iTrain (M) Sdn Bhd
KL: C-19-8, KL Trillion, 338 Jalan Tun Razak, 50400 Kuala Lumpur. Tel: +603 2733 0337
Website: www.itrain.com.my

Introduction to Deep Architectures of Neural Networks

By: Fares Hasan

What is Artificial Intelligence?

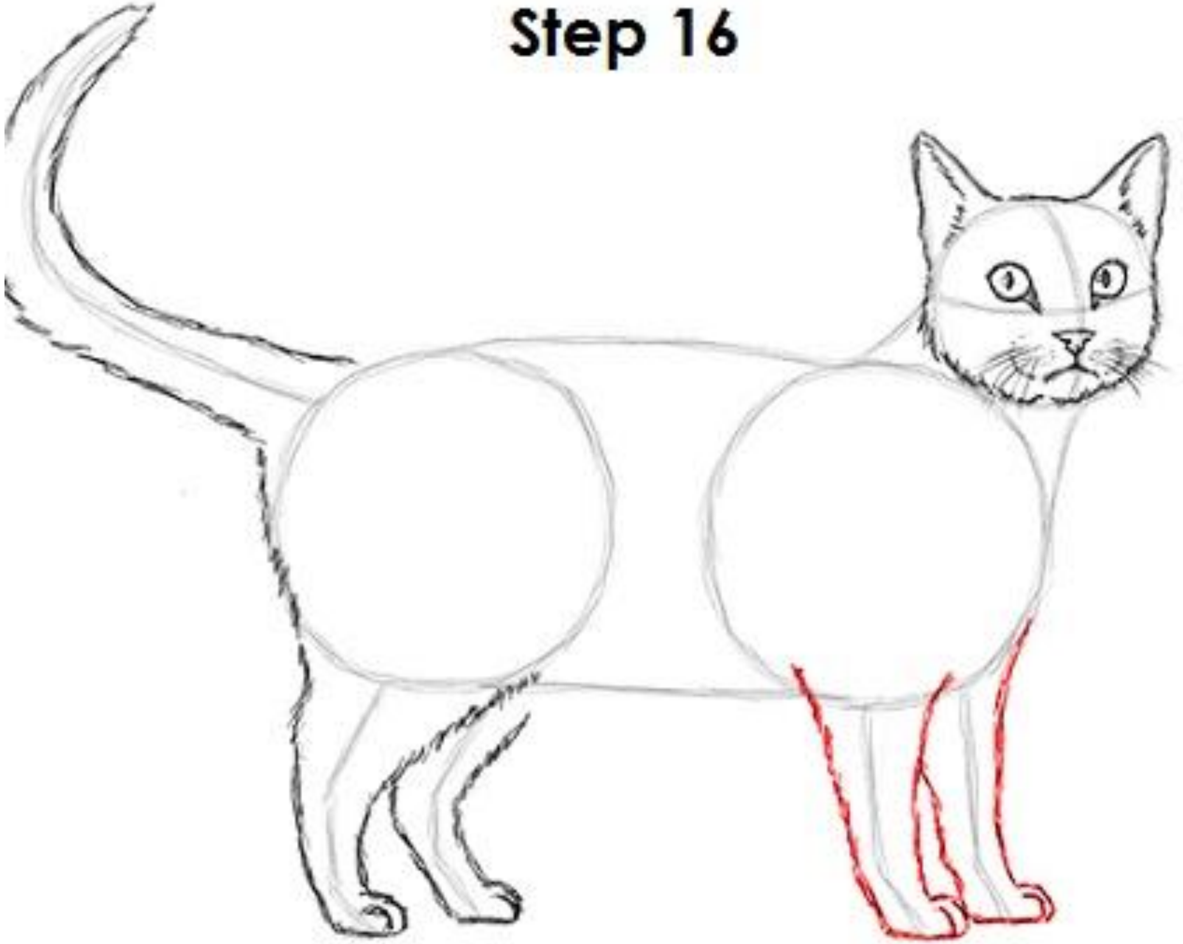
How does a machine learn to recognize objects?

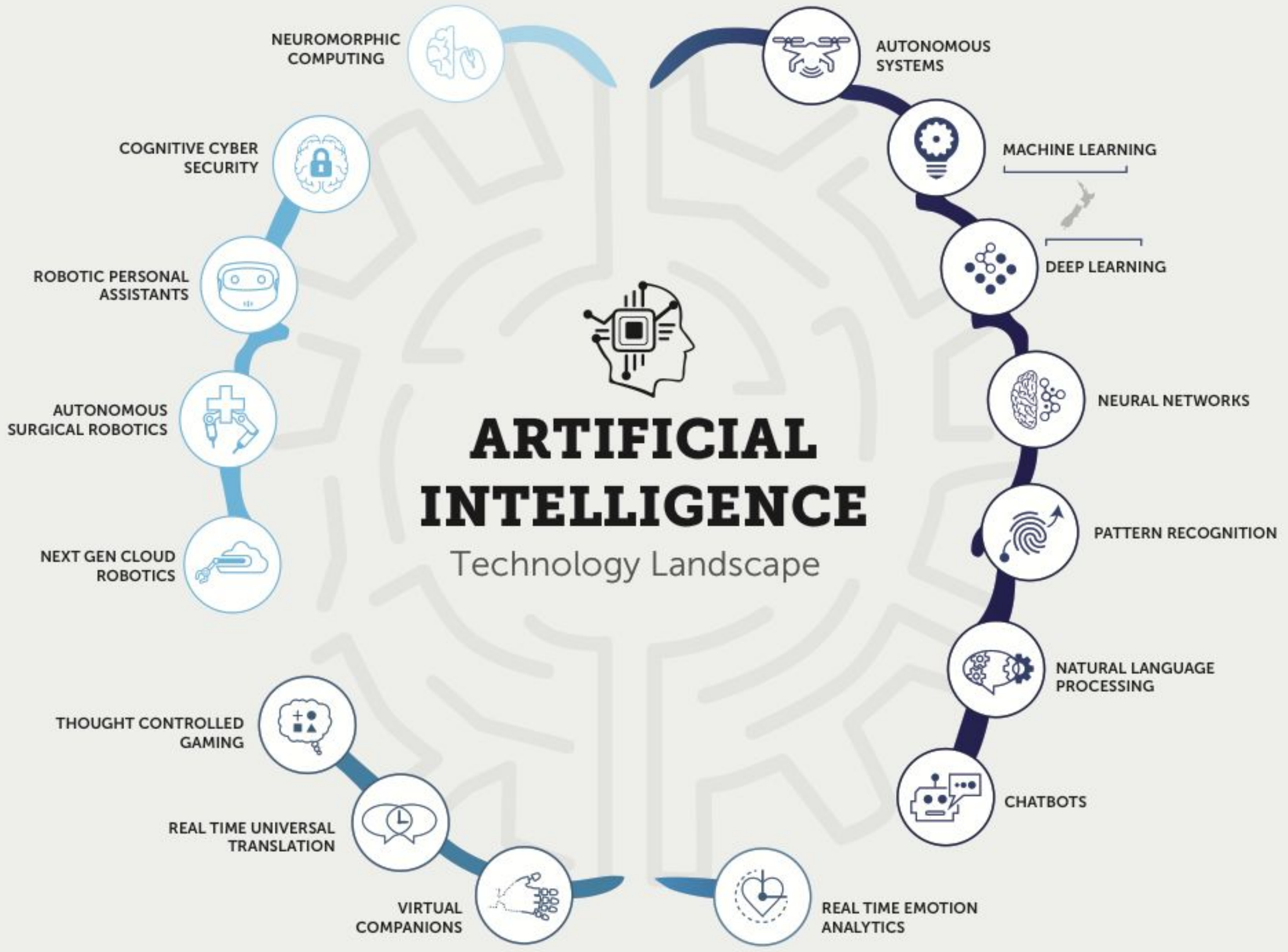
How do we humans learn?

How did you learn to recognize blue color?

How did you recognize cars/dogs/cats?

How does a machine learn to recognize objects?


Step 16



ARTIFICIAL INTELLIGENCE

Technology Landscape

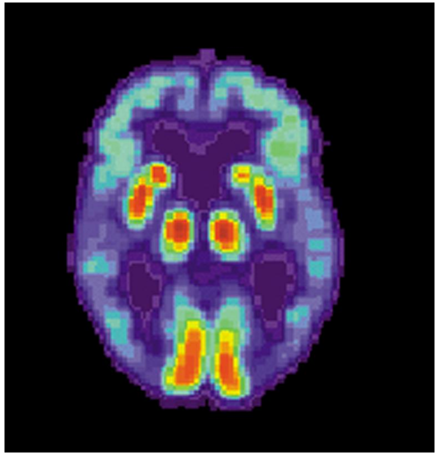
Technology Readiness

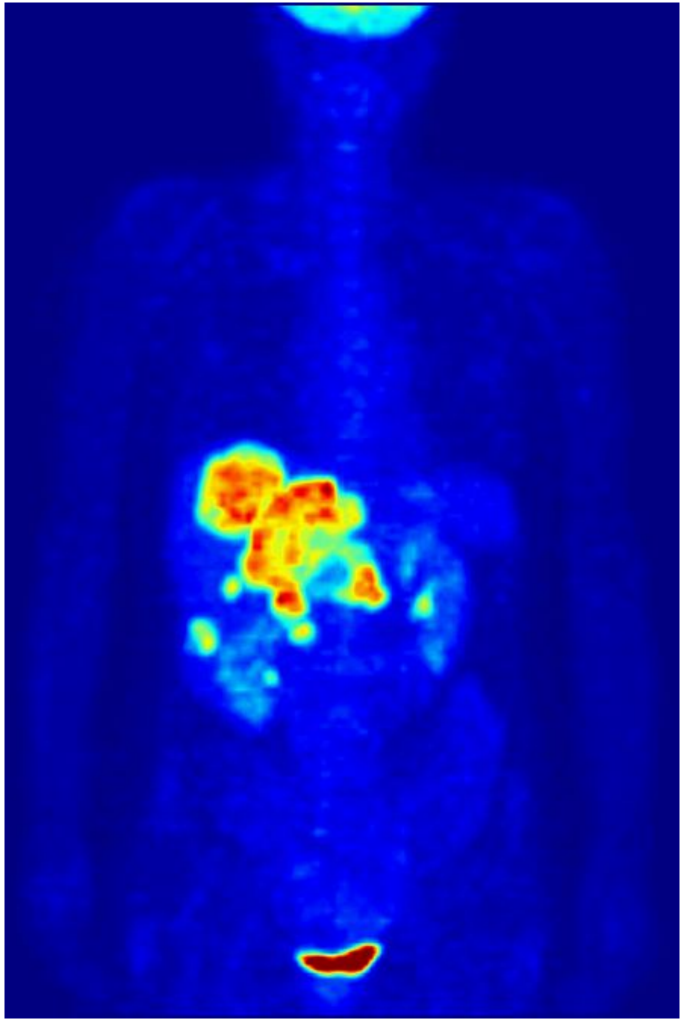
- ● NOW
- ● 1- 2 YEARS
- ● 2- 4 YEARS
- ● >4 YEARS
- Important technology for NZ business to be exploring

Sources:
Frost & Sullivan "Artificial Intelligence- R&D and Applications Road Map" (Dec 2016), Harvard Business Review- The competitive landscape for Machine Intelligence (Nov 2016), Shivon Zilis and James Chan "The State of Machine

Machine defeating human in Alphago

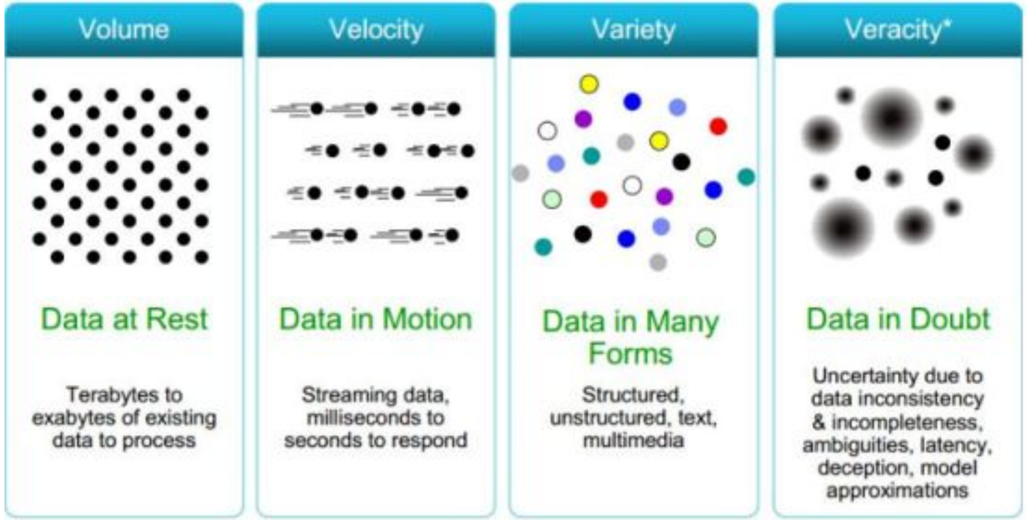
Self driving cars



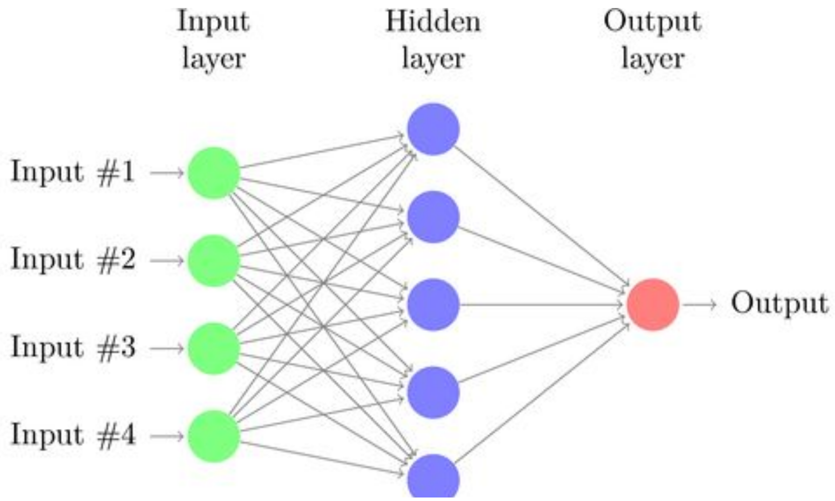


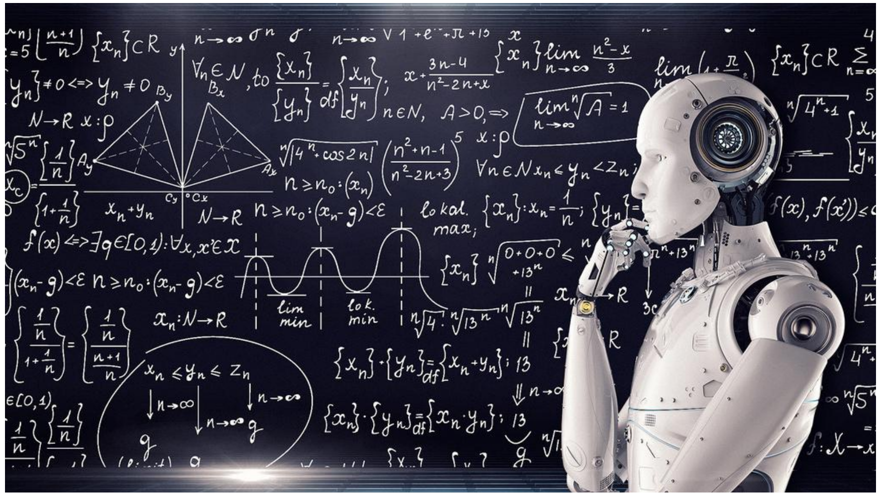
Ai in Healthcare

Big Data



Volume	Velocity	Variety	Veracity*
Data at Rest	Data in Motion	Data in Many Forms	Data in Doubt
Terabytes to exabytes of existing data to process	Streaming data, milliseconds to seconds to respond	Structured, unstructured, text, multimedia	Uncertainty due to data inconsistency & incompleteness, ambiguities, latency, deception, model approximations





Artificial Intelligence

If-else, search theory, reasoning, fuzzy logic,

Preface

Agenda

- Definition of machine learning
- Prospects in learning
- Artificial neural networks
- Activation functions
- Multilayer neural networks
- Backpropagation
- Underfitting vs Overfitting
- Evaluation metrics

Major approaches in AI

Logic & rule based:

- Representing processes or systems using logical rules.
- Top down rules are created for the computer.
- Computers reason about those rules.
- Can be used to automate processes.
- Example: expert systems.

Pattern recognition (Machine learning):

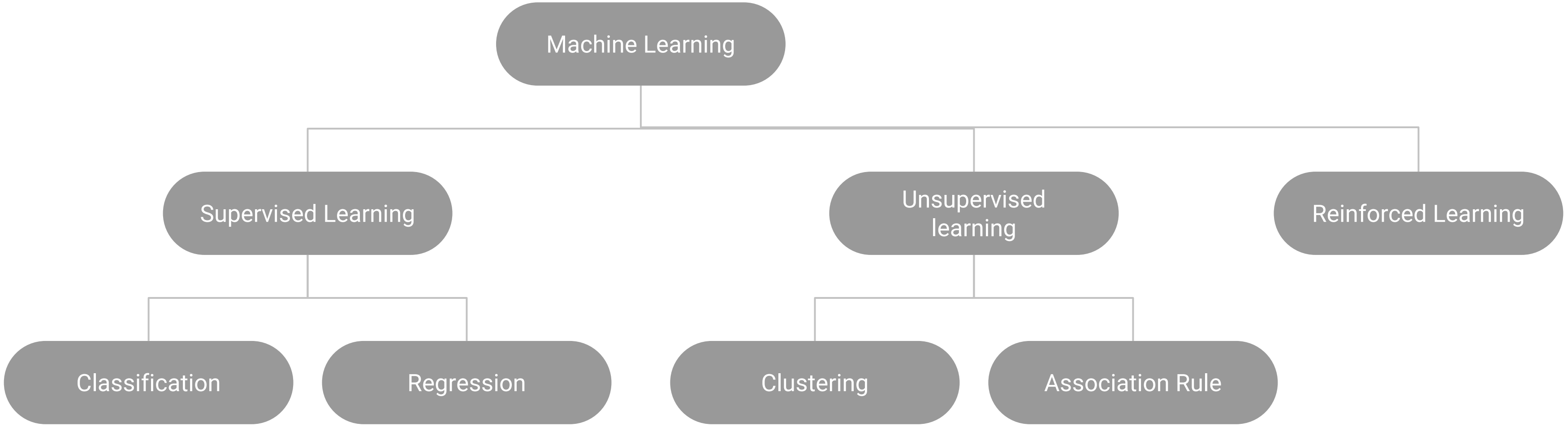
We gonna talk about this for the rest of the day

Machine Learning Definition

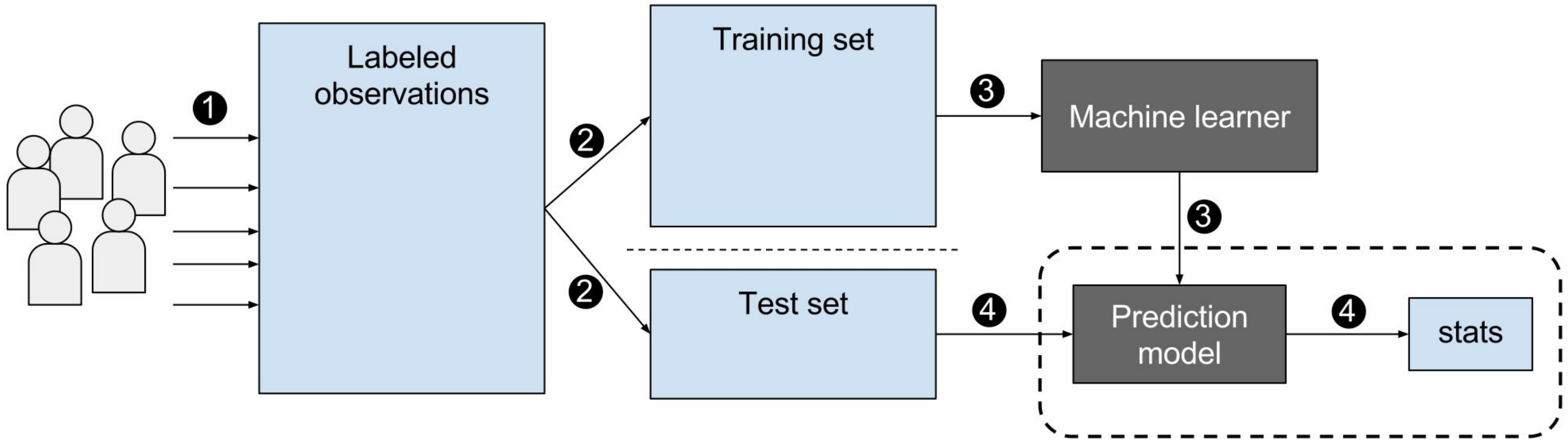
Arthur 1959: The subfield of computer science that gives computers the ability to learn without being explicitly programmed.

Mitchel 1997: A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E.

Types of Learning

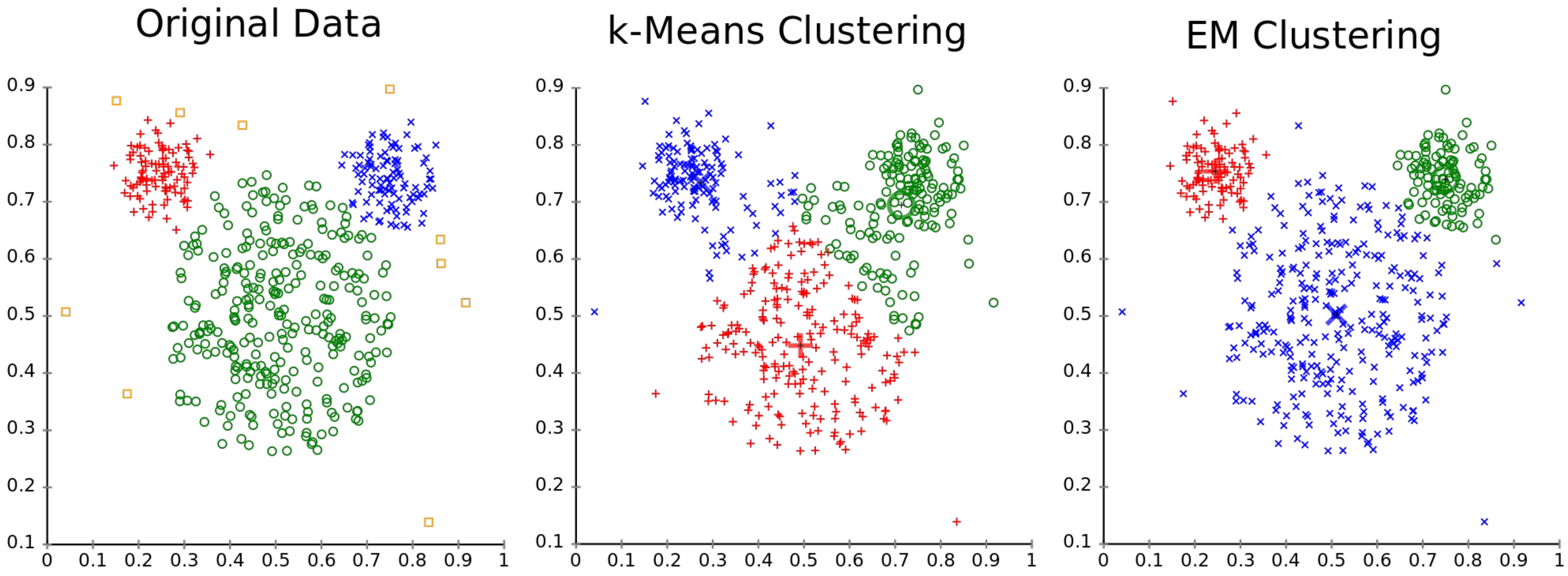


Supervised Learning

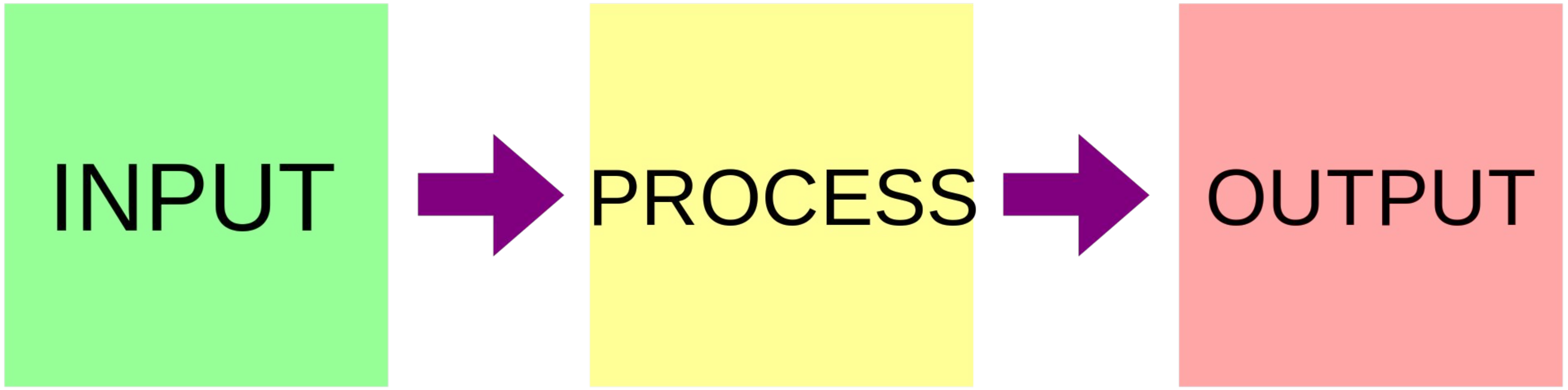


Unsupervised Learning

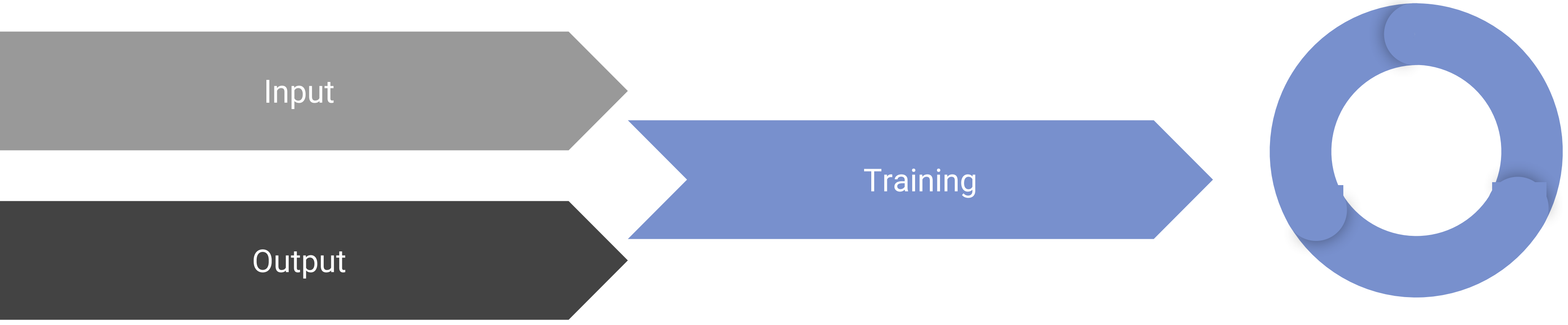
Different cluster analysis results on "mouse" data set:



The difference between ML and Sequential logic



Machine learning Paradigm



Input: the training data, features the represent an entity in our real world.

Output: the target that we want predict/detect. Basically we want to train the machine to figure out this part on it own.

Based on any model the training is the process of autonomously recognize the patterns and the relationship that connects the input to the output. Mathematically, figuring out the coefficients of an equation.

This model is a trained algorithm to perform certain type of tasks in its own without being explicitly programmed.

New input

New data coming from our business (user profiles)

The trained model will predict the type of the input value.

Output

Classification of the new user profiles.

Supervised Learning

Classification

Predicting Categories

Types of an entity

- Bad - Good - Medium
- Sick - not sick
- Hot dog- not hot dog
- Sad - happy - surprised - angry

Regression

Predicting values

Continuous values

- Sales
- Coordinates
- Time
- Age
- Power
- pressure

Artificial Neural Networks

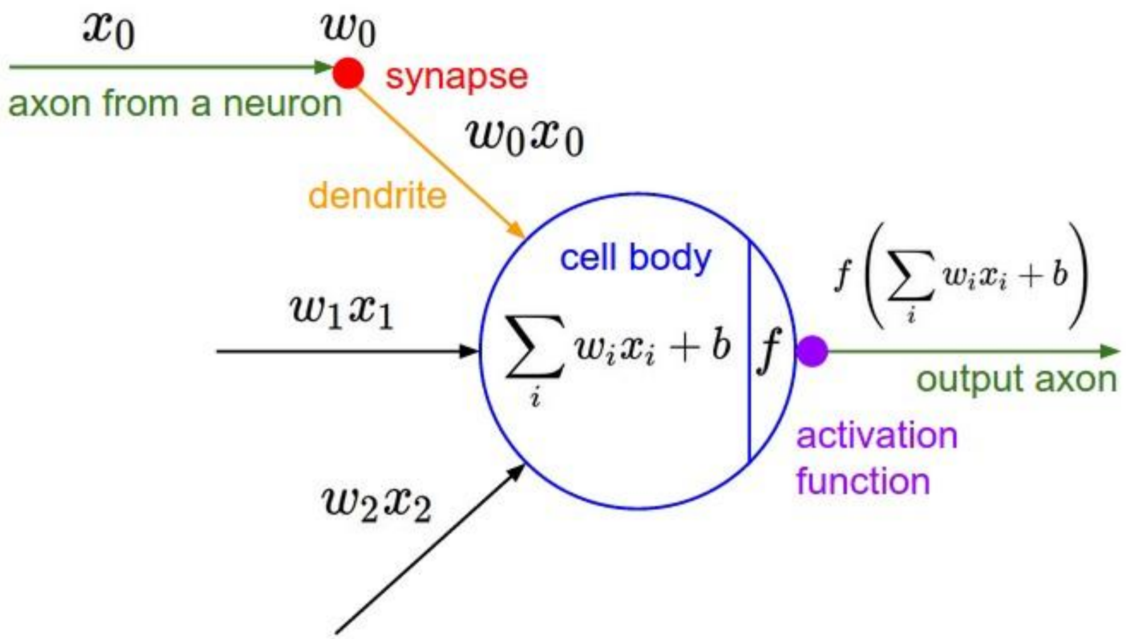
In 1943, neurophysiologist Warren McCulloch and mathematician Walter Pitts wrote a paper on how neurons might work. In order to describe how neurons in the brain might work, they modeled a simple neural network using electrical circuits.

In 1949, Donald Hebb wrote *The Organization of Behavior*, a work which pointed out the fact that neural pathways are strengthened each time they are used, a concept fundamentally essential to the ways in which humans learn. If two nerves fire at the same time, he argued, the connection between them is enhanced.

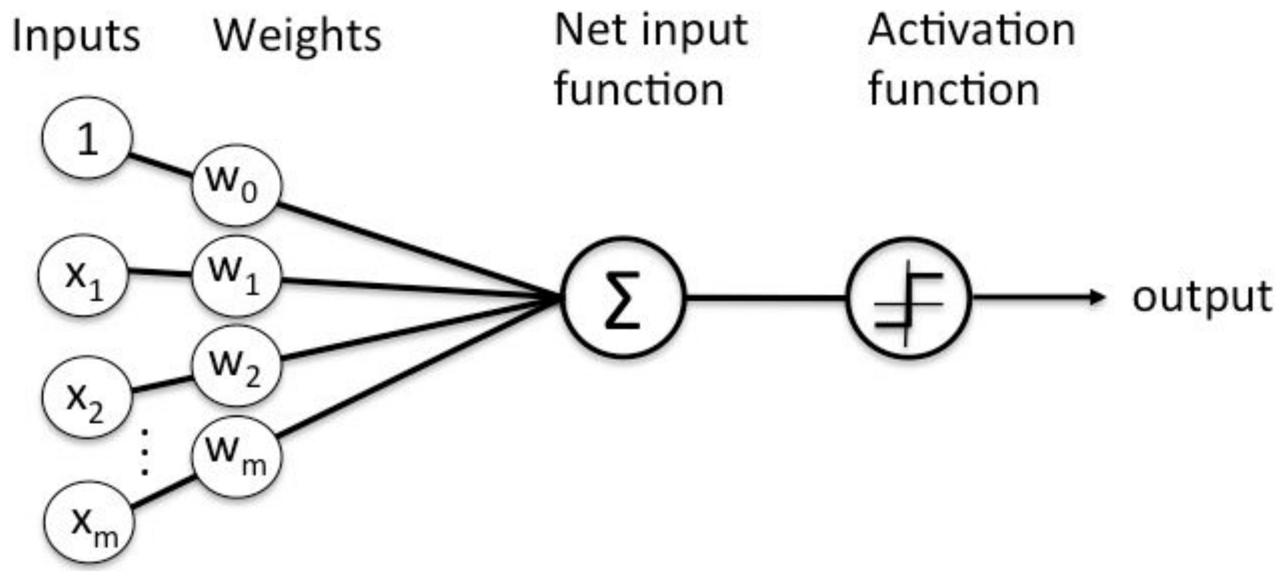
The first multi layered network was developed in 1975, an unsupervised network.

Stanford Courses

Single Perceptron



Single Perceptron



The Winter of Ai

The neural networks weren’t great and sooner they faced many challenges the most prominent was that they weren’t able to learn the XOR function. This have set to freeze for funding and publication for Ai projects after a wave of hype. This is called the winter of Ai.

So, things were not good for neural nets. But why? The idea, after all, was to combine a bunch of simple mathematical neurons to do complicated things, not to use a single one. In other terms, instead of just having one output layer, to send an input to arbitrarily many neurons which are called a hidden layer because their output acts as input to another hidden layer or the output layer of neurons. Only the output layer’s output is ‘seen’ - it is the answer of the neural net - but all the intermediate computations done by the hidden layer(s) can tackle vastly more complicated problems than just a single layer.

Let’s Train a perceptron:

$Y = \sum \omega . \chi$

W0 = 0.3

W1 = 0.5

W2 = -0.4

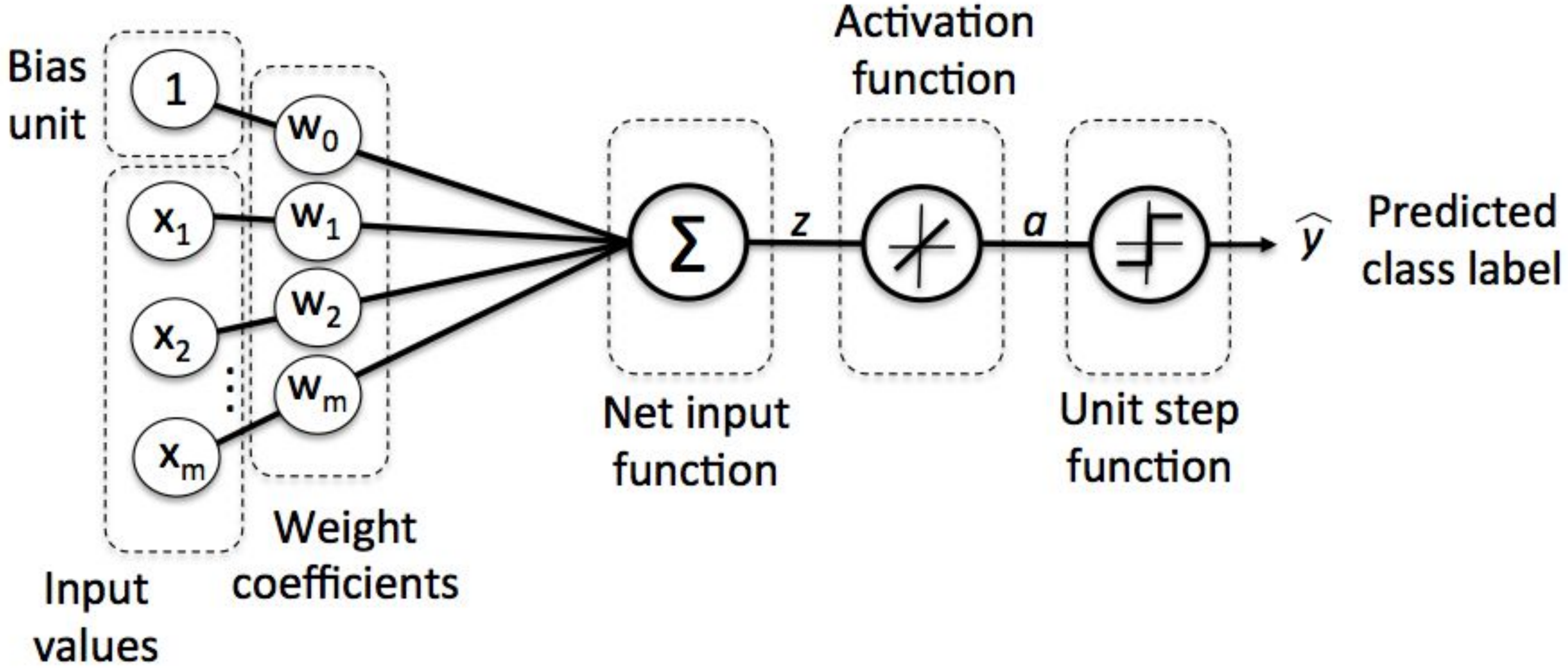
Hint: (-1*0.3)+(0*0.5)+(0*-0.4)

in1	in2	in3	Output
-1	0	0	0
-1	0	1	0
-1	1	0	1
-1	1	1	0

Perceptron example

Evolution of Machine Learning and ANN

Neural networks	1960
Multilayer Perceptrons	1985
Restricted Boltzmann Machine	1986
Support Vector Machine	1995
Hinton presents the **Deep Belief Network** (**DBN**) New interests in deep learning and RBM State of the art MNIST	2005
Deep Recurrent Neural Network	2009
Convolutional DBN	2010
Max-Pooling CDBN	2011

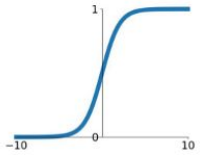

Bias unit
1
w_0
x_1
w_1
x_2
w_2
x_m
w_m
Input values
Weight coefficients
Net input function
Σ
z
Activation function
a
Unit step function
$\hat{y}$
Predicted class label

Activation Functions

Activation Functions

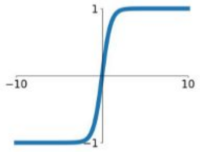
Sigmoid

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



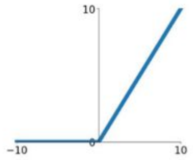
tanh

$\tanh(x)$



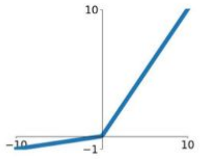
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

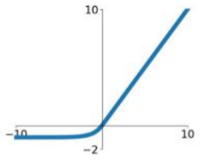


Maxout

$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



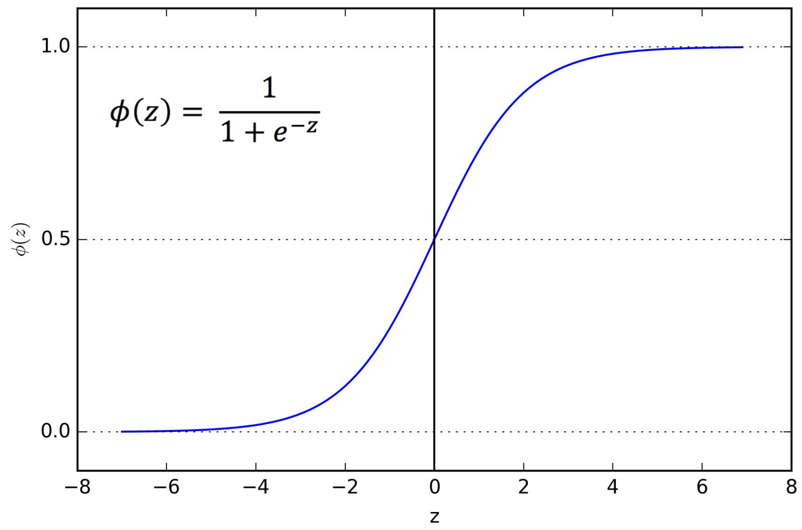
Sigmoid

Squashes numbers between [0,1]

Its popularity comes from it’s nice interpretation.

Exp() is computationally expensive.

Outputs are not zero centered.

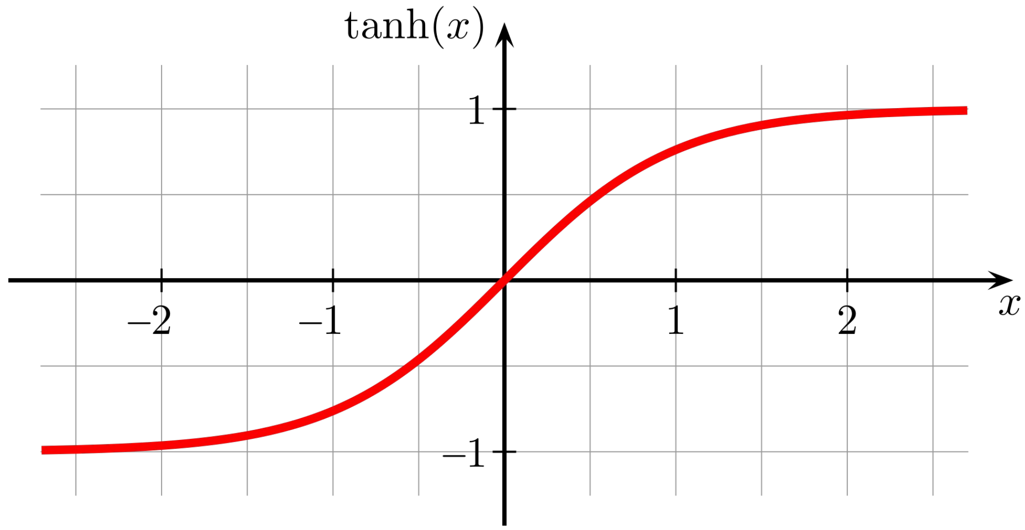


Tanh(x)

Squashes numbers between [-1,1]

Zero centered.

Kills gradient when saturated.



ReLU - Rectified Linear Unit

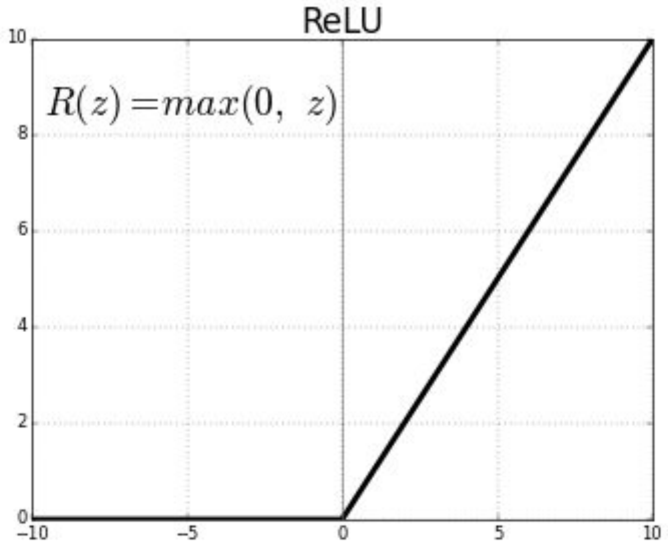
Doesn't saturate.

Computationally efficient.

Converges 6x times faster than tanh and sigmoid.

Not Zero Centered.

Can die.



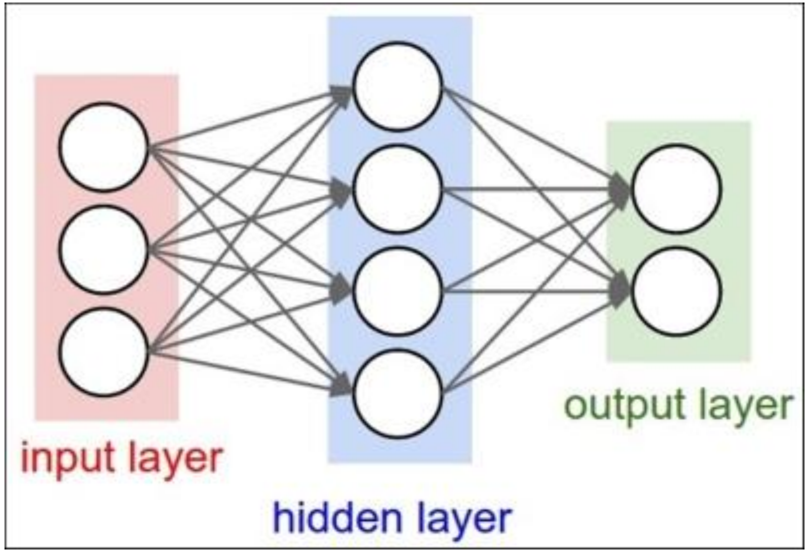
General lines

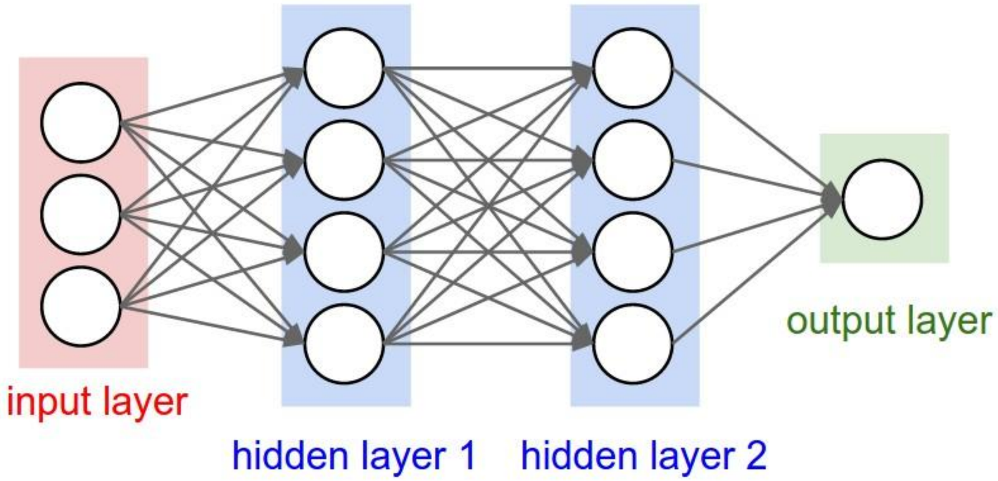
- Use ReLU with attention to the learning rate.
- Try the other variations of ReLU.
- People hate Sigmoid.
- Low expectations on tanh.

Multilayer Neural Networks

The reason hidden layers are good, in basic terms, is that the hidden layers can find features within the data and allow following layers to operate on those features rather than the noisy and large raw data. For example, in the very common neural net task of finding human faces in an image, the first hidden layer could take in the raw pixel values and find lines, circles, ovals, and so on within the image. The next layer would receive the position of these lines, circles, ovals, and so on within the image and use those to find the location of human faces - much easier! And people, basically, understood this. In fact, until recently machine learning techniques were commonly not applied directly to raw data inputs such as images or audio. Instead, machine learning was done on data after it had passed through feature extraction - that is, to make learning easier machine learning was done on preprocessed data from which more useful features such as angles or shapes had been already extracted.

Multilayer Feedforward Neural Networks





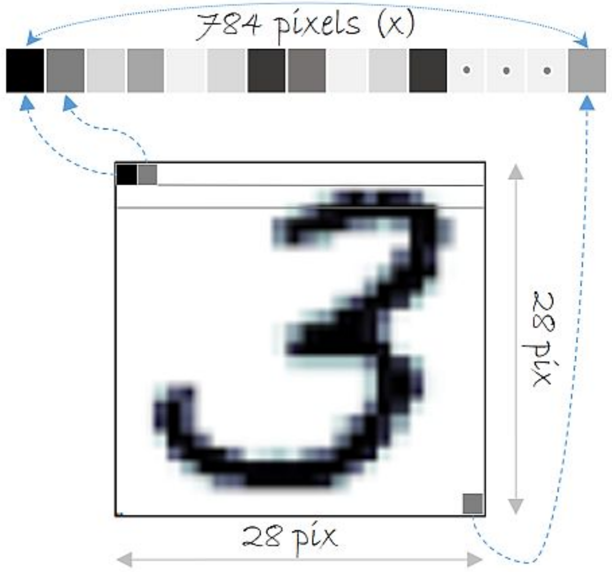
The size of a neural network

Total number of neurons = neurons in the hidden layer **+** neurons in the output layer

Total weights = [neurons in the input layer * neurons in the hidden layer] **+** [neurons in the hidden layer * neurons in the output layer]

Feedforward Neural network

The design of the input and output layers in a network is often straightforward. For example, suppose we're trying to determine whether a handwritten image depicts a "3" or not. A natural way to design the network is to encode the intensities of the image pixels into the input neurons. If the image is a 28 by 28 greyscale image, then we'd have 784=28×28 input neurons, with the intensities scaled appropriately between 0 and 1. The output layer will contain just a single neuron, with output values of less than 0.5 indicating "input image is not a 3", and values greater than 0.5 indicating "input image is a 3 ".



Learning Neuron Error

It is clear to you by now that neural networks are functions trying to proximate the weights. The delta rule is the most common way to adjust the weights of the neuron.

If t is the desired output and y is the predicted one, the error δ can be calculated by:

$$\delta = t - y$$

The formula to measure the change in weights Δw that results in error is:

$$\Delta w = \eta \delta x$$

The new weights formula is:

$$w \text{ (new weight)} = w \text{(old weight)} - \Delta w$$

Backpropagation

The back propagation algorithm is a generalization of the delta rule for training multilayer networks (MLN). This algorithm updates the weights wi of the network by means of successive iterations, that minimize the cost function of the error E. The minimization of the error is obtained using the gradient of the cost function, which consists of the first derivative of the function with respect to all the weights wi namely:

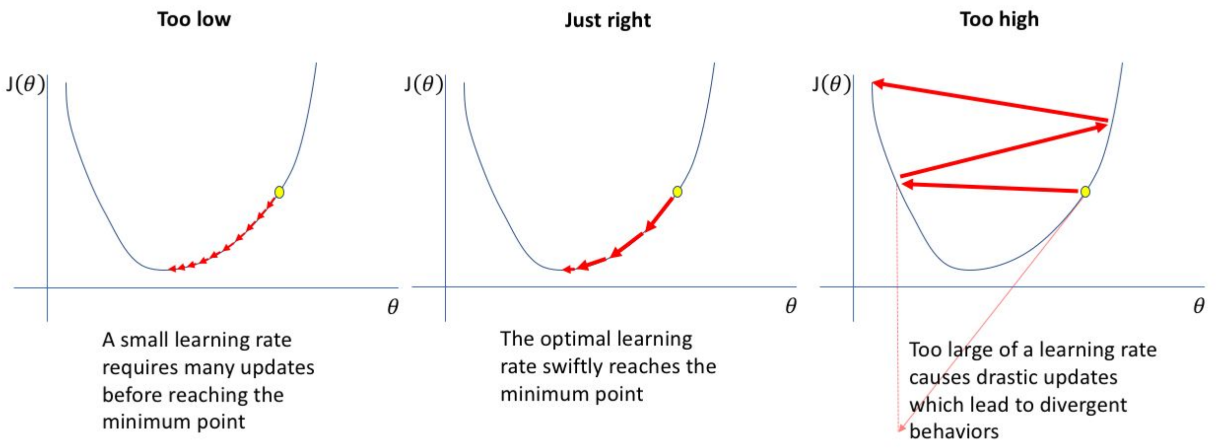
$$\frac{\partial}{\partial w_i}(E)$$

On the basis of this gradient the weights will be updated with the following mechanism:

$$w_i = w^0{}_i - \eta \frac{\partial}{\partial w_i}(E)$$

Learning Rate

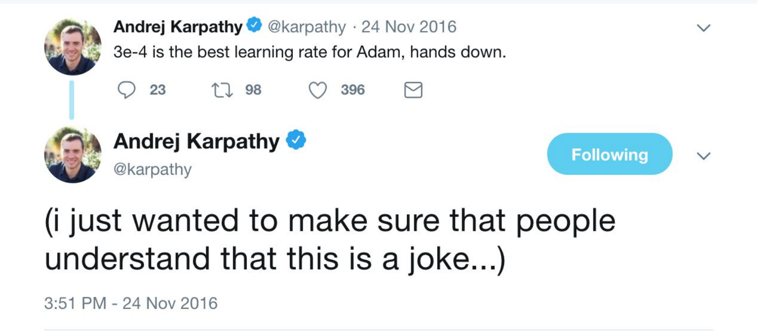
Learning rate is a hyper-parameter that controls how much we are adjusting the weights of our network with respect to the loss gradient. A low learning rate means slow movement in adjusting the weights.


Too low
Just right
Too high
J(θ)
θ
A small learning rate requires many updates before reaching the minimum point
The optimal learning rate swiftly reaches the minimum point
Too large of a learning rate causes drastic updates which lead to divergent behaviors

Jeremy Jordan Blog - “nn learning rate”

Determine Learning Rate value

Ultimately, we'd like a learning rate which results is a *steep decrease* in the network's loss. We can observe this by performing a simple experiment where we gradually increase the learning rate after each mini batch, recording the loss at each increment. This gradual increase can be on either a linear or exponential scale.

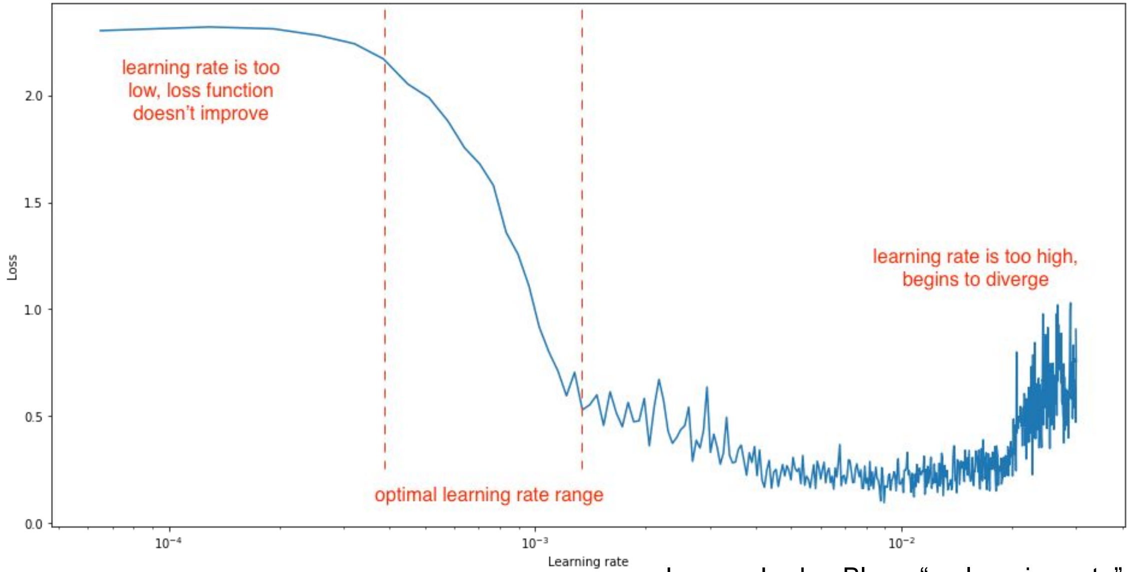


Andrej Karpathy @karpathy · 24 Nov 2016
3e-4 is the best learning rate for Adam, hands down.
23 98 396

Andrej Karpathy
@karpathy
Following

(i just wanted to make sure that people understand that this is a joke...)

3:51 PM - 24 Nov 2016

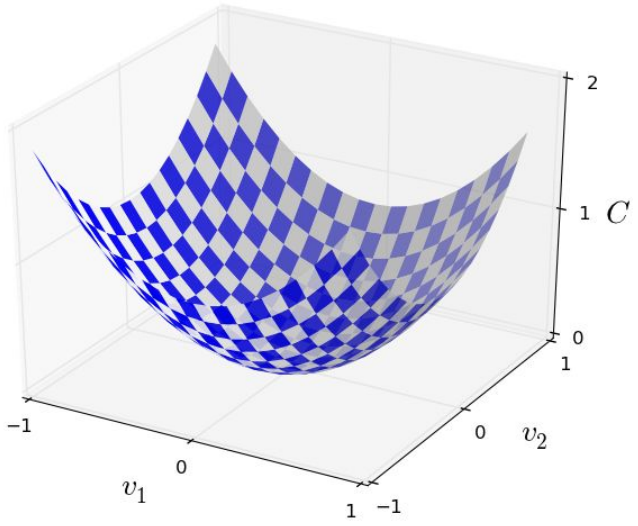


Jeremy Jordan Blog - “nn learning rate”

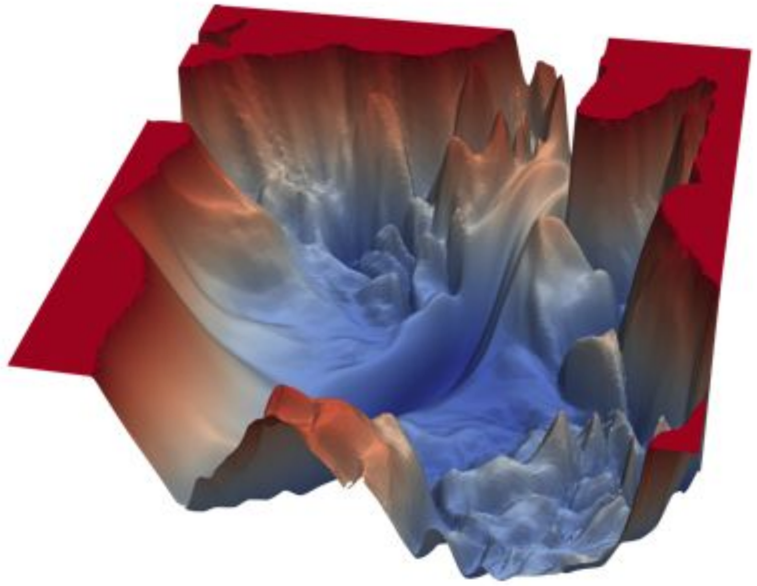
Gradient Descent

In any context be neural networks or otherwise, if we are trying to minimize a function say C(v1,v2). Gradient descent is a minimization technique that we use to find the global minima. The graph on the side is an easy example where you can identify the global minima with your eyes. In most cases our function C will have several variables and it won't be possible to find global minimum with our eyes.

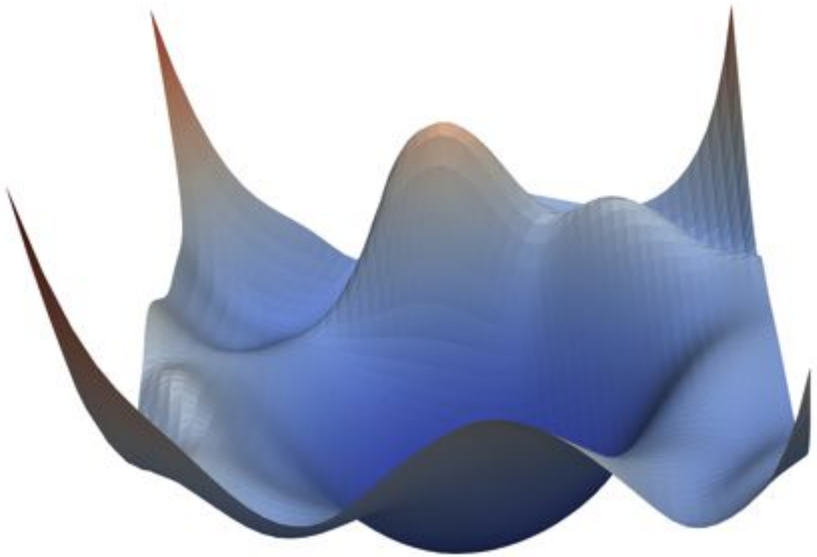
To be honest this is an oversimplification of the problem but it is necessary to imagine C as function of two variables and use calculus to find the minimization solution analytically. In real life it's more complicated.



No residual connections



With residual connections



Same general network architecture

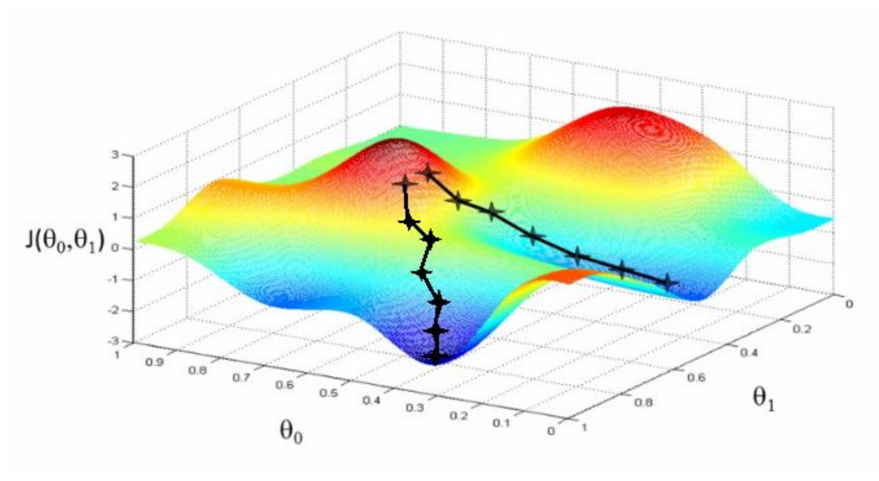
Jeremy Jordan Blog - “nn learning rate”

Solving the minimization problem

If calculus doesn't work then how to solve the problem. Let's make an analogy, imagine the function space as it looks in the image. There is a valley somewhere on this space. Imagine the solution as a ball if we have just started randomly on any point and let the ball rolling, we know for fact that the ball will rest in a low point at the valley.

We can do this simulation by calculating the derivative and second derivative of the function C.

This analogy is to make us imagine the simulation but not to put a constraint on our thinking and hustle with physics law.



Neural Networks and Deep Learning Book Chapter 1

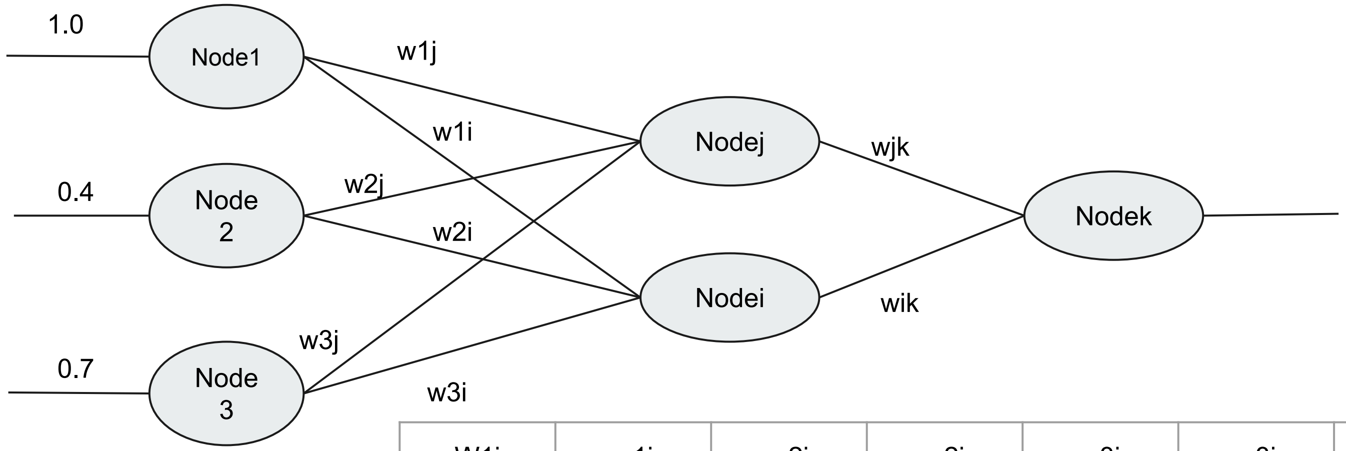
What happens when we move the ball a small amount ***Δ**v1* in the *v1* direction and small amount of ***Δ**v2* in the *v2* direction. Then our C functions will change accordingly by:

$$\Delta C \approx \frac{\partial C}{\partial v_1}\Delta v_1 + \frac{\partial C}{\partial v_2}\Delta v_2 .$$

You should ask then why do we have a learning rate and a gradient at the same time. The gradient is showing the ball which direction to roll into, whereas learning rate is the step size.

Calculate Backpropagation

Assume we have an input of [1.0, 0.4, 0.7] with an output value of [0.65] in the dataset. Feed this into a network of two hidden layers and compute the network output, (output = 0.582) then compute the error using the delta role and squared error function. $E_{(total)} = \Sigma \frac{1}{2} (\text{target} - \text{output})^2$



W1j	w1i	w2j	w2i	w3j	w3i	wjk	wik
0.20	0.10	0.30	-0.10	-0.10	0.20	0.10	0.50

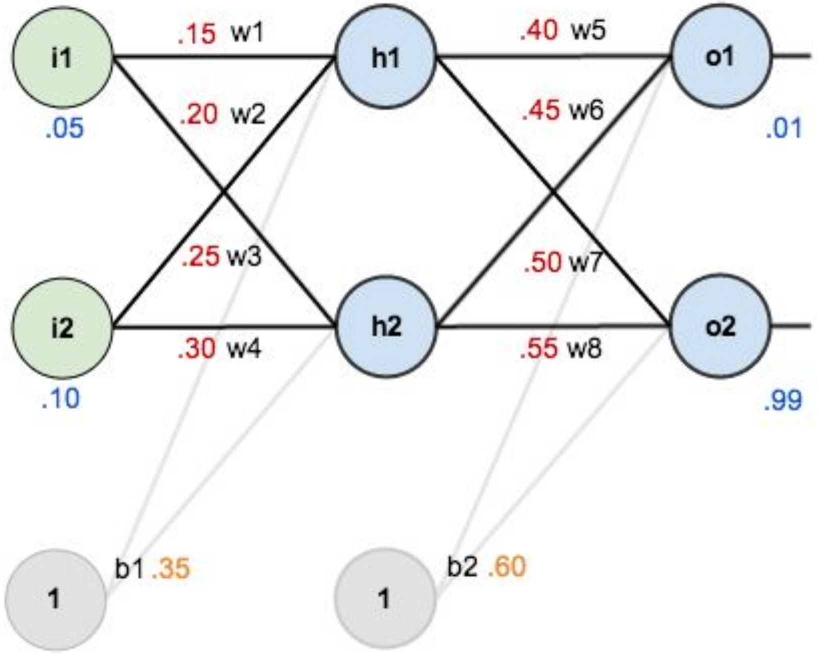
Backpropagation

Net_{h1} = w1*i1 + w2+i2 + b1*1

Calculate Net_{h1}, out_{h1}, Net_{h2}, and out_{h2}

Calculate Net_{o1}, out_{o1}, Net_{o2}, and out_{o2}

Calculate the error at o1, o2, and total error using squared error function.



Complete solution from here: steps

Gradient Descent

Cool and solves many issues that were face in the past but it came with it’s own problems:

- May stop at local minima.
- Local minima can be worst that global minima.
- There exist many local minimas.

Possible solution:

Training with different initial weights, train different networks architectures.

Regularization in neural networks

Machine learning models are vulnerable to overfitting and underfitting syndrome. The model overfit when it try too hard to learn and it captures noise. This noise is data points and details that are irrelevant or doesn’t contribute to the knowledge of the model. Balancing variance and bias is one of the famous approaches to solving overfitting.

The regularizer is a regression technique that shrinks the coefficient estimates towards zero. In simple words it discourage the model from learning too much to avoid the risk of overfitting.

$$Y \approx \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p$$

β is the coefficient estimate for different variables (X) and Y is the learned relationship

Residual Sum of Squares RSS

The fitting procedure involves a loss function, known as residual sum of squares or RSS. The coefficients are chosen, such that they minimize this loss function.

$$\mathrm{RSS} = \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 .$$

Now, this will adjust the coefficients based on your training data. *If there is noise in the training data, then the estimated coefficients won't generalize well to the future data. This is where regularization comes in and shrinks or regularizes these learned estimates towards zero.*

What does Regularization achieve?

A standard model tend to have some variance and therefore it doesn’t generalize in a data that is different than training dataset, it overfits. **Regularization significantly reduces the variance of the model without substantial increase its bias.**

By reducing the variance and avoiding increase in the bias the regularizer prevents the model from capturing the noise and maintains it’s useful knowledge. But the value of the regularizer have to be carefully selected as it can harm the model by discouraging it’s learning if it’s set to be high.

In most machine learning implementations and particularly in scikit-learn there exist a parameter alpha for the regularization term which **helps in avoiding overfitting by penalizing weights with large magnitudes.**

Model Evaluation & Performance Analysis

How to evaluate our model?

What measure to use?

What is the objective state?

How to improve the performance of the model?

The desirable state to achieve: Good Generalization.

We consider the model to perform well if and only if it can generalize well.

Two states the model could fall into:

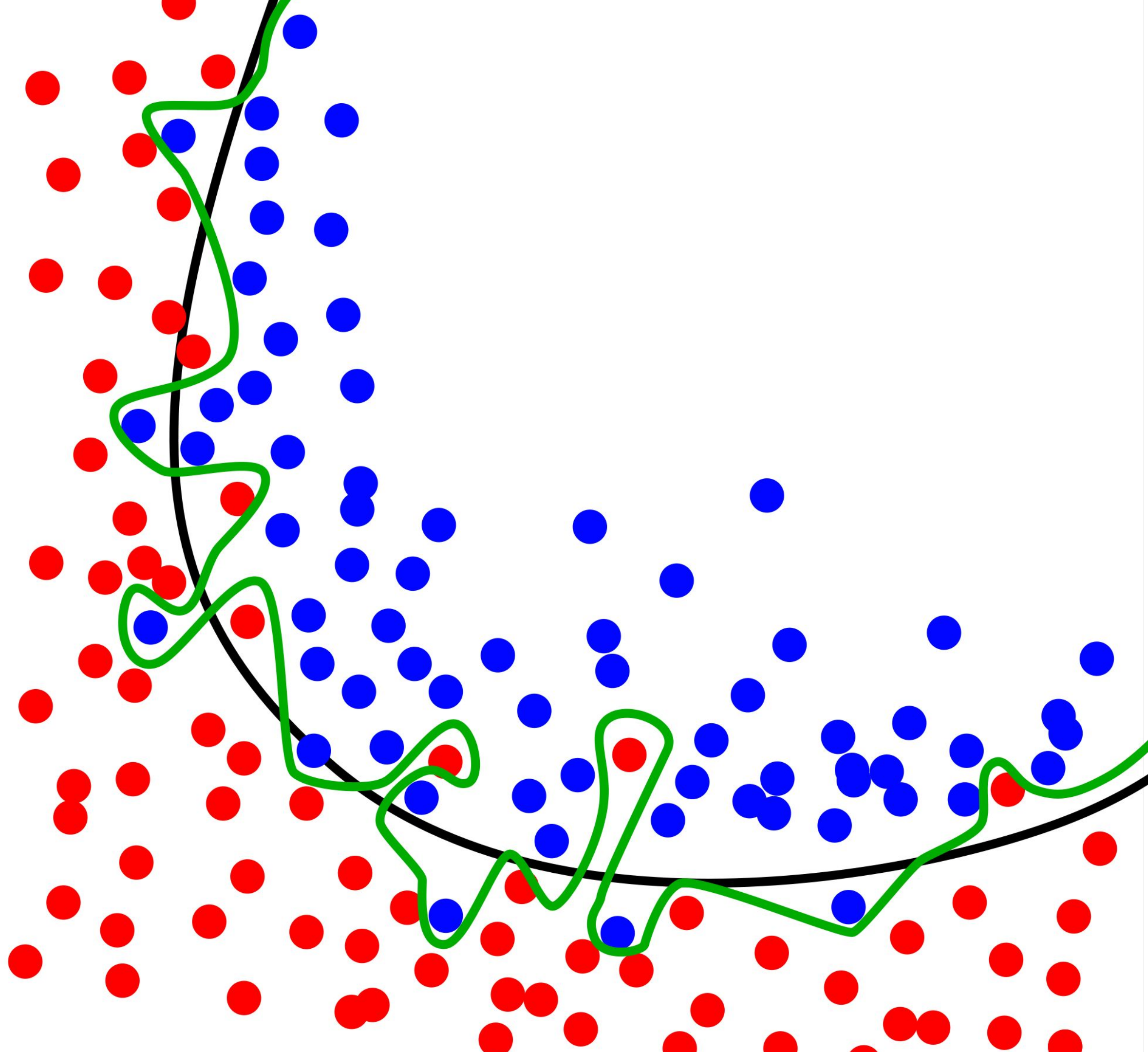
1. Overfitting.
2. Underfitting.

Both are considered ae error that result in an unsatisfactory performance and predictions.

Overfitting

&

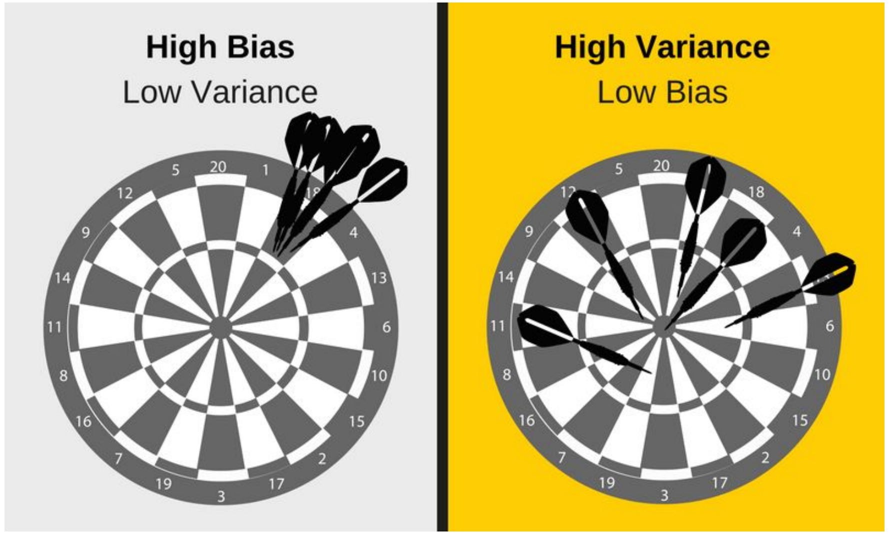
Underfitting



Both errors are the result of the tradeoffs between variance and bias.

Overfitting is when the model learns all the details and noise about your data to the extent that it performs 100% accurate in your training data, but when you test on a new data the model performs poorly.

Underfitting is when the model is not able to perform well in the dataset. The poor performance in both training and testing sets is an indicator of the underfitting error.



How to prevent this?

1. Cross validation.
2. Using Ensemble or bagging methods.
3. Add more data.
4. Reduce features (via feature selection or PCA).
5. Start with simple method.
6. Early stopping.
7. Regularization.

Evaluation of classification models.

Accuracy

Precision

Sensitivity/Recall

F1-Score

AUC (Area Under the ROC Curve)

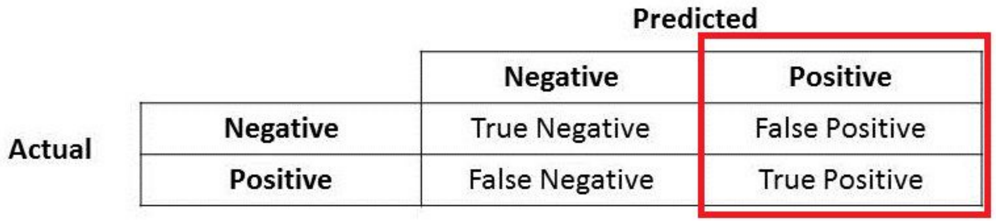
Precision

Spam Detection: you may lose few emails because you classify them as spam.

How precise your model is? Out of the predicted positive instances how many are actually positive.

$$\text{Precision} = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

$$= \frac{True\ Positive}{Total\ Predicted\ Positive}$$



		Predicted	
		Negative	**Positive**
Actual	**Negative**	True Negative	False Positive
	Positive	False Negative	True Positive

Recall
Sensitivity

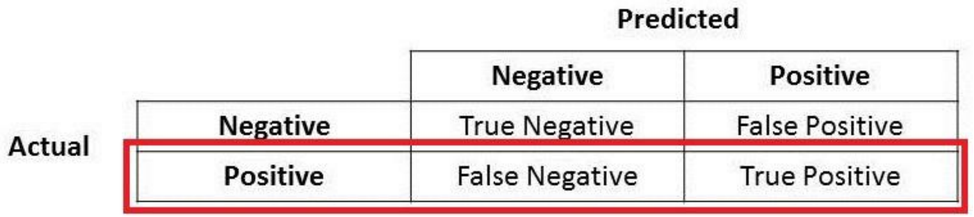
Intrusion Detection: if the model misclassify an intrusion a harmful attach will cost the organization millions and sensitive data.

How many of the actual positives are classified as so.

How sensitive is the model in detecting the right class.

$$\text{Recall} = \frac{\textit{True Positive}}{\textit{True Positive} + \textit{False Negative}}$$

$$= \frac{\textit{True Positive}}{\textit{Total Actual Positive}}$$

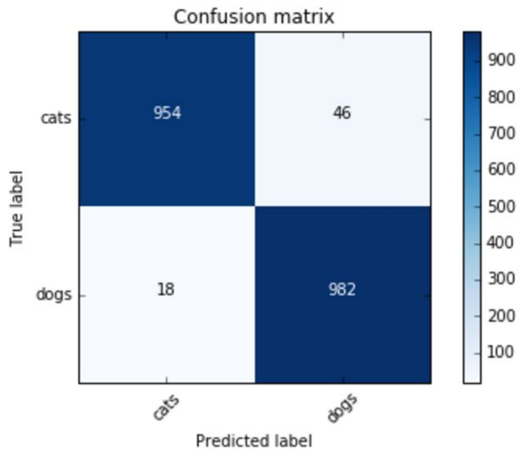


		Predicted	
		Negative	**Positive**
Actual	**Negative**	True Negative	False Positive
	Positive	False Negative	True Positive

Tools in scikit learn

Confusion Matrix

```
[[954  46]
 [ 18 982]]
```



Classification Report

```
             precision    recall  f1-score   support

        0.0       0.86      0.90      0.88       543
        1.0       0.83      0.76      0.79       342

avg / total       0.85      0.85      0.85       885
```

Evaluation of Regression models.

Root Mean Squared Error.

Relative Squared Error.

Mean Absolute Error.

Regression Evaluation Metrics

RMSE: measure the error rate of a regression model. However, it can only be compared between models whose errors are measured in the same units.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(p_i - a_i)^2}{n}} \qquad RSE = \frac{\sum_{i=1}^{n}(p_i - a_i)^2}{\sum_{i=1}^{n}(\bar{a} - a_i)^2}$$

RSE: relative squared error (RSE) can be compared between models whose errors are measured in the different units.

MAE: has the same unit as the original data, and it can only be compared between models whose errors are measured in the same units.

Model Optimization Fine Tuning

Decision Trees.

Artificial Neural Networks.

Support Vector Machines.

Parameters

Entropy OR Gini Index

Number of branches allowed, levels.

Activation Functions

Number of layers

Regularization term

Error Functions

Kernel functions

Epsilon values

Grid Search Method

Read more

A search consists of:

- an estimator (regressor or classifier such as sklearn.svm.SVC());
- a parameter space;
- a method for searching or sampling candidates;
- a cross-validation scheme; and
- a score function.

```
param_grid = [
  {'C': [1, 10, 100, 1000], 'kernel': ['linear']},
  {'C': [1, 10, 100, 1000], 'gamma': [0.001, 0.0001], 'kernel':
['rbf']},]
```

Tips on Practical Use

- MLP is sensitive to feature scaling, so always make sure you scale your data.
- Apply the same scaling method for testing data as well.
- Find a reasonable regularizer using Gridsearch.
- Empirically observed that L-BFGS converges faster and with better solutions on small datasets. For relatively large datasets, however, Adam is very robust. It usually converges quickly and gives pretty good performance. SGD with momentum or nesterov's momentum, on the other hand, can perform better than those two algorithms if learning rate is correctly tuned.

Scikit-learn modules