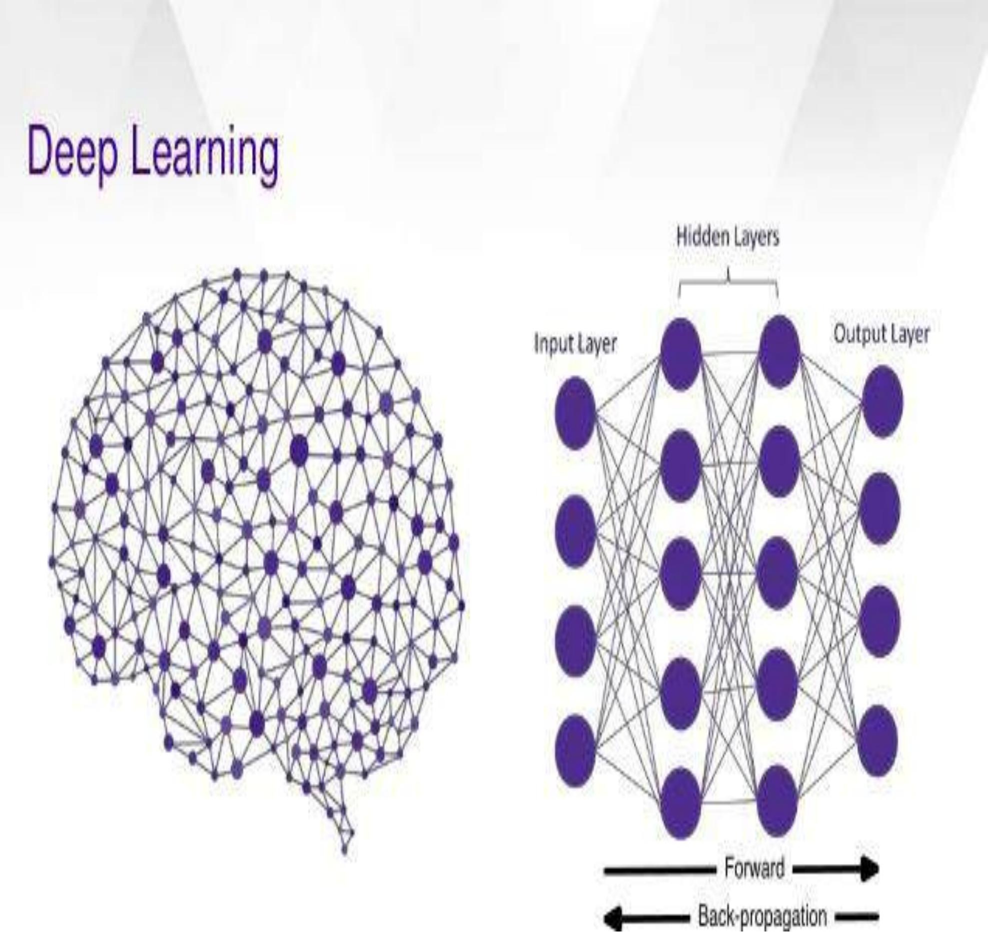


Machine Learning VS Deep Learning

Presented by : Dr. Hanaa Bayomi
h.mobarz@fci-cu.edu.eg

Agenda

1- Machine learning

- Definition and types
- machine Learning road map
- feature selection
  - filter, wrapper and embedded
- Model selection
  - cross validation(K-fold)

2- Deep learning

- Definition
- ML VS DL
- DL architecture
  - fully connected NN
  - convolution NN
  - Recurrent NN (LSTM)

3- NLP Tasks

Machine Learning definition

- One definition of machine learning: A computer program <u>improves its performance</u> on a given <u>task</u> with <u>experience (i.e. examples, data)</u>.
- Task: What is the problem that the program is solving?
- Experience: What is the data (examples) that the program is using to improve its performance?
- Performance measure: How is the performance of the program (when solving the given task) evaluated?

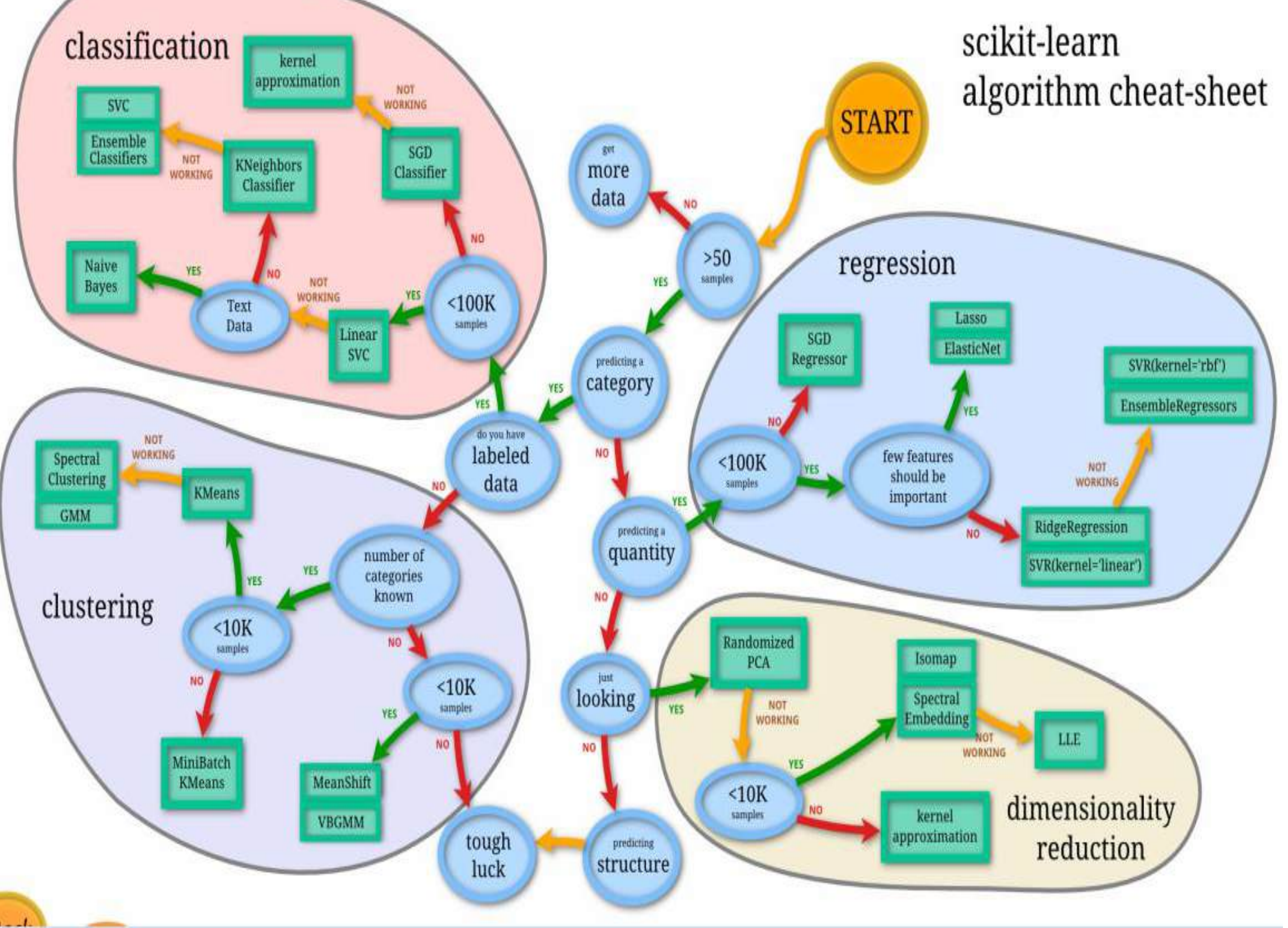
Machine Learning types

- Supervised learning
  - Learning from labelled data
  - Classification, Regression, Prediction, Function Approximation
- Unsupervised learning
  - Learning from unlabelled data
  - Clustering, Visualization, Dimensionality Reduction

Machine Learning types

- Semi-supervised learning
  - mix of Supervised and Unsupervised learning
  - usually small part of data is labelled
- Reinforcement learning
  - Model learns from a series of actions by maximizing a reward function
  - The reward function can either be maximized by penalizing bad actions and/or rewarding good actions
  - Example - training of self-driving car using feedback from the environment

http://scikit-learn.org/stable/tutorial/machine_learning_map/index.html



Learning Types

	Supervised	Unsupervised
Discrete	Classification	Clustering
Continuous	Regression	Dimensionality reduction

Feature selection

Performance of Machine Learning model depend on

- Choice of algorithm
- Feature selection
- Feature creation
- Model selection

https://archive.ics.uci.edu/ml/datasets.html
UCI Machine Learning Repository: Data Sets

Classification of FS methods

- Filter (single factor analysis)
  - Assess the relevance of features only by looking at the essential properties of the data.
  - Usually, calculate the feature relevance score and remove low-scoring features.
- Wrapper
  - Bundle the search for best model with the FS.
  - Generate and evaluate various subsets of features. The evaluation is obtained by training and testing a specific ML model.
- Embedded
  - Embedded methods learn which features best contribute to the accuracy of the model while the model is being created. The most common type of embedded feature selection methods are **regularization methods**.

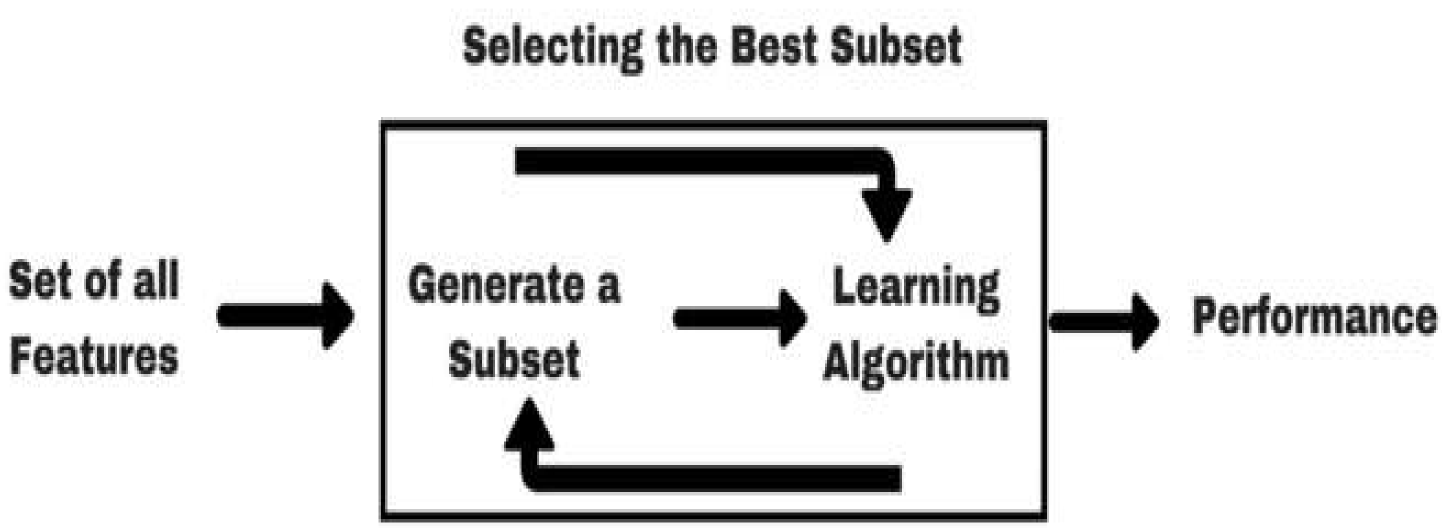
Filter methods

- Filter methods are generally used as a *preprocessing step*. The selection of features is independent of any machine learning algorithms.
- Two steps (score-and-filter approach)
  1. assess each feature individually for its potential in discriminating among classes in the data
  2. features falling beyond threshold are eliminated

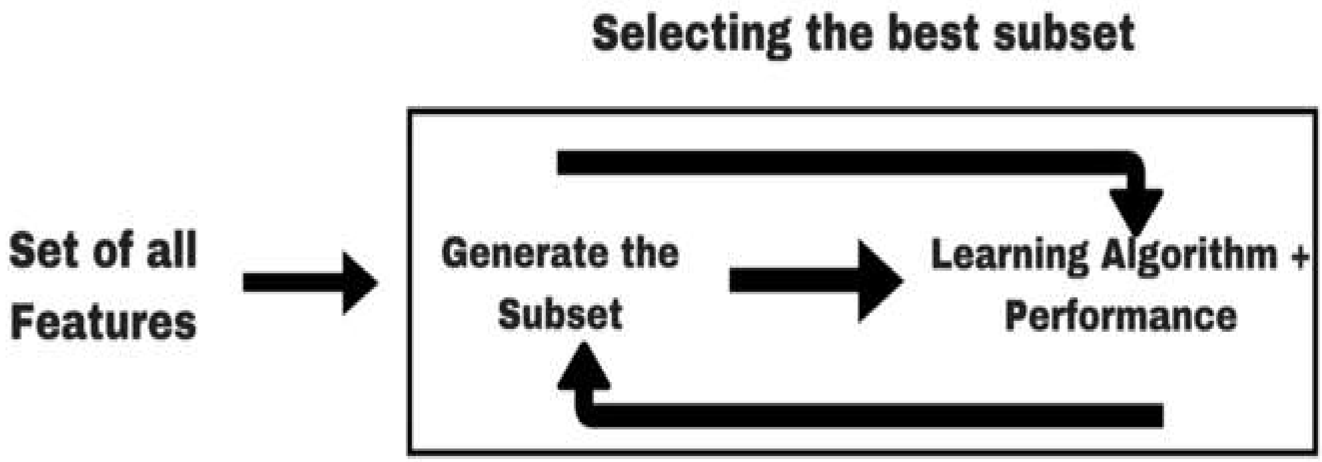
Wrappers

- Search for the best feature subset in combination with a fixed classification method.
- The goodness of a feature subset is determined

-one-out



Embedded



Some of the most popular examples of these methods are LASSO and RIDGE regression which have inbuilt penalization functions to reduce over fitting.

Lasso regression performs L1 regularization which adds penalty equivalent to absolute value of the magnitude of coefficients.

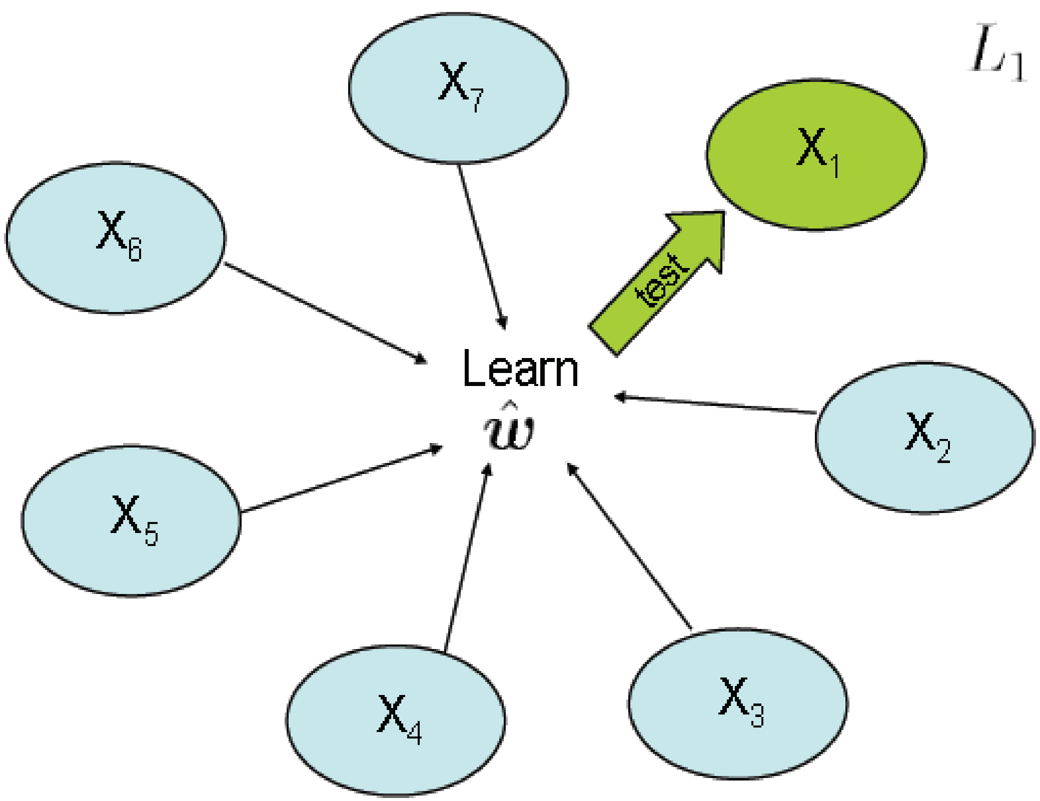
Ridge regression performs L2 regularization which adds penalty equivalent to square of the magnitude of coefficients.

Choosing the best model

K-fold cross validation

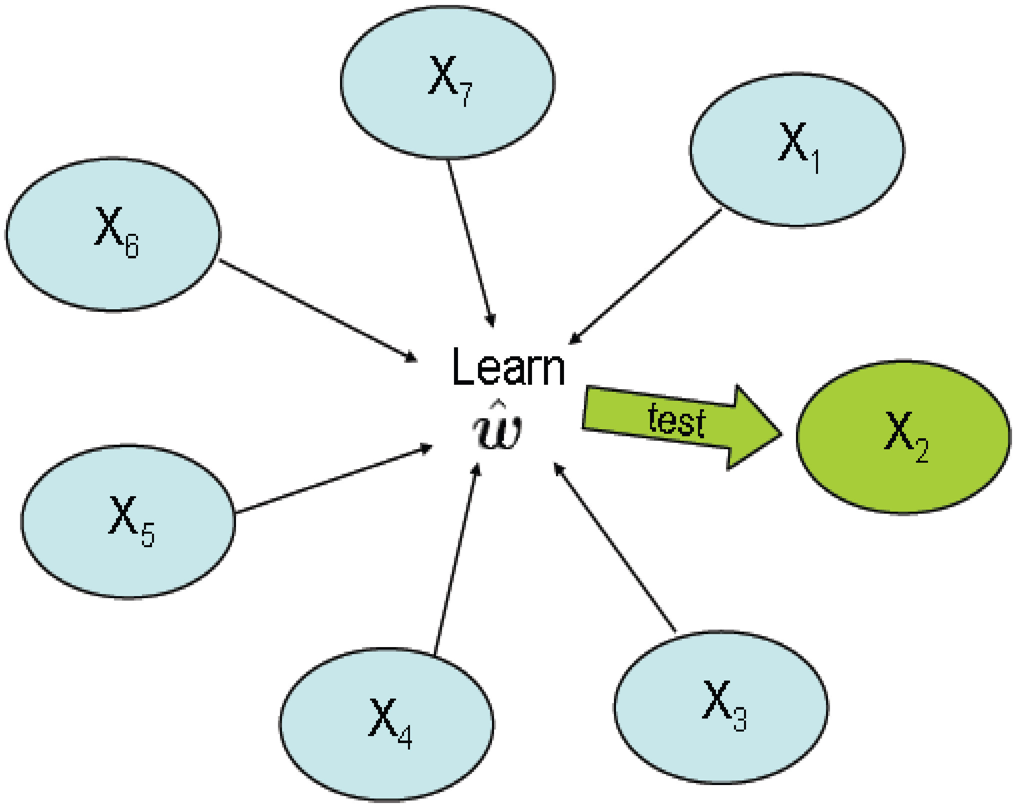
- A technique for estimating test error
- Uses *all* of the data to validate
- Divide data into K groups $\{X_1, X_2, \ldots, X_K\}$.
- Use each group as a validation set, then average all validation errors

$$L_1 = \sum_{(\boldsymbol{x},y)\in X_1} (y - \hat{\boldsymbol{w}}^\top \boldsymbol{x})$$



K-fold cross validation

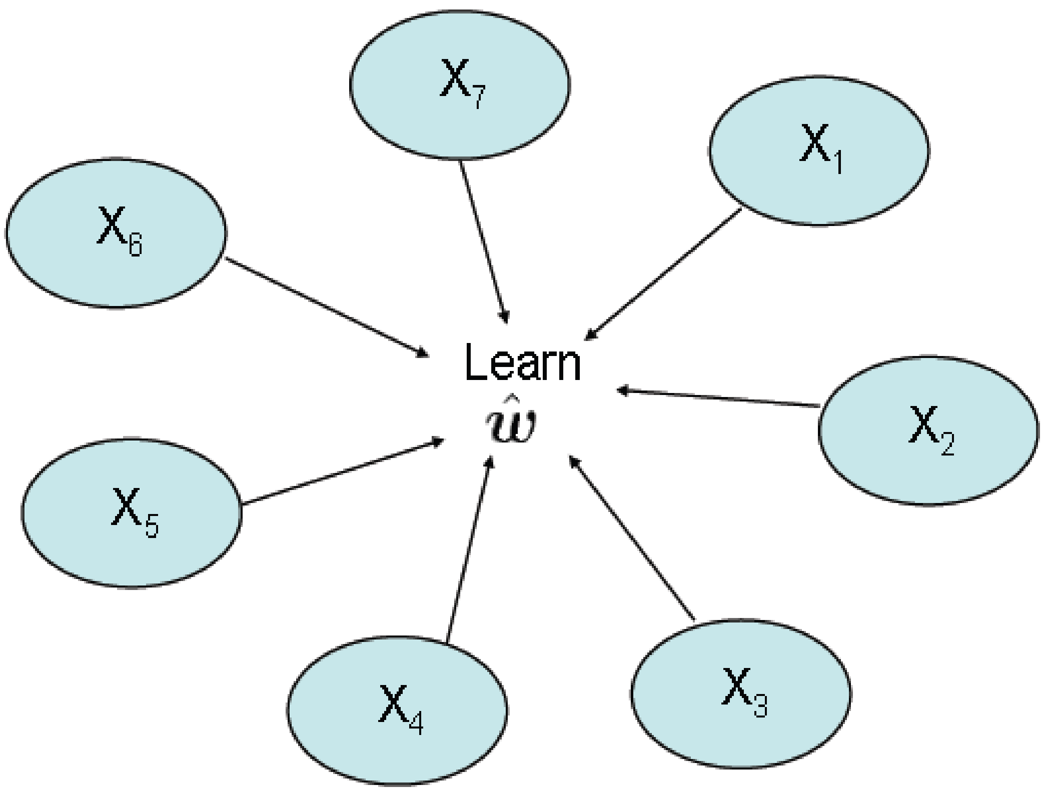
- A technique for estimating test error
- Uses *all* of the data to validate
- Divide data into K groups $\{X_1, X_2, \ldots, X_K\}$.
- Use each group as a validation set, then average all validation errors



$$L_2 = \sum_{(\boldsymbol{x},y)\in X_2} (y - \hat{\boldsymbol{w}}^\top \boldsymbol{x})$$

K-fold cross validation

- A technique for estimating test error
- Uses *all* of the data to validate
- Divide data into K groups $\{X_1, X_2, \ldots, X_K\}$.
- Use each group as a validation set, then average all validation errors



$$CV(s) = \frac{1}{K}\sum_{i=1}^{K} L_i$$

Deep Learning definition

- *Deep learning is a particular kind of machine learning that achieves great power and flexibility by learning to represent the world as nested hierarchy of concepts, with each concept defined in relation to simpler concepts, and more abstract representations computed in terms of less abstract ones.*

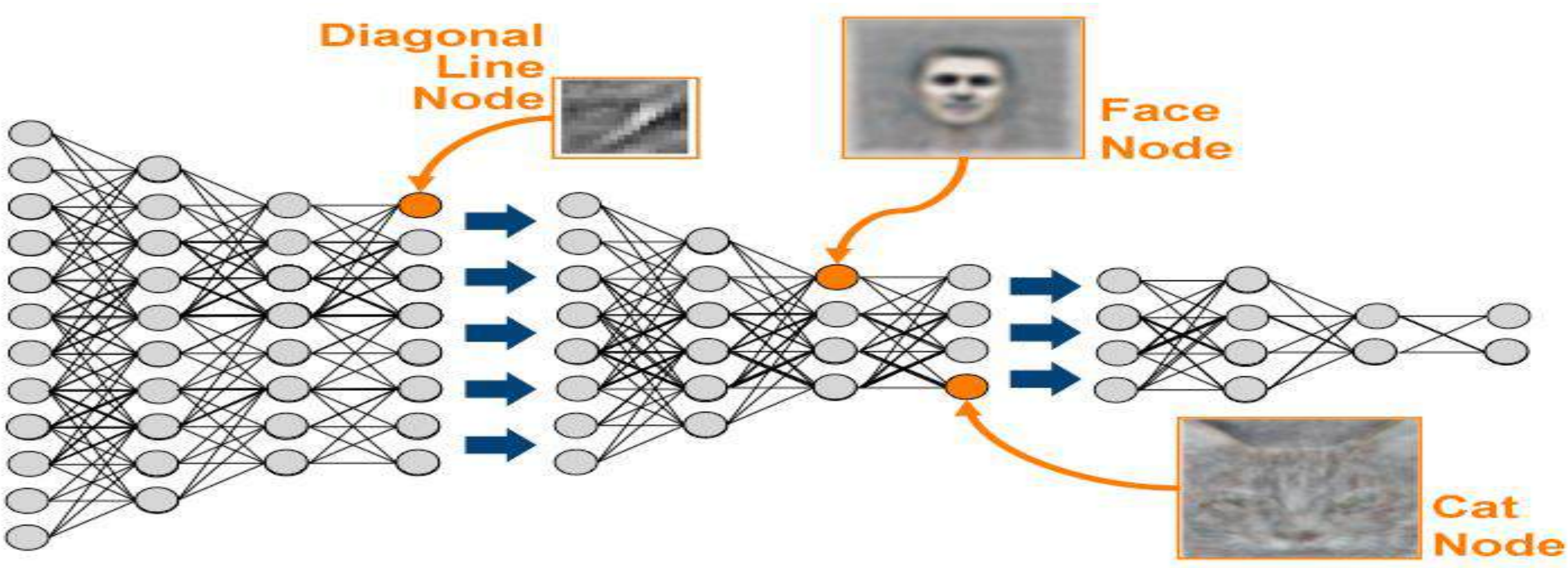
- Learning deep (many layered) neural networks

- The more layers in a Neural Network, the more abstract features can be represented

Deep Learning definition

E.g. Classify a cat:

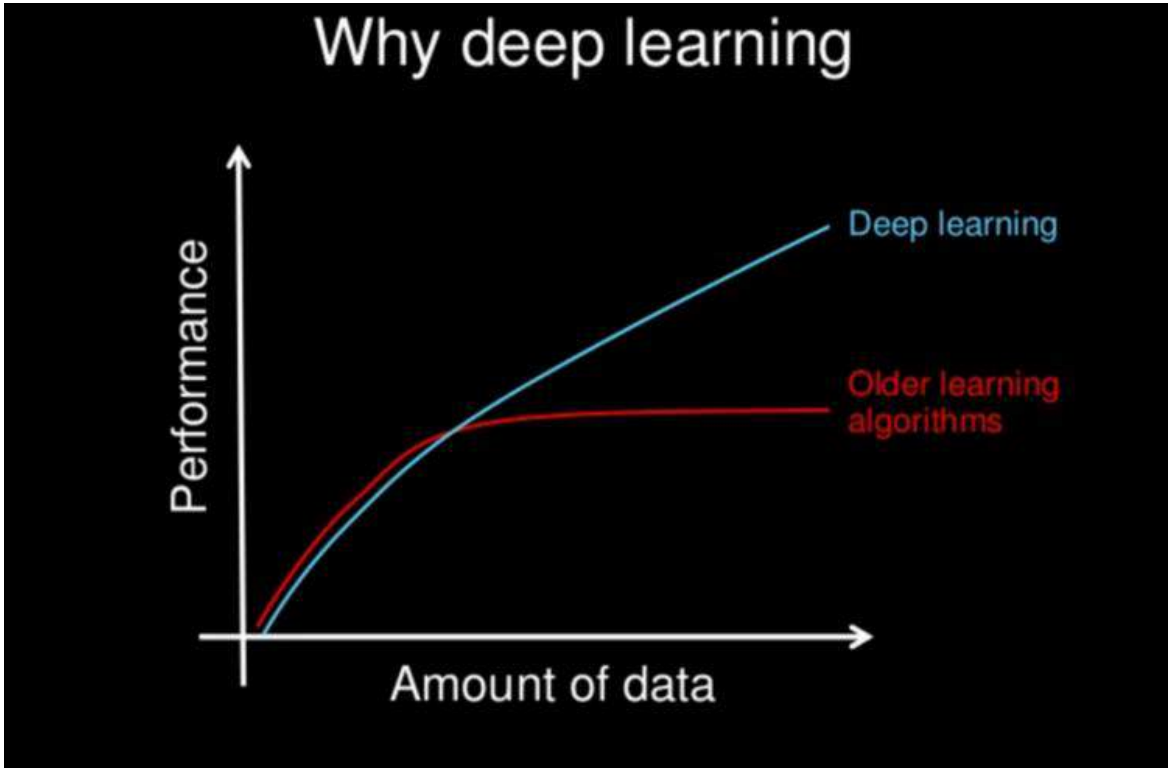
– Bottom Layers: Edge detectors, curves, corners straight lines
– Middle Layers: Fur patterns, eyes, ears
– Higher Layers: Body, head, legs
– Top Layer: Cat or Dog



Machine Learning VS Deep Learning

1- Data Dependency

- Deep learning need large amount of data to understand it perfectly



Machine Learning VS Deep Learning

2- Hardware Dependency

- Deep learning algorithms heavily depend on high-end machines This is because the requirements of deep learning algorithm include GPUs which are an integral part of its working.
- Machine Learning which can work on low-end machines.

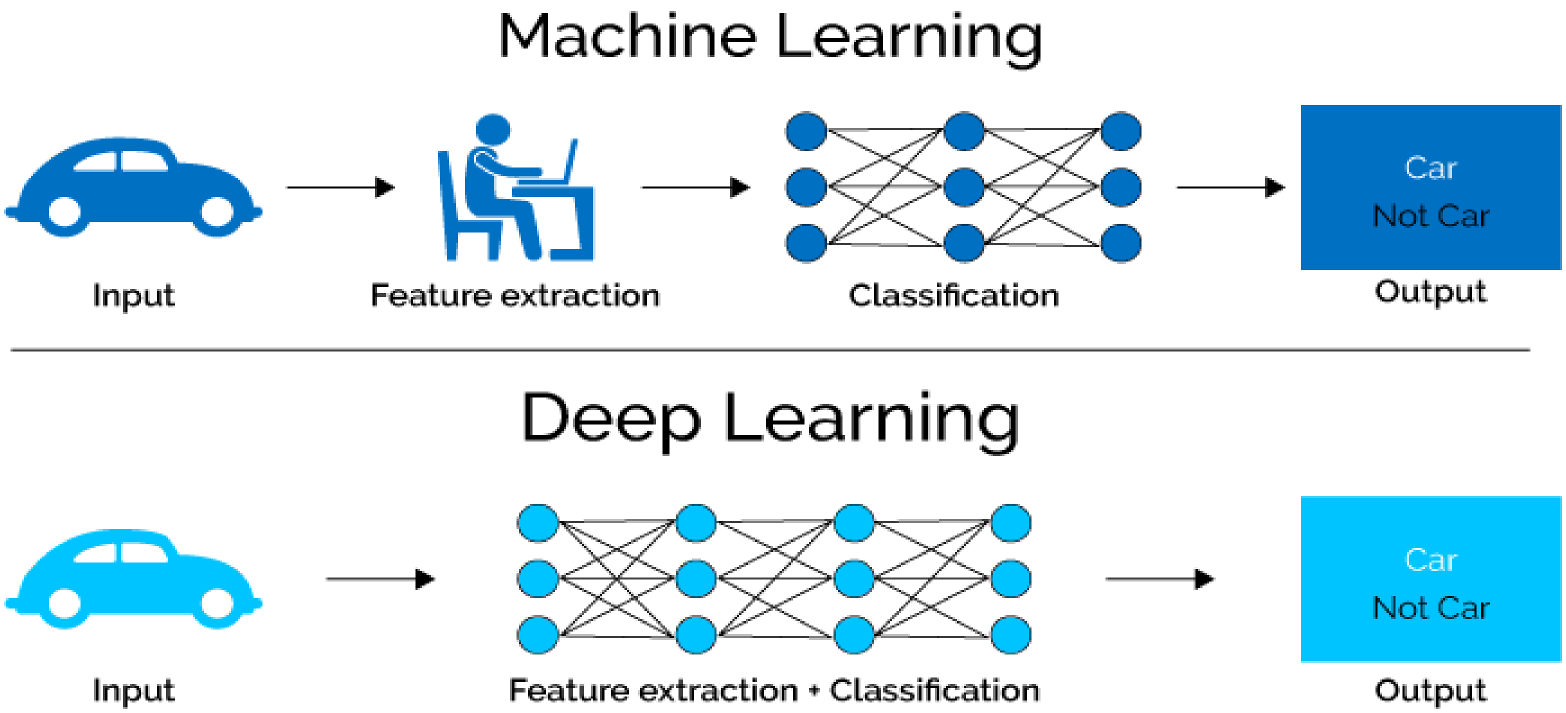
3- Execution time

- deep learning algorithm takes a long time to train. This is because there are so many parameters in a deep learning algorithm that training them takes longer than usual.

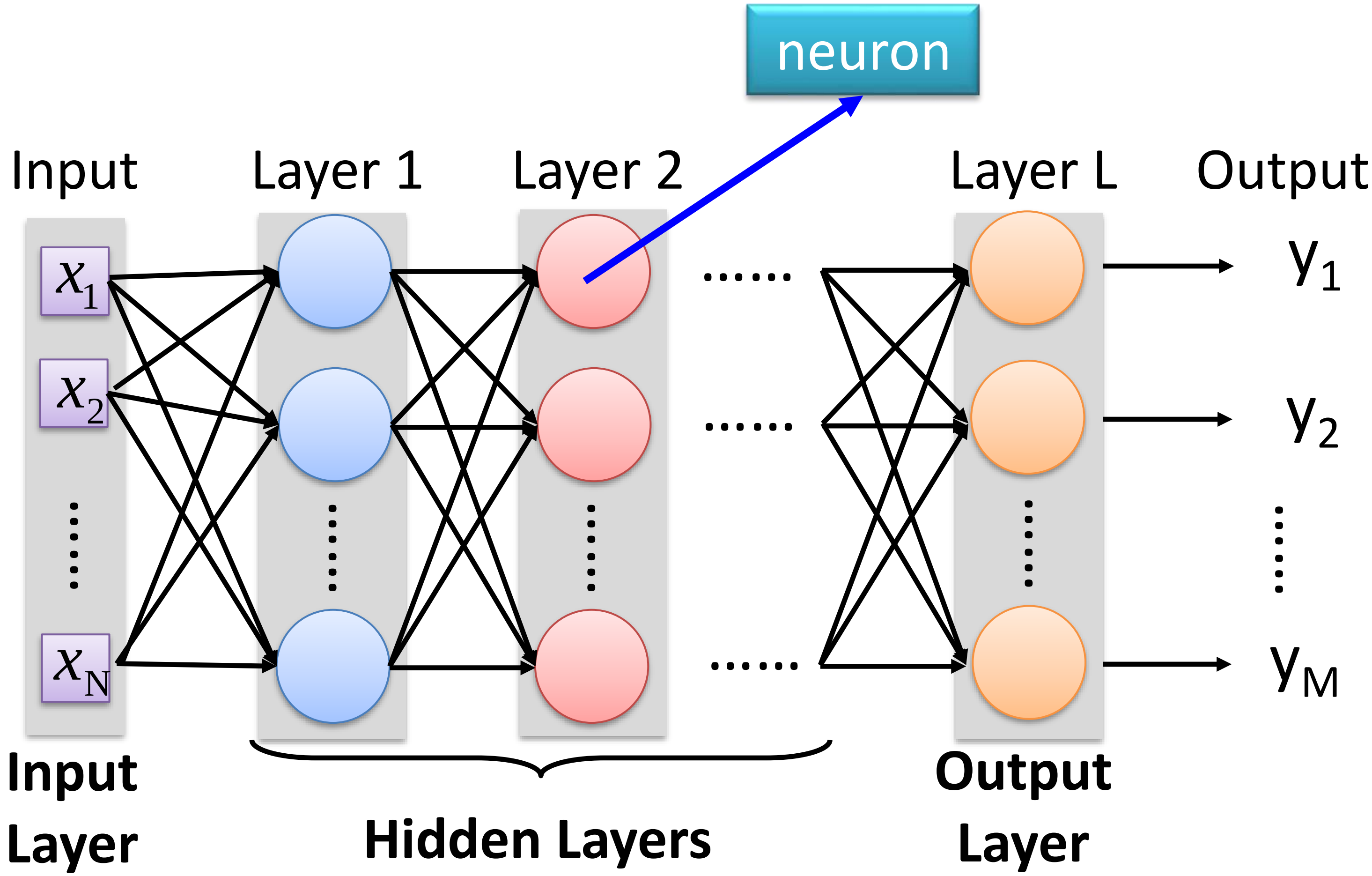
Machine Learning VS Deep Learning

4- Feature engineering

- Deep learning algorithms try to learn high-level features from data.
- Machine Learning which can work on low-end machines.



Element of Neural Network

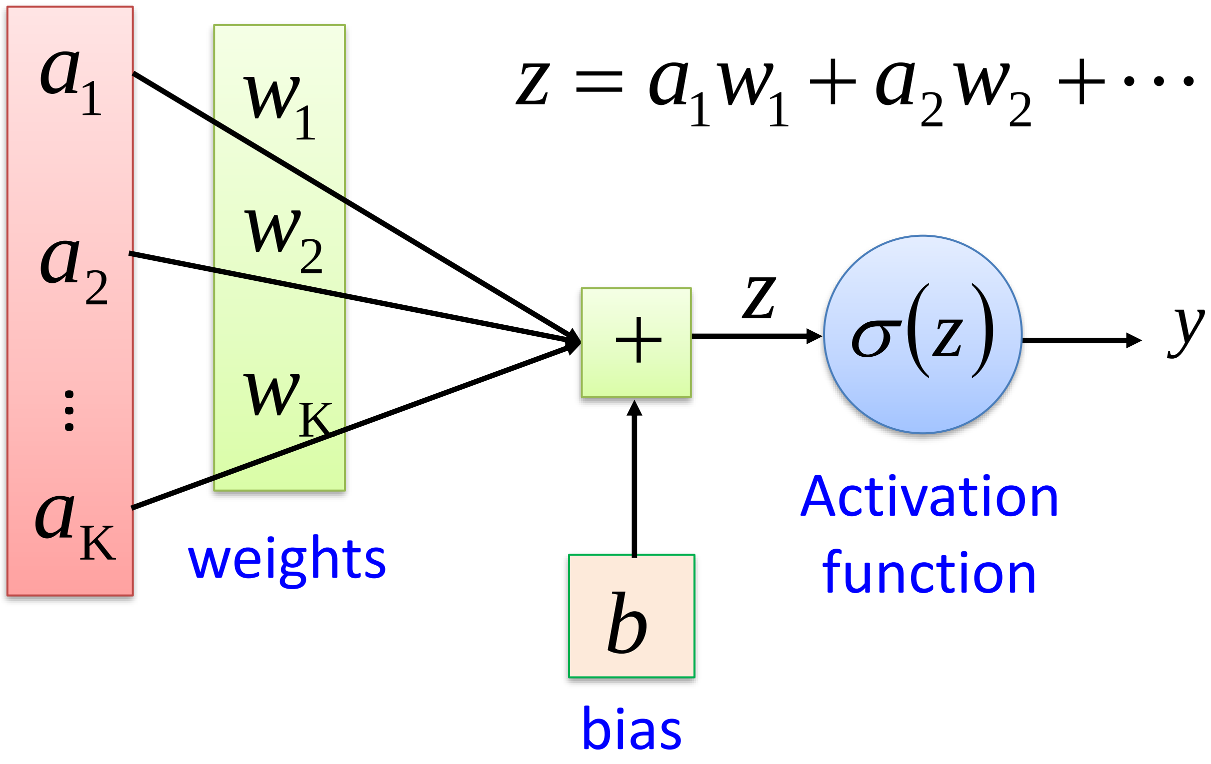


Deep means many hidden layers

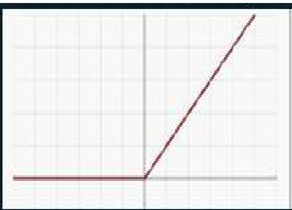
Neural Network

Neuron $f: R^K \rightarrow R$

$$z = a_1 w_1 + a_2 w_2 + \cdots + a_K w_K + b$$

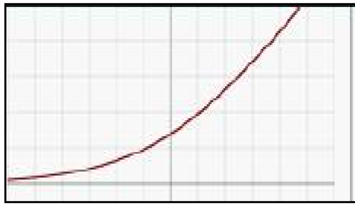


Activation Function types



$$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases}$$

ReLU



$$f(x) = \ln(1 + e^x)$$

Softplus

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

Sigmoid/logistic

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

Tanh

$$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ 1 & \text{for} \quad x \geq 0 \end{cases}$$

Binary

$$f(x) = \begin{cases} -1 & \text{for} \quad x < 0 \\ 1 & \text{for} \quad x > 0 \end{cases}$$

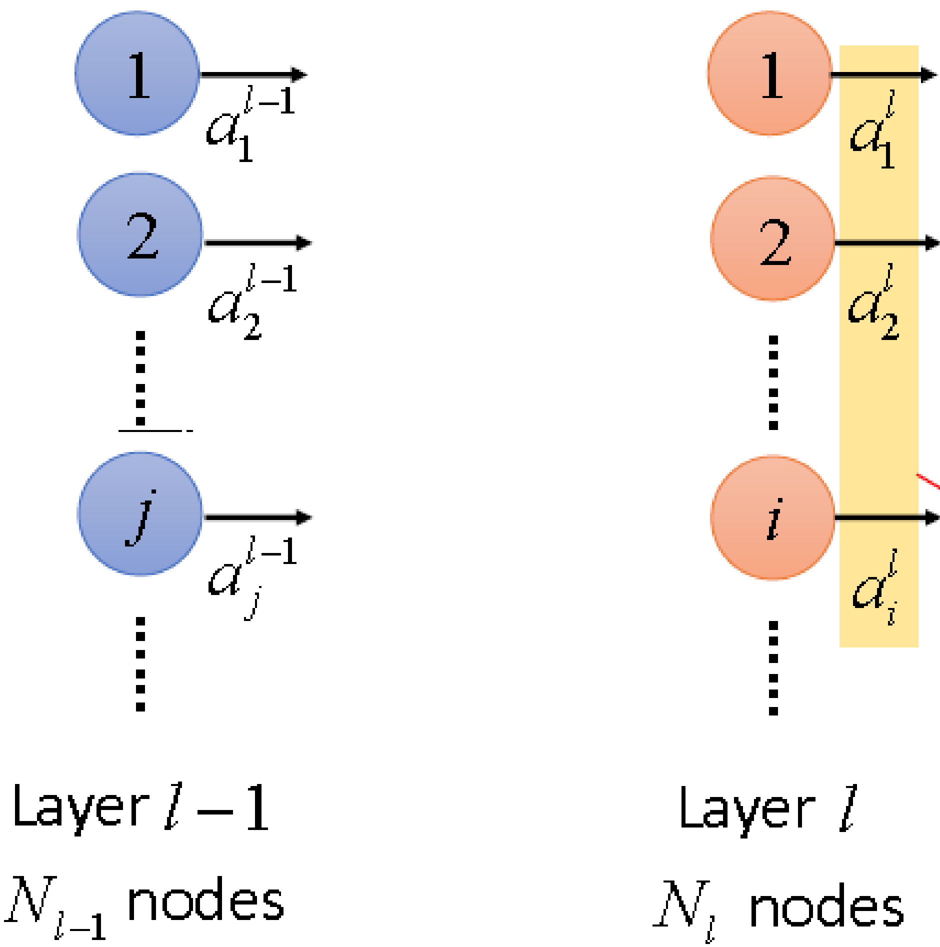
Signum

$$f_i(\vec{x}) = \frac{e^{x_i}}{\sum_{j=1}^{J} e^{x_j}} \quad \text{for } i = 1, \ldots, J$$

Softmax

Fully Connected Layer

Vanilla



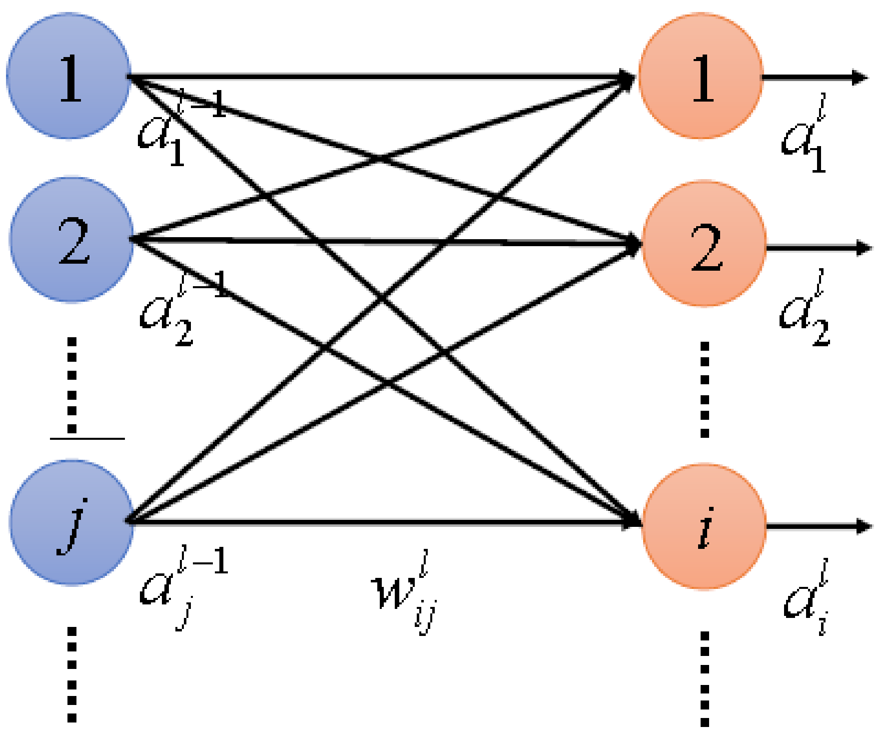
Output of a neuron:

a_i^l → Layer l, Neuron i

Output of one layer:

a^l : a vector

Fully Connected Layer



Layer $l-1$

N_{l-1} nodes

Layer l

N_l nodes

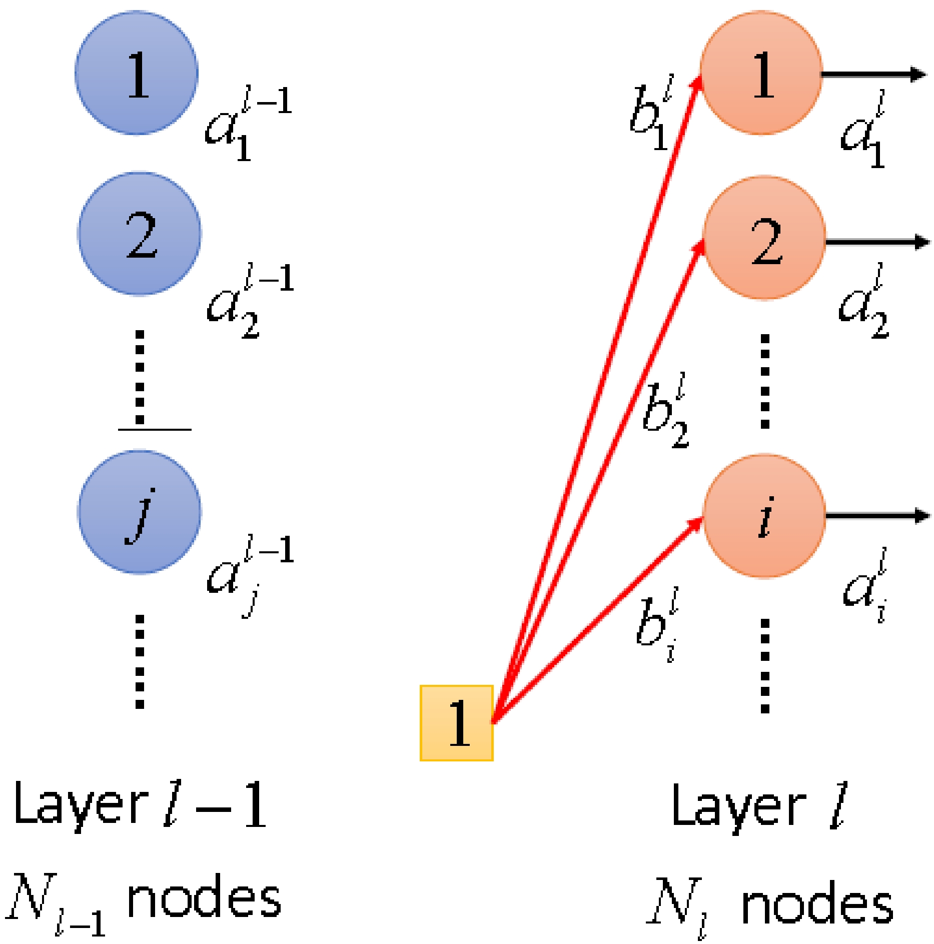
w_{ij}^{l} → Layer $l-1$ to Layer l

↓

from neuron j (Layer $l-1$)
to neuron i (Layer l)

$$W^l = \left.\overbrace{\begin{bmatrix} w_{11}^l & w_{12}^l & \cdots \\ w_{21}^l & w_{22}^l & \\ \vdots & & \ddots \\ & & \end{bmatrix}}^{N_{l-1}}\right\} N_l$$

Fully Connected Layer



b_i^l : bias for neuron i at layer l

$$b^l = \begin{bmatrix} b_1^l \\ b_2^l \\ \vdots \\ b_i^l \\ \vdots \end{bmatrix}$$

bias for all neurons in layer l

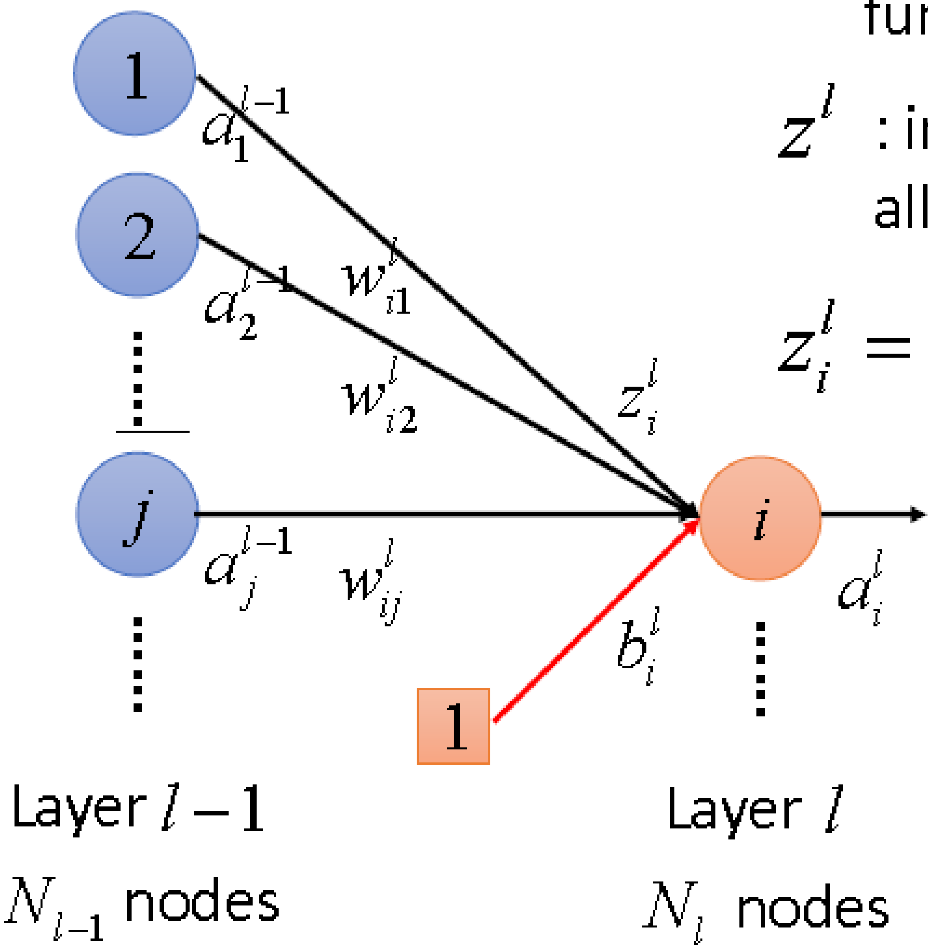
Fully Connected Layer

z_i^l : input of the activation function for neuron i at layer l

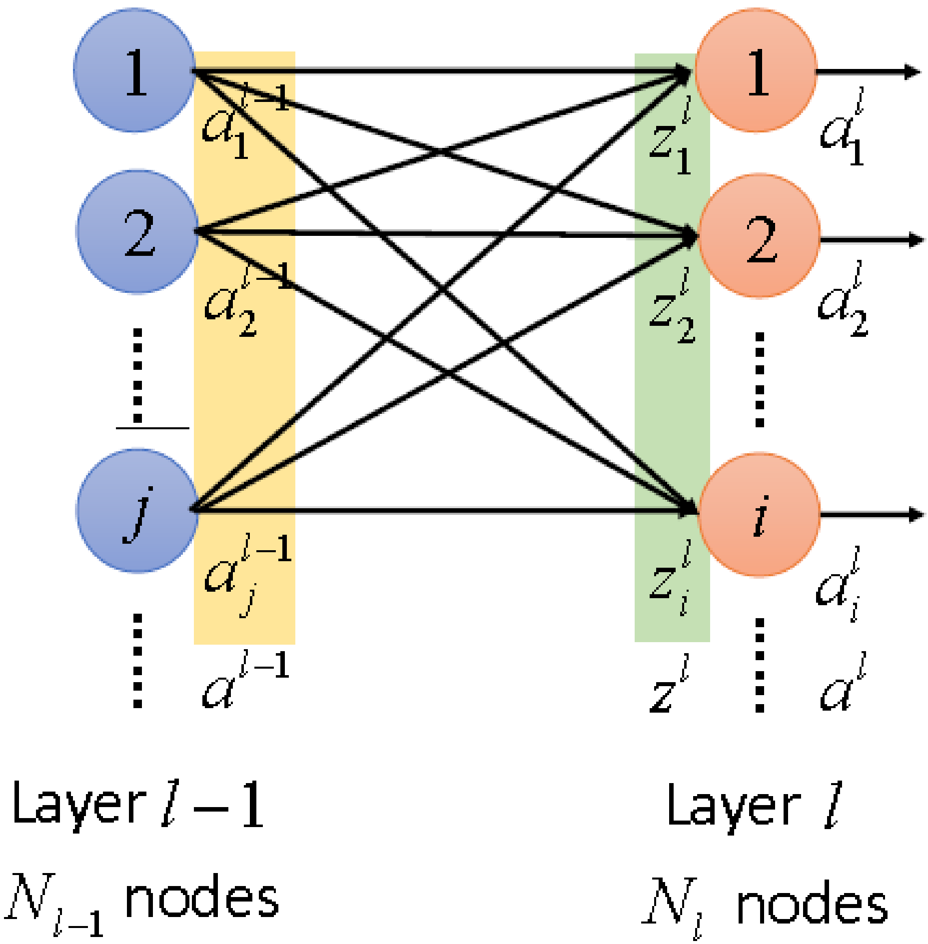
z^l : input of the activation function all the neurons in layer l

$$z_i^l = w_{i1}^l a_1^{l-1} + w_{i2}^l a_2^{l-1} \ldots + b_i^l$$

$$z_i^l = \sum_{j=1}^{N_{l-1}} w_{ij}^l a_j^{l-1} + b_i^l$$



Relations between Layer Outputs

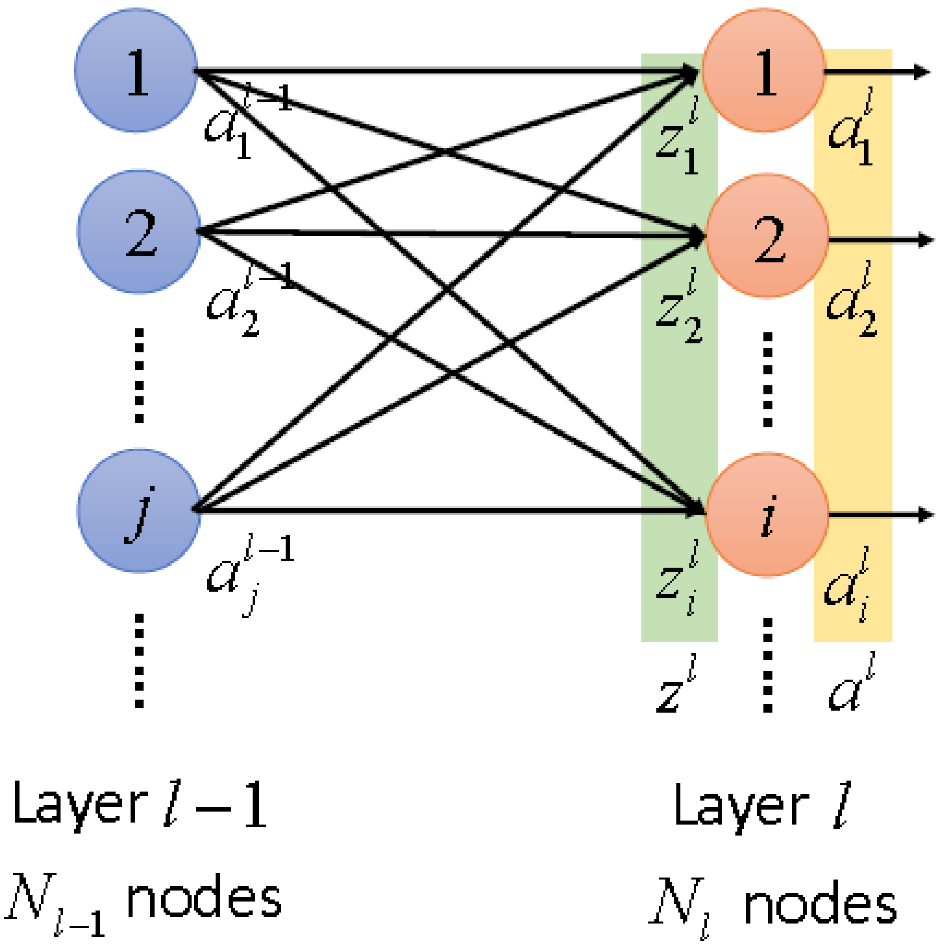


$$z_1^l = w_{11}^l a_1^{l-1} + w_{12}^l a_2^{l-1} + \cdots + b_1^l$$
$$z_2^l = w_{21}^l a_1^{l-1} + w_{22}^l a_2^{l-1} + \cdots + b_2^l$$
$$\vdots$$
$$z_i^l = w_{i1}^l a_1^{l-1} + w_{i2}^l a_2^{l-1} + \cdots + b_i^l$$
$$\vdots$$

$$\begin{bmatrix} z_1^l \\ z_2^l \\ \vdots \\ z_i^l \\ \vdots \end{bmatrix} = \begin{bmatrix} w_{11}^l & w_{12}^l & \cdots \\ w_{21}^l & w_{22}^l & \\ \vdots & & \ddots \end{bmatrix} \begin{bmatrix} a_1^{l-1} \\ a_2^{l-1} \\ \vdots \\ a_i^{l-1} \\ \vdots \end{bmatrix} + \begin{bmatrix} b_1^l \\ b_2^l \\ \vdots \\ b_i^l \\ \vdots \end{bmatrix}$$

$$z^l = W^l a^{l-1} + b^l$$

Relations between Layer Outputs

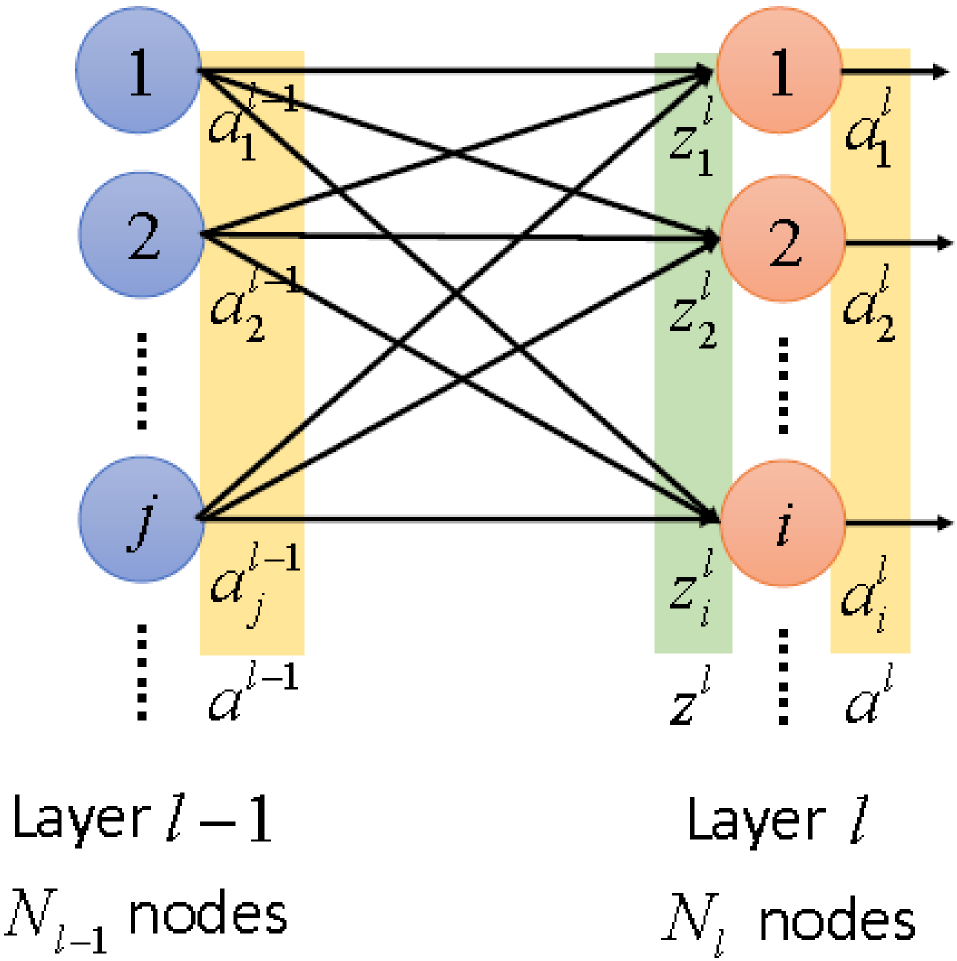


$$a_i^l = \sigma\left(z_i^l\right)$$

$$\begin{bmatrix} a_1^l \\ a_2^l \\ \vdots \\ a_i^l \\ \vdots \end{bmatrix} = \begin{bmatrix} \sigma\left(z_1^l\right) \\ \sigma\left(z_2^l\right) \\ \vdots \\ \sigma\left(z_i^l\right) \\ \vdots \end{bmatrix}$$

$$a^l = \sigma\left(z^l\right)$$

Relations between Layer Outputs

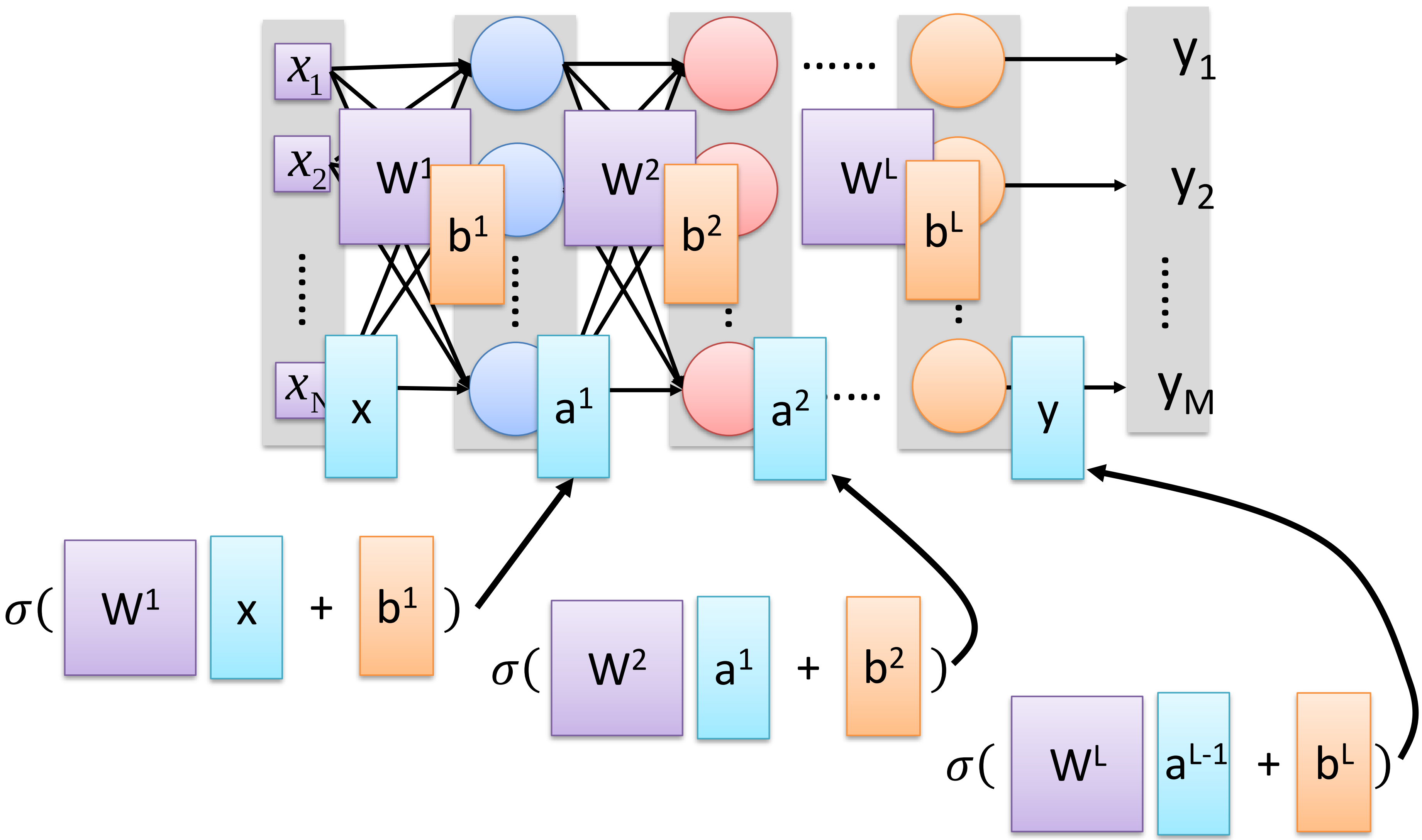


$$z^l = W^l a^{l-1} + b^l$$

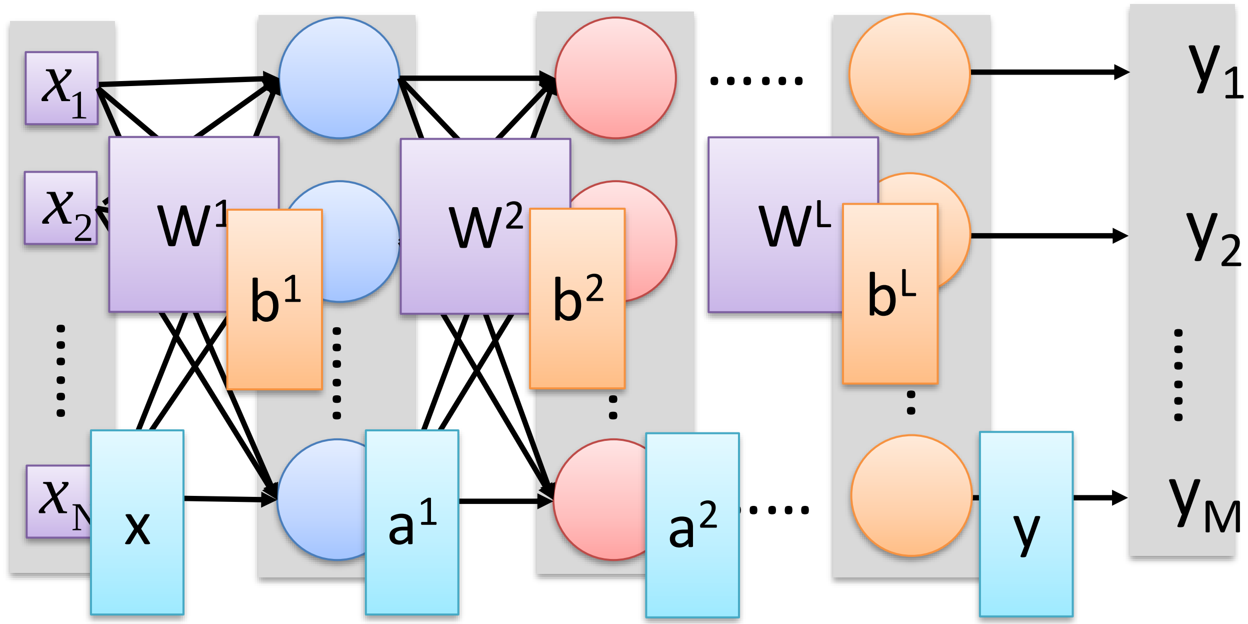
$$a^l = \sigma\left(z^l\right)$$

$$a^l = \sigma\left(W^l a^{l-1} + b^l\right)$$

Neural Network



Neural Network



$$y = f(x)$$

$$= \sigma(W^L \cdots \sigma(W^2 \, \sigma(W^1 x + b^1) + b^2) \cdots + b^L)$$

Neural Network training steps

1. Weight Initialization
2. Inputs Application
3. Sum of inputs - Weights product
4. Activation functions
5. Weights Adaptations
6. Back to step 2

Regarding 5th step: Weights Adaptation

First method:

- If the predicted output Y is not the same as the desired output d, then weights are to be adapted according to the following equation:

$$W(n+1) = W(n) + \eta[d(n) - Y(n)]X(n)$$

Where

$$W(n) = [b(n), W_1(n), W_2(n), W_3(n), \ldots, W_m(n)]$$

Learning Rate η

$0 \leq \eta \leq 1$

$0 \leq \alpha \leq 1$

Q. Why new weights are better than old weights?

Q. What is the effect of each weight over the prediction error?

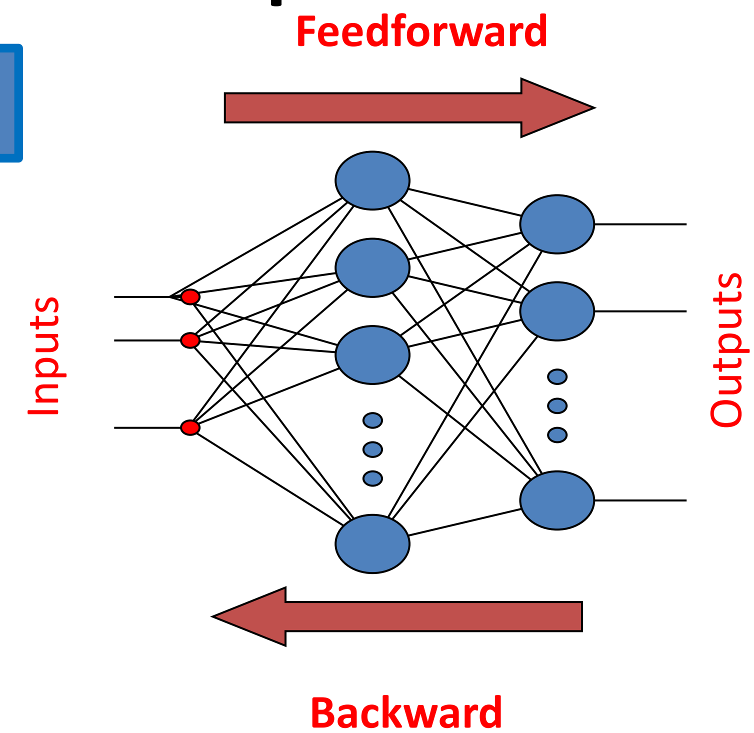
Q. How increasing or decreasing weights affects the prediction error?

Regarding 5th step: Weights Adaptation

second method: Back propagation

- **Fowrward VS Backword passes**

The Backpropagation algorithm is a sensible approach for dividing the contribution of each weight.



Fowrward	**Input weights** →	**SOP** →	**Prediction Output** →	**Prediction Error**
backward	**Prediction Error** →	**Prediction Output** →	**SOP** →	**Input weights**

Regarding 5th step: Weights Adaptation

second method: Back propagation

- **Backword pass**

What is the change in prediction Error (E) given the change in weight (W) ?

Get partial derivative of E W.R.T W $\frac{\partial E}{\partial W}$

Prediction Error → **Prediction Output** → **SOP** → **Input weights**

$E = \frac{1}{2}(d - y)^2$

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

$s = \sum_{j}^{m} x_i \, w_{ji} + b_i$

w_1, w_2

d (desired output) Const

y (predicted output)

s (Sum Of Product SOP)

$$E = \frac{1}{2}\left(d - \frac{1}{e^{-\sum_{j}^{n} x_i w_{ij} + b_i}}\right)^2$$

Regarding 5th step: Weights Adaptation

second method: Back propagation

Chain Rule

- **Weight derivative**

Prediction Error → **Prediction Output** → **SOP** → **Input weights**

$$E = \frac{1}{2}(d-y)^2 \qquad y = f(s) = \frac{1}{1+e^{-s}} \qquad s = x_1 w_1 + x_2 w_2 + b \qquad w_1, w_2$$

$$\frac{\partial E}{\partial W} = \frac{\partial E}{\partial y} \times \frac{\partial y}{\partial s} \times \frac{\partial s}{\partial w_1}, \frac{\partial s}{\partial w_2}$$

$$\frac{\partial E}{\partial w_1} = \frac{\partial E}{\partial y} x \frac{\partial y}{\partial s} x \frac{\partial s}{\partial w_1}$$

$$\frac{\partial E}{\partial w_2} = \frac{\partial E}{\partial y} x \frac{\partial y}{\partial s} x \frac{\partial s}{\partial w_2}$$

Regarding 5th step: Weights Adaptation

second method: Back propagation

- **Weight derivative**

$$\frac{\partial E}{\partial y} = \frac{\partial}{\partial y}\frac{1}{2}(d-y)^2 = y-d$$

$$\frac{\partial y}{\partial s} = \frac{\partial}{\partial s}\frac{1}{1+\mathrm{e}^{-s}} = \frac{1}{1+\mathrm{e}^{-s}}\left(1-\frac{1}{1+\mathrm{e}^{-s}}\right)$$

$$\frac{\partial s}{\partial w_1} = \frac{\partial}{\partial w_1} x_1 w_1 + x_2 w_2 + b \;= x_1$$

$$\frac{\partial s}{\partial w_2} = \frac{\partial}{\partial w_2} x_1 w_1 + x_2 w_2 + b \;= x_2$$

$$\frac{\partial E}{\partial w_i} = (y-d)\frac{1}{1+\mathrm{e}^{-s}}\left(1-\frac{1}{1+\mathrm{e}^{-s}}\right)x_i$$

Regarding 5th step: Weights Adaptation

second method: Back propagation

- **interpreting derivatives** ∇W

$$\frac{\partial E}{\partial w_i} = (y-d)\frac{\partial f(s)}{\partial s} x_i$$

Derivatives sign		Derivatives Magnitude
Increasing/decreasing weight increases/decreases error.	**Positive** directly proportional	Increasing/decreasing weight by P increases/decreases error by MAG*P.
Increasing/decreasing weight decreases/increases error.	**Negative** opposite	Increasing/decreasing weight by P decreases/increases error by MAG*P.

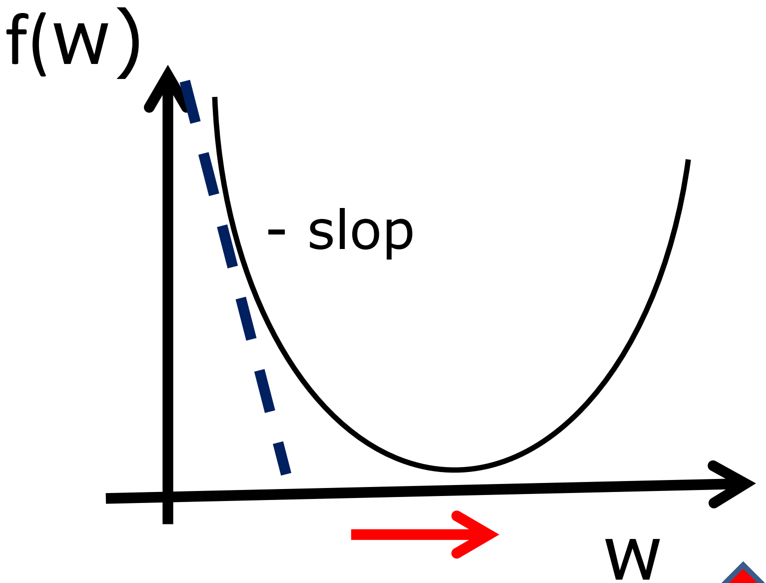
Regarding 5th step: Weights Adaptation

second method: Back propagation

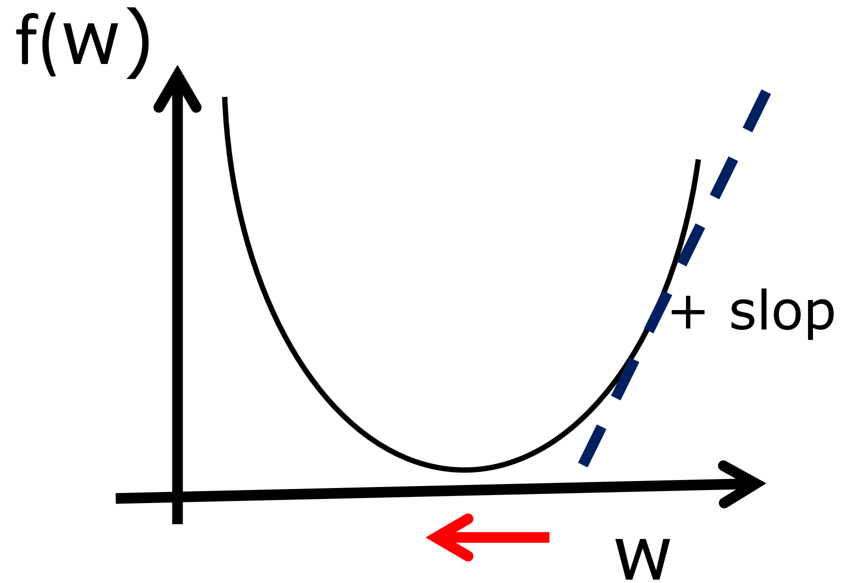
- **Update the Weights**

In order to update the weights , use the Gradient Descent

$$W_{inew} = W_{iold} - \eta * \frac{\partial E}{\partial W_i}$$



$W_{new} = W_{old} - \eta(-ve)$



$W_{new} = W_{old} - \eta(+ve)$

Convolution Neural Network
CNN

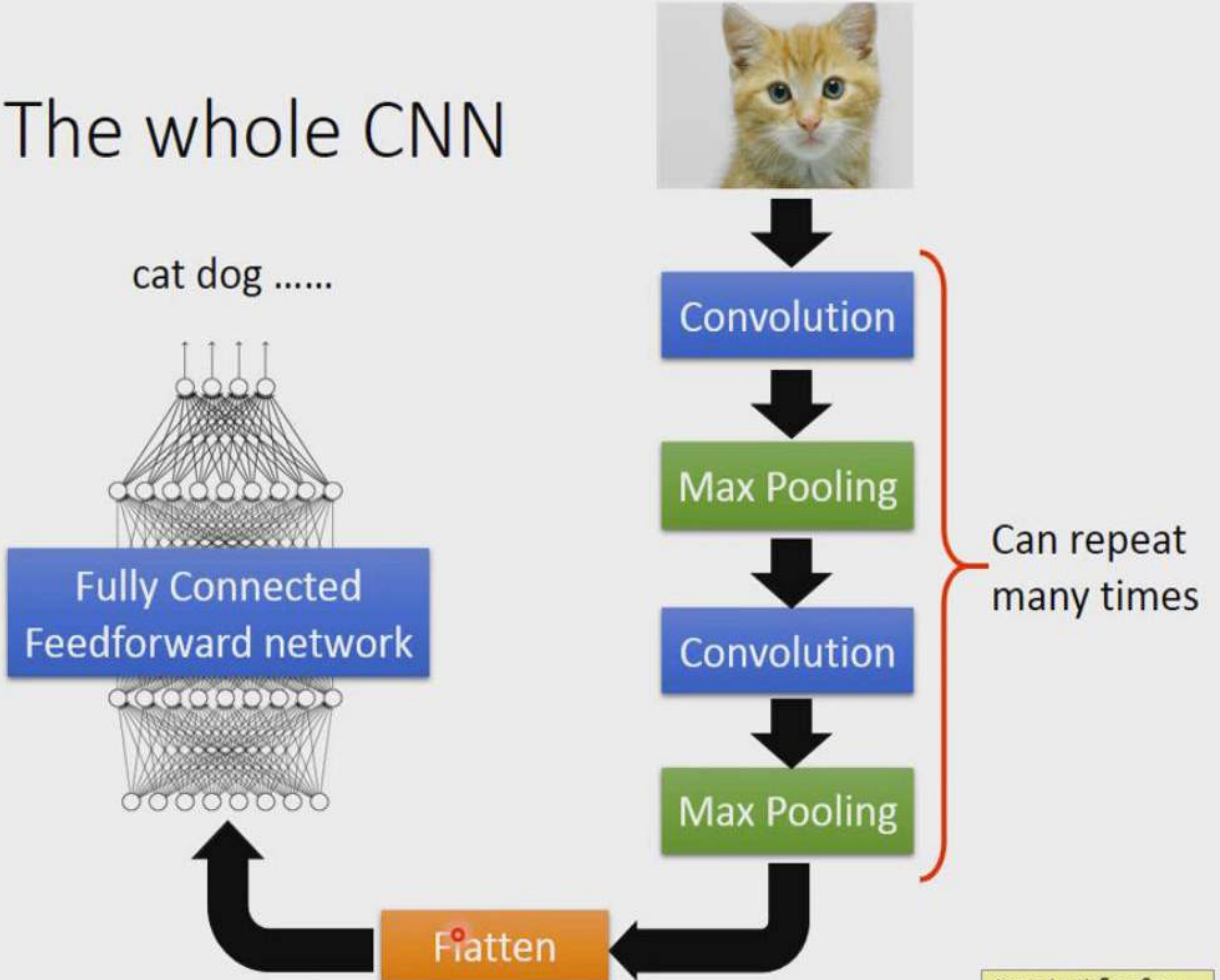
introduction

- Convolutional neural networks (or convnets for short) are used in situations where data can be expressed as a "map" wherein the proximity between two data points indicates how related they are.

- Convnets contain one or more of each of the following layers:

1. convolution layer
2. ReLU (rectified linear units) layer (element wise threshold)
3. pooling layer
4. fully connected layer
5. loss layer (during the training process)

The whole CNN



The whole CNN

Property 1

- Some patterns are much smaller than the whole image

Property 2

- The same patterns appear in different regions.

Property 3

- Subsampling the pixels will not change the object

Convolution

Max Pooling

Convolution

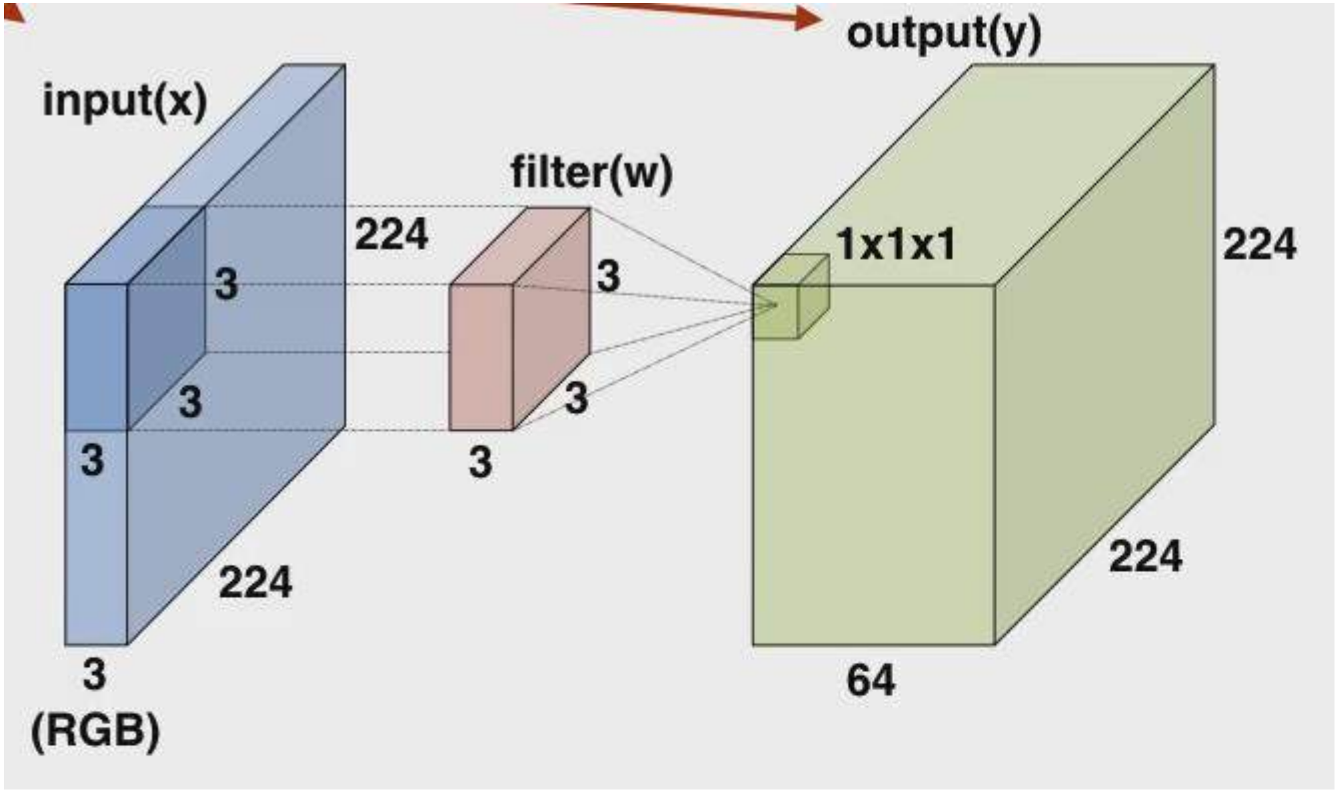
Max Pooling

Can repeat many times

Flatten

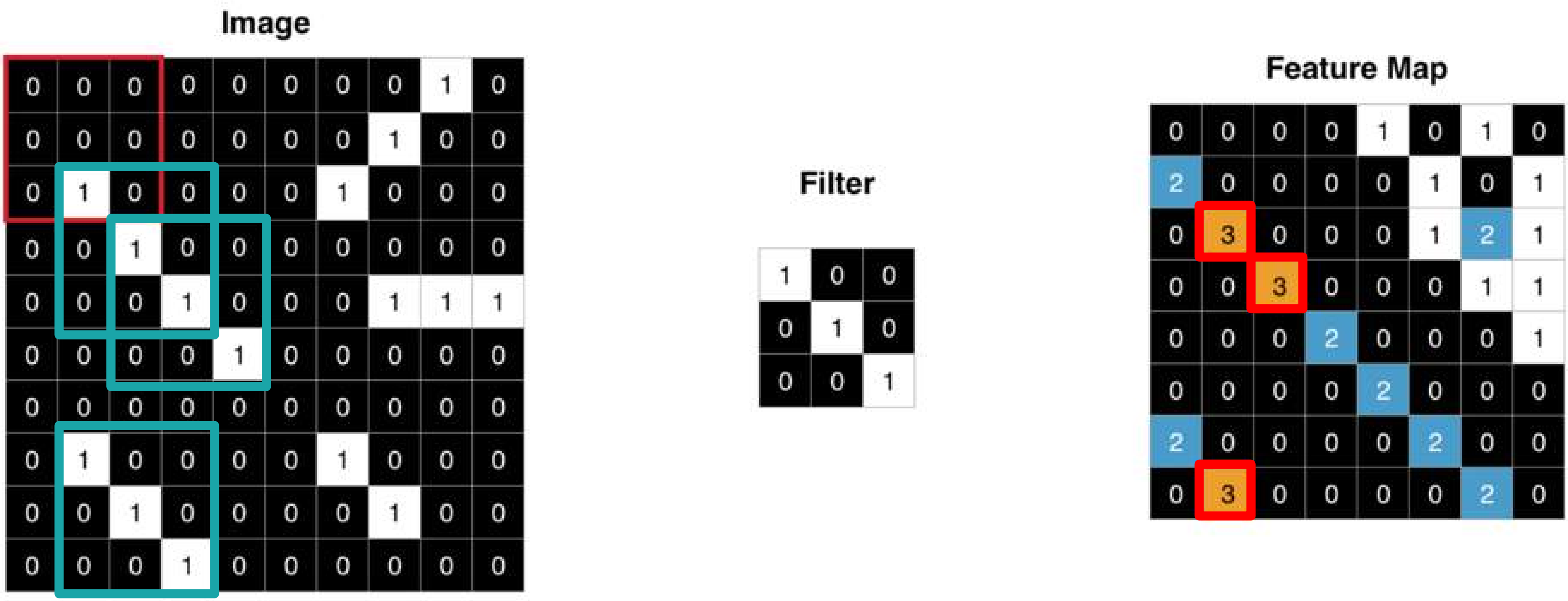
1- Convolution layer

a convnet processes an image using a matrix of weights called filters (or features) that detect specific attributes such as diagonal edges, vertical edges, etc. Moreover, as the image progresses through each layer, the filters are able to recognize more complex attributes.



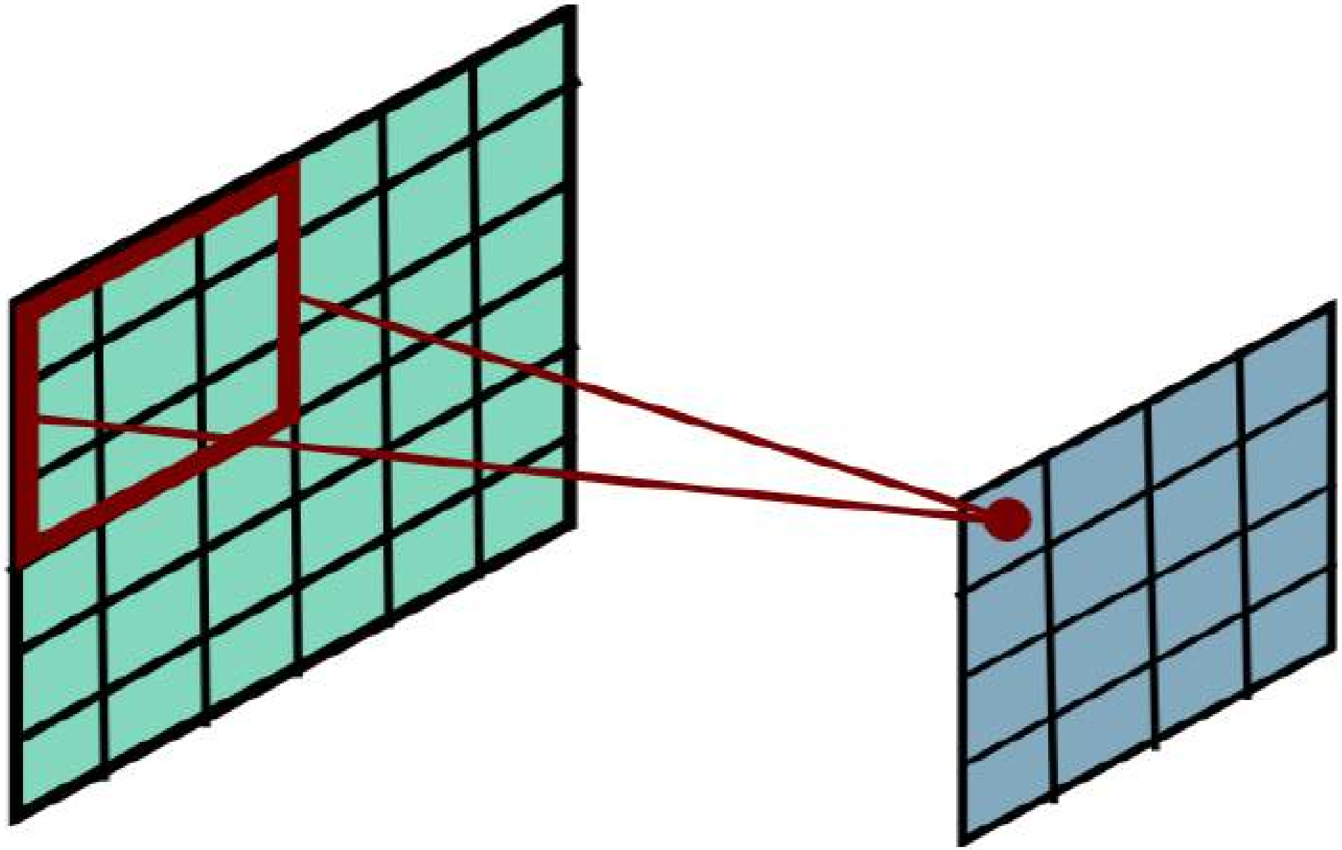
Convolution layer

The convolution layer is always the first step in a convnet. Let's say we have a 10 x 10 pixel image, here represented by a 10 x 10 x 1 matrix of numbers:

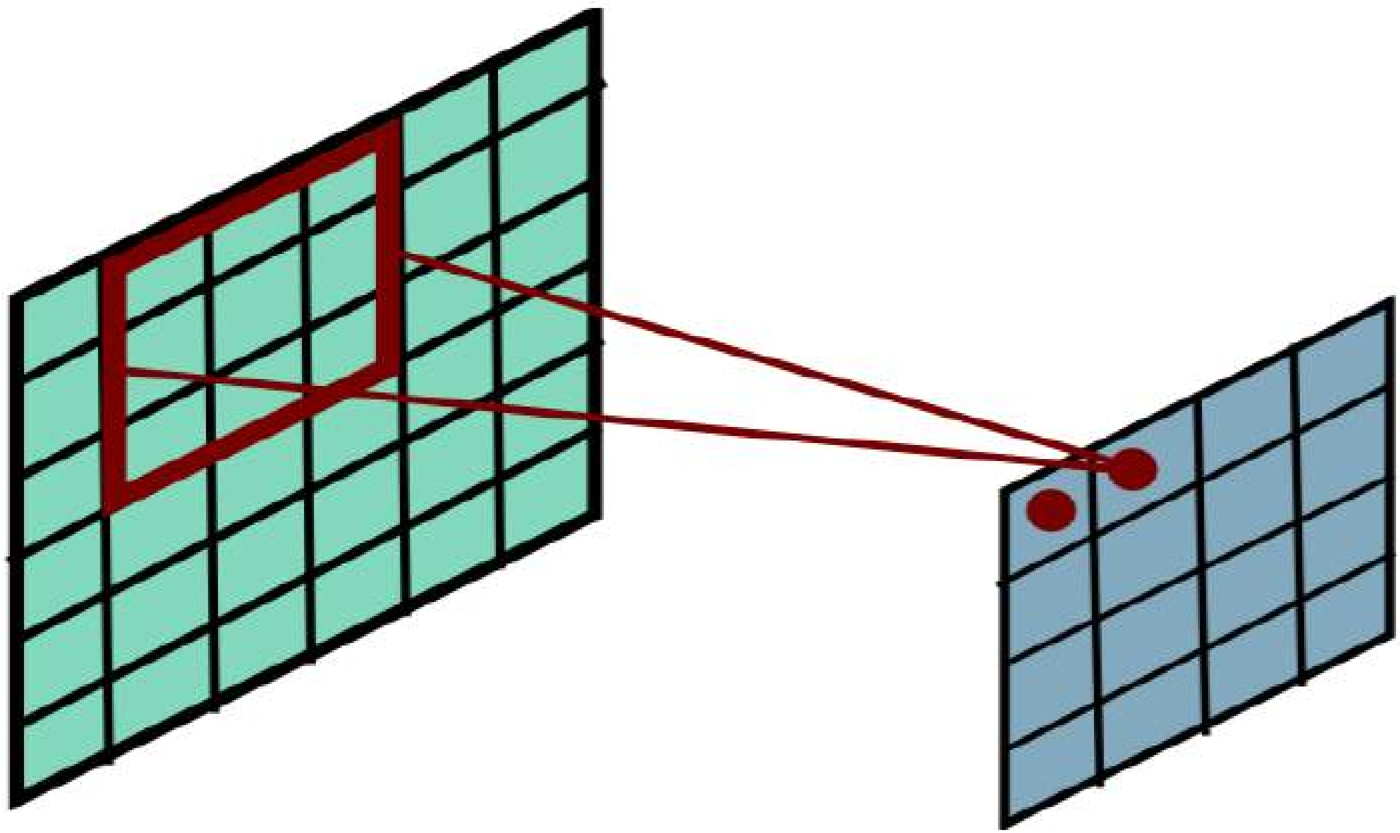


$0 \times 1 + 0 \times 0 + 0 \times 0 + 0 \times 0 + 0 \times 1 + 0 \times 0 + 0 \times 0 + 1 \times 0 + 0 \times 1 = 0$

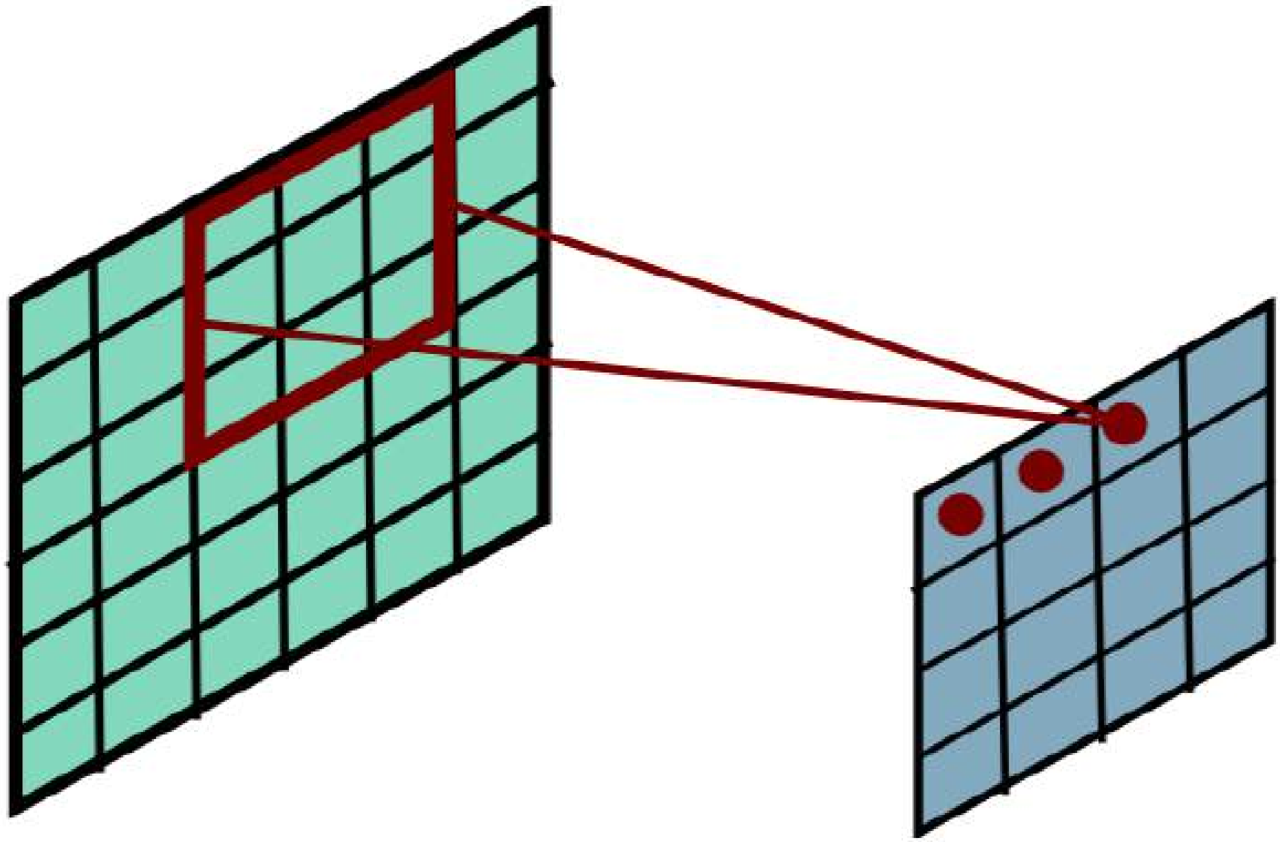
Convolutional Layer



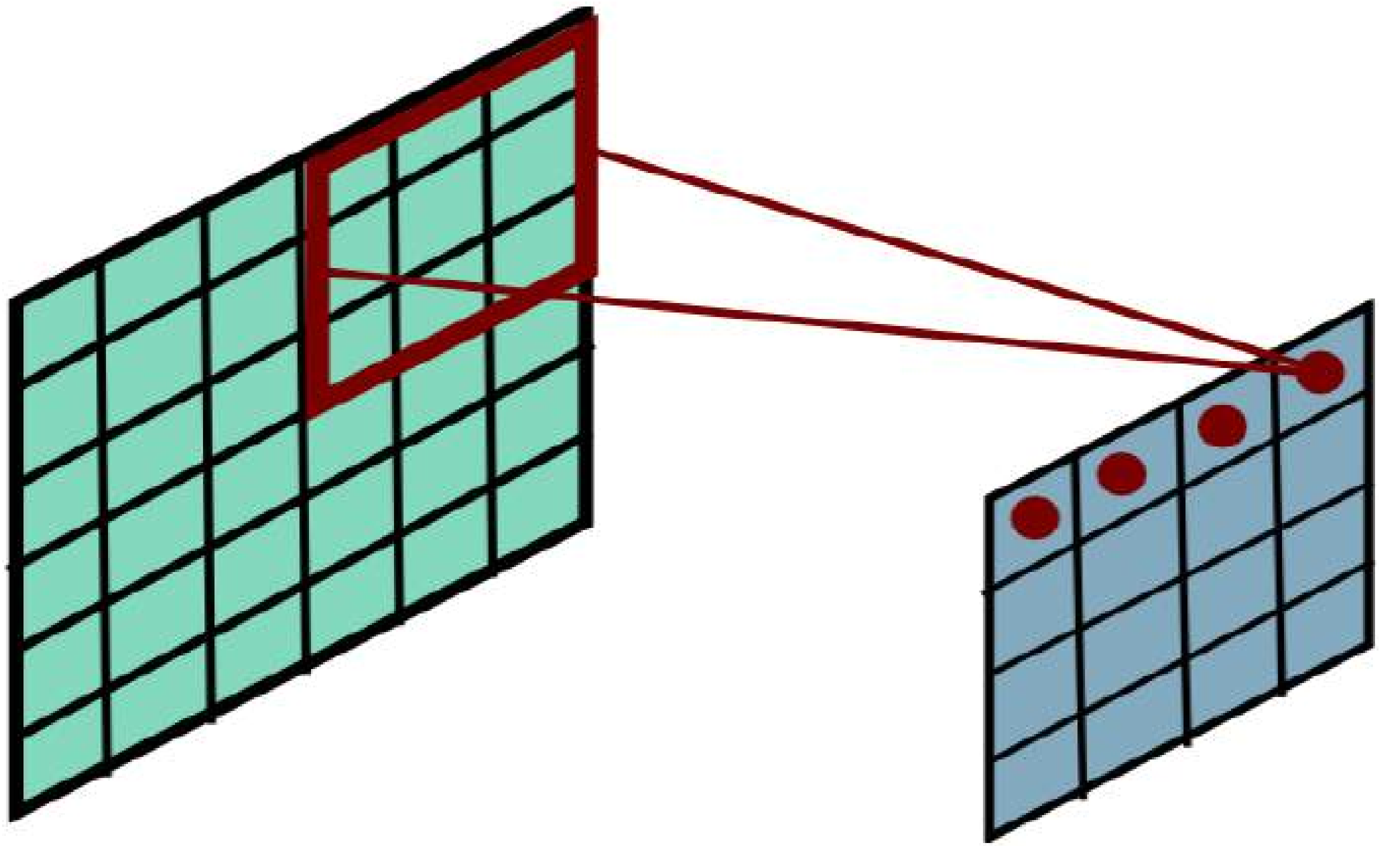
Convolutional Layer



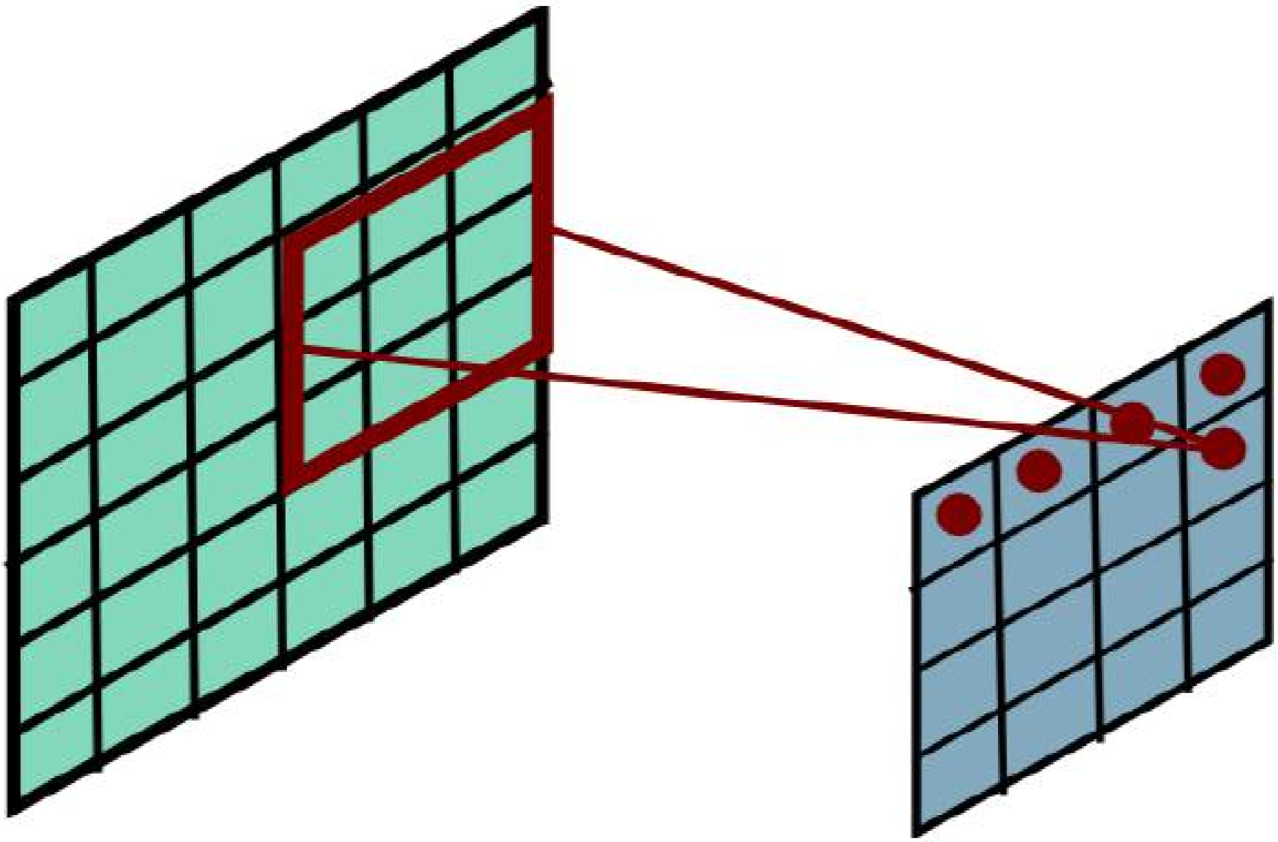
Convolutional Layer



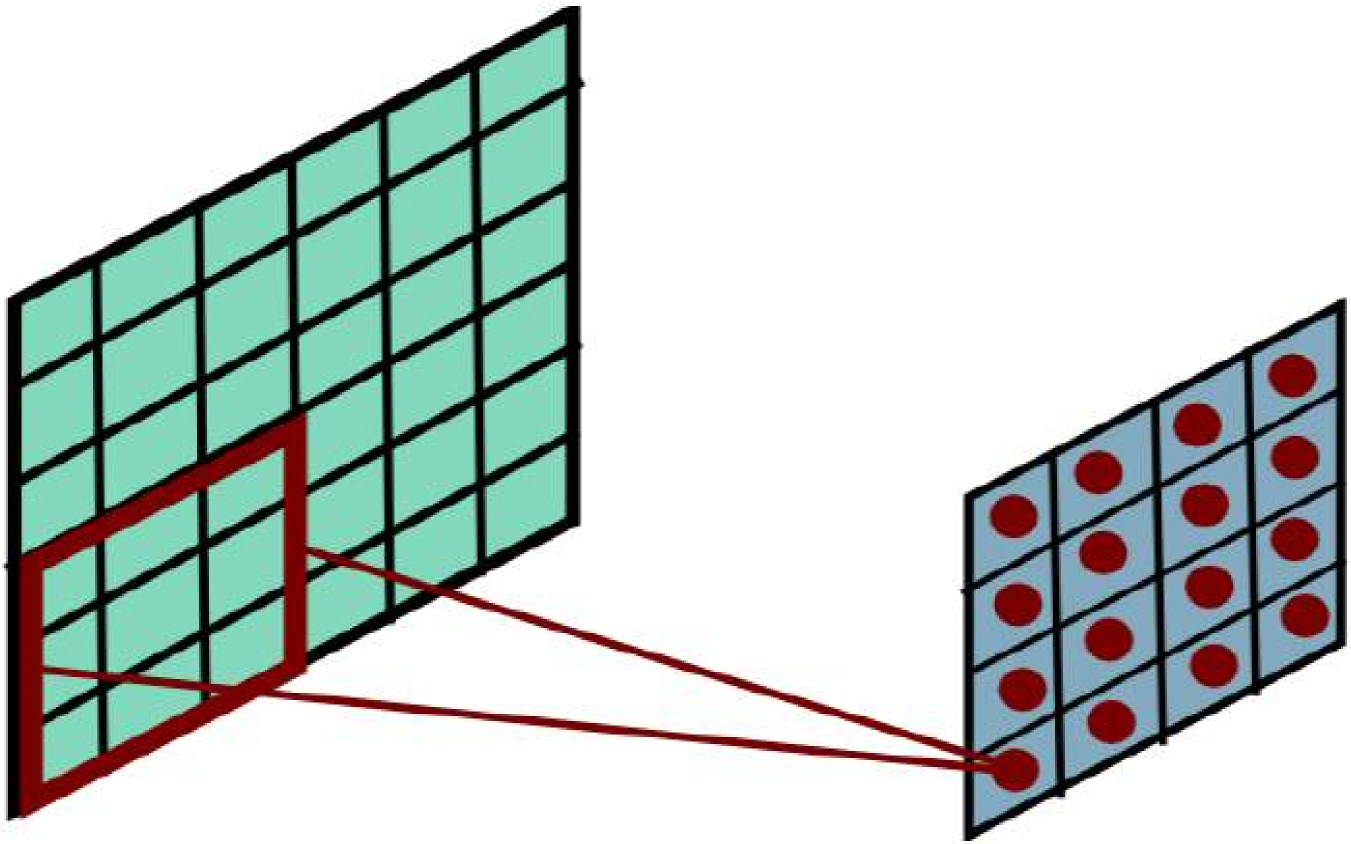
Convolutional Layer



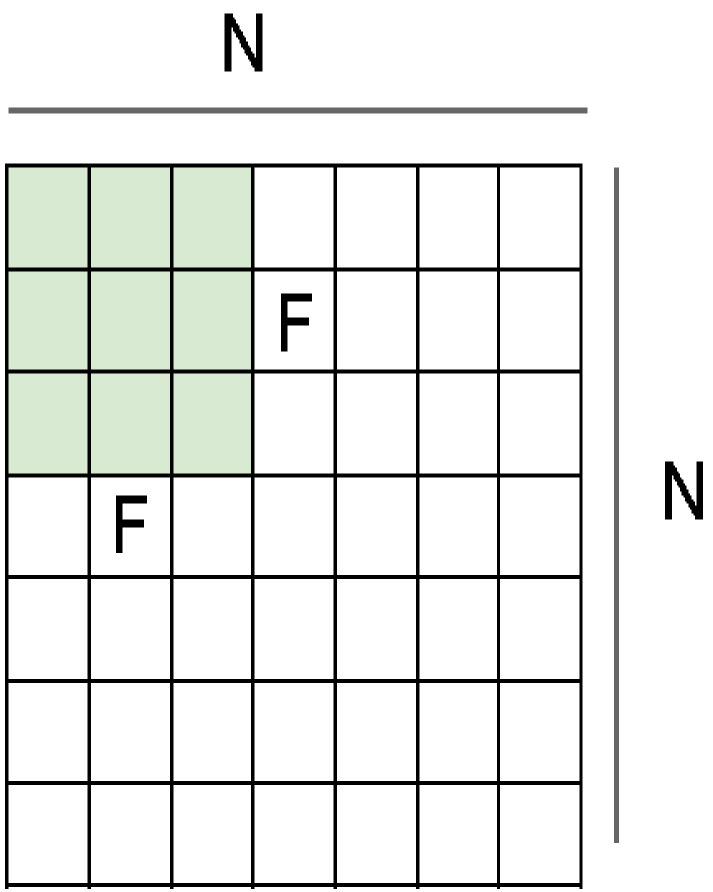
Convolutional Layer



Convolutional Layer



stride



Output size:

(N - F) / stride + 1

e.g. N = 7, F = 3:

stride 1 => (7 - 3)/1 + 1 = 5

stride 2 => (7 - 3)/2 + 1 = 3

stride 3 => (7 - 3)/3 + 1 = 2.33 :\

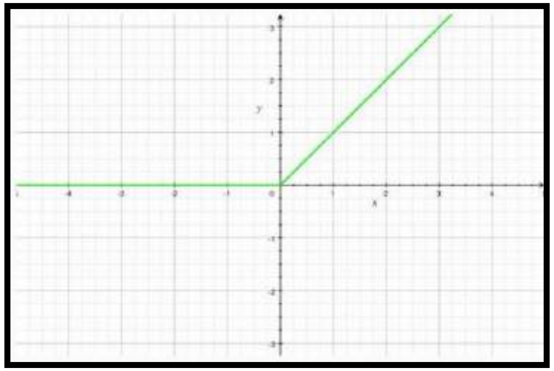
2- ReLU Layer $f(x) = \max(0,x)$

•The ReLU (short for rectified linear units) layer commonly follows the convolution layer.

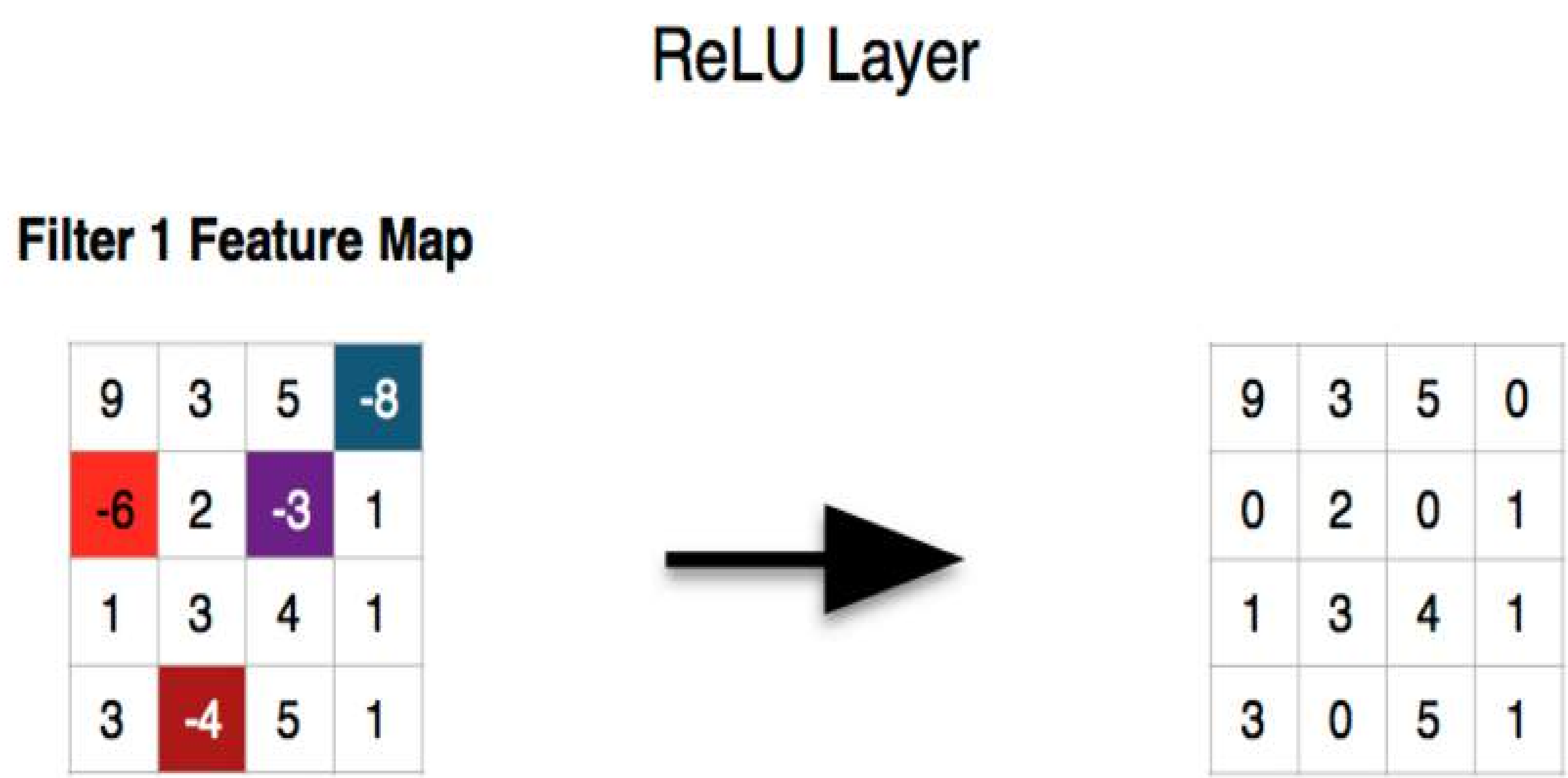
• The addition of the ReLU layer allows the neural network to account for non-linear relationships, i.e. the ReLU layer allows the convnet to account for situations in which the relationship between the pixel value inputs and the convnet output is not linear.

• the convolution operation is a linear one. $\mathbf{y = w_1x_1 + w_2x_2 + w_3x_3 + \ldots}$

• The ReLU function takes a value **x** and returns 0 if **x** is negative and **x** if **x** is positive.



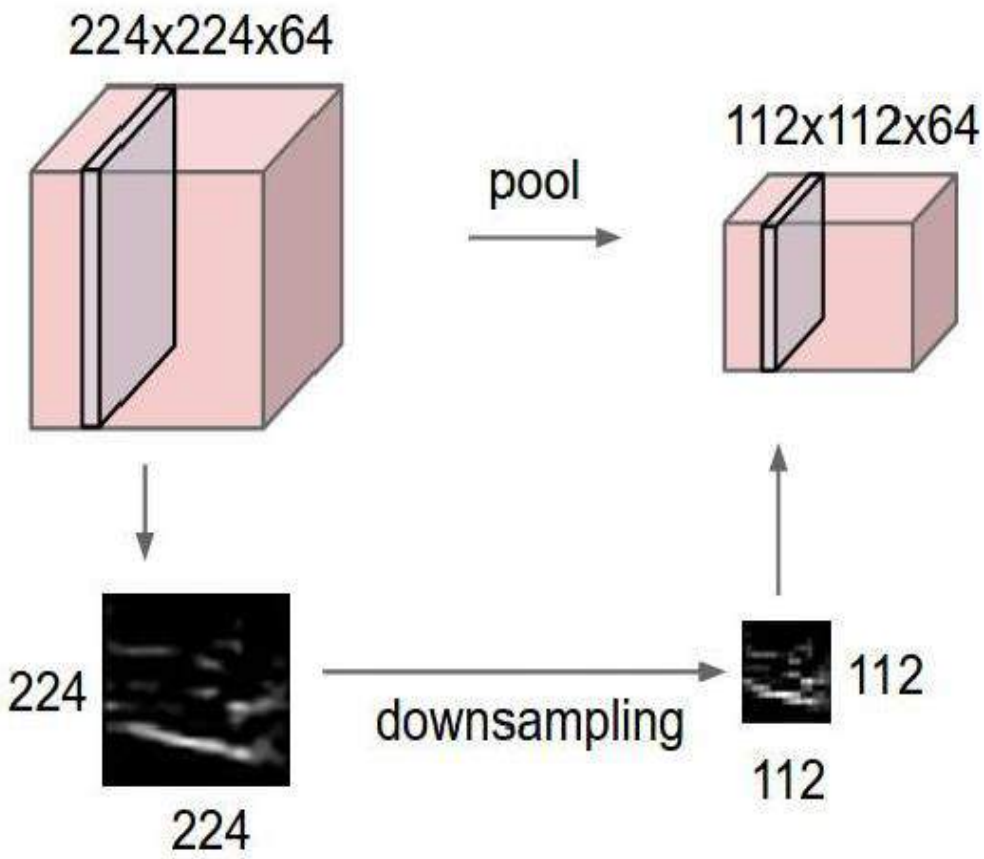
2- ReLU Layer $f(x) = \max(0,x)$



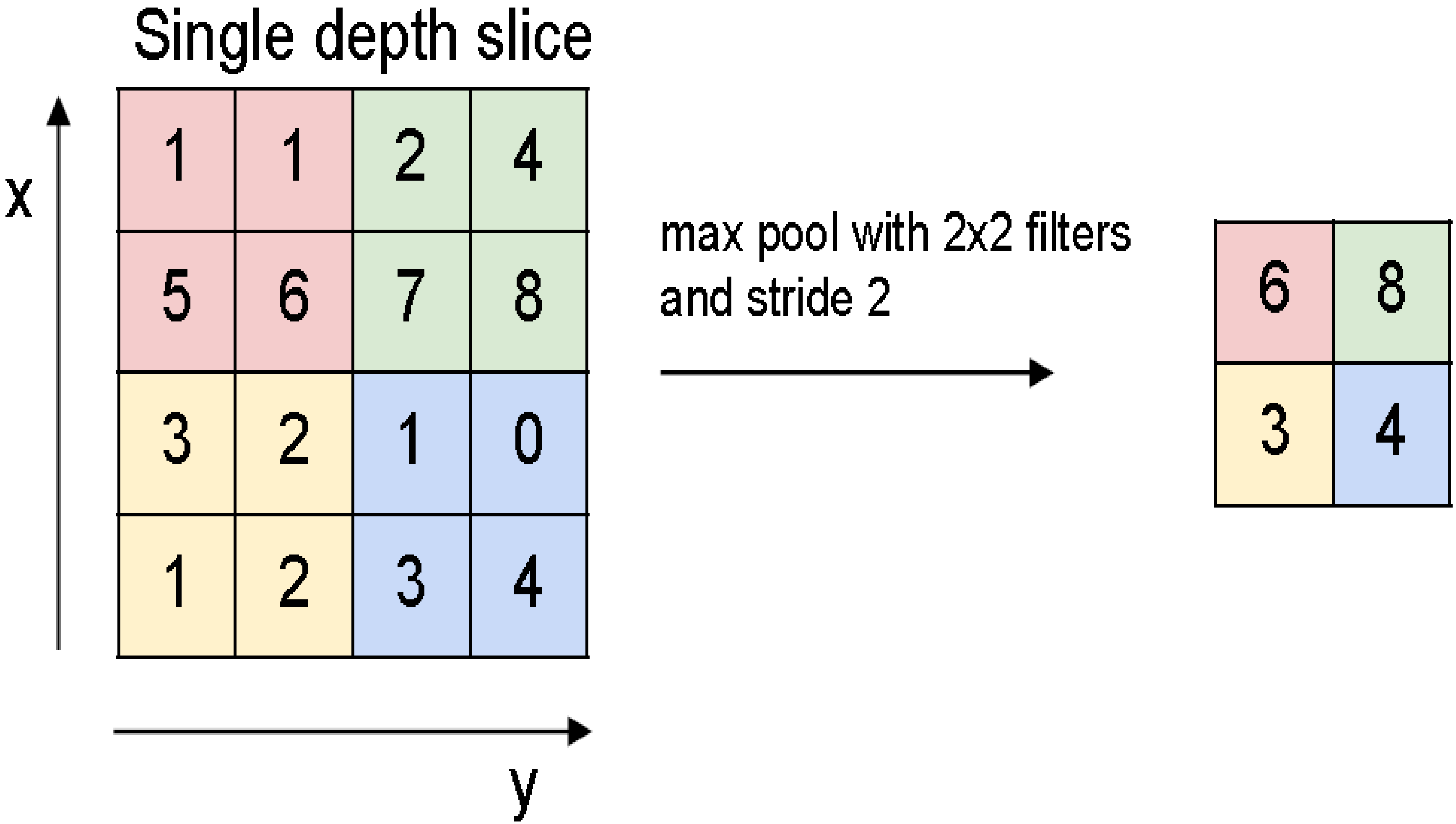
Other functions such as tanh or the sigmoid function can be used to add non-linearity to the network, but ReLU generally works better in practice.

3- Pooling layer

- the pooling layer makes the convnet less sensitive to small changes in the location of a feature
- Pooling also reduces the size of the feature map, thus simplifying computation in later layers.

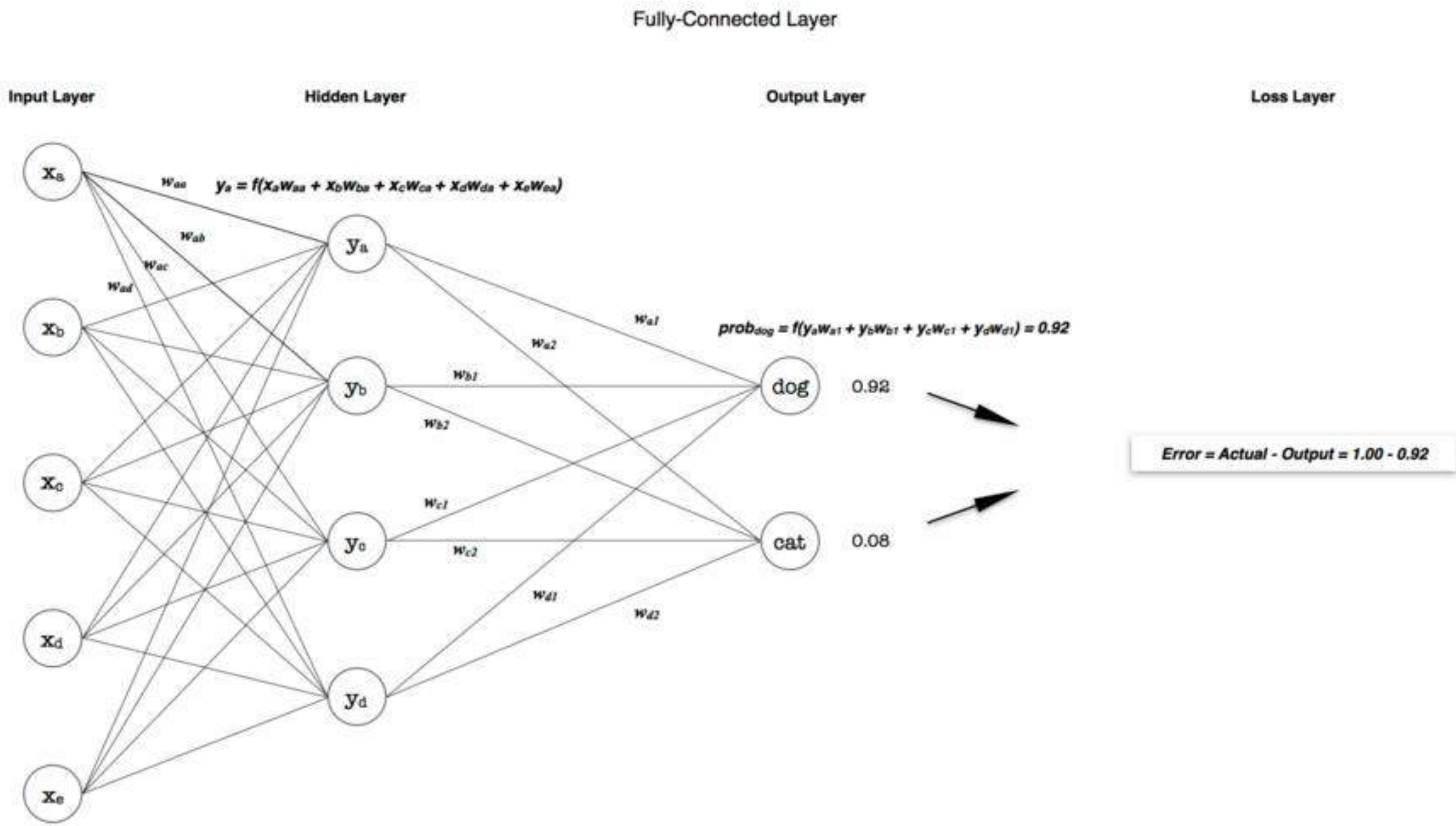


MAX POOLING

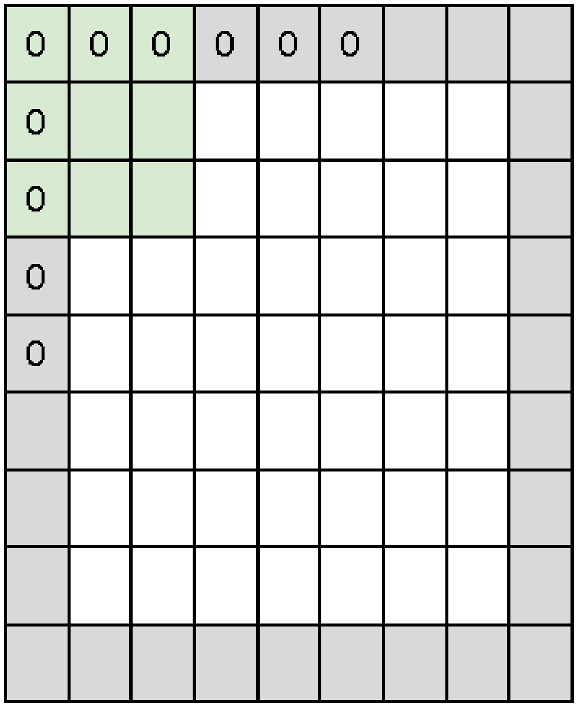

Single depth slice
x
y
1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
max pool with 2x2 filters
and stride 2
6 8
3 4

4- fully connected NN + loss layers

The fully-connected layer is where the final "decision" is made.



In practice: Common to zero pad the border



e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!

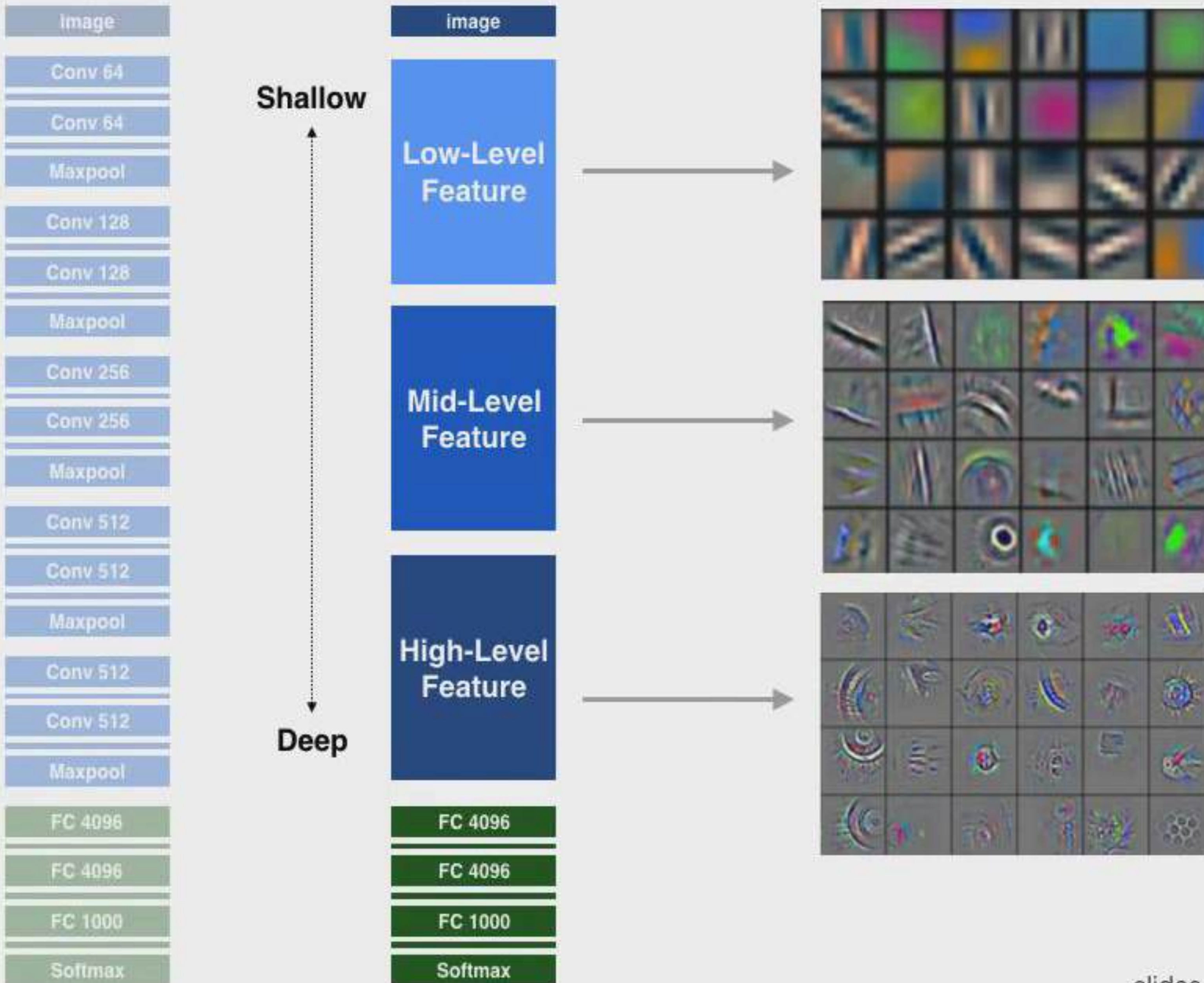
The formula for calculating the output size for any given conv layer is

$$O = \frac{(W - K + 2P)}{S} + 1$$

where O is the output height/length, W is the input height/length, K is the filter size, P is the padding, and S is the stride.

CNN

what do they learn?

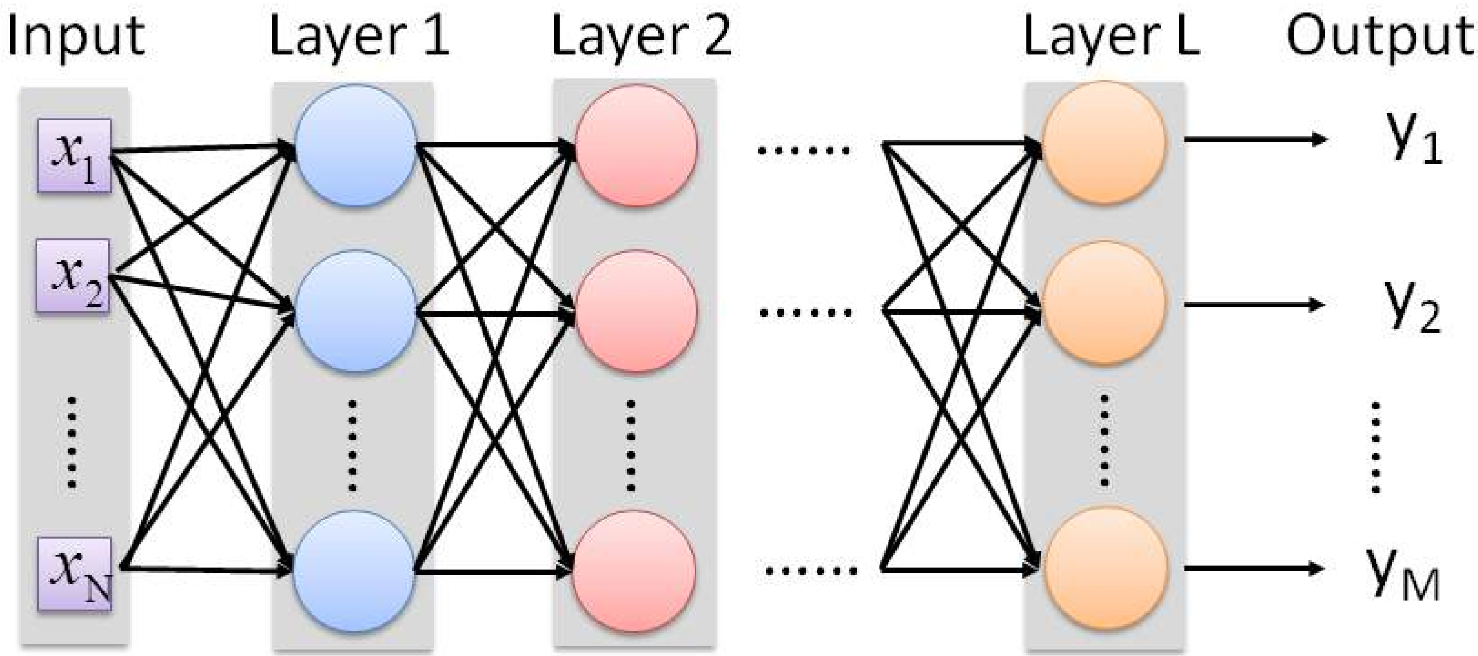


Recurrent Neural Network
RNN

Learning sequences

RNN VS Vanilla

Vanilla



- pass all input in the same time
- inputs are independent in each other
- fixed input and fixed output
- using different parameters with different layers in the network

Motivation

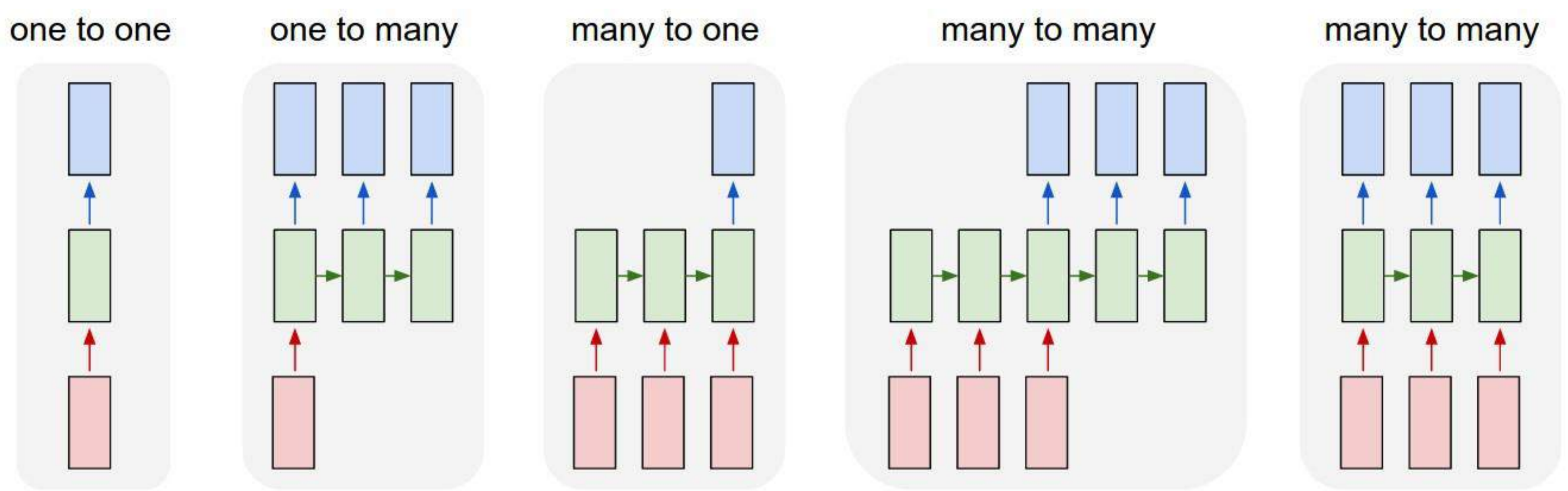


Image classification

Image captioning

Sentiment analysis

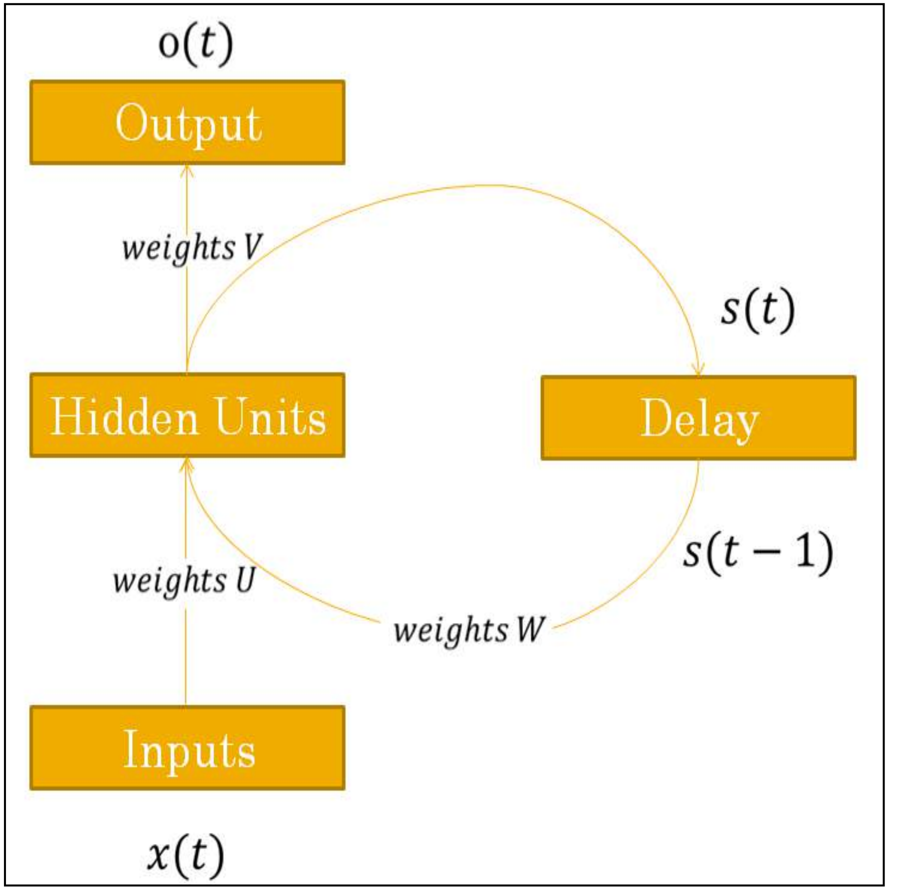
Machine translation

Synced sequence(video classification)

RNN architecture

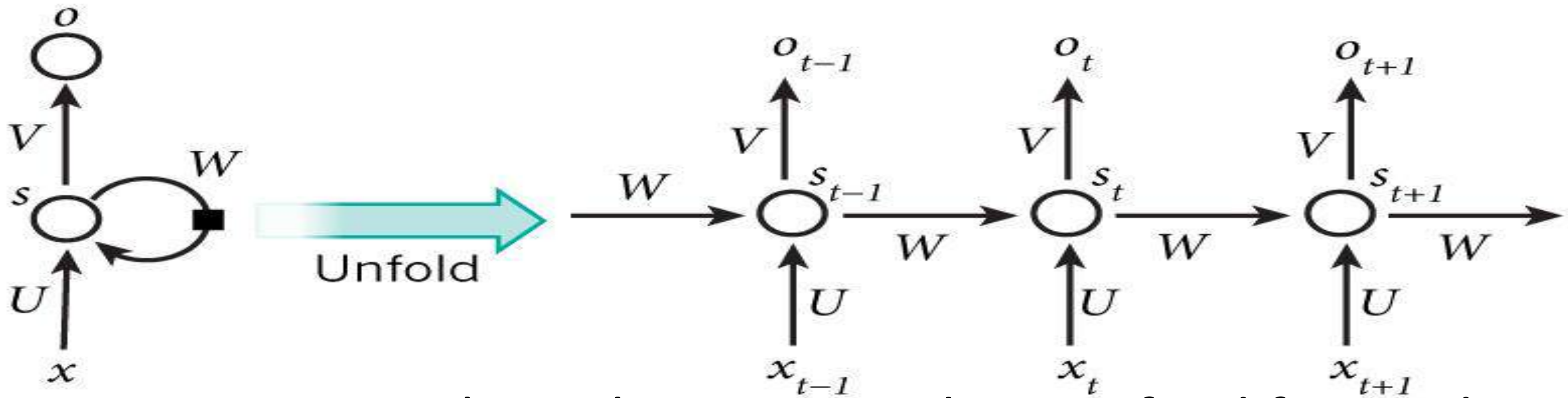
- RNNs are called *recurrent* because they perform the same task for every element of a sequence, with the output being depended on the previous computations (memory).

- Inputs ***x(t)*** outputs ***y(t)*** hidden state ***s(t)*** the memory of the network
A delay unit is introduced to hold activation until they are processed at the next step.



- The decision a recurrent net reached at time step ***t-1*** affects the decision it will reach one moment later at time step ***t***. So recurrent networks *have two sources of input*, **the present and the recent past**, which combine to determine how they respond to new data

RNN Architecture



- The recurrent network can be converted into a feed forward network by **unfolding over time**
  - The network input at time t:

$$a_h(t) = Ux(t) + Ws(t-1)$$

  - The activation of the input at time t:

$$s(t) = f_h(a_h(t))$$

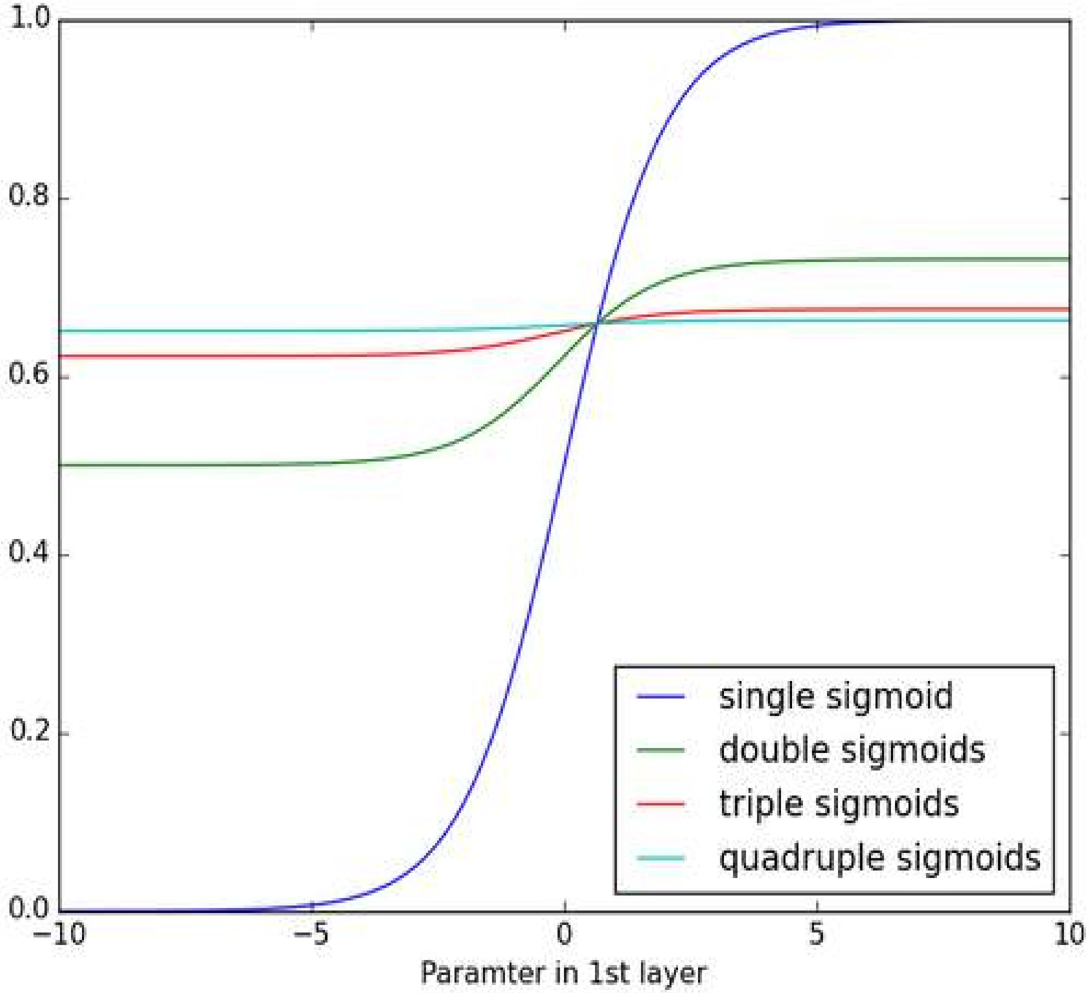
  - The network input to the output unit at time t:

$$a_o(t) = Vs(t)$$

  - The output of the network at time t is:

$$o(t) = f_o(a_o(t))$$

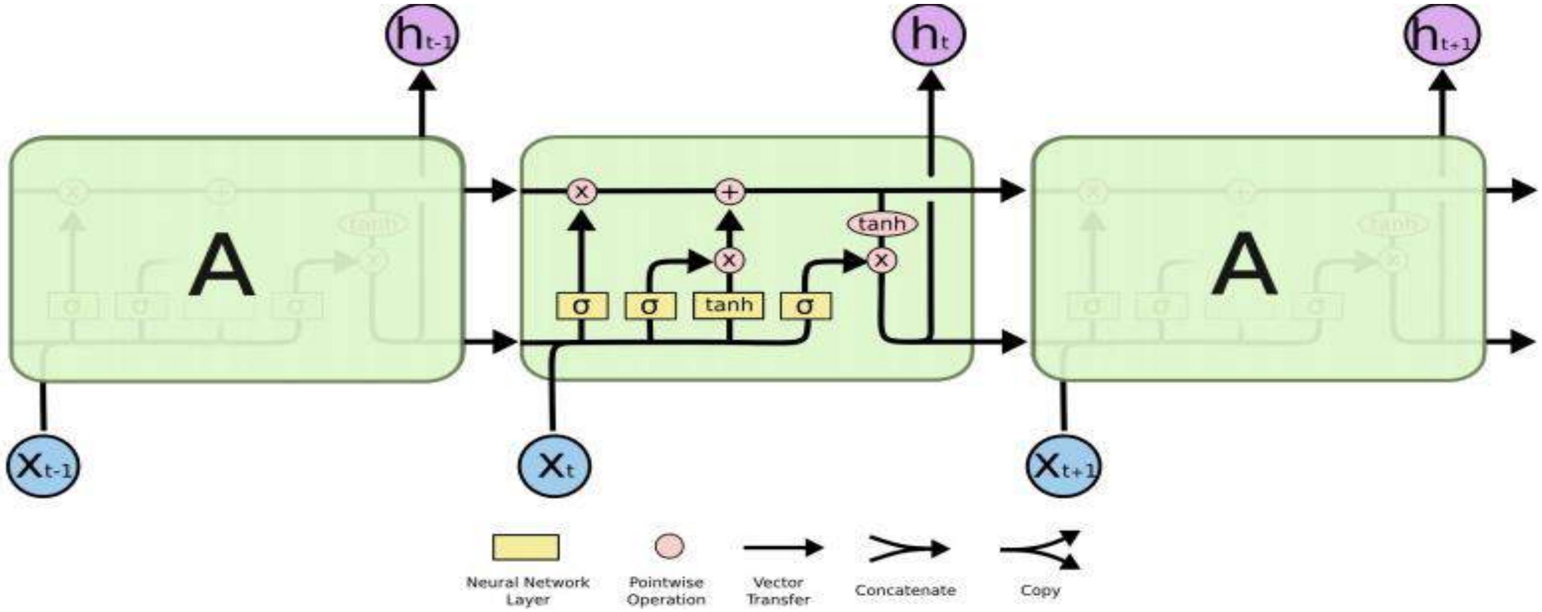
Vanishing Gradients



long-term dependencies

The key difference from regular networks is that we sum up the gradients for **W** at each time step

Recurrent NN - LSTM



The basic unit in the hidden layer of an LSTM network is a memory block, it replaces the hidden unit in a traditional RNN. A memory block contains one or more memory cell and a pair of adaptive multiplicative gating units which gates input and output to all cells in the block. Memory blocks allow cells to share the same gates thus reducing the number of parameters. Each cell has in its core a recurrently self connected linear unit called the "Constant error carousel" whose activation we call the cell state.

Natural Language Processing Tasks

1- Automatic Summarization

the process of shortening a text document with software, in order to create a summary with the major points of the original document.

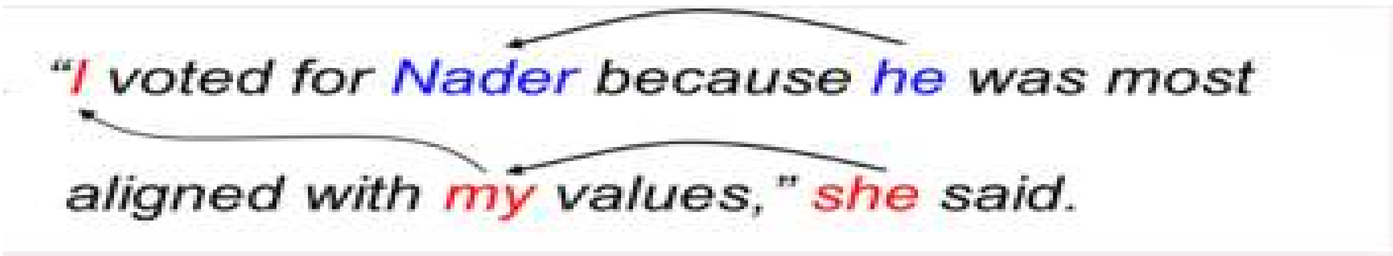
There are two methods

1-extracting sentences or parts thereof from the original text
2- generating abstract summaries.

Tools- The Python library sumy,

2- Co reference resolution

Coreference resolution is the task of finding all expressions that refer to the same entity in a text.



Tools- The Apache OpenNLP

tokenization, sentence segmentation, part-of-speech tagging, named entity extraction, chunking, parsing, and co reference resolution.

3- Named Entity Recognition

Named-entity recognition (**NER**) (also known as **entity identification**, **entity chunking** and **entity extraction**) is a subtask of information extraction that seeks to locate and classify named entities in text into pre-defined categories such as

- **person names**
- **company/organization names**
- **locations**
- **dates & time**
- **percentages**
- **monetary amounts (Currency)**
- **number**
- **Device**
- **Jop**
- **Car**
- **Cell Phone**

Tools- The Apache OpenNLP

4- Sentiment analysis

The task of finding the opinions of authors about specific entities.

Sentiment Analysis Problem

An *opinion* is a quintuple

$$(O, F, S\_P, OH, T)$$

Object

Feature

Subjectivity or Polarity classification

Opinion Holder

Time

https://github.com/Kyubyong/nlp_tasks#coreference-resolution

Natural Language Processing Tasks and Selected References

I've been working on several natural language processing tasks for a long time. One day, I felt like drawing a map of the NLP field where I earn a living. I'm sure I'm not the only person who wants to see at a glance which tasks are in NLP.

I did my best to cover as many as possible tasks in NLP, but admittedly this is far from exhaustive purely due to my lack of knowledge. And selected references are biased towards recent deep learning accomplishments. I expect these serve as a starting point when you're about to dig into the task. I'll keep updating this repo myself, but what I really hope is you collaborate on this work. Don't hesitate to send me a pull request!

Oct. 13, 2017.
by Kyubyong

Reviewed and updated by YJ Choe on Oct. 18, 2017.

Anaphora Resolution

- See Coreference Resolution

Automated Essay Scoring

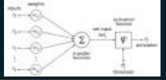
- PAPER Automatic Text Scoring Using Neural Networks
- PAPER A Neural Approach to Automated Essay Scoring
- CHALLENGE Kaggle: The Hewlett Foundation: Automated Essay Scoring
- PROJECT EASE (Enhanced AI Scoring Engine)

Automatic Speech Recognition

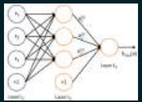
- WIKI Speech recognition
- PAPER Deep Speech 2: End-to-End Speech Recognition in English and Mandarin
- PAPER WaveNet: A Generative Model for Raw Audio
- PROJECT A TensorFlow implementation of Baidu's DeepSpeech architecture
- PROJECT Speech-to-Text-WaveNet : End-to-end sentence level English speech recognition using DeepMind's WaveNet
- CHALLENGE The 5th CHiME Speech Separation and Recognition Challenge
- DATA The 5th CHiME Speech Separation and Recognition Challenge
- DATA CSTR VCTK Corpus
- DATA LibriSpeech ASR corpus
- DATA Switchboard-1 Telephone Speech Corpus

Concepts

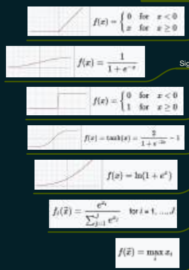
- **Unit (Neurons)**
  - A unit often refers to the activation function in a layer by which the inputs are transformed via a nonlinear activation function (for example by the logistic sigmoid function). Usually, a unit has several incoming connections and several outgoing connections.



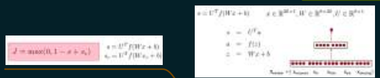
- **Input Layer**
  - Comprised of multiple Real-Valued inputs. Each input must be linearly independent from each other.
- **Hidden Layers**
  - Layers other than the input and output layers. A layer is the highest-level building block in deep learning. A layer is a container that usually receives weighted input, transforms it with a set of mostly non-linear functions and then passes these values as output to the next layer.



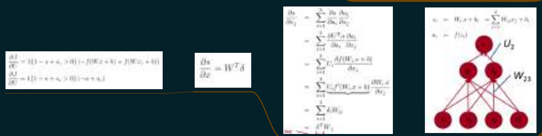
- **Batch Normalization**
  - Using mini-batches of examples, as opposed to one example at a time, is helpful in several ways. First, the gradient of the loss over a mini-batch is an estimate of the gradient over the training set, whose quality improves as the batch size increases. Second, computation over a batch can be much more efficient than m computations for individual examples, due to the parallelism afforded by the modern computing platforms.
  - With SGD, the training proceeds in steps, and at each step we consider a mini- batch x1...m of size m. The mini-batch is used to approx- imate the gradient of the loss function with respect to the parameters.
- **Cost/Loss(Min) Objective(Max) Functions**
  - Maximum Likelihood Estimation (MLE)
    - Many cost functions are the result of applying Maximum Likelihood. For instance, the Least Squares cost function can be obtained via Maximum Likelihood. Cross-Entropy is another example.
    - The likelihood of a parameter value (or vector of parameter values), θ, given outcomes x, is equal to the probability (density) assumed for those observed outcomes given those parameter values, that is $\mathcal{L}(\theta \mid x) = P(x \mid \theta)$.
    - The natural logarithm of the likelihood function, called the log-likelihood, is more convenient to work with. Because the logarithm is a monotonically increasing function, the logarithm of a function achieves its maximum value at the same points as the function itself, and hence the log-likelihood can be used in place of the likelihood in maximum likelihood estimation and related techniques.
    - In general, for a fixed set of data and underlying statistical model, the method of maximum likelihood selects the set of values of the model parameters that maximizes the likelihood function. Intuitively, this maximizes the "agreement" of the selected model with the observed data, and for discrete random variables it indeed maximizes the probability of the observed data under the resulting distribution. Maximum-likelihood estimation gives a unified approach to estimation, which is well-defined in the case of the normal distribution and many other problems.
      - $f(x_1, x_2, \ldots, x_n \mid \theta) = f(x_1 \mid \theta) \times f(x_2 \mid \theta) \times \cdots \times f(x_n \mid \theta)$
      - $\mathcal{L}(\theta; x_1, \ldots, x_n) = f(x_1, x_2, \ldots, x_n \mid \theta) = \prod_{i=1}^{n} f(x_i \mid \theta)$
      - $\ln \mathcal{L}(\theta; x_1, \ldots, x_n) = \sum_{i=1}^{n} \ln f(x_i \mid \theta)$
      - $\hat{\ell}(\theta; x) = \frac{1}{n} \sum_{i=1}^{n} \ln f(x_i \mid \theta)$
      - $\{\hat{\theta}_{mle}\} \subseteq \{\underset{\theta \in \Theta}{\arg\max}\ \hat{\ell}(\theta; x_1, \ldots, x_n)\}$
  - Cross-Entropy
    - Cross entropy can be used to define the loss function in machine learning and optimization. The true probability pi is the true label, and the given distribution qi is the predicted value of the current model.
      - $H(p,q) = \mathrm{E}_p[l_i] = \mathrm{E}_p\left[\log \frac{1}{q(x_i)}\right]$
      - $H(p,q) = \sum_{x_i} p(x_i) \log \frac{1}{q(x_i)}$
      - $H(p,q) = -\sum_x p(x) \log q(x)$
      - $L(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N} H(p_n, q_n) = -\frac{1}{N}\sum_{n=1}^{N}\left[y_n \log \hat{y}_n + (1 - y_n)\log(1 - \hat{y}_n)\right]$
      - Cross-entropy error function and logistic regression
  - Logistic
    - The logistic loss function is defined as: $V(f(\vec{x}), y) = \frac{1}{\ln 2}\ln(1 + e^{-yf(\vec{x})})$
  - Quadratic
    - The use of a quadratic loss function is common, for example when using least squares techniques. It is often more mathematically tractable than other loss functions because of the properties of variances, as well as being symmetric: an error above the target causes the same loss as the same magnitude of error below the target. If the target is t, then a quadratic loss function is: $\lambda(x) = C(t - x)^2$
  - 0-1 Loss
    - In statistics and decision theory, a frequently used loss function is the 0-1 loss function $L(\hat{y}, y) = I(\hat{y} \neq y)$
  - Hinge Loss
    - The hinge loss is a loss function used for training classifiers. For an intended output t = ±1 and a classifier score y, the hinge loss of the prediction y is defined as: $\ell(y) = \max(0, 1 - t \cdot y)$
  - Exponential
    - $E_x(y, t) = \exp\left(\frac{1}{\tau}\sum_i (y_i - t_i)^2\right)$
  - Hellinger Distance
    - It is used to quantify the similarity between two probability distributions. It is a type of f-divergence.
    - $H^2(P, Q) = \frac{1}{2}\int \left(\sqrt{\frac{dP}{d\lambda}} - \sqrt{\frac{dQ}{d\lambda}}\right)^2 d\lambda$
    - To define the Hellinger distance in terms of measure theory, let P and Q denote two probability measures that are absolutely continuous with respect to a third probability measure λ. The square of the Hellinger distance between P and Q is defined as the quantity
  - Kullback-Leibler Divengence
    - Is a measure of how one probability distribution diverges from a second expected probability distribution. Applications include characterizing the relative (Shannon) entropy in information systems, randomness in continuous time-series, and information gain when comparing statistical models of inference.
      - Discrete: $D_{\mathrm{KL}}(P \| Q) = \sum_i P(i) \log \frac{P(i)}{Q(i)}$
      - Continuous: $D_{\mathrm{KL}}(P \| Q) = \int_{-\infty}^{\infty} p \log \frac{p}{q}\, d\mu$
  - Itakura–Saito distance
    - is a measure of the difference between an original spectrum P(ω) and an approximation P^(ω) of that spectrum. Although it is not a perceptual measure, it is intended to reflect perceptual (dis)similarity.
    - $D_{IS}(P(\omega), \hat{P}(\omega)) = \frac{1}{2\pi}\int_{-\pi}^{\pi}\left[\frac{P(\omega)}{\hat{P}(\omega)} - \log\frac{P(\omega)}{\hat{P}(\omega)} - 1\right] d\omega$
  - https://stats.stackexchange.com/questions/154879/a-list-of-cost-functions-used-in-neural-networks-alongside-applications
  - https://en.wikipedia.org/wiki/Loss_functions_for_classification
- **Regularization**
  - L1 norm — Manhattan Distance
    - L1-norm is also known as least absolute deviations (LAD), least absolute errors (LAE). It is basically minimizing the sum of the absolute differences (S) between the target value and the estimated values.
    - $S = \sum_{i=1}^{n} |y_i - f(x_i)|$
  - L2 norm — Euclidean Distance
    - L2-norm is also known as least squares. It is basically minimizing the sum of the square of the differences (S) between the target value and the estimated values:
    - $S = \sum_{i=1}^{n} (y_i - f(x_i))^2$
  - Early Stopping
    - Early stopping rules provide guidance as to how many iterations can be run before the learner begins to over-fit, and stop the algorithm then.
  - Dropout
    - Is a regularization technique for reducing overfitting in neural networks by preventing complex co-adaptations on training data. It is a very efficient way of performing model averaging with neural networks. The term "dropout" refers to dropping out units (both hidden and visible) in a neural network
  - Sparse regularizer on columns
    - This regularizer defines an L2 norm on each column and an L1 norm over all columns. It can be solved by proximal methods.
    - $R(w) = \sum_{i=1}^{D} \|W\|_{2,1}$
  - Nuclear norm regularization
    - $R(w) = \|\sigma(W)\|_1$ where $\sigma(W)$ is the eigenvalues in the singular value decomposition of W.
  - Mean-constrained regularization
    - This regularizer constrains the functions learned for each task to be similar to the overall average of the functions across all tasks. This is useful for expressing prior information that each task is expected to share similarities with each other task. An example is predicting blood iron levels measured at different times of the day, where each task represents a different person.
    - $R(f_1 \cdots f_T) = \sum_{t=1}^{T} \|f_t - \frac{1}{T}\sum_{s=1}^{T} f_s\|^2_{H_k}$
  - Clustered mean-constrained regularization
    - This regularizer is similar to the mean-constrained regularizer, but instead enforces similarity between tasks within the same cluster. This can capture more complex prior information. This technique has been used to predict Netflix recommendations.
    - $R(f_1 \cdots f_T) = \sum_{r=1}^{C} \sum_{t \in I(r)} \|f_t - \frac{1}{I(r)}\sum_{s \in I(r)} f_s\|^2_{H_k}$ where $I(r)$ is a cluster of tasks.
  - Graph-based similarity
    - More general than above, similarity between tasks can be defined by a function. The regularizer encourages the model to learn similar functions for similar tasks.
    - $R(f_1 \cdots f_T) = \sum_{t,s=1, t \neq s}^{T} \|f_t - f_s\|^2 M_{ts}$ for a given symmetric similarity matrix M.
- **Activation Functions**
  - Defines the output of that node given an input or set of inputs.
  - Types
    - ReLU: $f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$
    - Sigmoid / Logistic: $f(x) = \frac{1}{1 + e^{-x}}$
    - Binary: $f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
    - Tanh: $f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$
    - Softplus: $f(x) = \ln(1 + e^x)$
    - Softmax: $f_i(\vec{x}) = \frac{e^{x_i}}{\sum_{j=1}^{J} e^{x_j}}$ for $i = 1, \ldots, J$
    - Maxout: $f(\vec{x}) = \max_i x_i$
    - Leaky ReLU, PReLU, RReLU, ELU, SELU, and others.



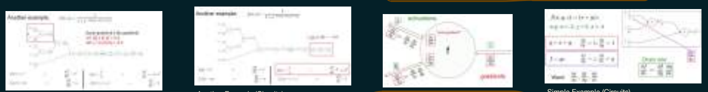
- **Backpropagation**
  - Is a method used in artificial neural networks to calculate the error contribution of each neuron after a batch of data. It calculates the gradient of the loss function. It is commonly used in the gradient descent optimization algorithm. It is also called backward propagation of errors, because the error is calculated at the output and distributed back through the network layers.
    - Neural Network taking 4 dimension vector representation of words.



  - In this method, we reuse partial derivatives computed for higher layers in lower layers, for efficiency.

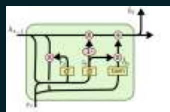


- **Intuition for backpropagation**
  - Another Example (Circuits)
  - Simple Example (Circuits)



  - Simple Example (Flowgraphs)

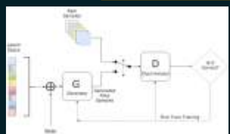
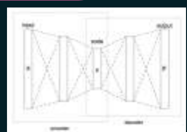
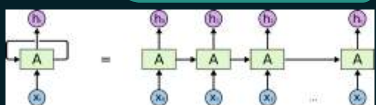
- **Learning Rate**
  - Neural networks are often trained by gradient descent on the weights. This means at each iteration we use backpropagation to calculate the derivative of the loss function with respect to each weight and subtract it from that weight.
    - However, if you actually try that, the weights will change far too much each iteration, which will make them "overcorrect" and the loss will actually increase/diverge. So in practice, people usually multiply each derivative by a small value called the "learning rate" before they subtract it from its corresponding weight.
  - Tricks
    - Simplest recipe: keep it fixed and use the same for all parameters. $\theta^{new} = \theta^{old} - \alpha \nabla_\theta J_t(\theta)$
    - Better results by allowing learning rates to decrease Options:
      - Reduce by 0.5 when validation error stops improving
      - Reduction by O(1/t) because of theoretical convergence guarantees, with hyper-parameters ϵ_0 and τ and t is iteration numbers. $\alpha = \frac{\alpha_0 \tau}{\max(t, \tau)}$
      - Better yet: No hand-set learning of rates by using AdaGrad
- **Optimization**
  - Gradient Descent
    - Is a first-order iterative optimization algorithm for finding the minimum of a function. To find a local minimum of a function using gradient descent, one takes steps proportional to the negative of the gradient (or of the approximate gradient) of the function at the current point. If instead one takes steps proportional to the positive of the gradient, one approaches a local maximum of that function; the procedure is then known as gradient ascent.
  - Stochastic Gradient Descent (SGD)
    - Gradient descent uses total gradient over all examples per update, SGD updates after only 1 or few examples: $\theta^{new} = \theta^{old} - \alpha \nabla_\theta J_t(\theta)$
  - Mini-batch Stochastic Gradient Descent (SGD)
    - Gradient descent uses total gradient over all examples per update, SGD updates after only 1 example
  - Momentum
    - Idea: Add a fraction v of previous update to current one. When the gradient keeps pointing in the same direction, this will increase the size of the steps taken towards the minimum.
  - Adagrad
    - Adaptive learning rates for each parameter
- **Weight Initialization**
  - All Zero Initialization
    - In the ideal situation, with proper data normalization it is reasonable to assume that approximately half of the weights will be positive and half of them will be negative. A reasonable-sounding idea then might be to set all the initial weights to zero, which you expect to be the "best guess" in expectation.
      - But, this turns out to be a mistake, because if every neuron in the network computes the same output, then they will also all compute the same gradients during back-propagation and undergo the exact same parameter updates. In other words, there is no source of asymmetry between neurons if their weights are initialized to be the same.
  - Initialization with Small Random Numbers
    - Thus, you still want the weights to be very close to zero, but not identically zero. In this way, you can random these neurons to small numbers which are very close to zero, and it is treated as symmetry breaking. The idea is that the neurons are all random and unique in the beginning, so they will compute distinct updates and integrate themselves as diverse parts of the full network.
      - The implementation for weights might simply drawing values from a normal distribution with zero mean, and unit standard deviation. It is also possible to use small numbers drawn from a uniform distribution, but this seems to have relatively little impact on the final performance in practice.
  - Calibrating the Variances
    - One problem with the above suggestion is that the distribution of the outputs from a randomly initialized neuron has a variance that grows with the number of inputs. It turns out that you can normalize the variance of each neuron's output to 1 by scaling its weight vector by the square root of its fan-in (i.e., its number of inputs)
      - This ensures that all neurons in the network initially have approximately the same output distribution and empirically improves the rate of convergence. The detailed derivations can be found from Page. 18 to 23 of the slides. Please note that, in the derivations, it does not consider the influence of ReLU neurons.

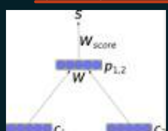
- Architectures
  - Feed Forward
    - Is an artificial neural network wherein connections between the units do not form a cycle. In this network, the information moves in only one direction, forward, from the input nodes, through the hidden nodes (if any) and to the output nodes. There are no cycles or loops in the network.
    - Kinds
      - Single-Layer Perceptron
        - The inputs are fed directly to the outputs via a series of weights. By adding an Logistic activation function to the outputs, the model is identical to a classical Logistic Regression model.
      - Multi-Layer Perceptron
        - This class of networks consists of multiple layers of computational units, usually interconnected in a feed-forward way. Each neuron in one layer has directed connections to the neurons of the subsequent layer. In many applications the units of these networks apply a sigmoid function as an activation function.
  - LSTMs
    - 

      $z_t = \sigma(W_z \cdot [h_{t-1}, x_t])$

      $r_t = \sigma(W_r \cdot [h_{t-1}, x_t])$

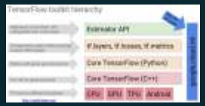
      $\tilde{h}_t = \tanh(W \cdot [r_t * h_{t-1}, x_t])$

      $h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$
    - Long short-term memory - It is a type of recurrent (RNN), allowing data to flow both forwards and backwards within the network.
      - An LSTM is well-suited to learn from experience to classify, process and predict time series given time lags of unknown size and bound between important events. Relative insensitivity to gap length gives an advantage to LSTM over alternative RNNs, hidden Markov models and other sequence learning methods in numerous applications.
  - GANs
    - GANs or Generative Adversarial Networks are a class of artificial intelligence algorithms used in unsupervised machine learning, implemented by a system of two neural networks contesting with each other in a zero-sum game framework.
      - 
  - Auto-Encoders
    - 
    - Is an artificial neural network used for unsupervised learning of efficient codings.
      - The aim of an autoencoder is to learn a representation (encoding) for a set of data, typically for the purpose of dimensionality reduction. Recently, the autoencoder concept has become more widely used for learning generative models of data.
  - Convolutional Neural Networks (CNN)
    - 
    - They have applications in image and video recognition, recommender systems and natural language processing.
      - Pooling
      - Convolution
      - Subsampling
  - RNNs (Recurrent)
    - 

    - Is a class of artificial neural network where connections between units form a directed cycle. This allows it to exhibit dynamic temporal behavior. Unlike feedforward neural networks, RNNs can use their internal memory to process arbitrary sequences of inputs.
    - This makes them applicable to tasks such as unsegmented, connected handwriting recognition or speech recognition.
  - RNNs (Recursive)
    - Is a kind of deep neural network created by applying the same set of weights recursively over a structure, to produce a structured prediction over variable-size input structures, or a scalar prediction on it, by traversing a given structure in topological order.
      - 

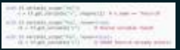
      - RNNs have been successful for instance in learning sequence and tree structures in natural language processing, mainly phrase and sentence continuous representations based on word embedding.
- Strategy
  - 1. Select Network Structure appropriate for problem
    - Structure: Single words, fixed windows, sentence based, document level; bag of words, recursive vs. recurrent, CNN
    - Nonlinearity (Activation Functions)
  - 2. Check for implementation bugs with gradient checks
    - 1. Implement your gradient
    - 2. Implement a finite difference computation by looping through the parameters of your network, adding and subtracting a small epsilon (~10-4) and estimate derivatives
      - $f'(\theta) \approx \frac{J(\theta^{(i+)}) - J(\theta^{(i-)})}{2\epsilon}$ $\quad \theta^{(i+)} = \theta + \epsilon \times e_i$
    - 3. Compare the two and make sure they are almost the same
    - Using Gradient Checks
      - If you gradient fails and you don't know why?
        - Simplify your model until you have no bug!
        - What now? Create a very tiny synthetic model and dataset
      - Example: Start from simplest model then go to what you want:
        - Only softmax on fixed input
        - Backprop into word vectors and softmax
        - Add single unit single hidden layer
        - Add multi unit single layer
        - Add second layer single unit, add multiple units, bias • Add one softmax on top, then two softmax layers
        - Add bias
  - 3. Parameter initialization
    - Initialize hidden layer biases to 0 and output (or reconstruction) biases to optimal value if weights were 0 (e.g., mean target or inverse sigmoid of mean target).
    - Initialize weights ~ Uniform(−r, r), r inversely proportional to fan-in (previous layer size) and fan-out (next layer size):
      - $\sqrt{6/(\text{fan-in} + \text{fan-out})}$
  - 4. Optimization
    - Gradient Descent
      - Is a first-order iterative optimization algorithm for finding the minimum of a function. To find a local minimum of a function using gradient descent, one takes steps proportional to the negative of the gradient (or of the approximate gradient) of the function at the current point. If instead one takes steps proportional to the positive of the gradient, one approaches a local maximum of that function; the procedure is then known as gradient ascent.
    - Stochastic Gradient Descent (SGD)
      - Gradient descent uses total gradient over all examples per update, SGD updates after only 1 or few examples:
        - $\theta^{new} = \theta^{old} - \alpha \nabla_\theta J_t(\theta)$
      - Ordinary gradient descent as a batch method is very slow, should never be used. Use 2nd order batch method such as L-BFGS.
      - On large datasets, SGD usually wins over all batch methods. On smaller datasets L-BFGS or Conjugate Gradients win. Large-batch L-BFGS extends the reach of L-BFGS [Le et al. ICML 2011].
    - Mini-batch Stochastic Gradient Descent (SGD)
      - Gradient descent uses total gradient over all examples per update, SGD updates after only 1 example
      - Most commonly used now, Size of each mini batch B: 20 to 1000
        - $\theta^{new} = \theta^{old} - \alpha \nabla_\theta J_{t:t+B}(\theta)$
      - Helps parallelizing any model by computing gradients for multiple elements of the batch in parallel
    - Momentum
      - Idea: Add a fraction v of previous update to current one. When the gradient keeps pointing in the same direction, this will increase the size of the steps taken towards the minimum.
      - Reduce global learning rate when using a lot of momentum
      - Update Rule
        - $$\begin{aligned} v &= \mu v - \alpha \nabla_\theta J_t(\theta) \\ \theta^{new} &= \theta^{old} + v \end{aligned}$$
          - v is initialized at 0
          - Momentum often increased after some epochs (0.5 à 0.99)
    - Adagrad
      - Adaptive learning rates for each parameter!
      - Learning rate is adapting differently for each parameter and rare parameters get larger updates than frequently occurring parameters. Word vectors!
        - Let $g_{t,i} = \frac{\partial}{\partial \theta_i^t} J_t(\theta)$, then: $\theta_{t,i} = \theta_{t-1,i} - \frac{\alpha}{\sqrt{\sum_{\tau=1}^{t} g_{\tau,i}^2}} g_{t,i}$
  - 5. Check if the model is powerful enough to overfit
    - If not, change model structure or make model "larger"
    - If you can overfit: Regularize to prevent overfitting:
      - Simple first step: Reduce model size by lowering number of units and layers and other parameters
      - Standard L1 or L2 regularization on weights
      - Early Stopping: Use parameters that gave best validation error
      - Sparsity constraints on hidden activations, e.g., add to cost:
      - Dropout
        - Training time: at each instance of evaluation (in online SGD-training), randomly set 50% of the inputs to each neuron to 0
        - Test time: halve the model weights (now twice as many) This prevents feature co-adaptation: A feature cannot only be useful in the presence of particular other features
        - In a single layer: A kind of middle-ground between Naïve Bayes (where all feature weights are set independently) and logistic regression models (where all weights are set in the context of all others)
        - Can be thought of as a form of model bagging
        - It also acts as a strong regularizer

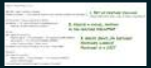
Tensorflow

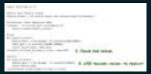


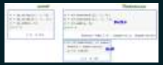
- Intuition
  - TensorFlow is a deep learning library recently open-sourced by Google. It provides primitives for defining functions on tensors and automatically computing their derivatives, expressed as a graph.
  - The Tensorflow Graph is build to contain all placeholders for X and y, all variables for W's and b's, all mathematical operations, the cost function, and the optimisation procedure. Then, at runtime, the values for the data are fed into that Graph, by placing the data batches in the placeholders and running the Graph.

    
  - Each node in the Graph can then be connected to each other node over the network, and thus running Tensorflow models can be parallelised.
- Tensorboard
  - TensorFlow has some neat built-in visualization tools (TensorBoard).
- Phases
  - 1. Construction
    - Assembles a computational graph
    - The computation graph has no numerical value until evaluated.
    - All computations add nodes to global default graph
  - 2. Execution
    - A Session object encapsulates the environment in which Tensor objects are evaluated
    - Uses a session to execute ops in the graph
    - Declared variables must be initialised before they have values.
    - When you train a model you use variables to hold and update parameters. Variables are in-memory buffers containing tensors.
- Main Components
  - Variables
    - Stateful nodes that output their current value, their state is retained across multiple executions of the graph.
    - Mostly Parameters we're interested in tuning, such as Weights (W) and Biases (b).
    - Sharing
      - Variables can be shared by Explicitly passing tf.Variable objects around, or...
      - Implicitly wrapping tf.Variable objects within tf.variable_scope objects.
        - Scopes
          - tf.variable_scope()
            - Provides simple name spacing to avoid cases when querying

              
          - tf.get_variable()
            - Creates/Access variables from a variable scope
  - Placeholders
    - Nodes whose value is fed at execution time.
    - Inputs, Features (X) and Labels (y)
  - Mathematical Operations
    - MatMul, Add, ReLU, etc.
  - Graph
    - Nodes
      - They are Operations, containing any number of inputs and outputs.
    - Edges
      - The tensors that flow between the nodes.
  - Session
    - It a binding to a particular execution context: CPU, GPU.
    - Running a Session
      - Inputs
        - Fetches
          - List of graph nodes. Returns the output of these nodes.
        - Feeds
          - Dictionary mapping from graph nodes to concrete values.
          - Specified the value of each graph node given in the dictionary.
- Comparison to Numpy
  - Does lazy evaluation. Need to build the graph, and then run it in a session.

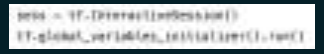
    

    
- Packages
  - tf
    - Main Steps
      - 1. Create the Model

        
      - 2. Define Target

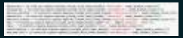
        
      - 3. Define Loss function and Optimizer

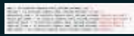
        
      - 4. Define the Session and Initialise Variables

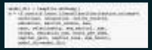
        
      - 5. Train the Model

        
      - 6. Test Trained Model

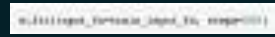
        
  - tf.estimator
    - TensorFlow's high-level machine learning API (tf.estimator) makes it easy to configure, train, and evaluate a variety of machine learning models.
      - tf.estimator.LinearClassifier: Constructs a linear classification model.
      - tf.estimator.LinearRegressor: Constructs a linear regression model.
      - tf.estimator.DNNClassifier: Construct a neural network classification model.
      - tf.estimator.DNNRegressor: Construct a neural network regression model.
      - tf.estimator.DNNLinearCombinedClassifier: Construct a neural network and linear combined classification model.
      - tf.estimator.DNNRegressor: Construct a neural network and linear combined regression model.
    - Main Steps
      - 1. Define Feature Columns
        - FeatureColumns are the primary way of encoding features for pre-canned tf.learn Estimators.
          - Categorical

            
          - Numerical

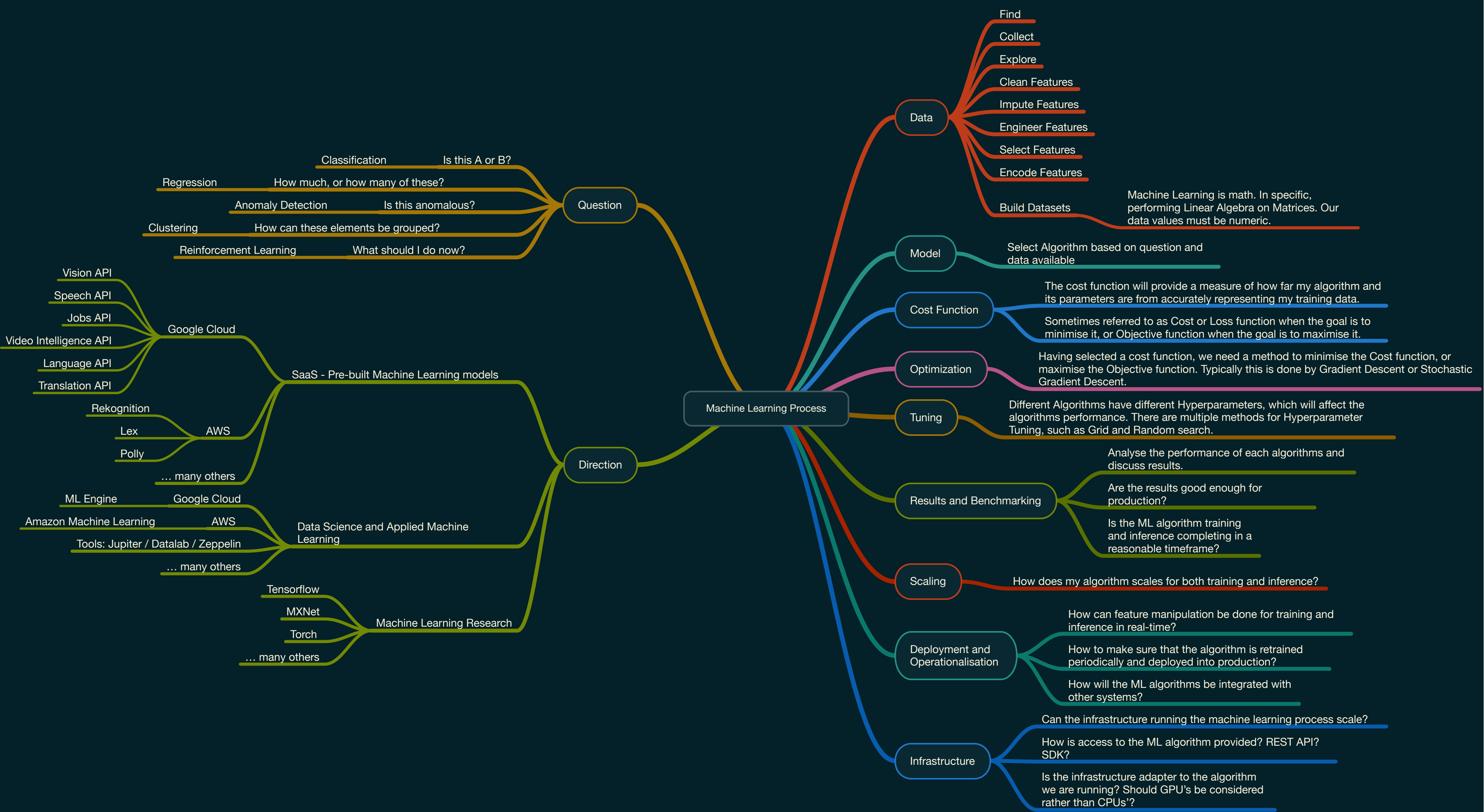
            
        - When using FeatureColumns with tf.learn models, the type of feature column you should choose depends on the feature type and the model type.
          - Continuous Features
            - Can be represented by real_valued_column
          - Categorical Features
            - Can be represented by any sparse_column_with_* column (sparse_column_with_keys, sparse_column_with_vocabulary_file, sparse_column_with_hash_bucket, sparse_column_with_integerized_feature
      - 2. Define your Layers, or use a prebuilt model

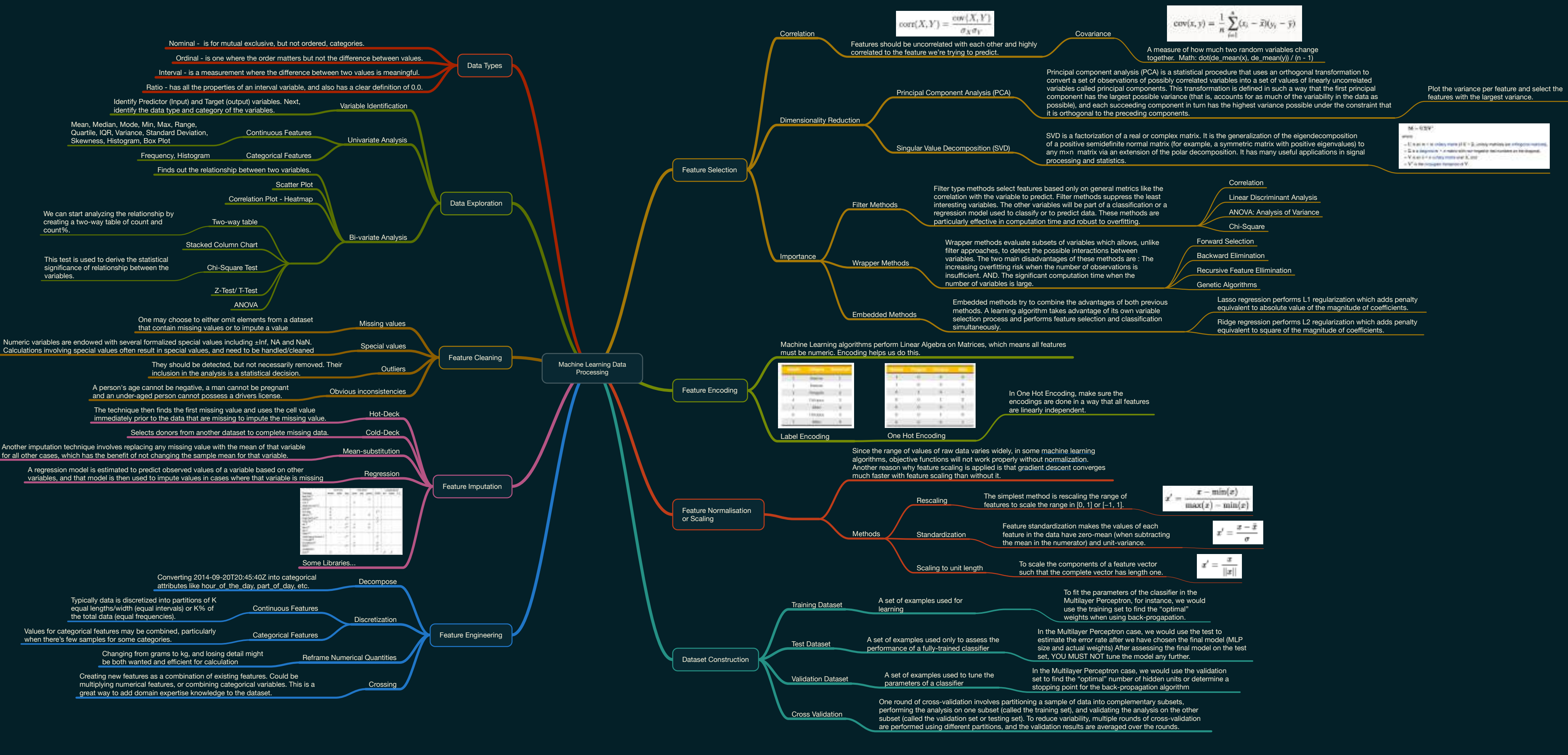
        
        - Using a pre-built Logistic Regression Classifier
      - 3. Write the input_fn function

        
        - This function holds the actual data (features and labels). Features is a python dictionary.
      - 4. Train the model

        
        - Using the fit function, on the input_fn. Notice that the feature columns are fed to the model as arguments.
      - 5. Predict and Evaluate

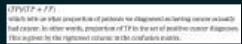
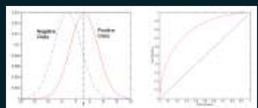
        
        - Using the eval_input_fn defined previously.


Machine Learning Process
Question
Classification
Is this A or B?
Regression
How much, or how many of these?
Anomaly Detection
Is this anomalous?
Clustering
How can these elements be grouped?
Reinforcement Learning
What should I do now?
Direction
SaaS - Pre-built Machine Learning models
Google Cloud
Vision API
Speech API
Jobs API
Video Intelligence API
Language API
Translation API
AWS
Rekognition
Lex
Polly
… many others
Data Science and Applied Machine Learning
Google Cloud
ML Engine
AWS
Amazon Machine Learning
Tools: Jupiter / Datalab / Zeppelin
… many others
Machine Learning Research
Tensorflow
MXNet
Torch
… many others
Data
Find
Collect
Explore
Clean Features
Impute Features
Engineer Features
Select Features
Encode Features
Build Datasets
Machine Learning is math. In specific, performing Linear Algebra on Matrices. Our data values must be numeric.
Model
Select Algorithm based on question and data available
Cost Function
The cost function will provide a measure of how far my algorithm and its parameters are from accurately representing my training data.
Sometimes referred to as Cost or Loss function when the goal is to minimise it, or Objective function when the goal is to maximise it.
Optimization
Having selected a cost function, we need a method to minimise the Cost function, or maximise the Objective function. Typically this is done by Gradient Descent or Stochastic Gradient Descent.
Tuning
Different Algorithms have different Hyperparameters, which will affect the algorithms performance. There are multiple methods for Hyperparameter Tuning, such as Grid and Random search.
Results and Benchmarking
Analyse the performance of each algorithms and discuss results.
Are the results good enough for production?
Is the ML algorithm training and inference completing in a reasonable timeframe?
Scaling
How does my algorithm scales for both training and inference?
Deployment and Operationalisation
How can feature manipulation be done for training and inference in real-time?
How to make sure that the algorithm is retrained periodically and deployed into production?
How will the ML algorithms be integrated with other systems?
Infrastructure
Can the infrastructure running the machine learning process scale?
How is access to the ML algorithm provided? REST API? SDK?
Is the infrastructure adapter to the algorithm we are running? Should GPU's be considered rather than CPUs'?



Machine Learning Data Processing
Data Types
Nominal - is for mutual exclusive, but not ordered, categories.
Ordinal - is one where the order matters but not the difference between values.
Interval - is a measurement where the difference between two values is meaningful.
Ratio - has all the properties of an interval variable, and also has a clear definition of 0.0.
Data Exploration
Variable Identification
Identify Predictor (Input) and Target (output) variables. Next, identify the data type and category of the variables.
Univariate Analysis
Continuous Features
Mean, Median, Mode, Min, Max, Range, Quartile, IQR, Variance, Standard Deviation, Skewness, Histogram, Box Plot
Categorical Features
Frequency, Histogram
Bi-variate Analysis
Finds out the relationship between two variables.
Scatter Plot
Correlation Plot - Heatmap
Two-way table
We can start analyzing the relationship by creating a two-way table of count and count%.
Stacked Column Chart
Chi-Square Test
This test is used to derive the statistical significance of relationship between the variables.
Z-Test/ T-Test
ANOVA
Feature Cleaning
Missing values
One may choose to either omit elements from a dataset that contain missing values or to impute a value
Special values
Numeric variables are endowed with several formalized special values including ±Inf, NA and NaN. Calculations involving special values often result in special values, and need to be handled/cleaned
Outliers
They should be detected, but not necessarily removed. Their inclusion in the analysis is a statistical decision.
Obvious inconsistencies
A person's age cannot be negative, a man cannot be pregnant and an under-aged person cannot possess a drivers license.
Feature Imputation
Hot-Deck
The technique then finds the first missing value and uses the cell value immediately prior to the data that are missing to impute the missing value.
Cold-Deck
Selects donors from another dataset to complete missing data.
Mean-substitution
Another imputation technique involves replacing any missing value with the mean of that variable for all other cases, which has the benefit of not changing the sample mean for that variable.
Regression
A regression model is estimated to predict observed values of a variable based on other variables, and that model is then used to impute values in cases where that variable is missing
Some Libraries...
Feature Engineering
Decompose
Converting 2014-09-20T20:45:40Z into categorical attributes like hour_of_the_day, part_of_day, etc.
Discretization
Continuous Features
Typically data is discretized into partitions of K equal lengths/width (equal intervals) or K% of the total data (equal frequencies).
Categorical Features
Values for categorical features may be combined, particularly when there's few samples for some categories.
Reframe Numerical Quantities
Changing from grams to kg, and losing detail might be both wanted and efficient for calculation
Crossing
Creating new features as a combination of existing features. Could be multiplying numerical features, or combining categorical variables. This is a great way to add domain expertise knowledge to the dataset.
Feature Selection
Correlation
$\mathrm{corr}(X,Y) = \frac{\mathrm{cov}(X,Y)}{\sigma_X \sigma_Y}$
Features should be uncorrelated with each other and highly correlated to the feature we're trying to predict.
Covariance
$\mathrm{cov}(x,y) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})$
A measure of how much two random variables change together. Math: dot(de_mean(x), de_mean(y)) / (n - 1)
Dimensionality Reduction
Principal Component Analysis (PCA)
Principal component analysis (PCA) is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components. This transformation is defined in such a way that the first principal component has the largest possible variance (that is, accounts for as much of the variability in the data as possible), and each succeeding component in turn has the highest variance possible under the constraint that it is orthogonal to the preceding components.
Plot the variance per feature and select the features with the largest variance.
Singular Value Decomposition (SVD)
SVD is a factorization of a real or complex matrix. It is the generalization of the eigendecomposition of a positive semidefinite normal matrix (for example, a symmetric matrix with positive eigenvalues) to any m×n matrix via an extension of the polar decomposition. It has many useful applications in signal processing and statistics.
Importance
Filter Methods
Filter type methods select features based only on general metrics like the correlation with the variable to predict. Filter methods suppress the least interesting variables. The other variables will be part of a classification or a regression model used to classify or to predict data. These methods are particularly effective in computation time and robust to overfitting.
Correlation
Linear Discriminant Analysis
ANOVA: Analysis of Variance
Chi-Square
Wrapper Methods
Wrapper methods evaluate subsets of variables which allows, unlike filter approaches, to detect the possible interactions between variables. The two main disadvantages of these methods are : The increasing overfitting risk when the number of observations is insufficient. AND. The significant computation time when the number of variables is large.
Forward Selection
Backward Elimination
Recursive Feature Ellimination
Genetic Algorithms
Embedded Methods
Embedded methods try to combine the advantages of both previous methods. A learning algorithm takes advantage of its own variable selection process and performs feature selection and classification simultaneously.
Lasso regression performs L1 regularization which adds penalty equivalent to absolute value of the magnitude of coefficients.
Ridge regression performs L2 regularization which adds penalty equivalent to square of the magnitude of coefficients.
Feature Encoding
Machine Learning algorithms perform Linear Algebra on Matrices, which means all features must be numeric. Encoding helps us do this.
Label Encoding
One Hot Encoding
In One Hot Encoding, make sure the encodings are done in a way that all features are linearly independent.
Feature Normalisation or Scaling
Since the range of values of raw data varies widely, in some machine learning algorithms, objective functions will not work properly without normalization. Another reason why feature scaling is applied is that gradient descent converges much faster with feature scaling than without it.
Methods
Rescaling
The simplest method is rescaling the range of features to scale the range in [0, 1] or [−1, 1].
$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$
Standardization
Feature standardization makes the values of each feature in the data have zero-mean (when subtracting the mean in the numerator) and unit-variance.
$x' = \frac{x - \bar{x}}{\sigma}$
Scaling to unit length
To scale the components of a feature vector such that the complete vector has length one.
$x' = \frac{x}{\|x\|}$
Dataset Construction
Training Dataset
A set of examples used for learning
To fit the parameters of the classifier in the Multilayer Perceptron, for instance, we would use the training set to find the "optimal" weights when using back-progapation.
Test Dataset
A set of examples used only to assess the performance of a fully-trained classifier
In the Multilayer Perceptron case, we would use the test to estimate the error rate after we have chosen the final model (MLP size and actual weights) After assessing the final model on the test set, YOU MUST NOT tune the model any further.
Validation Dataset
A set of examples used to tune the parameters of a classifier
In the Multilayer Perceptron case, we would use the validation set to find the "optimal" number of hidden units or determine a stopping point for the back-propagation algorithm
Cross Validation
One round of cross-validation involves partitioning a sample of data into complementary subsets, performing the analysis on one subset (called the training set), and validating the analysis on the other subset (called the validation set or testing set). To reduce variability, multiple rounds of cross-validation are performed using different partitions, and the validation results are averaged over the rounds.

Machine Learning Concepts

- **Motivation**
  - Prediction
    - When we are interested mainly in the predicted variable as a result of the inputs, but not on the each way of the inputs affect the prediction. In a real estate example, Prediction would answer the question of: Is my house over or under valued? Non-linear models are very good at these sort of predictions, but not great for inference because the models are much less interpretable.
  - Inference
    - When we are interested in the way each one of the inputs affect the prediction. In a real estate example, Prediction would answer the question of: How much would my house cost if it had a view of the sea? Linear models are more suited for inference because the models themselves are easier to understand than their non-linear counterparts.
- **Performance Analysis**
  - Confusion Matrix
    - 
  - Accuracy
    - Fraction of correct predictions, not reliable as skewed when the data set is unbalanced (that is, when the number of samples in different classes vary greatly)
  - f1 score
    - Precision
      - 
      - Out of all the examples the classifier labeled as positive, what fraction were correct?
    - Recall
      - 
      - Out of all the positive examples there were, what fraction did the classifier pick up?
    - Harmonic Mean of Precision and Recall: (2 * p * r / (p + r))
  - ROC Curve - Receiver Operating Characteristics
    - 
    - True Positive Rate (Recall / Sensitivity) vs False Positive Rate (1-Specificity)
  - Bias-Variance Tradeoff
    - Bias refers to the amount of error that is introduced by approximating a real-life problem, which may be extremely complicated, by a simple model. If Bias is high, and/or if the algorithm performs poorly even on your training data, try adding more features, or a more flexible model.
    - Variance is the amount our model's prediction would change when using a different training data set. High: Remove features, or obtain more data.
  - Goodness of Fit = R^2
    - 1.0 - sum_of_squared_errors / total_sum_of_squares(y)
  - Mean Squared Error (MSE)
    - $$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{f}(x_i))^2,$$
    - The mean squared error (MSE) or mean squared deviation (MSD) of an estimator (of a procedure for estimating an unobserved quantity) measures the average of the squares of the errors or deviations—that is, the difference between the estimator and what is estimated.
  - Error Rate
    - $$\frac{1}{n}\sum_{i=1}^{n} I(y_i \neq \hat{y}_i).$$
    - The proportion of mistakes made if we apply out estimate model function the the training observations in a classification setting.
- **Tuning**
  - Cross-validation
    - 
    - One round of cross-validation involves partitioning a sample of data into complementary subsets, performing the analysis on one subset (called the training set), and validating the analysis on the other subset (called the validation set or testing set). To reduce variability, multiple rounds of cross-validation are performed using different partitions, and the validation results are averaged over the rounds.
    - Methods
      - Leave-p-out cross-validation
      - Leave-one-out cross-validation
      - k-fold cross-validation
      - Holdout method
      - Repeated random sub-sampling validation
  - Hyperparameters
    - Grid Search
      - The traditional way of performing hyperparameter optimization has been grid search, or a parameter sweep, which is simply an exhaustive searching through a manually specified subset of the hyperparameter space of a learning algorithm. A grid search algorithm must be guided by some performance metric, typically measured by cross-validation on the training set or evaluation on a held-out validation set.
    - Random Search
      - Since grid searching is an exhaustive and therefore potentially expensive method, several alternatives have been proposed. In particular, a randomized search that simply samples parameter settings a fixed number of times has been found to be more effective in high-dimensional spaces than exhaustive search.
    - Gradient-based optimization
      - For specific learning algorithms, it is possible to compute the gradient with respect to hyperparameters and then optimize the hyperparameters using gradient descent. The first usage of these techniques was focused on neural networks. Since then, these methods have been extended to other models such as support vector machines or logistic regression.
  - Early Stopping (Regularization)
    - Early stopping rules provide guidance as to how many iterations can be run before the learner begins to over-fit, and stop the algorithm then.
  - Overfitting
    - When a given method yields a small training MSE (or cost), but a large test MSE (or cost), we are said to be overfitting the data. This happens because our statistical learning procedure is trying too hard to find pattens in the data, that might be due to random chance, rather than a property of our function. In other words, the algorithms may be learning the training data too well. If model underfits, try removing some features, decreasing degrees of freedom, or adding more data.
  - Underfitting
    - Opposite of Overfitting. Underfitting occurs when a statistical model or machine learning algorithm cannot capture the underlying trend of the data. It occurs when the model or algorithm does not fit the data enough. Underfitting occurs if the model or algorithm shows low variance but high bias (to contrast the opposite, overfitting from high variance and low bias). It is often a result of an excessively simple model.
  - Bootstrap
    - Test that applies Random Sampling with Replacement of the available data, and assigns measures of accuracy (bias, variance, etc.) to sample estimates.
  - Bagging
    - An approach to ensemble learning that is based on bootstrapping. Shortly, given a training set, we produce multiple different training sets (called bootstrap samples), by sampling with replacement from the original dataset. Then, for each bootstrap sample, we build a model. The results in an ensemble of models, where each model votes with the equal weight. Typically, the goal of this procedure is to reduce the variance of the model of interest (e.g. decision trees).
- **Types**
  - Regression
    - A supervised problem, the outputs are continuous rather than discrete.
  - Classification
    - Inputs are divided into two or more classes, and the learner must produce a model that assigns unseen inputs to one or more (multi-label classification) of these classes. This is typically tackled in a supervised way.
  - Clustering
    - A set of inputs is to be divided into groups. Unlike in classification, the groups are not known beforehand, making this typically an unsupervised task.
  - Density Estimation
    - Finds the distribution of inputs in some space.
  - Dimensionality Reduction
    - Simplifies inputs by mapping them into a lower-dimensional space.
- **Kind**
  - Parametric
    - Step 1: Making an assumption about the functional form or shape of our function (f), i.e.: f is linear, thus we will select a linear model.
    - Step 2: Selecting a procedure to fit or train our model. This means estimating the Beta parameters in the linear function. A common approach is the (ordinary) least squares, amongst others.
  - Non-Parametric
    - When we do not make assumptions about the form of our function (f). However, since these methods do not reduce the problem of estimating f to a small number of parameters, a large number of observations is required in order to obtain an accurate estimate for f. An example would be the thin-plate spline model.
- **Categories**
  - Supervised
    - The computer is presented with example inputs and their desired outputs, given by a "teacher", and the goal is to learn a general rule that maps inputs to outputs.
  - Unsupervised
    - No labels are given to the learning algorithm, leaving it on its own to find structure in its input. Unsupervised learning can be a goal in itself (discovering hidden patterns in data) or a means towards an end (feature learning).
  - Reinforcement Learning
    - A computer program interacts with a dynamic environment in which it must perform a certain goal (such as driving a vehicle or playing a game against an opponent). The program is provided feedback in terms of rewards and punishments as it navigates its problem space.
- **Approaches**
  - Decision tree learning
  - Association rule learning
  - Artificial neural networks
  - Deep learning
  - Inductive logic programming
  - Support vector machines
  - Clustering
  - Bayesian networks
  - Reinforcement learning
  - Representation learning
  - Similarity and metric learning
  - Sparse dictionary learning
  - Genetic algorithms
  - Rule-based machine learning
  - Learning classifier systems
- **Taxonomy**
  - Generative Methods
    - Model class-conditional pdfs and prior probabilities. "Generative" since sampling can generate synthetic data points.
    - Popular models
      - Gaussians, Naïve Bayes, Mixtures of multinomials
      - Mixtures of Gaussians, Mixtures of experts, Hidden Markov Models (HMM)
      - Sigmoidal belief networks, Bayesian networks, Markov random fields
  - Discriminative Methods
    - Directly estimate posterior probabilities. No attempt to model underlying probability distributions. Focus computational resources on given task– better performance
    - Popular Models
      - Logistic regression, SVMs
      - Traditional neural networks, Nearest neighbor
      - Conditional Random Fields (CRF)
- **Selection Criteria**
  - Prediction Accuracy vs Model Interpretability
    - There is an inherent tradeoff between Prediction Accuracy and Model Interpretability, that is to say that as the model get more flexible in the way the function (f) is selected, they get obscured, and are hard to interpret. Flexible methods are better for inference, and inflexible methods are preferable for prediction.
- **Libraries**
  - Python
    - Numpy
      - Adds support for large, multi-dimensional arrays and matrices, along with a large library of high-level mathematical functions to operate on these arrays
    - Pandas
      - Offers data structures and operations for manipulating numerical tables and time series
    - Scikit-Learn
      - It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.
    - Tensorflow
      - Components
        - 
      - Does lazy evaluation. Need to build the graph, and then run it in a session.
        - 
        - 
        - 
    - MXNet
      - Is an modern open-source deep learning framework used to train, and deploy deep neural networks. MXNet library is portable and can scale to multiple GPUs and multiple machines. MXNet is supported by major Public Cloud providers including AWS and Azure. Amazon has chosen MXNet as its deep learning framework of choice at AWS.
    - Keras
      - Is an open source neural network library written in Python. It is capable of running on top of MXNet, Deeplearning4j, Tensorflow, CNTK or Theano. Designed to enable fast experimentation with deep neural networks, it focuses on being minimal, modular and extensible.
    - Torch
      - Torch is an open source machine learning library, a scientific computing framework, and a script language based on the Lua programming language. It provides a wide range of algorithms for deep machine learning, and uses the scripting language LuaJIT, and an underlying C implementation.
    - Microsoft Cognitive Toolkit
      - Previously known as CNTK and sometimes styled as The Microsoft Cognitive Toolkit, is a deep learning framework developed by Microsoft Research. Microsoft Cognitive Toolkit describes neural networks as a series of computational steps via a directed graph.

Machine Learning Mathematics

- **Linear Algebra**
  - Matrices
    - Almost all Machine Learning algorithms use Matrix algebra in one way or another. This is a broad subject, too large to be included here in it's full length. Here's a start: https://en.wikipedia.org/wiki/Matrix_(mathematics)
      - Basic Operations: Addition, Multiplication, Transposition
      - Transformations
      - Trace, Rank, Determinante, Inverse
  - Eigenvectors and Eigenvalues
    - In linear algebra, an eigenvector or characteristic vector of a linear transformation T from a vector space V over a field F into itself is a non-zero vector that does not change its direction when that linear transformation is applied to it.
      - http://setosa.io/ev/eigenvectors-and-eigenvalues/
  - Derivatives Chain Rule
    - Rule: $F'(x) = f'(g(x))g'(x)$
    - Leibniz Notation: $\frac{dz}{dx} = \frac{dz}{dy}\cdot\frac{dy}{dx}$
  - Jacobian Matrix
    - The matrix of all first-order partial derivatives of a vector-valued function. When the matrix is a square matrix, both the matrix and its determinant are referred to as the Jacobian in literature
  - Gradient
    - The gradient is a multi-variable generalization of the derivative. The gradient is a vector-valued function, as opposed to a derivative, which is scalar-valued.
      - $\nabla f = \frac{\partial f}{\partial x}\mathbf{i} + \frac{\partial f}{\partial y}\mathbf{j} + \frac{\partial f}{\partial z}\mathbf{k}$
  - Tensors
    - For Machine Learning purposes, a Tensor can be described as a Multidimentional Matrix Matrix. Depending on the dimensions, the Tensor can be a Scalar, a Vector, a Matrix, or a Multidimentional Matrix.
      - When measuring the forces applied to an infinitesimal cube, one can store the force values in a multidimensional matrix.
  - Curse of Dimensionality
    - When the dimensionality increases, the volume of the space increases so fast that the available data become sparse. This sparsity is problematic for any method that requires statistical significance. In order to obtain a statistically sound and reliable result, the amount of data needed to support the result often grows exponentially with the dimensionality.

- **Statistics**
  - Measures of Central Tendency
    - Mean
    - Median: Value in the middle or an ordered list, or average of two in middle.
    - Mode: Most Frequent Value
    - Quantile: Division of probability distributions based on contiguous intervals with equal probabilities. In short: Dividing observations numbers in a sample list equally.
  - Dispersion
    - Range
    - Medium Absolute Deviation (MAD): The average of the absolute value of the deviation of each value from the mean
    - Inter-quartile Range (IQR): Three quartiles divide the data in approximately four equally divided parts
    - Variance
      - Definition: The average of the squared differences from the Mean. Formally, is the expectation of the squared deviation of a random variable from its mean, and it informally measures how far a set of (random) numbers are spread out from their mean.
      - Types
        - Continuous: $\mathrm{Var}(X) = \sigma^2 = \int (x-\mu)^2 f(x)\,dx = \int x^2 f(x)\,dx - \mu^2$
        - Discrete
    - Standard Deviation
      - sqrt(variance): $\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i-\mu)^2}$
      - z-score/value/factor: The signed number of standard deviations an observation or datum is above the mean.
  - Relationship
    - Covariance
      - A measure of how much two random variables change together. http://stats.stackexchange.com/questions/18058/how-would-you-explain-covariance-to-someone-who-understands-only-the-mean
      - $\mathrm{cov}(x,y) = \frac{1}{n}\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y})$
      - dot(de_mean(x), de_mean(y)) / (n - 1)
    - Correlation
      - Pearson: Benchmarks linear relationship, most appropriate for measurements taken from an interval scale, is a measure of the linear dependence between two variables
        - $\rho_{X,Y} = \frac{\mathrm{cov}(X,Y)}{\sigma_X\sigma_Y}$
      - Spearman: Benchmarks monotonic relationship (whether linear or not), Spearman's coefficient is appropriate for both continuous and discrete variables, including ordinal variables.
      - Kendall
        - Is a statistic used to measure the ordinal association between two measured quantities.
        - Contrary to the Spearman correlation, the Kendall correlation is not affected by how far from each other ranks are but only by whether the ranks between observations are equal or not, and is thus only appropriate for discrete variables but not defined for continuous variables.
    - Co-occurence: The results are presented in a matrix format, where the cross tabulation of two fields is a cell value. The cell value represents the percentage of times that the two fields exist in the same events.
  - Techniques
    - Null Hypothesis: Is a general statement or default position that there is no relationship between two measured phenomena, or no association among groups. The null hypothesis is generally assumed to be true until evidence indicates otherwise.
    - p-value
      - In this method, as part of experimental design, before performing the experiment, one first chooses a model (the null hypothesis) and a threshold value for p, called the significance level of the test, traditionally 5% or 1% and denoted as α. If the p-value is less than the chosen significance level (α), that suggests that the observed data is sufficiently inconsistent with the null hypothesis that the null hypothesis may be rejected. However, that does not prove that the tested hypothesis is true. For typical analysis, using the standard α = 0.05 cutoff, the null hypothesis is rejected when p < .05 and not rejected when p > .05. The p-value does not, in itself, support reasoning about the probabilities of hypotheses but is only a tool for deciding whether to reject the null hypothesis.
      - Five heads in a row Example
        - Suppose a researcher flips a coin five times in a row and assumes a null hypothesis that the coin is fair. The test statistic of "total number of heads" can be one-tailed or two-tailed: a one-tailed test corresponds to seeing if the coin is biased towards heads, but a two-tailed test corresponds to seeing if the coin is biased either way. The researcher flips the coin five times and observes heads each time (HHHHH), yielding a test statistic of 5. In a one-tailed test, this is the upper extreme of all possible outcomes, and yields a p-value of (1/2)5 = 1/32 ≈ 0.03. If the researcher assumed a significance level of 0.05, this result would be deemed significant and the hypothesis that the coin is fair would be rejected. In a two-tailed test, a test statistic of zero heads (TTTTT) is just as extreme and thus the data of HHHHH would yield a p-value of 2×(1/2)5 = 1/16 ≈ 0.06, which is not significant at the 0.05 level.
      - This demonstrates that specifying a direction (on a symmetric test statistic) halves the p-value (increases the significance) and can mean the difference between data being considered significant or not.
    - p-hacking: The process of data mining involves automatically testing huge numbers of hypotheses about a single data set by exhaustively searching for combinations of variables that might show a correlation. Conventional tests of statistical significance are based on the probability that an observation arose by chance, and necessarily accept some risk of mistaken test results, called the significance.
  - Central Limit Theorem: States that a random variable defined as the average of a large number of independent and identically distributed random variables is itself approximately normally distributed.
    - http://blog.vctr.me/posts/central-limit-theorem.html

- **Optimization**
  - Gradient Descent: Is a first-order iterative optimization algorithm for finding the minimum of a function. To find a local minimum of a function using gradient descent, one takes steps proportional to the negative of the gradient (or of the approximate gradient) of the function at the current point. If instead one takes steps proportional to the positive of the gradient, one approaches a local maximum of that function; the procedure is then known as gradient ascent.
  - Stochastic Gradient Descent (SGD): Gradient descent uses total gradient over all examples per update, SGD updates after only 1 or few examples:
    - $\theta^{new} = \theta^{old} - \alpha\nabla_\theta J_t(\theta)$
  - Mini-batch Stochastic Gradient Descent (SGD): Gradient descent uses total gradient over all examples per update, SGD updates after only 1 example
  - Momentum: Idea: Add a fraction v of previous update to current one. When the gradient keeps pointing in the same direction, this will increase the size of the steps taken towards the minimum.
  - Adagrad: Adaptive learning rates for each parameter

- **Regularization**
  - L1 norm
    - Manhattan Distance: L1-norm is also known as least absolute deviations (LAD), least absolute errors (LAE). It is basically minimizing the sum of the absolute differences (S) between the target value and the estimated values.
      - $S = \sum_{i=1}^{n}|y_i - f(x_i)|$
  - L2 norm
    - Euclidean Distance: L2-norm is also known as least squares. It is basically minimizing the sum of the square of the differences (S) between the target value and the estimated values:
      - $S = \sum_{i=1}^{n}(y_i - f(x_i))^2$
  - Early Stopping: Early stopping rules provide guidance as to how many iterations can be run before the learner begins to over-fit, and stop the algorithm then.
  - Dropout: Is a regularization technique for reducing overfitting in neural networks by preventing complex co-adaptations on training data. It is a very efficient way of performing model averaging with neural networks. The term "dropout" refers to dropping out units (both hidden and visible) in a neural network
  - Sparse regularizer on columns: This regularizer defines an L2 norm on each column and an L1 norm over all columns. It can be solved by proximal methods.
    - $R(w) = \sum_{i=1}^{D}\|W\|_{2,1}$
  - Nuclear norm regularization
    - $R(w) = \|\sigma(W)\|_1$ where $\sigma(W)$ is the eigenvalues in the singular value decomposition of W.
  - Mean-constrained regularization: This regularizer constrains the functions learned for each task to be similar to the overall average of the functions across all tasks. This is useful for expressing prior information that each task is expected to share similarities with each other task. An example is predicting blood iron levels measured at different times of the day, where each task represents a different person.
  - Clustered mean-constrained regularization: This regularizer is similar to the mean-constrained regularizer, but instead enforces similarity between tasks within the same cluster. This can capture more complex prior information. This technique has been used to predict Netflix recommendations.
  - Graph-based similarity: More general than above, similarity between tasks can be defined by a function. The regularizer encourages the model to learn similar functions for similar tasks.

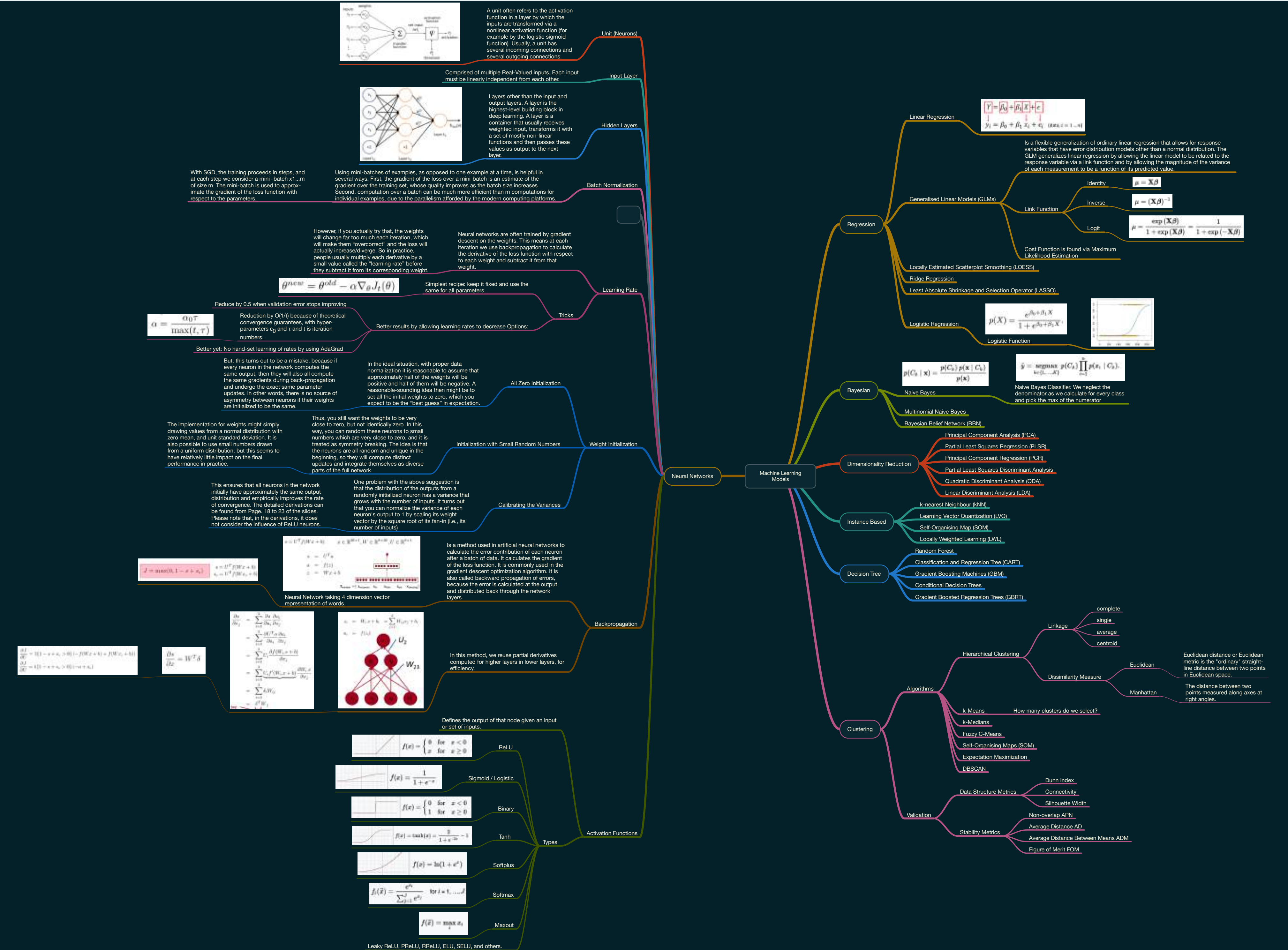
- **Cost/Loss(Min) Objective(Max) Functions**
  - Maximum Likelihood Estimation (MLE)
    - Many cost functions are the result of applying Maximum Likelihood. For instance, the Least Squares cost function can be obtained via Maximum Likelihood. Cross-Entropy is another example.
    - The likelihood of a parameter value (or vector of parameter values), θ, given outcomes x, is equal to the probability (density) assumed for those observed outcomes given those parameter values, that is
      - $\mathcal{L}(\theta \mid x) = P(x \mid \theta)$
    - The natural logarithm of the likelihood function, called the log-likelihood, is more convenient to work with. Because the logarithm is a monotonically increasing function, the logarithm of a function achieves its maximum value at the same points as the function itself, and hence the log-likelihood can be used in place of the likelihood in maximum likelihood estimation and related techniques.
    - In general, for a fixed set of data and underlying statistical model, the method of maximum likelihood selects the set of values of the model parameters that maximizes the likelihood function. Intuitively, this maximizes the "agreement" of the selected model with the observed data, and for discrete random variables it indeed maximizes the probability of the observed data under the resulting distribution. Maximum-likelihood estimation gives a unified approach to estimation, which is well-defined in the case of the normal distribution and many other problems.
      - $f(x_1,x_2,\ldots,x_n \mid \theta) = f(x_1\mid\theta)\times f(x_2\mid\theta)\times\cdots\times f(x_n\mid\theta)$
      - $\mathcal{L}(\theta\,;x_1,\ldots,x_n) = f(x_1,x_2,\ldots,x_n\mid\theta) = \prod_{i=1}^{n} f(x_i\mid\theta)$
      - $\ln\mathcal{L}(\theta\,;x_1,\ldots,x_n) = \sum_{i=1}^{n}\ln f(x_i\mid\theta)$
      - $\hat{\ell}(\theta\,;x) = \frac{1}{n}\sum_{i=1}^{n}\ln f(x_i\mid\theta)$
      - $\{\hat{\theta}_{mle}\} \subseteq \{\arg\max_{\theta\in\Theta}\hat{\ell}(\theta\,;x_1,\ldots,x_n)\}$
  - Cross-Entropy
    - Cross entropy can be used to define the loss function in machine learning and optimization. The true probability pi is the true label, and the given distribution qi is the predicted value of the current model.
      - $H(p,q) = \mathrm{E}_p[l_i] = \mathrm{E}_p\left[\log\frac{1}{q(x_i)}\right]$
      - $H(p,q) = \sum_{x_i} p(x_i)\,\log\frac{1}{q(x_i)}$
      - $H(p,q) = -\sum_x p(x)\,\log q(x)$
      - Cross-entropy error function and logistic regression: $L(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N}H(p_n,q_n) = -\frac{1}{N}\sum_{n=1}^{N}\left[y_n\log\hat{y}_n + (1-y_n)\log(1-\hat{y}_n)\right]$
  - Logistic: The logistic loss function is defined as:
    - $V(f(\vec{x}),y) = \frac{1}{\ln 2}\ln(1+e^{-yf(\vec{x})})$
  - Quadratic: The use of a quadratic loss function is common, for example when using least squares techniques. It is often more mathematically tractable than other loss functions because of the properties of variances, as well as being symmetric: an error above the target causes the same loss as the same magnitude of error below the target. If the target is t, then a quadratic loss function is:
    - $\lambda(x) = C(t-x)^2$
  - 0-1 Loss: In statistics and decision theory, a frequently used loss function is the 0-1 loss function
    - $L(\hat{y},y) = I(\hat{y}\neq y)$
  - Hinge Loss: The hinge loss is a loss function used for training classifiers. For an intended output t = ±1 and a classifier score y, the hinge loss of the prediction y is defined as:
    - $\ell(y) = \max(0, 1 - t\cdot y)$
  - Exponential
    - $E_e(y,t) = \exp\left(-\frac{1}{2}\sum_i (y_i - t_i)^2\right)$
  - Hellinger Distance: It is used to quantify the similarity between two probability distributions. It is a type of f-divergence.
    - $H^2(P,Q) = \frac{1}{2}\int\left(\sqrt{\frac{dP}{d\lambda}} - \sqrt{\frac{dQ}{d\lambda}}\right)^2 d\lambda$
    - To define the Hellinger distance in terms of measure theory, let P and Q denote two probability measures that are absolutely continuous with respect to a third probability measure λ. The square of the Hellinger distance between P and Q is defined as the quantity
  - Kullback-Leibler Divengence: Is a measure of how one probability distribution diverges from a second expected probability distribution. Applications include characterizing the relative (Shannon) entropy in information systems, randomness in continuous time-series, and information gain when comparing statistical models of inference.
    - Discrete: $D_{\mathrm{KL}}(P\|Q) = \sum_i P(i)\,\log\frac{P(i)}{Q(i)}$
    - Continuous: $D_{\mathrm{KL}}(P\|Q) = \int_X p\,\log\frac{p}{q}\,d\mu$
  - Itakura–Saito distance: Is a measure of the difference between an original spectrum P(ω) and an approximation P^(ω) of that spectrum. Although it is not a perceptual measure, it is intended to reflect perceptual (dis)similarity.
    - $D_{IS}(P(\omega),\hat{P}(\omega)) = \frac{1}{2\pi}\int_{-\pi}^{\pi}\left[\frac{P(\omega)}{\hat{P}(\omega)} - \log\frac{P(\omega)}{\hat{P}(\omega)} - 1\right]d\omega$
  - https://stats.stackexchange.com/questions/154879/a-list-of-cost-functions-used-in-neural-networks-alongside-applications
  - https://en.wikipedia.org/wiki/Loss_functions_for_classification

- **Probability**
  - Concepts
    - Frequentist vs Bayesian Probability
      - Frequentist: Basic notion of probability: # Results / # Attempts
      - Bayesian: The probability is not a number, but a distribution itself.
      - http://www.behind-the-enemy-lines.com/2008/01/are-you-bayesian-or-frequentist-or.html
    - Random Variable: In probability and statistics, a random variable, random quantity, aleatory variable or stochastic variable is a variable whose value is subject to variations due to chance (i.e. randomness, in a mathematical sense). A random variable can take on a set of possible different values (similarly to other mathematical variables), each with an associated probability, in contrast to other mathematical variables.
      - Expectation (Expected Value) of a Random Variable: $\mathrm{E}[X] = \sum_{i=1}^{k} x_i\,p_i$
      - Same, for continuous variables: $\mathrm{E}[X] = \int_{-\infty}^{\infty} x f(x)\,dx$
    - Independence: Two events are independent, statistically independent, or stochastically independent if the occurrence of one does not affect the probability of the other.
      - $p(x,y) = p(x)\cdot p(y).$
    - Conditionality: $P(A\cap B) = P(A|B)P(B)$
    - Bayes Theorem (rule, law)
      - Simple Form: $P(A|B) = \frac{P(B|A)\,P(A)}{P(B)}$
      - With Law of Total probability: $P(B) = \sum_j P(B|A_j)P(A_j)$, $\Rightarrow P(A_i|B) = \frac{P(B|A_i)\,P(A_i)}{\sum_j P(B|A_j)\,P(A_j)}$
    - Marginalisation: The marginal distribution of a subset of a collection of random variables is the probability distribution of the variables contained in the subset. It gives the probabilities of various values of the variables in the subset without reference to the values of the other variables.
      - Continuous: $p(x) = \int_y p(x,y)\,dy$
      - Discrete: $p(x) = \sum_y p(x,y)$
      - $\Pr(X=x) = \sum_y \Pr(X=x, Y=y) = \sum_y \Pr(X=x\mid Y=y)\Pr(Y=y)$
    - Law of Total Probability: Is a fundamental rule relating marginal probabilities to conditional probabilities. It expresses the total probability of an outcome which can be realized via several distinct events - hence the name.
      - $\Pr(A) = \sum_n \Pr(A\cap B_n)$
      - $\Pr(A) = \sum_n \Pr(A\mid B_n)\Pr(B_n)$
    - Chain Rule: Permits the calculation of any member of the joint distribution of a set of random variables using only conditional probabilities.
      - $P\left(\bigcap_{k=1}^{n}A_k\right) = \prod_{k=1}^{n}P\left(A_k \,\middle|\, \bigcap_{j=1}^{k-1}A_j\right)$
    - Bayesian Inference: Bayesian inference derives the posterior probability as a consequence of two antecedents, a prior probability and a "likelihood function" derived from a statistical model for the observed data. Bayesian inference computes the posterior probability according to Bayes' theorem. It can be applied iteratively so to update the confidence on out hypothesis.
      - $P(H\mid E) = \frac{P(E\mid H)\cdot P(H)}{P(E)}$

- **Distributions**
  - Definition: Is a table or an equation that links each outcome of a statistical experiment with the probability of occurence. When Continuous, is is described by the Probability Density Function
  - Types (Density Function)
    - Normal (Gaussian): $f(x\mid\mu,\sigma^2) = \frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{(x-\mu)^2}{2\sigma^2}}$
    - Poisson: $f(k;\lambda) = \Pr(X=k) = \frac{\lambda^k e^{-\lambda}}{k!}$
    - Uniform: $f(x) = \begin{cases}\frac{1}{b-a} & \text{for } a\le x\le b,\\ 0 & \text{for } x<a \text{ or } x>b\end{cases}$
    - Bernoulli: $f(k;p) = \begin{cases}p & \text{if } k=1,\\ 1-p & \text{if } k=0.\end{cases}$, $f(k;p) = p^k(1-p)^{1-k}$ for $k\in\{0,1\}$
    - Gamma
    - Binomial: $f(k) = \binom{n}{k}p^k(1-p)^{n-k}$
  - Cumulative Distribution Function (CDF): $F_X(x) = \int_{-\infty}^{x} f_X(t)\,dt$

- **Information Theory**
  - Entropy: $H = -\sum_i p_i\log_2(p_i)$
    - Entropy is a measure of unpredictability of information content.
    - To evaluate a language model, we should measure how much surprise it gives us for real sequences in that language. For each real word encountered, the language model will give a probability p. And we use -log(p) to quantify the surprise. And we average the total surprise over a long enough sequence. So, in case of a 1000-letter sequence with 500 A and 500 B, the surprise given by the 1/3-2/3 model will be: [-500*log(1/3) - 500*log(2/3)]/1000 = 1/2 * Log(9/2) While the correct 1/2-1/2 model will give: [-500*log(1/2) - 500*log(1/2)]/1000 = 1/2 * Log(8/2) So, we can see, the 1/3, 2/3 model gives more surprise, which indicates it is worse than the correct model. Only when the sequence is long enough, the average effect will mimic the expectation over the 1/2-1/2 distribution. If the sequence is short, it won't give a convincing result.
  - Cross Entropy: $H(p,q) = -\sum_x p(x)\,\log q(x).$
    - Cross entropy between two probability distributions p and q over the same underlying set of events measures the average number of bits needed to identify an event drawn from the set, if a coding scheme is used that is optimized for an "unnatural" probability distribution q, rather than the "true" distribution p.
  - Joint Entropy: $H(X,Y) = \mathbb{E}_{X,Y}[-\log p(x,y)] = -\sum_{x,y}p(x,y)\log p(x,y)$
  - Conditional Entropy
  - Mutual Information: $I(X;Y) = \sum_{y\in Y}\sum_{x\in X} p(x,y)\log\left(\frac{p(x,y)}{p(x)\,p(y)}\right)$
  - Kullback-Leibler Divergence

- **Density Estimation**
  - Mostly Non-Parametric. Parametric makes assumptions on my data/random-variables, for instance, that they are normally distributed. Non-parametric does not.
  - The methods are generally intended for description rather than formal inference
  - Methods
    - Kernel Density Estimation
      - non-negative
      - it's a type of PDF that it is symmetric
      - real-valued
      - symmetric
      - integral over function is equal to 1
      - non-parametric
      - calculates kernel distributions for every sample point, and then adds all the distributions
      - Uniform, Triangle, Quartic, Triweight, Gaussian, Cosine, others...
    - Cubic Spline: A cubic spline is a function created from cubic polynomials on each between-knot interval by pairing them together twice continuously differentiable at the knots.


Machine Learning Models
Neural Networks
Unit (Neurons)
A unit often refers to the activation function in a layer by which the inputs are transformed via a nonlinear activation function (for example by the logistic sigmoid function). Usually, a unit has several incoming connections and several outgoing connections.
Input Layer
Comprised of multiple Real-Valued inputs. Each input must be linearly independent from each other.
Hidden Layers
Layers other than the input and output layers. A layer is the highest-level building block in deep learning. A layer is a container that usually receives weighted input, transforms it with a set of mostly non-linear functions and then passes these values as output to the next layer.
Batch Normalization
Using mini-batches of examples, as opposed to one example at a time, is helpful in several ways. First, the gradient of the loss over a mini-batch is an estimate of the gradient over the training set, whose quality improves as the batch size increases. Second, computation over a batch can be much more efficient than m computations for individual examples, due to the parallelism afforded by the modern computing platforms.
With SGD, the training proceeds in steps, and at each step we consider a mini- batch x1...m of size m. The mini-batch is used to approx- imate the gradient of the loss function with respect to the parameters.
Learning Rate
Neural networks are often trained by gradient descent on the weights. This means at each iteration we use backpropagation to calculate the derivative of the loss function with respect to each weight and subtract it from that weight.
However, if you actually try that, the weights will change far too much each iteration, which will make them “overcorrect” and the loss will actually increase/diverge. So in practice, people usually multiply each derivative by a small value called the “learning rate” before they subtract it from its corresponding weight.
$\theta^{new} = \theta^{old} - \alpha \nabla_\theta J_t(\theta)$
Tricks
Simplest recipe: keep it fixed and use the same for all parameters.
Better results by allowing learning rates to decrease Options:
Reduce by 0.5 when validation error stops improving
Reduction by O(1/t) because of theoretical convergence guarantees, with hyper-parameters ϵ_0 and τ and t is iteration numbers.
$\alpha = \frac{\alpha_0 \tau}{\max(t, \tau)}$
Better yet: No hand-set learning of rates by using AdaGrad
Weight Initialization
All Zero Initialization
In the ideal situation, with proper data normalization it is reasonable to assume that approximately half of the weights will be positive and half of them will be negative. A reasonable-sounding idea then might be to set all the initial weights to zero, which you expect to be the “best guess” in expectation.
But, this turns out to be a mistake, because if every neuron in the network computes the same output, then they will also all compute the same gradients during back-propagation and undergo the exact same parameter updates. In other words, there is no source of asymmetry between neurons if their weights are initialized to be the same.
Initialization with Small Random Numbers
Thus, you still want the weights to be very close to zero, but not identically zero. In this way, you can random these neurons to small numbers which are very close to zero, and it is treated as symmetry breaking. The idea is that the neurons are all random and unique in the beginning, so they will compute distinct updates and integrate themselves as diverse parts of the full network.
The implementation for weights might simply drawing values from a normal distribution with zero mean, and unit standard deviation. It is also possible to use small numbers drawn from a uniform distribution, but this seems to have relatively little impact on the final performance in practice.
Calibrating the Variances
One problem with the above suggestion is that the distribution of the outputs from a randomly initialized neuron has a variance that grows with the number of inputs. It turns out that you can normalize the variance of each neuron's output to 1 by scaling its weight vector by the square root of its fan-in (i.e., its number of inputs)
This ensures that all neurons in the network initially have approximately the same output distribution and empirically improves the rate of convergence. The detailed derivations can be found from Page. 18 to 23 of the slides. Please note that, in the derivations, it does not consider the influence of ReLU neurons.
Backpropagation
Is a method used in artificial neural networks to calculate the error contribution of each neuron after a batch of data. It calculates the gradient of the loss function. It is commonly used in the gradient descent optimization algorithm. It is also called backward propagation of errors, because the error is calculated at the output and distributed back through the network layers.
Neural Network taking 4 dimension vector representation of words.
In this method, we reuse partial derivatives computed for higher layers in lower layers, for efficiency.
$\frac{\partial s}{\partial z} = W^T \delta$
Activation Functions
Defines the output of that node given an input or set of inputs.
Types
ReLU
$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$
Sigmoid / Logistic
$f(x) = \frac{1}{1+e^{-x}}$
Binary
$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Tanh
$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$
Softplus
$f(x) = \ln(1+e^x)$
Softmax
$f_i(\vec{x}) = \frac{e^{x_i}}{\sum_{j=1}^{J} e^{x_j}}$ for $i = 1, \ldots, J$
Maxout
$f(\vec{x}) = \max_i x_i$
Leaky ReLU, PReLU, RReLU, ELU, SELU, and others.
Regression
Linear Regression
$Y = \beta_0 + \beta_1 X + \epsilon$
$y_i = \beta_0 + \beta_1 x_i + \epsilon_i$
Generalised Linear Models (GLMs)
Is a flexible generalization of ordinary linear regression that allows for response variables that have error distribution models other than a normal distribution. The GLM generalizes linear regression by allowing the linear model to be related to the response variable via a link function and by allowing the magnitude of the variance of each measurement to be a function of its predicted value.
Link Function
Identity
$\mu = \mathbf{X}\beta$
Inverse
$\mu = (\mathbf{X}\beta)^{-1}$
Logit
$\mu = \frac{\exp(\mathbf{X}\beta)}{1+\exp(\mathbf{X}\beta)} = \frac{1}{1+\exp(-\mathbf{X}\beta)}$
Cost Function is found via Maximum Likelihood Estimation
Locally Estimated Scatterplot Smoothing (LOESS)
Ridge Regression
Least Absolute Shrinkage and Selection Operator (LASSO)
Logistic Regression
$p(X) = \frac{e^{\beta_0+\beta_1 X}}{1+e^{\beta_0+\beta_1 X}}$
Logistic Function
Bayesian
$p(C_k \mid \mathbf{x}) = \frac{p(C_k)\, p(\mathbf{x} \mid C_k)}{p(\mathbf{x})}$
$\hat{y} = \underset{k \in \{1,\ldots,K\}}{\operatorname{argmax}}\ p(C_k) \prod_{i=1}^{n} p(x_i \mid C_k)$
Naive Bayes
Naive Bayes Classifier. We neglect the denominator as we calculate for every class and pick the max of the numerator
Multinomial Naive Bayes
Bayesian Belief Network (BBN)
Dimensionality Reduction
Principal Component Analysis (PCA)
Partial Least Squares Regression (PLSR)
Principal Component Regression (PCR)
Partial Least Squares Discriminant Analysis
Quadratic Discriminant Analysis (QDA)
Linear Discriminant Analysis (LDA)
Instance Based
k-nearest Neighbour (kNN)
Learning Vector Quantization (LVQ)
Self-Organising Map (SOM)
Locally Weighted Learning (LWL)
Decision Tree
Random Forest
Classification and Regression Tree (CART)
Gradient Boosting Machines (GBM)
Conditional Decision Trees
Gradient Boosted Regression Trees (GBRT)
Clustering
Algorithms
Hierarchical Clustering
Linkage
complete
single
average
centroid
Dissimilarity Measure
Euclidean
Euclidean distance or Euclidean metric is the "ordinary" straight-line distance between two points in Euclidean space.
Manhattan
The distance between two points measured along axes at right angles.
k-Means
How many clusters do we select?
k-Medians
Fuzzy C-Means
Self-Organising Maps (SOM)
Expectation Maximization
DBSCAN
Validation
Data Structure Metrics
Dunn Index
Connectivity
Silhouette Width
Stability Metrics
Non-overlap APN
Average Distance AD
Average Distance Between Means ADM
Figure of Merit FOM