ntic master
revolucionamos la comunicación

UNIVERSIDAD COMPLUTENSE DE MADRID

Deep Learning

Alberto Ezpondaburu

About Me

60% Math 35% Computers 5% Philosophy

SOME FRAMEWORK PROFICIENCIES

- NLP
- Machine Learning
- Deep Learning
- Signal Processing
- Computer Vision

SOME AREAS OF EXPERTISE

Analytical Index

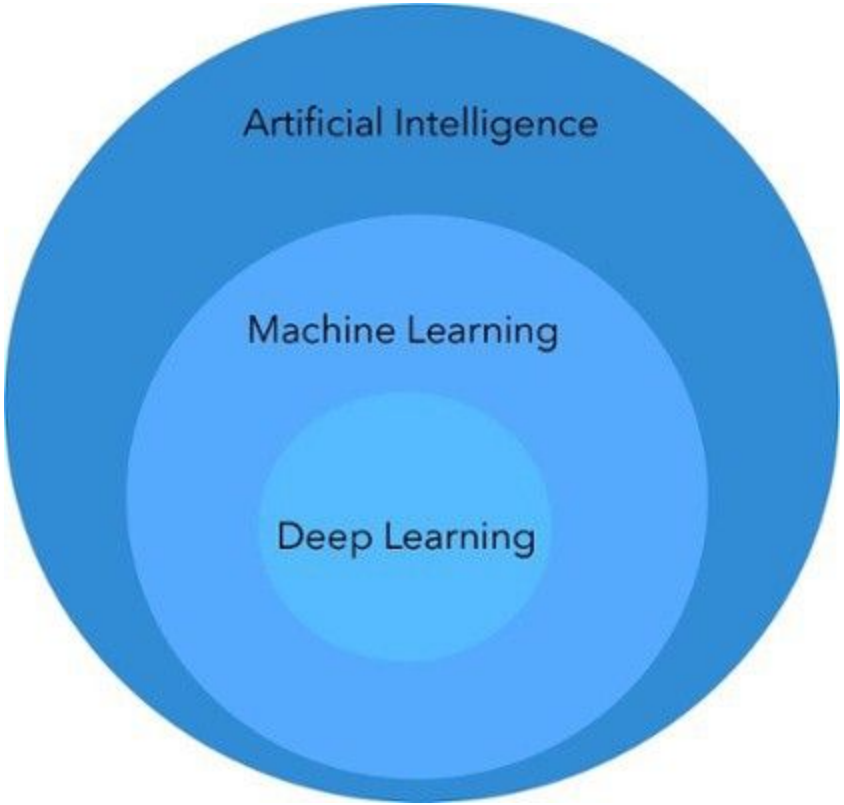
Analytical Index

1. Introduction to Deep Learning
    a. DL vs ML
    b. DL Frameworks
    c. DL Fundamentals
    d. Exercises
2. Convolutional Neural Networks (CNN)
3. Recurrent Neural Networks (RNN)
4. Natural Language Processing with Deep Learning
5. Deep Generative Modelling

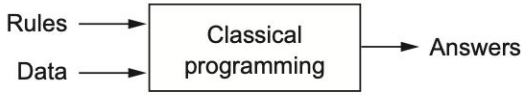
1.1

Introduction To Deep Learning

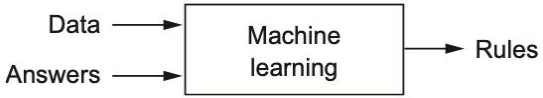
AI vs ML vs DL



- **Artificial Intelligence:** Automate intellectual tasks normally performed by humans.
  - Symbolic: Set of explicit rules



- **Machine Learning:** Algorithms that improve automatically through experience.
  - Trained rather than explicitly programmed

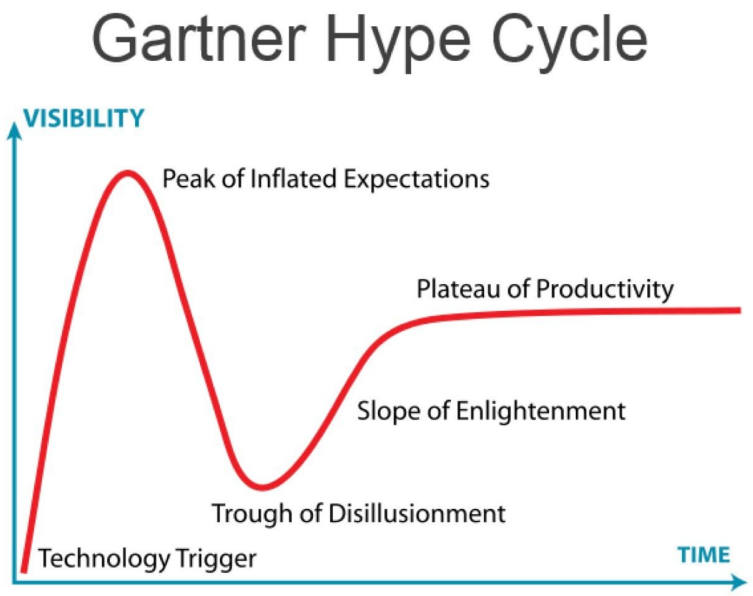


- **Deep Learning:** Artificial Neural Networks

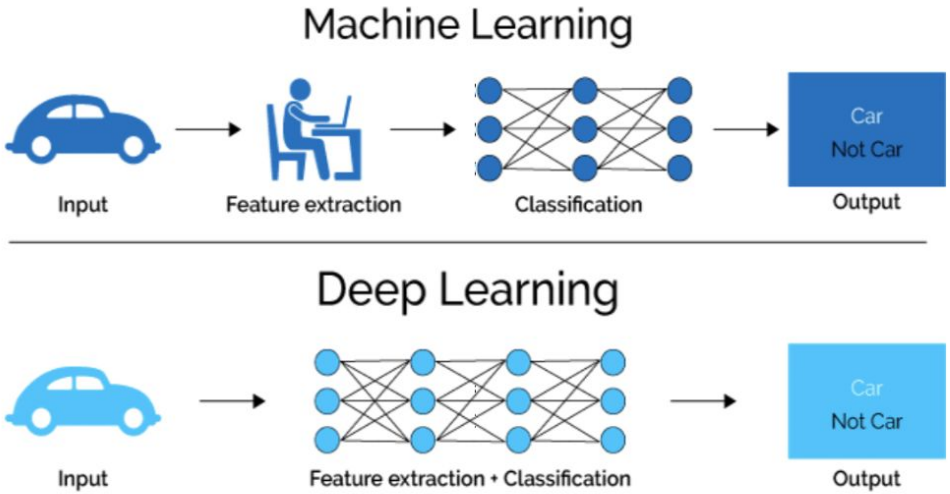
Why Deep Learning?

NLP	Speech recognition, chatbots, machine translation, handwriting generation.
Computer Vision	Face recognition, image captioning, handwriting transcriptions (nlp).
Medicine	Disease identification/diagnosis, many genomic tasks, drug discovery.
Playing Games	Superhuman Chess, Go, Atari games.
Robotics	Self-driving cars
Image/Audio Generation	Text-to-speech, colorizing images.
Recommender Systems	Web search improvements, better product recommendations.

Why Deep Learning?

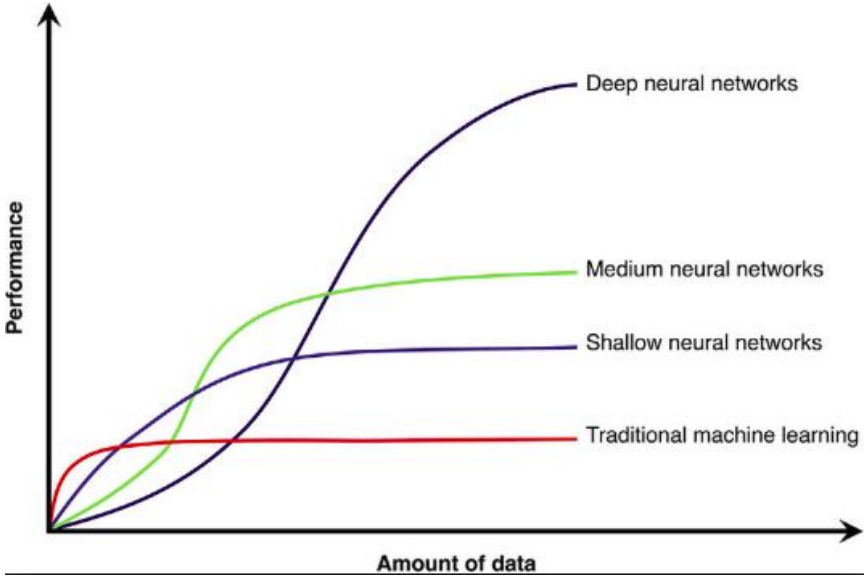


Why Deep Learning?



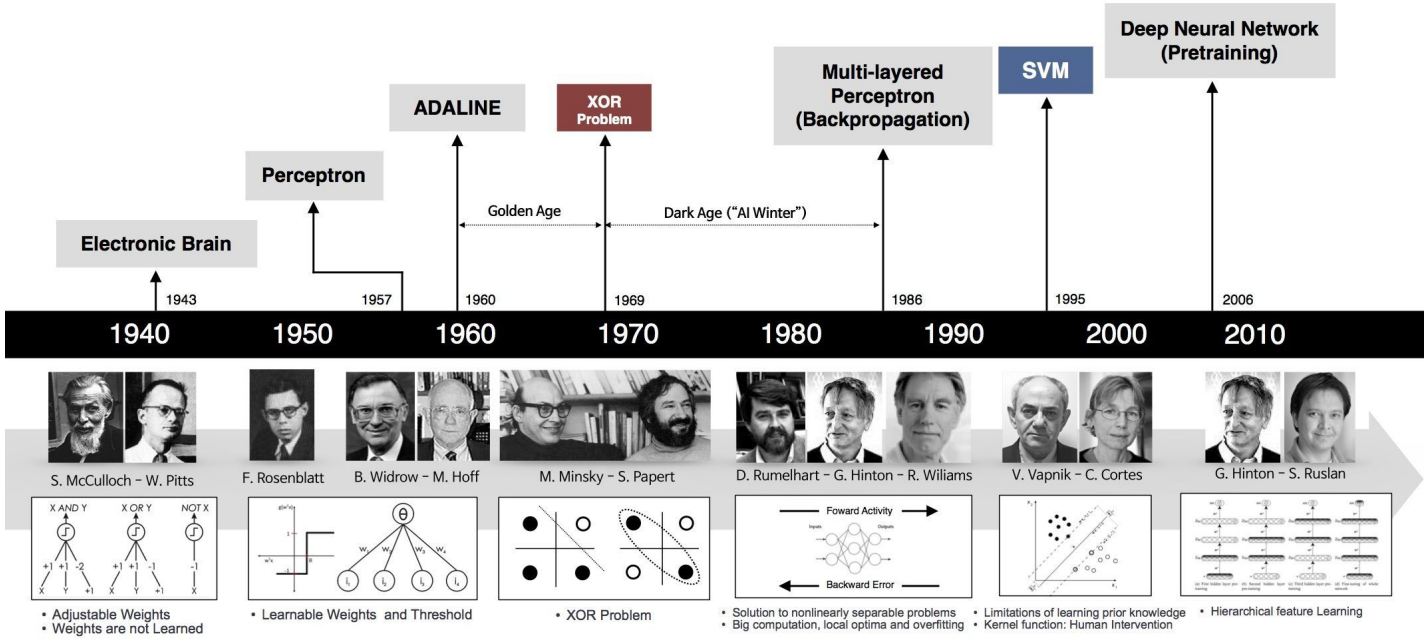
Why Deep Learning?

Why Deep Learning?

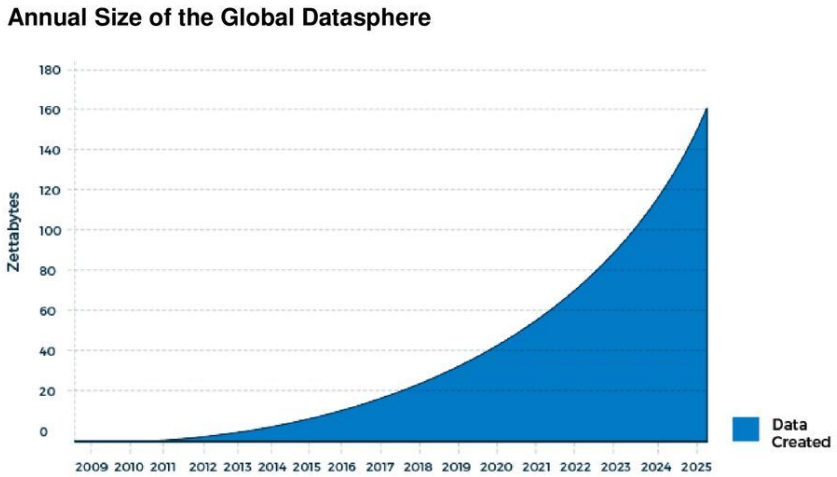


- Data:
  - Structured: tables
  - Unstructured: Audio, image, text
- Performance increases a lot with depth and with the amount of data. Specially with unstructured data.
- With small datasets, traditional machine learning usually performs better.

Why Now?

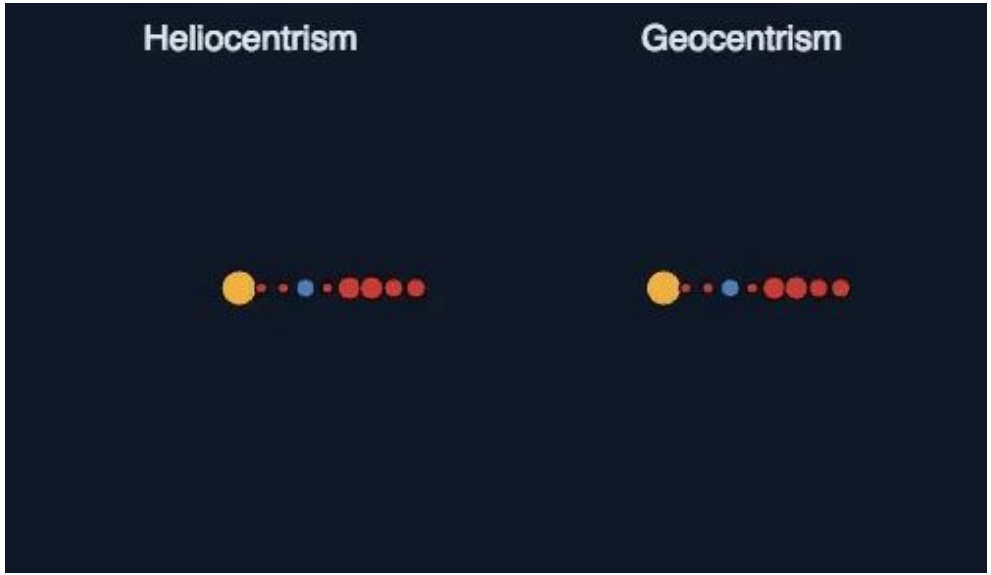

Deep Neural Network (Pretraining)
SVM
Multi-layered Perceptron (Backpropagation)
XOR Problem
ADALINE
Perceptron
Electronic Brain
Golden Age
Dark Age ("AI Winter")
1943
1957
1960
1969
1986
1995
2006
1940
1950
1960
1970
1980
1990
2000
2010
S. McCulloch - W. Pitts
F. Rosenblatt
B. Widrow - M. Hoff
M. Minsky - S. Papert
D. Rumelhart - G. Hinton - R. Wiliams
V. Vapnik - C. Cortes
G. Hinton - S. Ruslan
X AND Y
X OR Y
NOT X
Foward Activity
Backward Error
• Adjustable Weights
• Weights are not Learned
• Learnable Weights and Threshold
• XOR Problem
• Solution to nonlinearly separable problems
• Big computation, local optima and overfitting
• Limitations of learning prior knowledge
• Kernel function: Human Intervention
• Hierarchical feature Learning

Why Now? Data



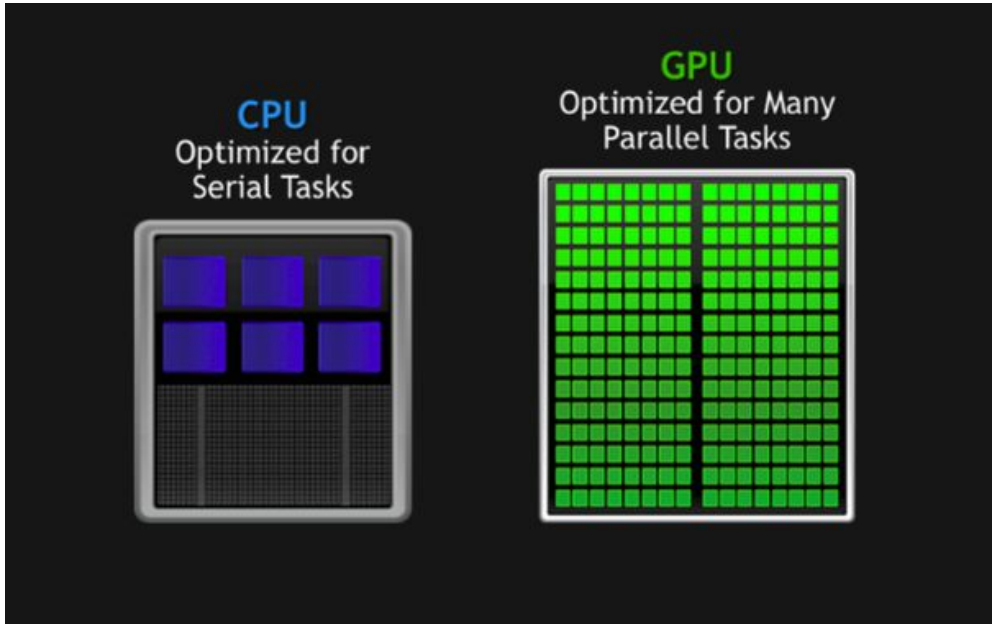
- Raw material that powers AI. The new electricity.
- ZettaByte Era:
  - Connected devices per person : 0.08 in 2000, 3.47 in 2015, 6.58 in 2020.
  - Before computers, humanity accumulated **12 exabytes** (10^18) of data (500 000 years DVD video). In 2002, **5 exabytes** of data.
- Web mining(content, use, structure), IoT, 5G, smartPhones, Wikipedia …
- 4th revolution

Why Now? Data



Why Now? Hardware

- Between 1990-2010 CPUS 5000x faster
- Video games industry: GPU (massive parallel chips)
- NVIDIA GeForce RTX 3090:
  - 2000 $
  - 10496 cores
  - 35.58 TFLOPS, 2 kW
- 2000: IBM ASCI White
  - Best supercomputer
  - 106 tons, 6MW (wind turbine)
  - 12 TFLOPS



Why Now? Software, DL Frameworks:

We cannot build a neural net from the scratch every time we need one, right? ;)

The most important frameworks need to be/have:

- Optimized for performance.
- Easy to code
- Need to have a good community
- Need to automatically compute gradients

R.I.P.

theano

Caffe

Contenders

mxnet

DEEPLEARNING4J

Rockstars

PYTORCH

TensorFlow

Keras

Pytorch

- Open-source, Python-based machine and deep learning framework.
- Primarily developed by Facebook's AI Research lab (FAIR), first release in 2016.
- Easy to use API.
- **Dynamic Computation Graphs**: the graph is built along and can be manipulated at run-time.
- The framework is more “Pythonic”
- Extended in academia

Tensorflow basics

- Was released in 2015 by the Google Brain Team.
- It can automatically compute the gradient of any differentiable expression
- It can run not only on CPU, but also on GPUs and TPUs.
- Since Tensorflow 2.0 (2019): dynamic graph definitions similar to PyTorch.
- Scalable production and deployment options, and support for various devices like Android.
- It is easy to deploy TensorFlow models on lightweight platforms such as mobile or IoT devices with Tensorflow Lite

Keras

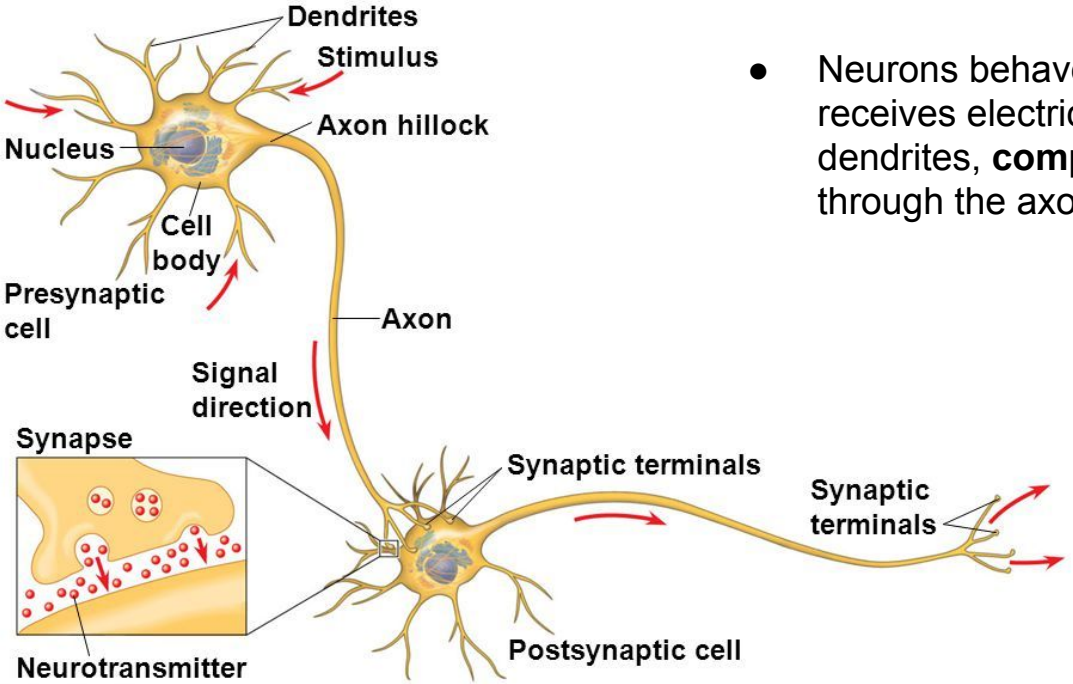
- TensorFlow project has adopted Keras as the high-level API for TensorFlow 2.0 release.
- Simple architecture. It is more readable and concise
- Keras was developed and maintained by François Chollet, a Google engineer:
  - Modularity
  - Minimalism
  - Extensibility:
  - Python: Everything is native Python.

Example: Digit Recognition

1.2

Deep Learning Fundamentals

Bio-inspired Model

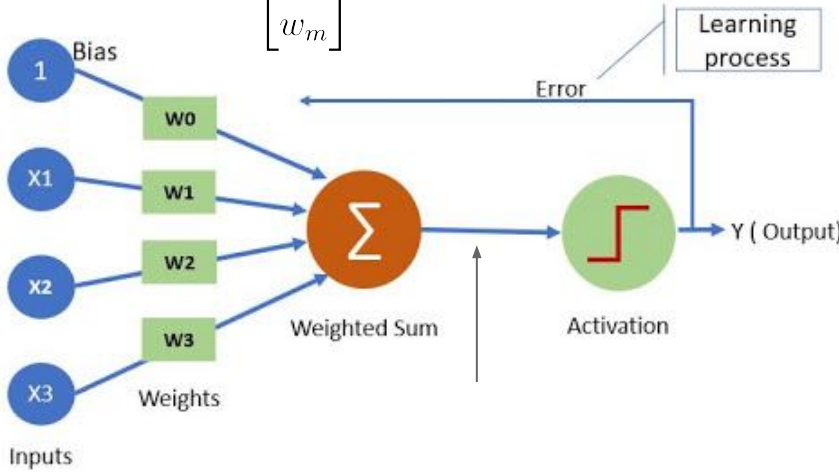


- Neurons behave as computing units. Each neuron receives electric input signals (called *spikes*) through the dendrites, **computes** an output with them and sends it through the axon to other neurons connected to it

- Axon-dendrite transmission is done in a mechanism called synapse.
- This process never changes, yet the brain is always learning

Perceptron: Forward Propagation

$$\mathbf{w} = \begin{bmatrix} w_0 \\ \vdots \\ w_m \end{bmatrix}$$



Output

Linear combination of inputs

$$y = g(w_0 x_0 + w_1 x_1 + w_2 x_2 + w_3 x_3) = g\left(w_0 x_0 + \sum_{i=1}^{m} w_i x_i\right) = g(\mathbf{w}^T \mathbf{x})$$

Non-linear activation function

$$\mathbf{x} = \begin{bmatrix} x_0 \\ \vdots \\ x_m \end{bmatrix}$$

Single Layer Perceptron

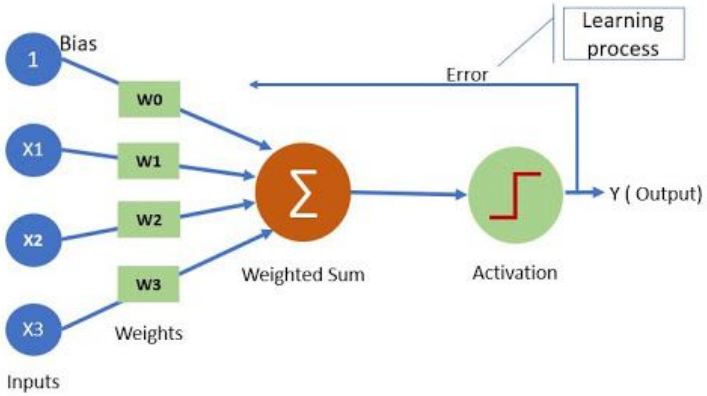
Perceptron: Logistic Regression

- Activation function:
  - sigmoid
- Loss function:
  - binary cross-entropy
- Gradients:

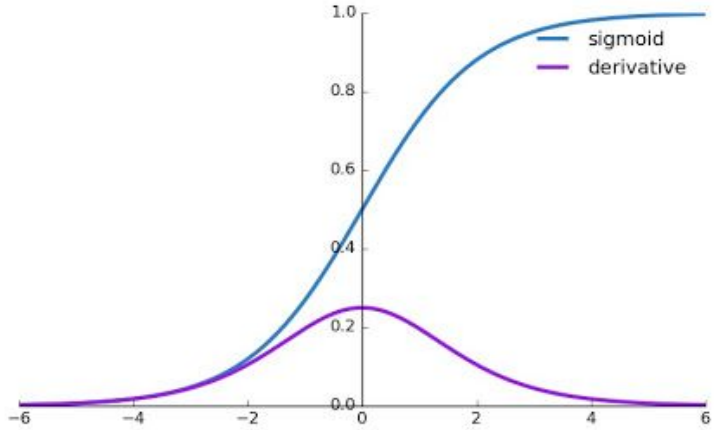
$$\hat{y} \leftarrow P(y=1|\mathbf{x}) = \sigma(\mathbf{w}^T\mathbf{x}) = \frac{1}{1+e^{-\mathbf{w}^T\mathbf{x}}}$$

$$L_{\mathbf{w}}(\hat{y}, y) = \frac{1}{m}\sum_{i=1}^{m}\left[-y_i \log\left(\sigma(\mathbf{w}^T\mathbf{x}_i)\right) - (1-y_i)\log\left(1-\sigma(\mathbf{w}^T\mathbf{x}_i)\right)\right]$$

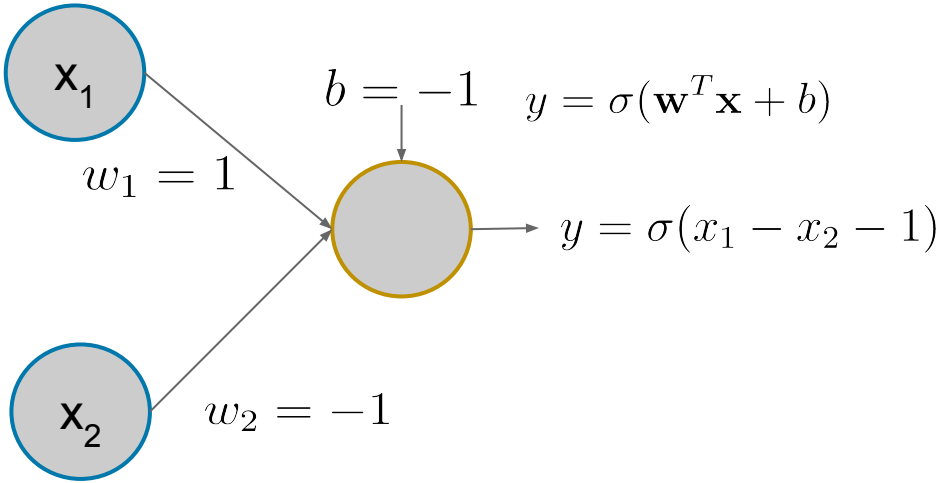
$$\nabla_{w_j} L_{\mathbf{w}}(\hat{y}, y) = \frac{1}{m}\sum_{i=1}^{m}\left(\sigma(\mathbf{w}^T\mathbf{x}_i) - y_i\right)x_j$$

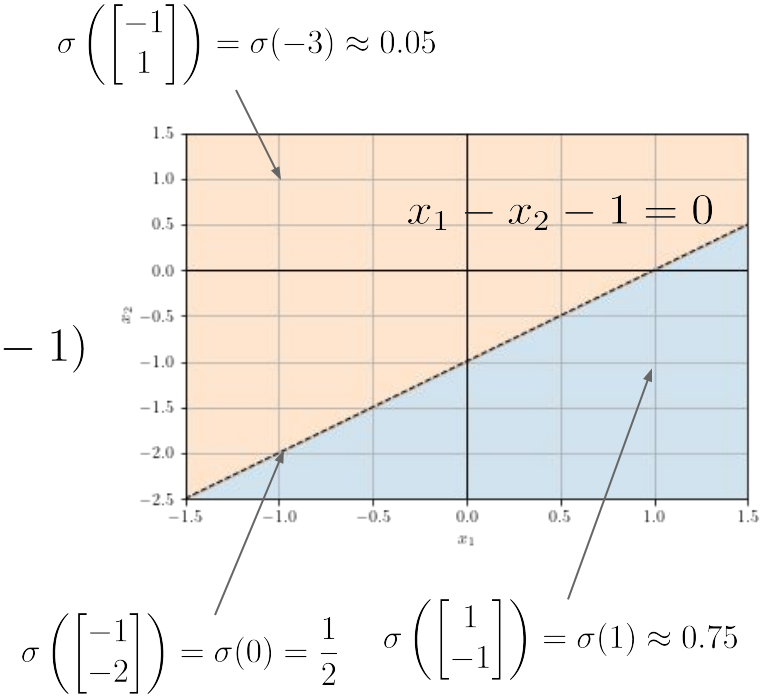


Single Layer Perceptron

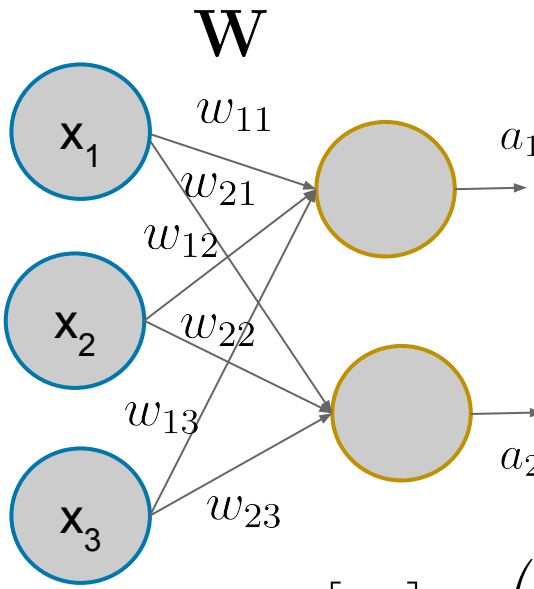


Perceptron: Example





Perceptron: Multi Output



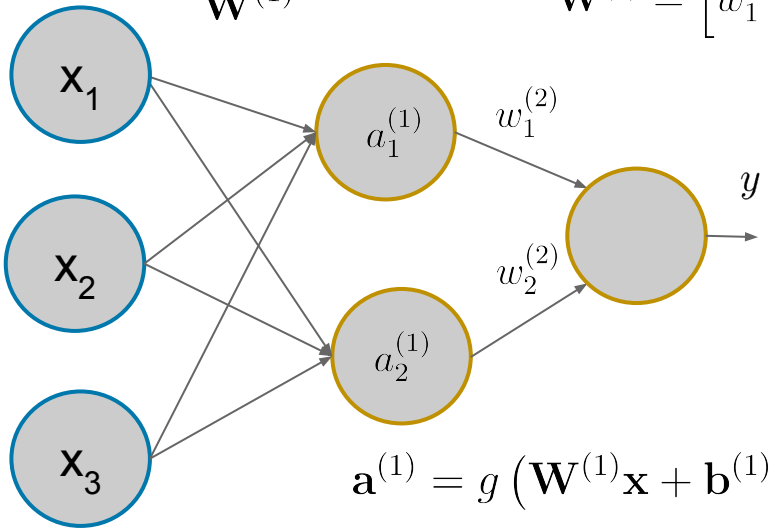
$$a_1 = g(w_{11}x_1 + w_{12}x_2 + w_{13}x_3 + b_1)$$

$$\begin{cases} a_1 = g(w_{11}x_1 + w_{12}x_2 + w_{13}x_3 + b_1) \\ a_2 = g(w_{21}x_1 + w_{22}x_2 + w_{23}x_3 + b_2) \end{cases}$$

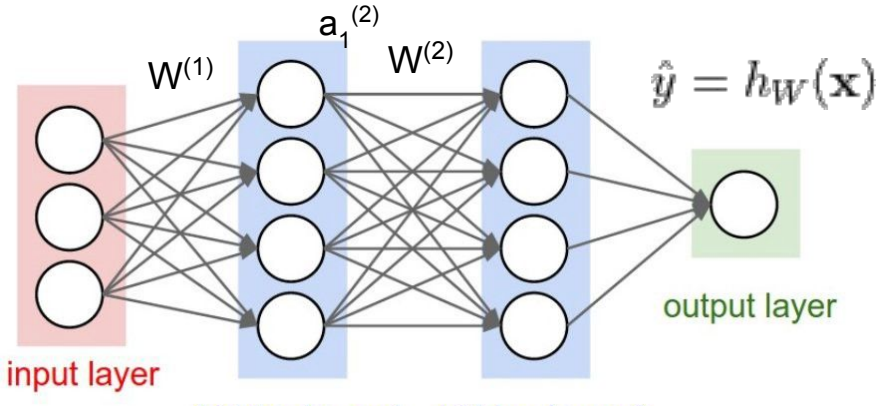
$$a_2 = g(w_{21}x_1 + w_{22}x_2 + w_{23}x_3 + b_2)$$

$$\mathbf{a} = \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} = g\left(\begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & \omega_{23} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \right) = g\Big(\mathbf{W}\mathbf{x} + \mathbf{b}\Big)$$

Single Layer Neural Network



Multi-Layer Perceptron MLP



$$\mathbf{a}^{(i)} = g(W^{(i)}\mathbf{a}^{(\mathbf{i-1})} + \mathbf{b}^{(\mathbf{i})})$$

$$\mathbf{a}^{(i)} = g(W^{(i)}g(W^{(i-1)}\mathbf{a}^{(\mathbf{i-2})} + \mathbf{b}^{(\mathbf{i-1})}) + \mathbf{b}^{(\mathbf{i})})$$

$$\hat{y} = h_W(\mathbf{x}) = g(g(g(g(\cdots))))$$

Dense, Feed forward ...

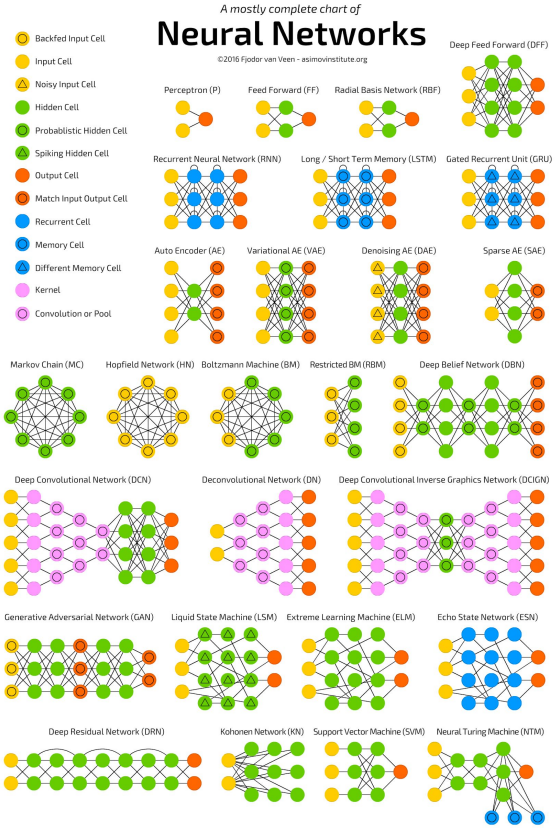
$a_i^{(j)}$ = activation of the i-th neuron of layer j

$W^{(j)}$ = matrix of parameters multiplied by the inputs (activations) from layer j to compute the activations of layer j+1

We see the whole network as a function $h_{\mathbf{w}} : \mathbb{R}^p \rightarrow \mathbb{R}^K$

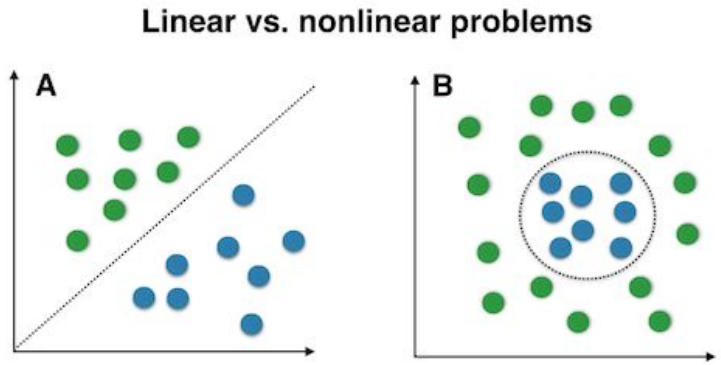
(*p* is the number of features)
(*K* is the number of classes)

NN Architectures



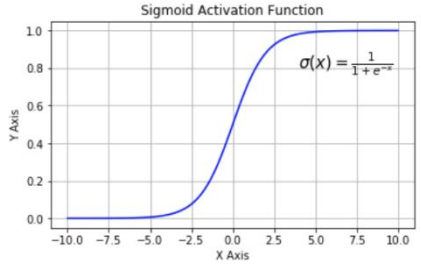
MLP: Introduction to Keras

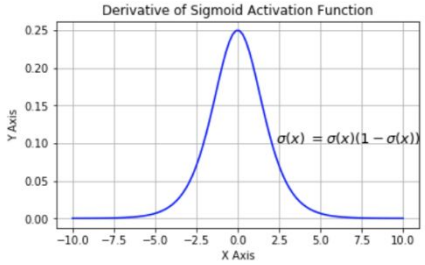
Activation Functions I

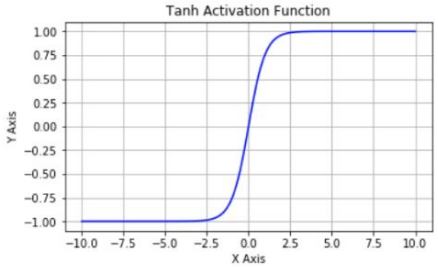


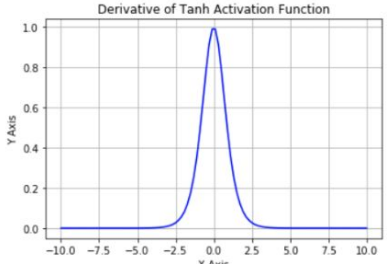
Introduce non-linearities into the network and helps to approximate nonlinear complex functions

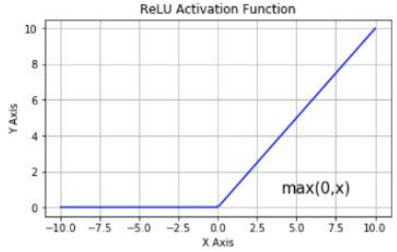
Activation Functions II

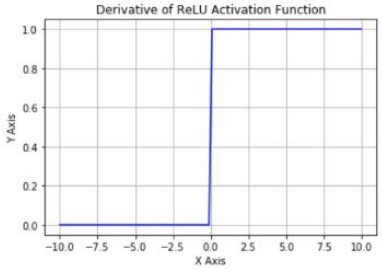












- Sigmoid: (0,1)
  - **Output:** binary classification.
  - Vanishing gradients
  - Non-zero centered

```
tf.keras.activations.sigmoid(x)
```

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

- Tanh: (-1,+1)
  - Vanishing gradients

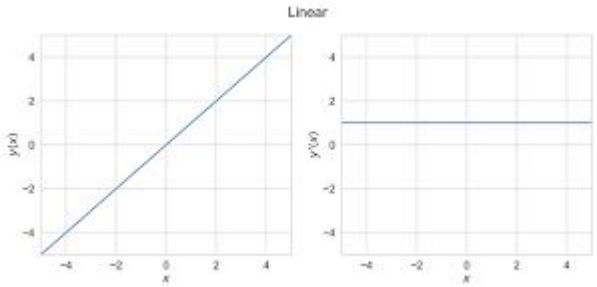
```
tf.keras.activations.tanh(x)
```

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

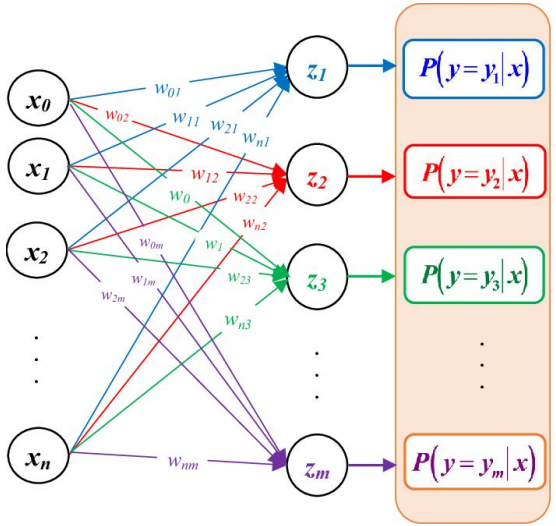
- ReLu: [0, +∞) $\mathrm{ReLu}(x) = \max(0, x)$
  - **Output:** Positive regression.
  - Non-zero centered

```
model.add(layers.Dense(64, activation='relu'))
```

Activation Functions III



- Linear: (-∞, +∞)
  - **Output:** Regression.
  - Only for output layer.
  - Any activation.



- Softmax:
  - Softmax converts a real vector to a vector of categorical probabilities.
  - Output: Multi-class classification.
  - Multiple neurons.

$$\text{softmax}(\mathbf{x})_i = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

```
tf.keras.activations.softmax(x, axis=-1)
```

Loss/Cost Function

We need to find the best parameters for model the given data.

Loss function: quantifies the error in prediction, how the model deviates from the correct results.

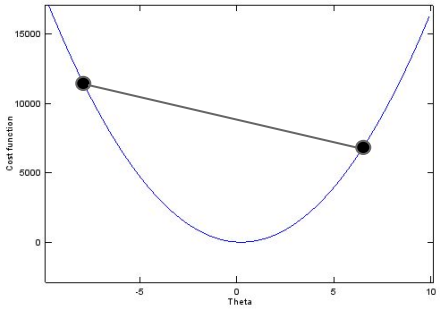
Objective function, cost function: Measure the total loss over the entire dataset

$$\mathcal{L}\left(f\left(\boldsymbol{x}^{(i)};\boldsymbol{W}\right),\boldsymbol{y}^{(i)}\right)$$

Prediction

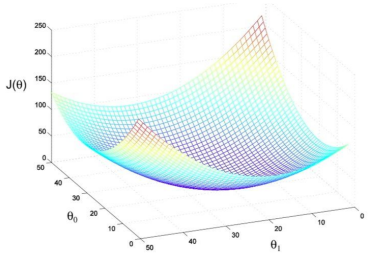
Correct

$$J(\boldsymbol{W}) = \frac{1}{n}\sum_{i=1}^{n}\mathcal{L}\left(f\left(\boldsymbol{x}^{(i)};\boldsymbol{W}\right),\boldsymbol{y}^{(i)}\right)$$



Cost function of one parameter

Ideally the **cost function is convex**: for every pair of points, the curve is always below the line (or hyper-plane) between them



Cost function of two parameters

Cost Functions: Regression

- Predict a numerical value given some input.
- **Mean squared error (MSE):** Average of the **square** of the difference between the predicted and actual target variables.
  - The most classic measure.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$$

```
model.compile(optimizer='sgd',
loss='mse')
```

- **Mean absolute error (MAE)**: Average of the **absolute value** of the difference between the predicted and actual target variables.
  - Harder differentiability and convergence

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|\hat{y}_i - y_i|$$

```
model.compile(optimizer='sgd',
loss='mae')
```

Cost Functions: Classification

- Specify which of k categories some input belongs to.

- **Binary Cross-Entropy (log-loss):** Similar to the logistic loss function

$$\mathcal{L}(\hat{y}, y) = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot \log(\hat{y}_i) + (1 - y_i) \cdot \log(1 - \hat{y}_i)$$

```
model.compile(optimizer='sgd', loss='binary_crossentropy')
```

- **Categorical Cross-Entropy:**
  - Minimizing the cross-entropy is equivalent to minimizing the **divergence** between the distribution of the real targets and the predictions.

$$\mathcal{L}(\hat{y}, y) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K} y_{ik} \cdot \log(\hat{y_{ik}})$$

```
model.compile(optimizer='sgd',
loss='categorical_crossentropy')
```

```
model.compile(optimizer='sgd', loss='sparse_categorical_crossentropy')
```

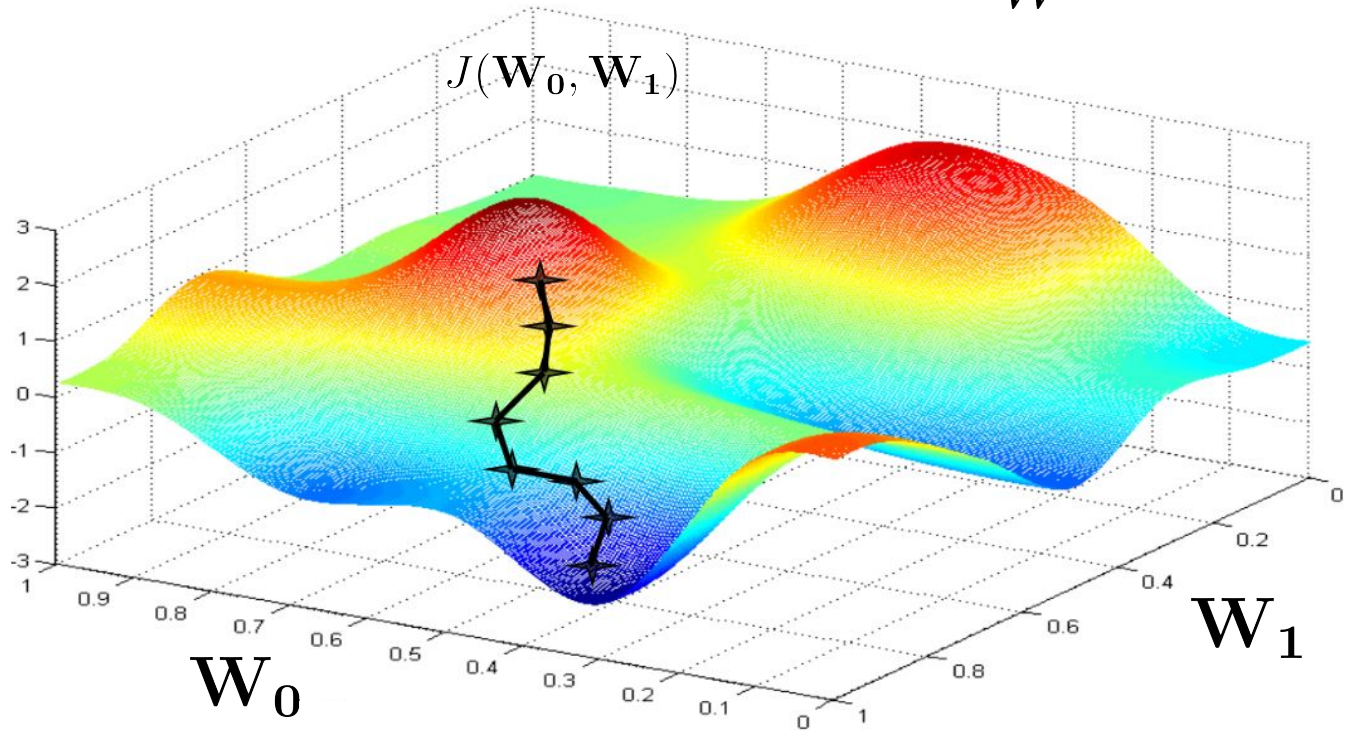
Optimization: Training The Network

$$\boldsymbol{W}^* = \underset{\boldsymbol{W}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}\left(f\left(x^{(i)}; \boldsymbol{W}\right), \boldsymbol{y}^{(i)}\right)$$

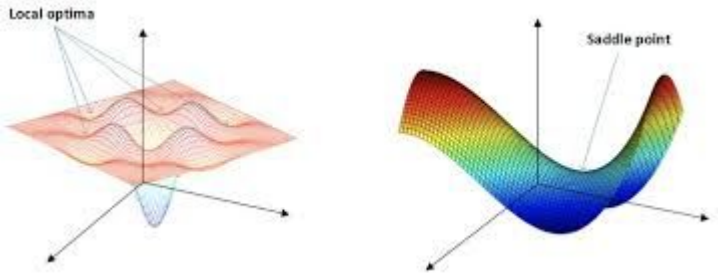
$$\boldsymbol{W}^* = \underset{\boldsymbol{W}}{\operatorname{argmin}} J(\boldsymbol{W})$$

Optimization

Follow gradient direction $\nabla_W J$

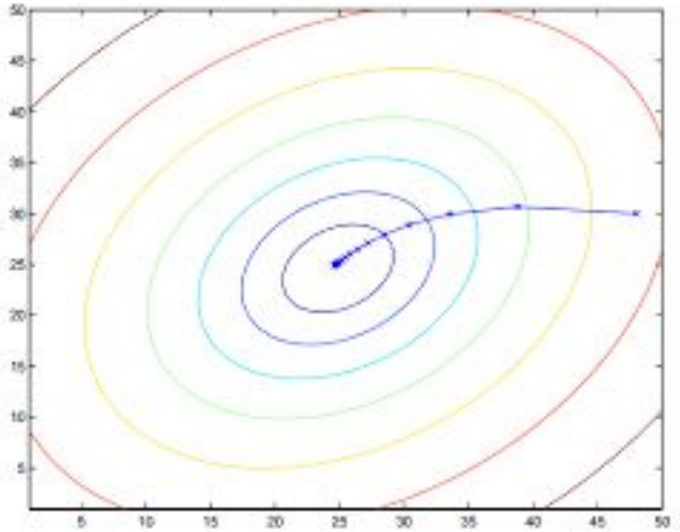


Optimization: Challenges



- The difficulty of training neural networks is mainly attributed to their optimization part.
- Plateaus, saddle points and local minima grows exponentially with the dimension.
- Classical convex optimization algorithms don’t perform well.

Optimization: Gradient Descend



- Initialize weights randomly
- In each step update weights until convergence:

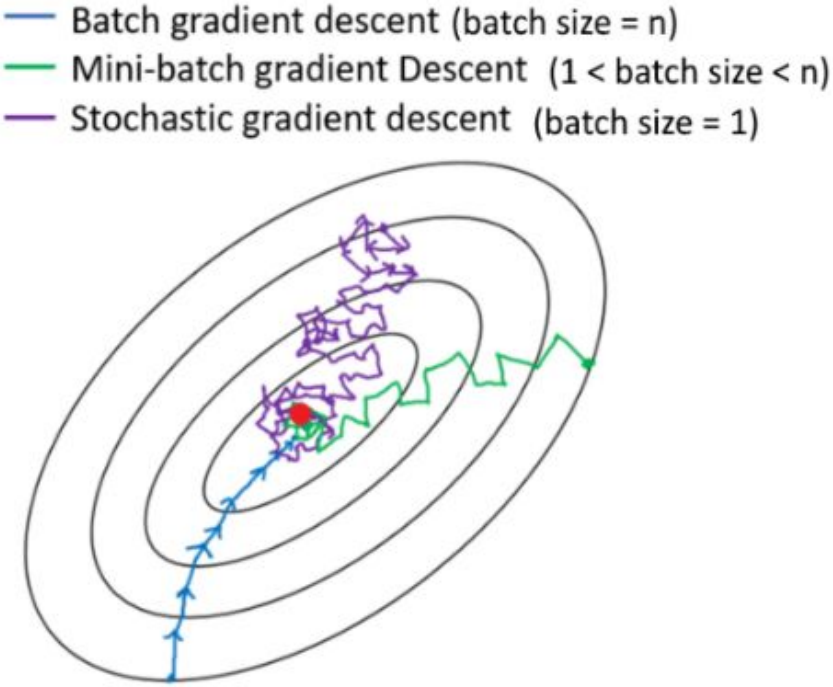
$$\mathbf{W} \leftarrow \mathbf{W} - \eta \cdot \nabla_W J(\mathbf{W})$$

- In every step goes through the entire dataset

$$\frac{1}{n}\sum_{i=1}^{n} \mathcal{L}\left(f\left(x^{(i)}; \boldsymbol{W}\right), \boldsymbol{y}^{(i)}\right)$$

- We need big datasets…
- Good for convex functions.

Optimization: Mini-batch Gradient Descent



- **Stochastic gradient descent:**
  - Estimate gradient with only one sample choose randomly

$$\nabla_W J(\mathbf{W}) \approx \mathcal{L}\left(f\left(x^{(i)}; \boldsymbol{W}\right)\right.$$

- **Mini-batch gradient descent:**
  - estimate gradient with some samples (m)

$$\nabla_W J(\mathbf{W}) \approx \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}\left(f\left(x^{(i)}; \boldsymbol{W}\right)\right.$$

- **Batch size:** Length of the minibatch
- **Iteration:** Every time we update the weights
- **Epoch:** One pass over the whole training set.

Optimization: Advances

$$\omega_t = \omega_{t-1} + \Delta\omega_t$$
$$\Delta\omega_t = -\frac{\mu}{\sqrt{v_t + \epsilon}} \cdot g_t$$
$$v_t = \rho \cdot v_{t-1} + (1 - \rho) \cdot g_t^2$$

v_t exponential average of squares of gradients

μ initial learning rate

g_t gradient at time t at ω

Root Mean Square Propagation (RMSProp)

$$\omega_t = \omega_{t-1} + \Delta\omega_t$$
$$\Delta\omega_t = -\frac{\mu \cdot v_t}{\sqrt{s_t + \epsilon}} \cdot g_t$$
$$v_t = \beta_1 \cdot v_{t-1} + (1 - \beta_1) \cdot g_t$$
$$s_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$$

v_t exponential average of gradients

s_t exponential average of squares of gradients

μ initial learning rate

β_1, β_2 hyperparameters

g_t gradient at time t at ω

Adaptive Moment Optimization (Adam)

Optimization: Keras

- RMSprop
- SDG
- Adam

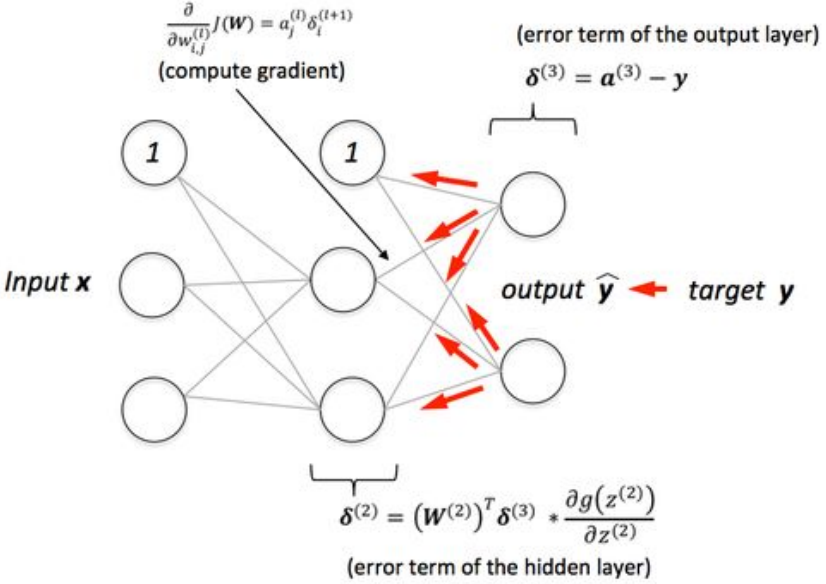
```
tf.keras.optimizers.RMSprop(learning_rate=0.1)
```

```
tf.keras.optimizers.SGD(learning_rate=0.1)
```

```
tf.keras.optimizers.Adam(learning_rate=0.1)
```

```
from tensorflow import keras
model = keras.Sequential(
    [
        keras.layers.Dense(
            256, activation="relu",
input_shape=input_shape
        ),
        keras.layers.Dense(1, activation="sigmoid"),
    ]
)
model.compile(
    optimizer=keras.optimizers.Adam(1e-2),
loss="binary_crossentropy"
)
```

Backpropagation

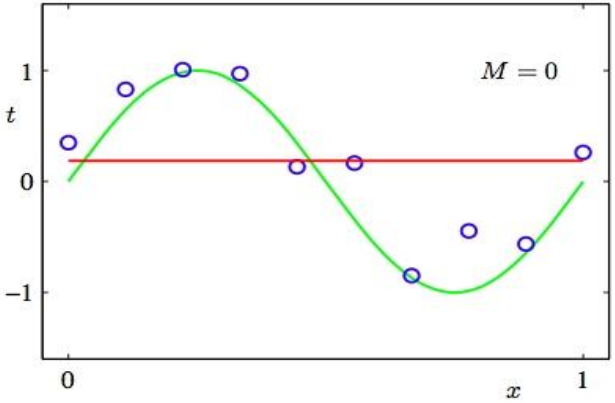


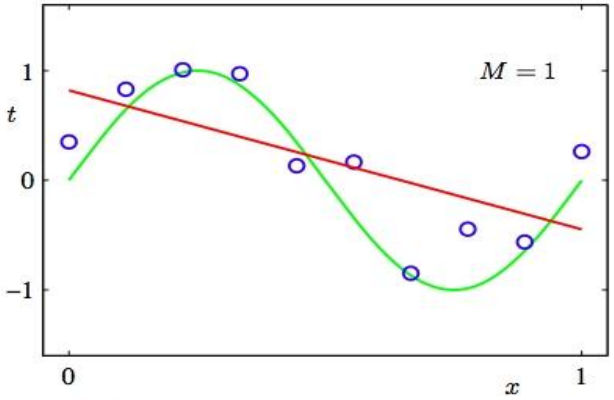
Algorithm to compute the partial derivatives of the cost function with respect to each parameter.

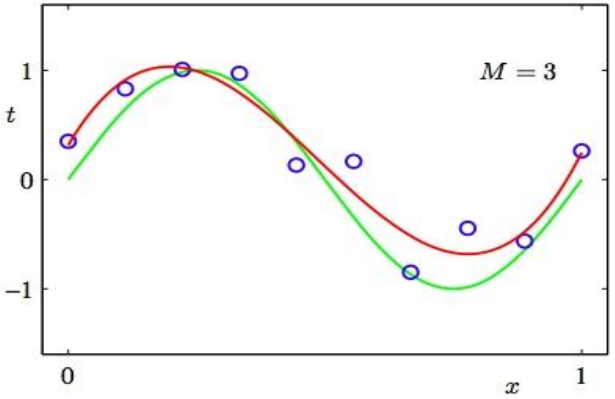
- **Intuition**: compute the contribution of each neuron to the final error, and change its weights accordingly
  - Compute the contribution (*deltas*) of each neuron to the error of each example separately, and then accumulate over all examples to obtain the total contribution of each neuron to the total error.

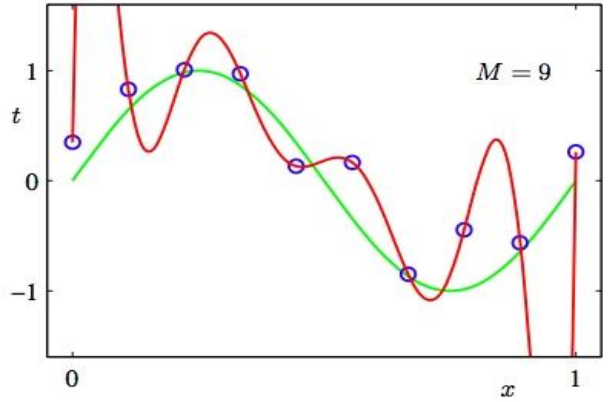
Example: we first apply forward propagation to compute every $a_j^{(\ell)}$ and the output of the network $h_B(\mathbf{x})$

Regularization: Overfitting

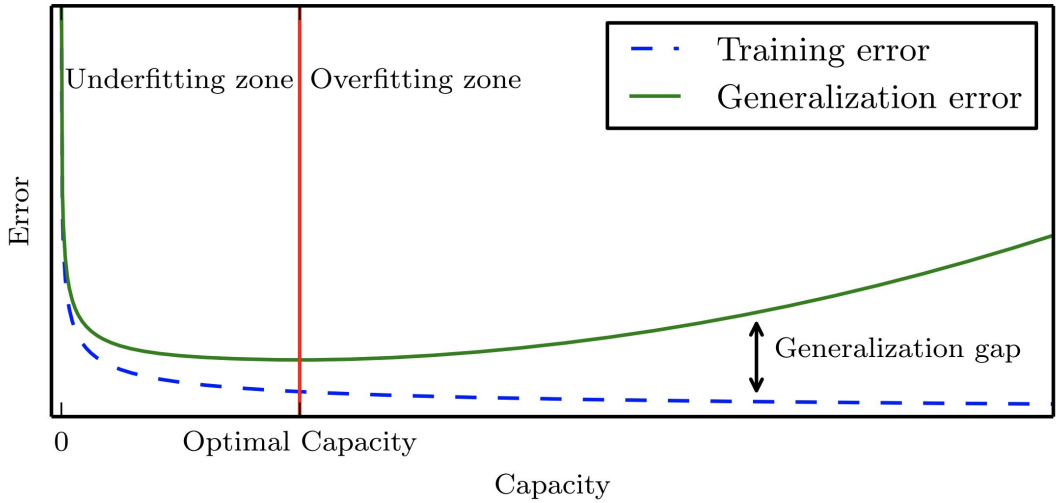








Regularization: Overfitting



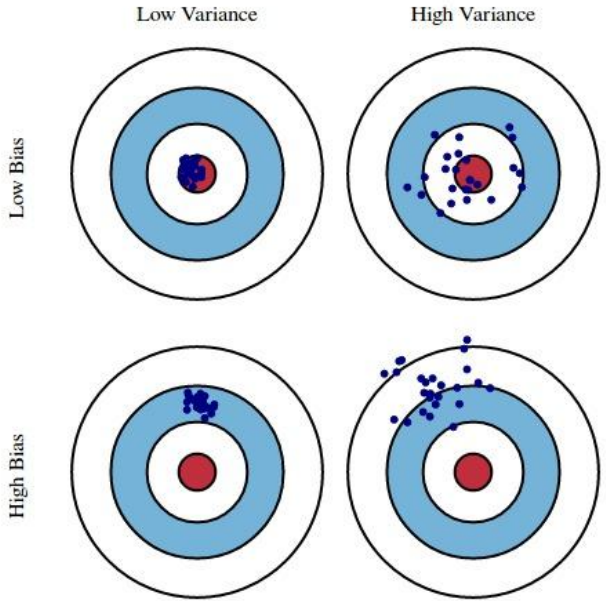
Generalization: The ability to perform well on previously unobserved inputs.

During training we reduce **training error**.

Generalization error (test error): Expected value of the error in a new unobserved input. We want reduce test error, not only training (as in an optimization task)

Capacity: ability to fit a wide variety of functions.

Regularization: Overfitting



Generalization : The ability to perform well on previously unobserved inputs.

Trade off :

- Overfitting :
  - Low bias, High variance.
  - Complex, flexible model.
  - Gap between training and test error is big
- Underfitting :
  - High bias, Low Variance
  - Simple, rigid model.
  - Training error is big

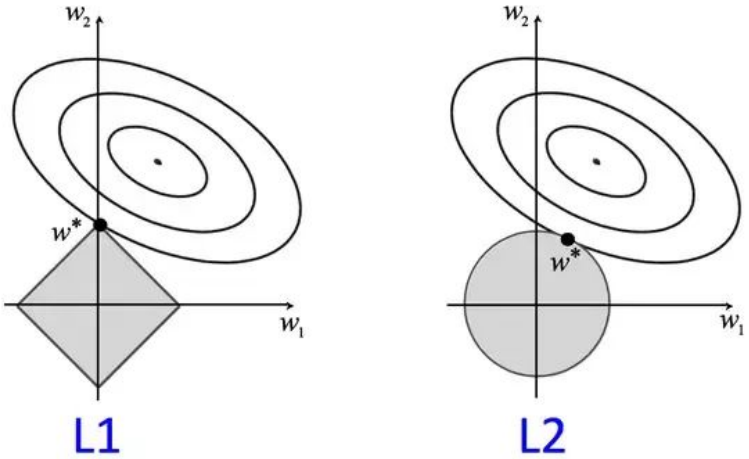
$$\text{Bias}_D\left[\hat{f}(x;D)\right] = \text{E}_D\left[\hat{f}(x;D)\right] - f(x)$$

$$\text{Var}_D\left[\hat{f}(x;D)\right] = \text{E}_D[\left(\text{E}_D[\hat{f}(x;D)] - \hat{f}(x;D)\right)^2]$$

Regularization: Fundamentals

- **Regularization:** Any modification we make to a learning algorithm for reducing its generalization error but not its training error.
- Central problem in machine learning.
- Lots of types:
  - Adding restrictions on the parameter values.
  - Adding extra terms in the objective function.
- In deep learning, trading increased bias for reduced variance.
- Ensemble methods.

Regularization: L2 y L1



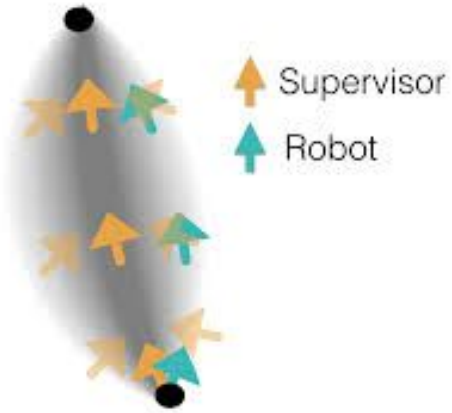
- Add a penalty to the cost function:

$$\tilde{J}(W) = J(W) + \lambda \cdot F(W)$$

- L2: $||W||_2^2 = \sum_i w_i^2$
  - Keeps weights near zero.
  - Simplest one, differentiable.

- L1: $||W||_1 = \sum_i |w_i|$
  - Sparse results, feature selection.
  - Not differentiable, slower.

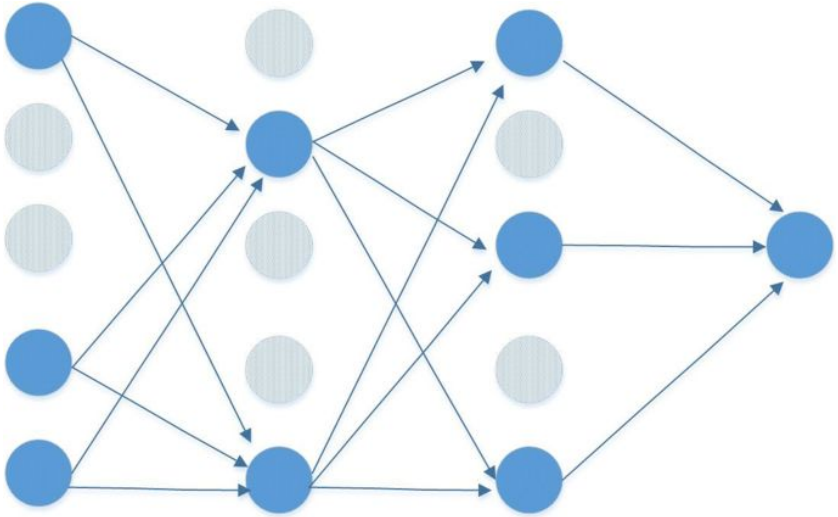
Regularization: Noise injection



Noise Injection

- Add Random Noise During Training.
- Injecting noise in the input to a neural network can also be seen as a form of data augmentation.
- With a bayesian point its equivalent to other regularization:
  - Gaussian noise: L2
  - Laplacian noise: L1

Regularization: Dropout

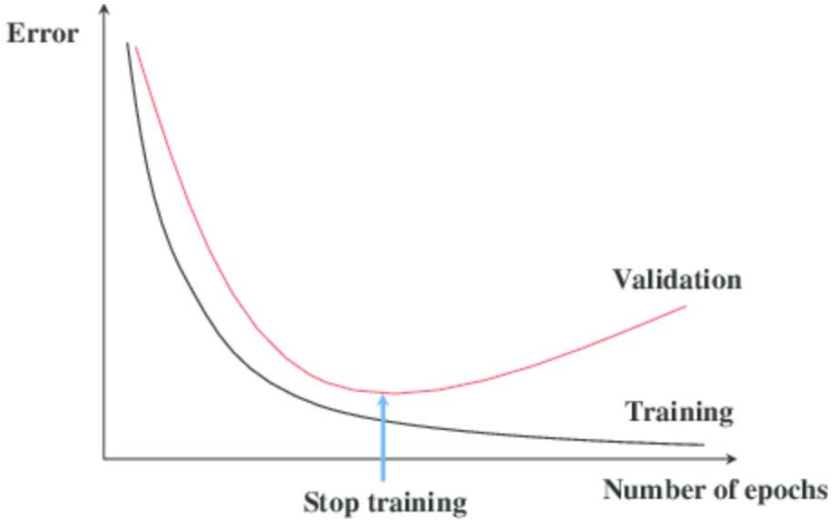


- During training, randomly set some activations to 0 with probability p.
- Not in prediction.

```
from tensorflow.keras.layers import
Dropout
model = Sequential()
model.add(Dense(60, activation='relu',
input_shape=input_shape))
model.add(Dropout(0.2))
model.add(Dense(30, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

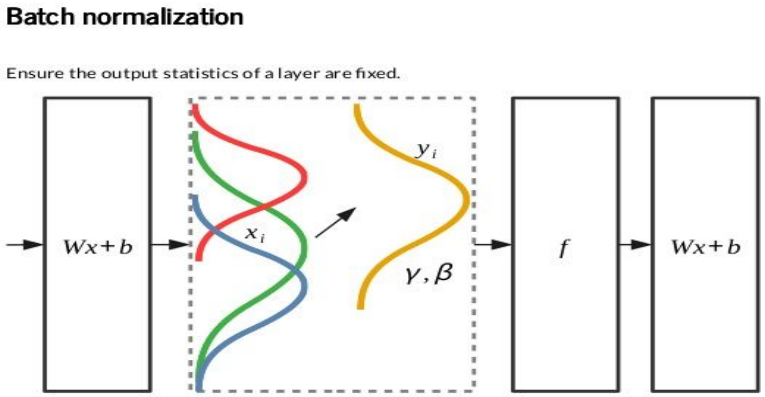
Regularization: Early Stopping

Stop training before overfit



Regularization: batch normalization

- Re-centering and re-scaling inputs to a layer in every mini-batch
- Accelerate the training




```
from keras.layers.normalization import
BatchNormalization
model = Sequential()
model.add(Dense(64, input_shape))
model.add(BatchNormalization())
model.add(Activation('tanh'))
model.add(Dropout(0.5))
```

Data Augmentation

1.3

Exercices

Exercises

UNIVERSIDAD
COMPLUTENSE
DE MADRID