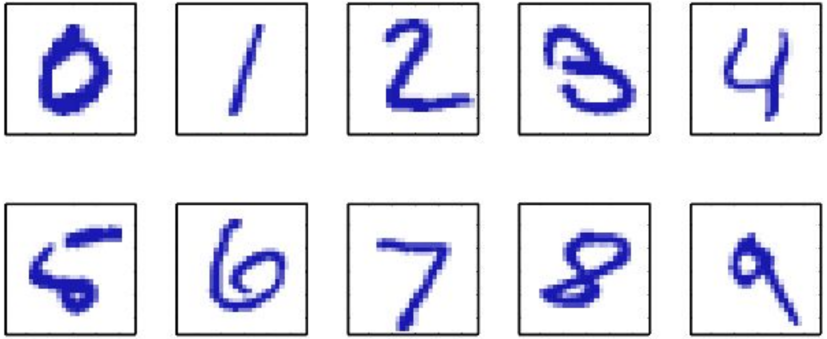
Encourages weights to be **close** to zero



L1 and L2 regularization: Takeaways

- We can encourage the weights to be small by choosing as our regularizer the L2 penalty.
- Note: To be precise, the L2 norm is Euclidean distance, so we’re regularizing the squared L2 norm.
- The regularized cost function makes a tradeoff between fit to the data and the norm of the weights.
- If you fit training data poorly, J is large. If your optimal weights have high values, R is large.
- Large λ penalizes weight values more.
- Like M , λ is a hyperparameter we can tune with a validation set.

Linear Classification: Categorical outputs



V7labs.com, anujadp.medium.com, machinelearningmastery.com

Linear Classification: Categorical outputs

- What do all these problems have in common?
    - Categorical outputs, called labels (e.g. yes/no, dog/cat/other)
- Assigning each input vector to one of a finite number of labels is called classification
- Binary classification: two possible labels (eg, yes/no, 0/1, cat/dog, happy/sad)
- Multi-class classification: multiple possible labels

We will first look at binary problems, and discuss multi-class problems later in class

Classification vs. Regression

- Actually: We can use all we’ve covered so far ignoring the categorical nature!
- Suppose we have a binary problem: $t \in \{-1, 1\}$
- Assuming the standard model used for (linear) regression:

$$y(\mathbf{w}) = f(\mathbf{x}, \mathbf{w}) = \mathbf{W}^T \mathbf{x}$$

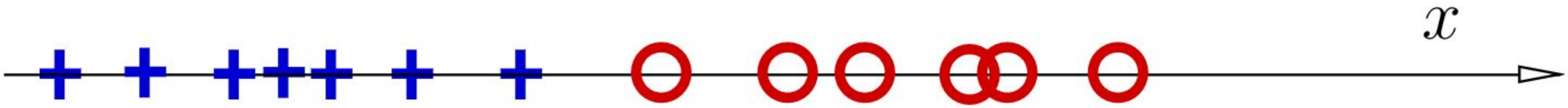
- Using Sum of Squared Error (SSE) as our cost function J(**w**):

$$J(\mathbf{w}) = \sum_{i=1}^{n} (y_i - \mathbf{W}^{\mathrm{T}} \mathbf{X}_{\mathrm{i}})^2$$

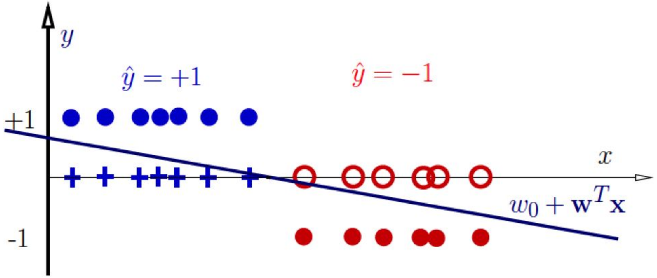
- But…how do we predict a label this way..?

Classification vs. Regression

- One dimensional example (input x is 1-dim): Red or blue?
- The colors indicate labels:
  - Blue plus denotes that x_i is from the first class
  - Red circle that x_i is from the second class



Classification vs. Regression



- Our classifier has the form:

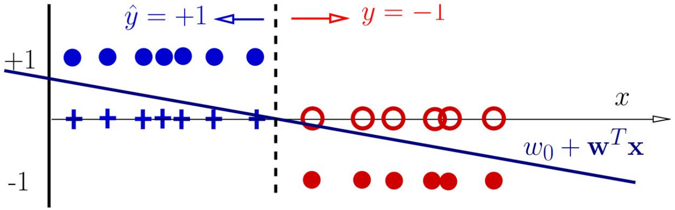
$$f(\mathbf{x}, \mathbf{w}) = w_0 + \mathbf{w}^T\mathbf{x}$$

- A reasonable decision rule is:

$$y = \begin{cases} 1 & \text{if } f(\mathbf{x}, \mathbf{w}) \geq 0. \\ -1 & \text{otherwise.} \end{cases}$$

- Mathematically write as: $y(\mathbf{x}) = sign(w_0 + \mathbf{w}^T\mathbf{x})$

Classification vs. Regression



- Mathematically write function as:

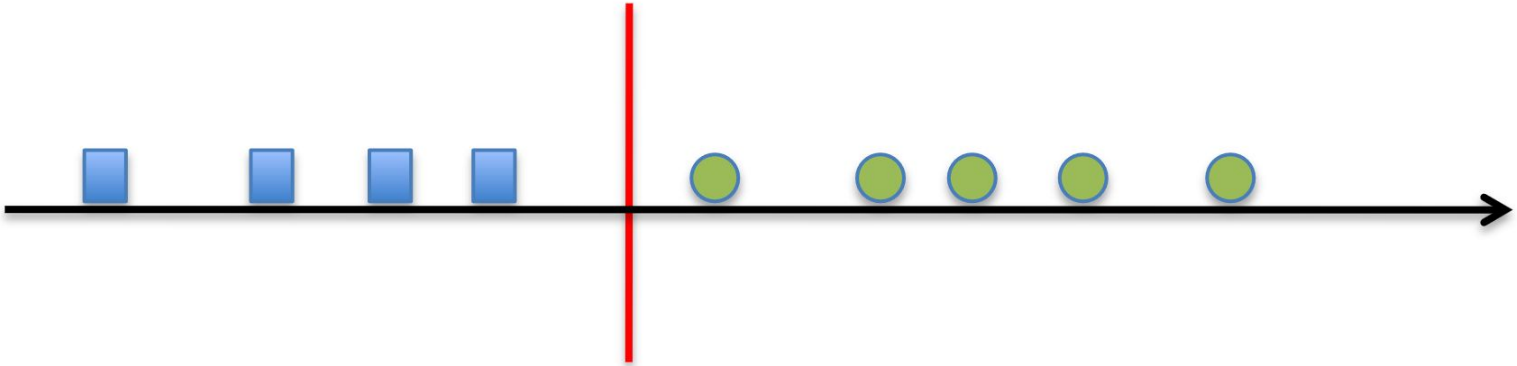
$$y(\mathbf{x}) = sign(w_0 + \mathbf{w}^T\mathbf{x})$$

- This specifies a linear classifier: it has a linear boundary (hyperplane):

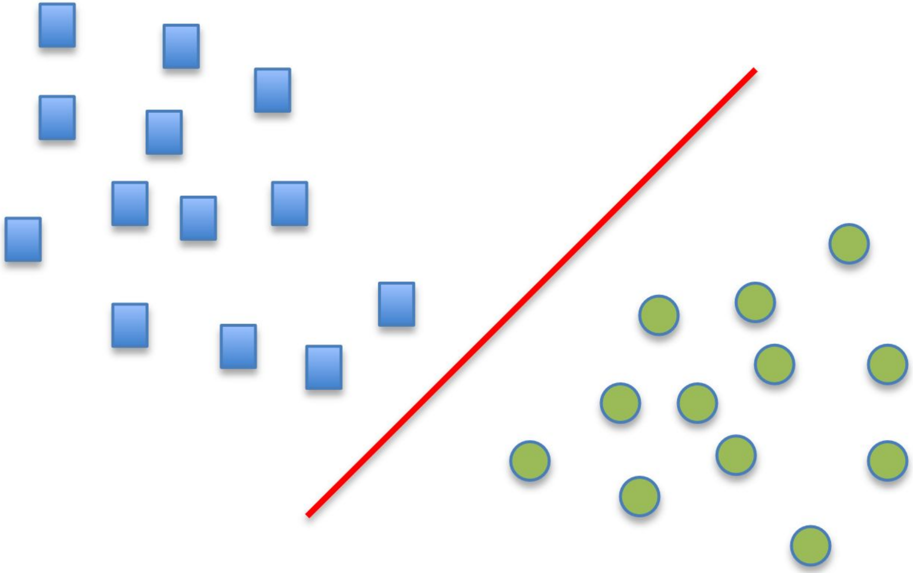
$$0 = w_0 + \mathbf{w}^T\mathbf{x}$$

- This hyperplane separates the space into two ”half-spaces”

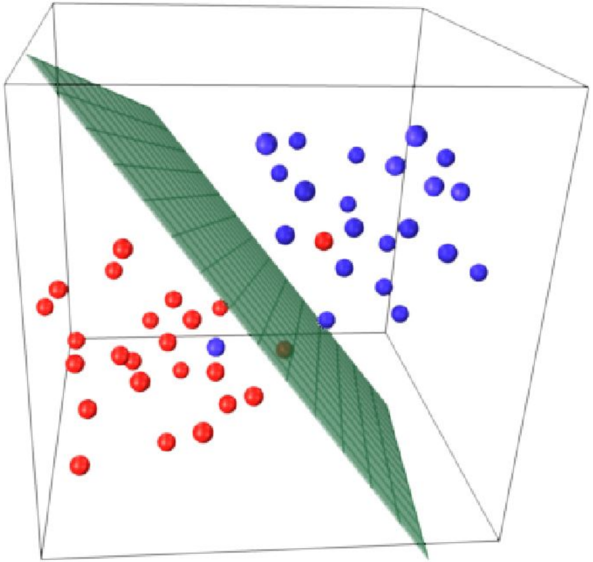
Decision plane in 1D: This is simply a threshold



Decision plane in 2D: This is a line



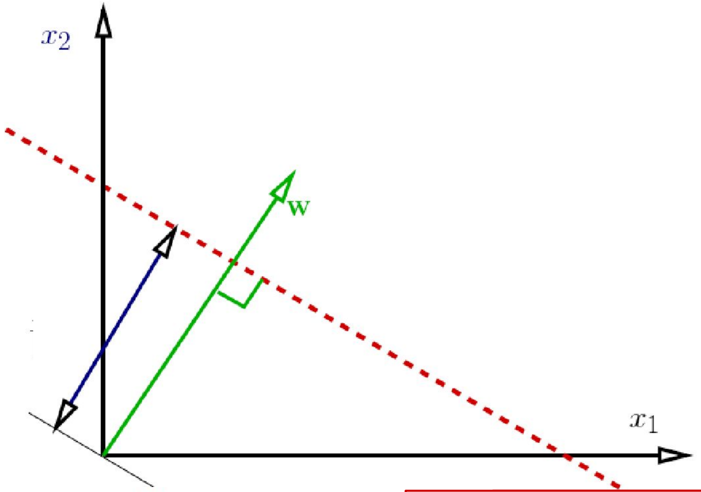
Decision plane in 3D: This is a plane



- What about higher dimensions?

Geometric interpretation of decision boundary

- A line through the origin and orthogonal to **w**: $0 = \mathbf{w}^T\mathbf{x}$
- Shifted by w_0: $0 = w_0 + \mathbf{w}^T\mathbf{x}$



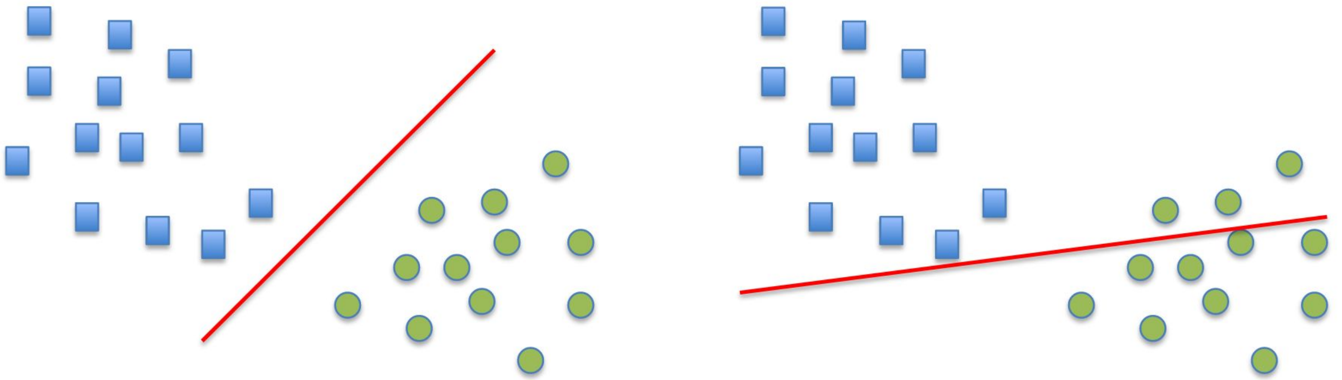
$$0 = \mathbf{w}^T\mathbf{x}$$

$$\boxed{0 = w_0 + \mathbf{w}^T\mathbf{x}}$$

G. Shakhnarovich

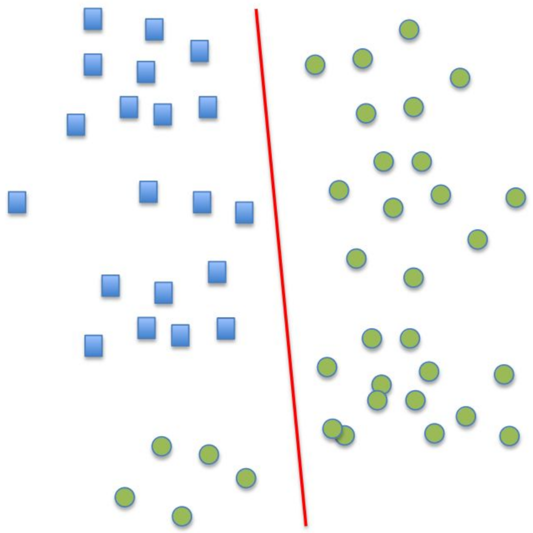
Learning is estimating a “good” decision boundary

- Goal: Find **w** (direction) and w_0 (location) of the boundary
- We need a criteria that tell us how to select the parameters

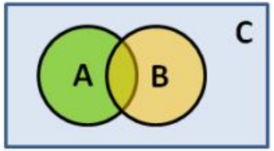
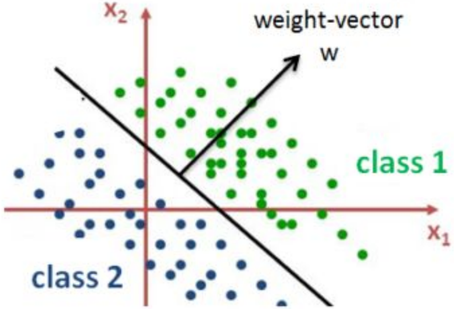


Cost functions are also used for linear classifiers

- In classification, the cost function is a metric for how well the data is separated by the boundary (red line)
- Causes for non-perfect separation:
  - Model is too simple (e.g. data is non-linear)
  - Noise in the inputs (i.e., data attributes)
  - Errors in data targets (mis-labeling)
  - Simple features that do not account for all variations
- We will cover more complex non-linear models later in class

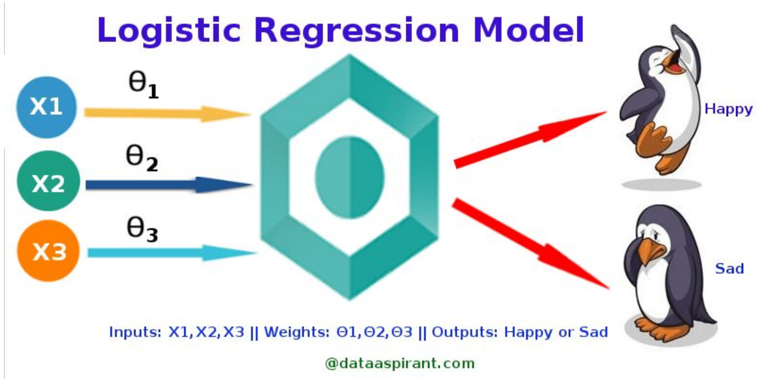


Binary Classification Algorithms

Functional Approach (Logistic Regression)	Statistical Approach (Naive Bayes)	Geometrical Approach (Perceptron, SVM)
$X = \begin{bmatrix} x^{(1)} \\ x^{(2)} \\ \vdots \\ x^{(n)} \end{bmatrix} \rightarrow \boxed{h_\theta(x)} \rightarrow Y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(n)} \end{bmatrix}$ $h_\theta(x) = \theta_0 + \theta_1 x^{(2)} + \cdots + \theta_n x^{(n)}$	$P(A\|B) = \frac{P(B\|A)P(A)}{P(B)}$ 	

Logistic Regression or Linear Regression?

- Consider a classification problems e.g.:
  - Spam detection (1 or 0)
  - Tumor detection (1 or 0)
  - Mood detection (1 or 0)


Logistic Regression Model
X1
X2
X3
θ_1
θ_2
θ_3
Happy
Sad
Inputs: X1,X2,X3 || Weights: ϴ1,ϴ2,ϴ3 || Outputs: Happy or Sad
@dataaspirant.com

- If the hypothesis is a linear regression model:

$$y = \mathbf{wx}$$

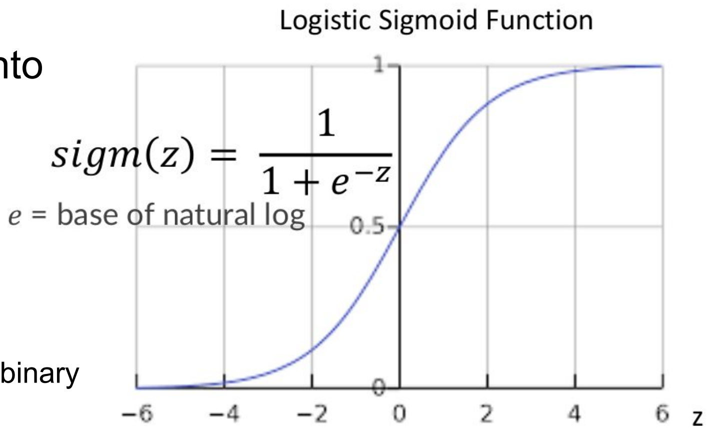
$$y^{(i)} = \begin{cases} 0; & predicted\ value < threshold \\ 1; & predicted\ value \geq threshold \end{cases}$$

- Drawbacks of using linear regression for classification:
  - Sensitive to outliers
  - Sensitivity to selected threshold

- Overall: **Logistic Regression** is a better way to perform classification!

Logistic Regression

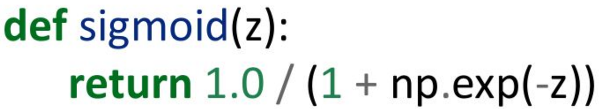
- When the goal is to classify the data points (samples) into categories (or labels), you can use Logistic Regression
- Output: 0 or 1, or a probability estimate
- Logistic regression has the form of a sigmoid function
    - S-shaped curve
    - Maps a real value input to a value between 0 and 1 (probability of a binary outcome)
    - Sigmoid function is also used as an activation function in Artificial Neural Networks
    - Sigmoid function is good for modeling non-linear relationships between the input and the output
    - The default threshold for logistic regression is 0.5
- Logistic regression is a special case of function expansion



Logistic Sigmoid Function

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

e = base of natural log



```
def sigmoid(z):
    return 1.0 / (1 + np.exp(-z))
```

Logistic Regression as a Basis Function Expansion

- Function expansion is a mathematical technique that involves approximating a complex function by a simpler function that can be represented as a sum of simpler functions
- The ‘simpler functions’ are called Basis Functions
- Useful to simplify a complicated expression involving a complex function
- Logistic Regression is a special case of Function Expansion, where it models the probability of the binary outcome as a logistic sigmoid function of a linear combination of input variables

$$z = f(x;w) = w^T x = w_0 x_0 + w_1 x_1 + \ldots + w_n x_n$$

$$g(z) = sigm(z) = \frac{1}{1+e^{-}} \Rightarrow g\left(w^T x^{(i)}\right) = sigm\left(w^T x\right) = \frac{1}{1+e^{-w^T x}}$$

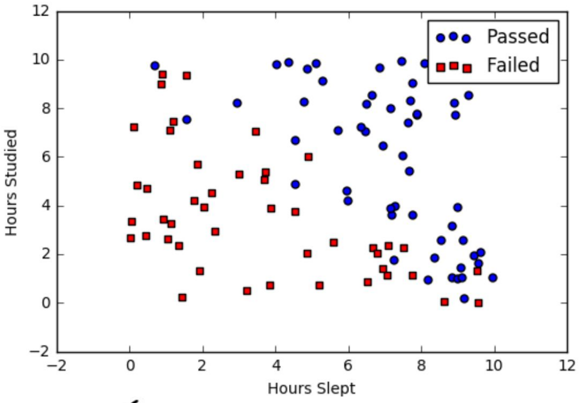
$$\begin{cases} 0 \; if \; g\left(w^T x^{(i)}\right) < threshold \\ 1 \; if g\left(w^T x^{(i)}\right) \geq threshold \end{cases}$$

0

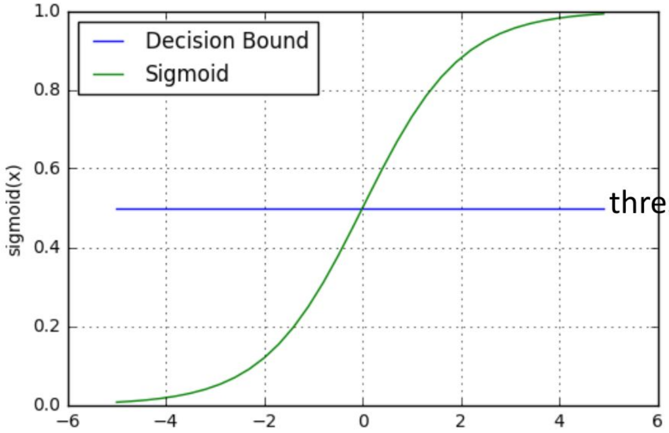
Logistic regression example

- The hours did each student study and sleep:
    - Did the student pass (1) or fail (0)?
- The default threshold is 0.5

Studied	Slept	Passed
4.85	9.63	1
8.62	3.23	0
5.43	8.23	1
9.21	6.34	0



$$y^{(i)} = \begin{cases} 0 & if\ passed < threshold \\ 1 & if\ passed \geq threshold \end{cases}$$

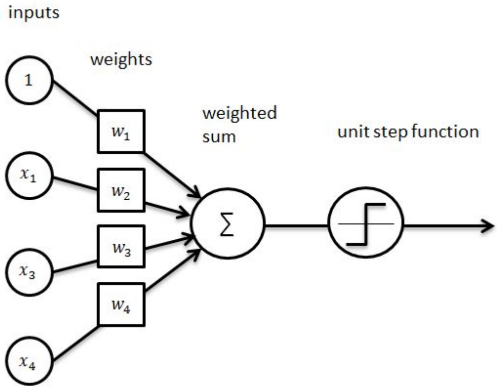


threshold

Perceptron Learning Algorithm

- The Perceptron Learning Algorithm is supervised learning that is good for binary classification

- The Perceptron model takes an input, aggregates it (calculates the weighted sum), and with the step function returns 1 if this is more than a threshold or 0 if it is equal or below



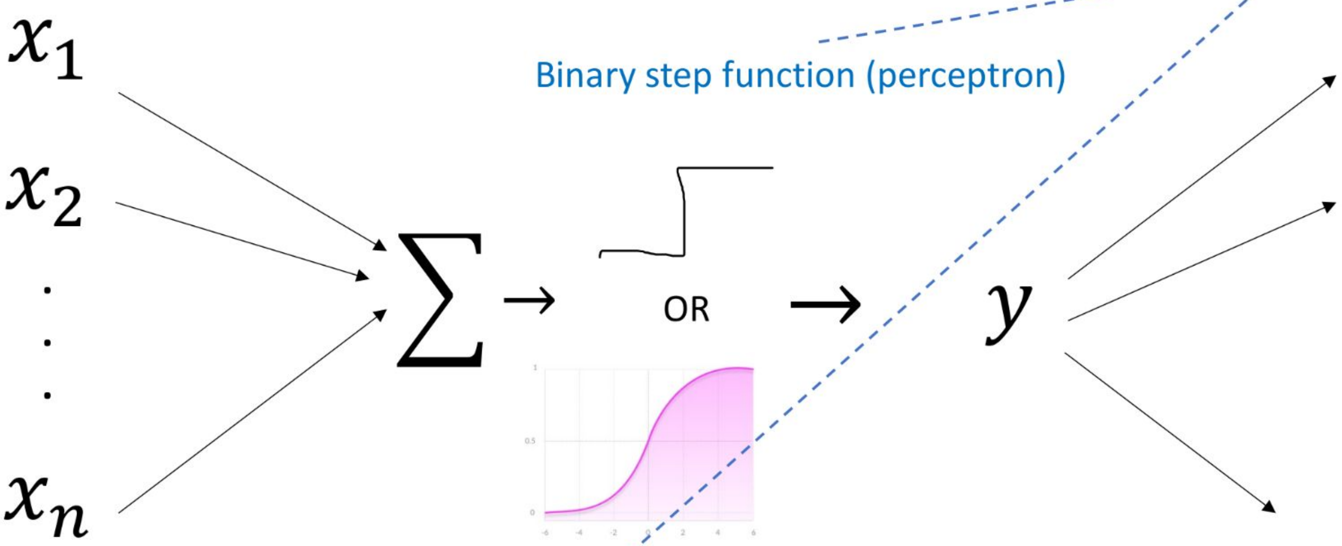
$$w_j = w_j + a\left(y^{(i)} - g\left(x^{(i)}; w\right)\right) x_j^{(i)}$$

$$w_j = w_j + a\left(y^{(i)} - g(z)\right) x_j^{(i)}$$

$$z = w^T x = w_0 + w_1 x_1 + w_2 x_2$$

$$g(z) = \begin{cases} 0\ if\ \ z < threshold \\ 1\ if\ \ z \geq threshold \end{cases}$$

ataspinar.com

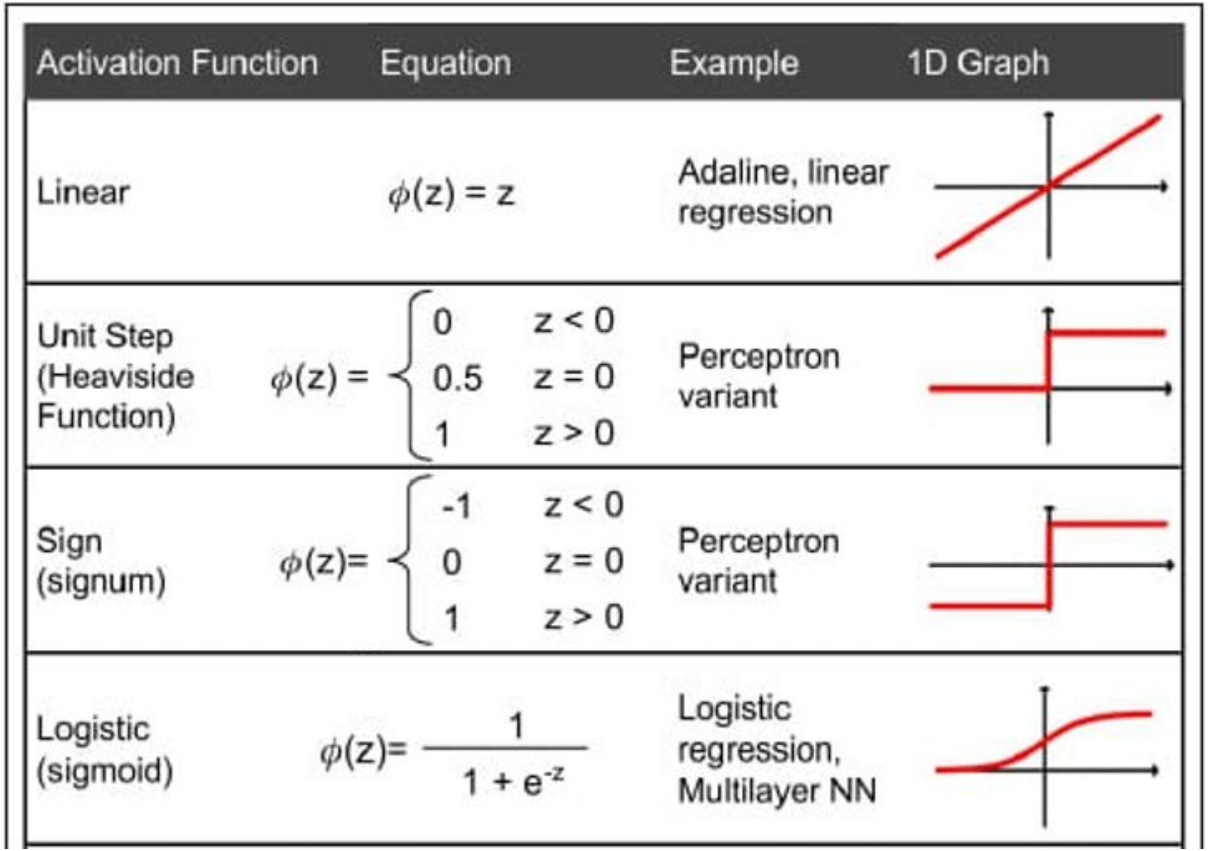


Activation function

Binary step function (perceptron)

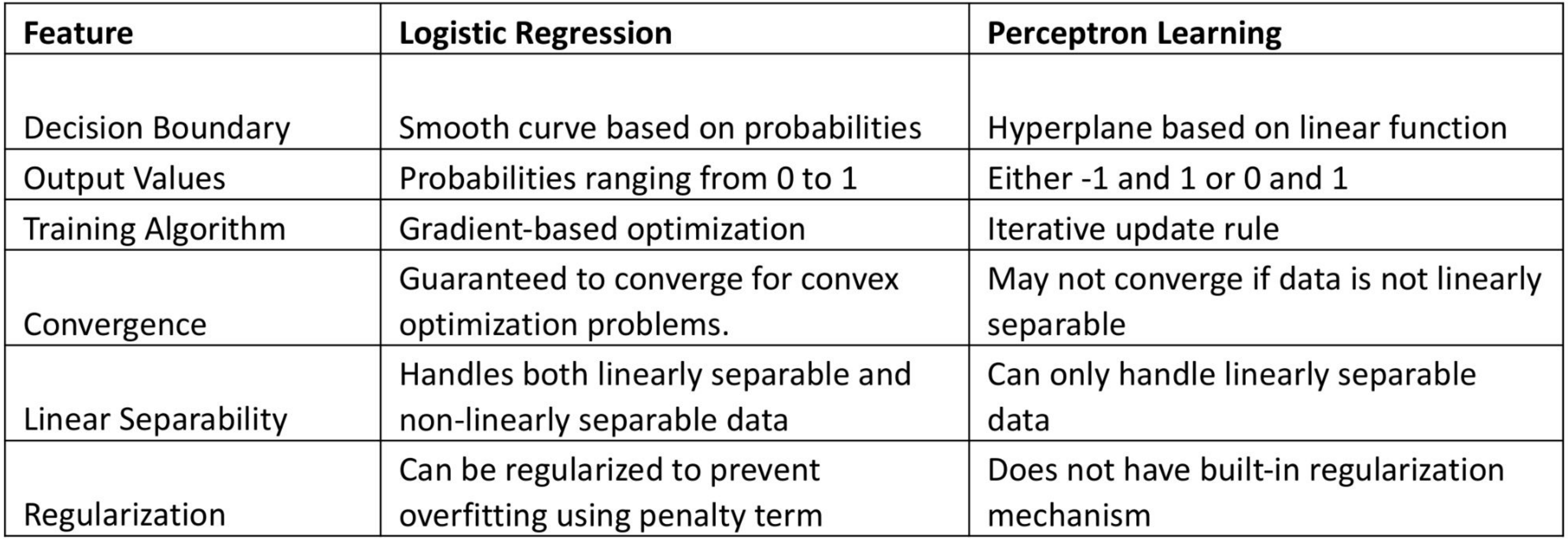
Sigmoid function (Logistic unit)

Different activation functions



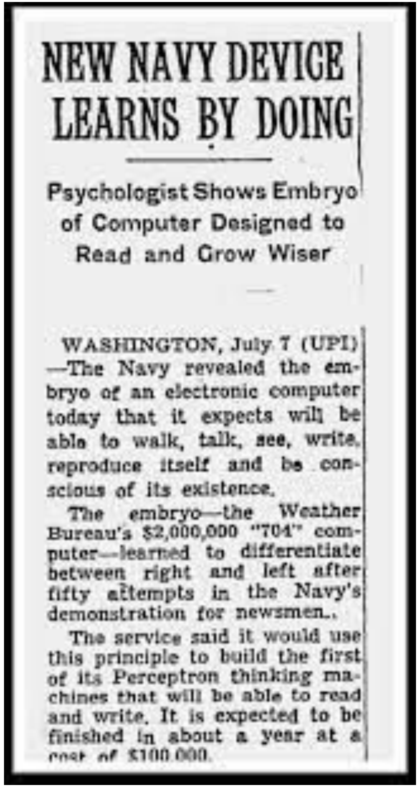
Activation Function	Equation	Example	1D Graph
Linear	$\phi(z) = z$	Adaline, linear regression	
Unit Step (Heaviside Function)	$\phi(z) = \begin{cases} 0 & z < 0 \\ 0.5 & z = 0 \\ 1 & z > 0 \end{cases}$	Perceptron variant	
Sign (signum)	$\phi(z)= \begin{cases} -1 & z < 0 \\ 0 & z = 0 \\ 1 & z > 0 \end{cases}$	Perceptron variant	
Logistic (sigmoid)	$\phi(z)= \dfrac{1}{1 + e^{-z}}$	Logistic regression, Multilayer NN	

Logistic Regression vs Perceptron Learning



Feature	Logistic Regression	Perceptron Learning
Decision Boundary	Smooth curve based on probabilities	Hyperplane based on linear function
Output Values	Probabilities ranging from 0 to 1	Either -1 and 1 or 0 and 1
Training Algorithm	Gradient-based optimization	Iterative update rule
Convergence	Guaranteed to converge for convex optimization problems.	May not converge if data is not linearly separable
Linear Separability	Handles both linearly separable and non-linearly separable data	Can only handle linearly separable data
Regularization	Can be regularized to prevent overfitting using penalty term	Does not have built-in regularization mechanism

The Perceptron (McCulloch–Pitts neuron)



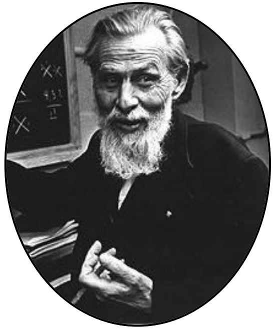
NEW NAVY DEVICE LEARNS BY DOING

Psychologist Shows Embryo of Computer Designed to Read and Grow Wiser

WASHINGTON, July 7 (UPI) —The Navy revealed the embryo of an electronic computer today that it expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence.

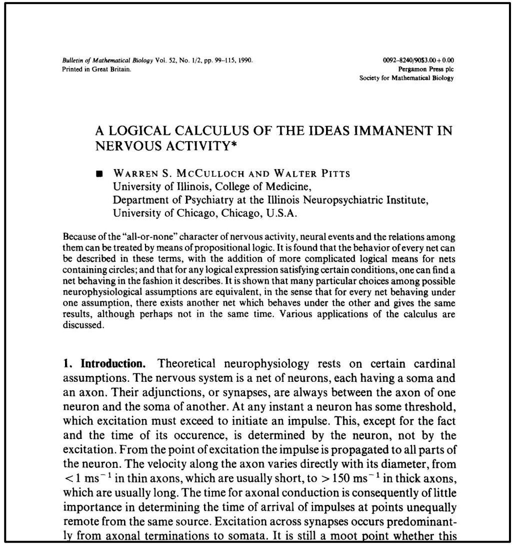
The embryo—the Weather Bureau's $2,000,000 "704" computer—learned to differentiate between right and left after fifty attempts in the Navy's demonstration for newsmen..

The service said it would use this principle to build the first of its Perceptron thinking machines that will be able to read and write. It is expected to be finished in about a year at a cost of $100,000.



Warren McCulloch (US)
1898-1969

Walter Pitts (US)
1923-1969



Bulletin of Mathematical Biology Vol. 52, No. 1/2, pp. 99–115, 1990.
Printed in Great Britain.

0092–8240/90$3.00 + 0.00
Pergamon Press plc
Society for Mathematical Biology

A LOGICAL CALCULUS OF THE IDEAS IMMANENT IN NERVOUS ACTIVITY*

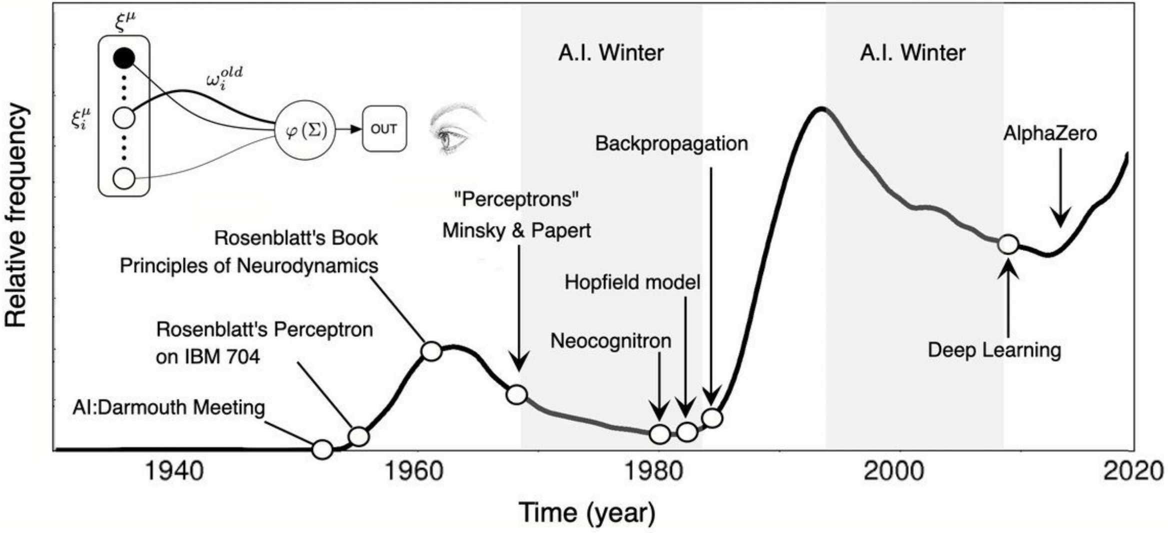
■ Warren S. McCulloch and Walter Pitts
University of Illinois, College of Medicine,
Department of Psychiatry at the Illinois Neuropsychiatric Institute,
University of Chicago, Chicago, U.S.A.

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

1. Introduction. Theoretical neurophysiology rests on certain cardinal assumptions. The nervous system is a net of neurons, each having a soma and an axon. Their adjunctions, or synapses, are always between the axon of one neuron and the soma of another. At any instant a neuron has some threshold, which excitation must exceed to initiate an impulse. This, except for the fact and the time of its occurence, is determined by the neuron, not by the excitation. From the point of excitation the impulse is propagated to all parts of the neuron. The velocity along the axon varies directly with its diameter, from $< 1\ \mathrm{ms}^{-1}$ in thin axons, which are usually short, to $> 150\ \mathrm{ms}^{-1}$ in thick axons, which are usually long. The time for axonal conduction is consequently of little importance in determining the time of arrival of impulses at points unequally remote from the same source. Excitation across synapses occurs predominantly from axonal terminations to somata. It is still a moot point whether this

1943

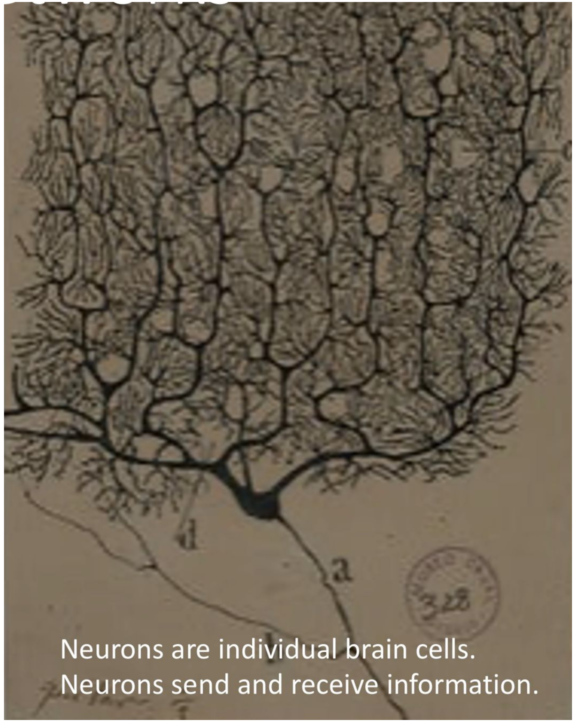
AI ‘spring’ and ‘winter’



Richard Sole

Introduction to Neural Networks

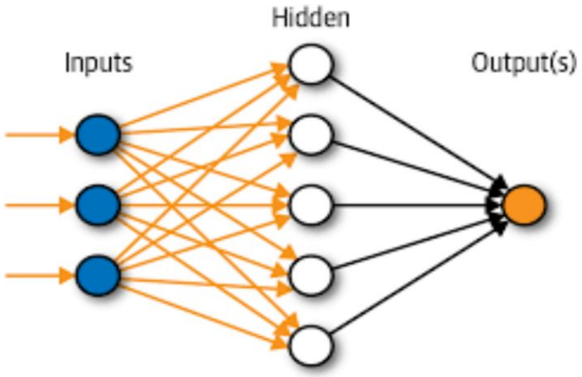
Ramón y Cajal



Neurons are individual brain cells.
Neurons send and receive information.

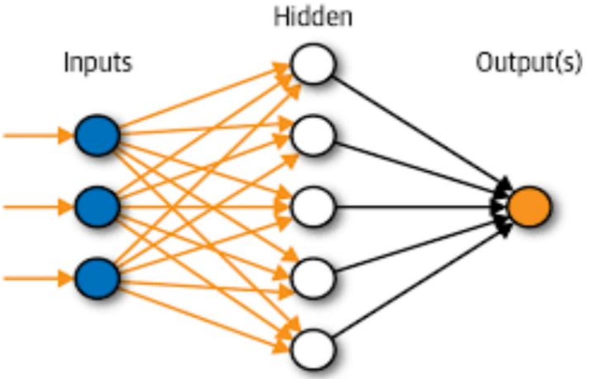
Neural Network: Layers

- A NN consists of a layered, feedforward, completely connected network of neurons.
- Layers : input layer, hidden layer, output layer
- A Feed-Forward NN (FFNN) is composed of two or more layers, but mostly 3 layers, with activation functions usually step or logistic function.
- Multi-Layer-Perceptron (MLP) has three or more layers.
- Perceptron Learning is a NN without a hidden later, (i.e., with two layers of input and output). It is good for emulating the functionality of logical AND and OR , but not good for XOR problem.



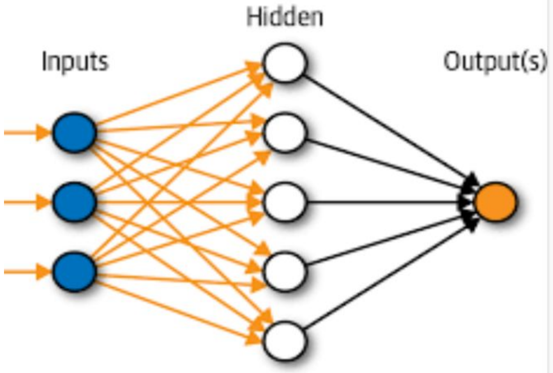
Neural Network: Layers

- Some networks may have more than one hidden layers, but in general 1-2 hidden layers is sufficient.
- Too many hidden layers increases the complexity of the model and training time, especially when the "error" is propagated backwards.
- Increasing the number of hidden layers leads to creating a Deep Neural Network (DNN).



Neural Network: Activation Function

- The activation function in NN makes them non-linear regressors.
- A NN without an activation function is a linear regressor. So, The final layer can be another logistic regression/perceptron (such as sigmoid, tanh, or softmax) or a linear regression model (such as no activation function) depending whether it is a classification or regression problem.


Hidden
Inputs
Output(s)