March 31st, 2020 Workshop

Deep Learning for Natural Language Understanding (NLU)

Basically, this workshop is about the deep learning for Arabic NLP. It will be hands on for linguists and engineers who are interested in building NLP models and applications. It will be held on the new campus El Alamein, north coast of Egypt, at the College of Artificial Intelligence, Arab Academy for Science Technology and Maritime Transport (AASTMT). The workshop will cover the following topics:

1. Brief introduction to neural networks (Dr. Hanaa Bayoumi).

   The talk will illustrate how neural networks learn a certain function by analyzing input data through simple illustrative examples

2. Distributive representation of texts: Word embedding (Dr. A. Sarah Hassan).

   Embedding means words or phrases from the vocabulary are mapped to vectors of real numbers. that words that are used in similar contexts will be given similar numerical vectors. These vectors will be placed close together within the high-dimensional semantic space. They will cluster together, and their distance to each other will be small.

   We will describe the different algorithms used to drive theses representations.

3. Language modeling using transformers (Dr. Aly Fahmy).

   Transformers are seen as the key breakthrough for the state-of-art performance of deep learning methods on challenging natural language processing problems. They are built around the notion of "Attention" introduced in the paper "Attention is All you Need". It is considered as one of top Machine Learning papers of the decade

   We will describe the structure of the transformers and emphasize what language characteristics are learned through attention.

4. Hands on BERT: A pre-trained model for NLP tasks (Eng. Islam Hassan).

   BERT, Bidirectional Encoder Representations from Transformers, is a technique for NLP pre-training developed by Google. BERT model can be fine-tuned with just one additional output layer to create state-of-the-art models for a wide range of NLP tasks such as Question-Answering, Named Entity Recognition, Textual Entailment, Coreference Resolution and many others.

We will provide hands-on fine-tuning pre-trained BERT to NLP tasks involving one or two input sentences such as grammatical checking task and paraphrasing checking task respectively. where the in single sentences.

It is recommended that you take your own laptop with you and be prepared to do the experiments yourself.

The Invited speakers:

Dr. Aly Fahmy, Dean of Artificial Intelligence College, Arab Academy for Sciences Technology and Maritime Transport (AASTMT) and his team. Dr. Fahmy was the Ex-Dean of Faculty of Computers and Artificial Intelligence, Cairo University.

Transportation

Transportation from Bibliotheca Alexandrina to Al-Elamein City's workshop will take place at 8:00 a.m. and back to Alexandrina Bibliotheca at 4:00 p.m.

The transportation will be provided free by the Arab Academy for Sciences Technology and Maritime Transport.

Workshop schedule

March 31st, 2020 Deep Learning for Natural Language Processing (NLU)	
Time	Activity
09:45 - 10:00	Planned Arrival to El-Alamein City – AAST
10:00 - 10:30	Workshop Opening Honorary Chairman: Dr. Mostafa Hussein, AAST Vice President
10:30 - 11:00	Brief Introduction to Neural Networks.
11:00 - 11:45	Distributive Representation of Texts: Word Embedding
11:45 - 12:15	Coffee Break and Networking
12:15 - 01:00	Language Modeling using Transformers.
01:00 - 01:45	Hands on BERT: Grammar Checking Task
01:45 - 02:15	Hands on BERT: Paraphrasing Checking Task
02:15 - 02:30	Conference Closure
02:30 - 03: 30	Lunch
03:30	Tour in New El-Alamein City and Back to Bibliotheca Alexandrina, Sidi Gaber Train Station and Cairo

ESOLE 2018

Deep Learning Technologies for NLP Tasks

DECEMBER 6, 2018 | 9:30 AM TO 2:30 PM

Overview

The objective of this workshop is to identify current and emerging natural language processing (NLP) efforts being applied using the different deep learning (DL) techniques. The workshop will provide insights into the different NLP tasks such as automated essay scoring, automatic classification, machine translation, paraphrase detection, question answering and text similarity. Also the basic fundamentals of DL and the different types of deep neural network such as Convolution Neural Network (CNN) and Recurrent Neural Network (RNN) will be covered. The amazing power of Word2Vec and Sent2Vec and their successful applications will be discussed in details.

Agenda

Time	Topic	Speaker
9.30	**Registration**	
10.00	**Welcome**	Prof. Salwa ElRamly
10.15	**Workshop Overview**	Prof. Aly Fahmy
10.30	**Introduction to Deep Learning, Machine learning and Neural Networks**	Prof. Aly Fahmy – Dr. Hanaa Bayomi
11.15	**Natural Language Processing Tasks**	Prof. Aly Fahmy – Dr. Hanaa Bayomi
11.45	**Break**	
12.15	**Hands On Chatbots and NLP**	Prof. Aly Fahmy - Dr. Nour El-Deen Mahmoud
1.15	**Hands on Applications of Word and Sentence Embedding (Automated Short Answer Scoring)**	Prof. Aly Fahmy – TA. Sarah Hassan
2.15	**Closing Remarks**	Prof. Aly Fahmy – Prof. Salwa ElRamly

Speakers

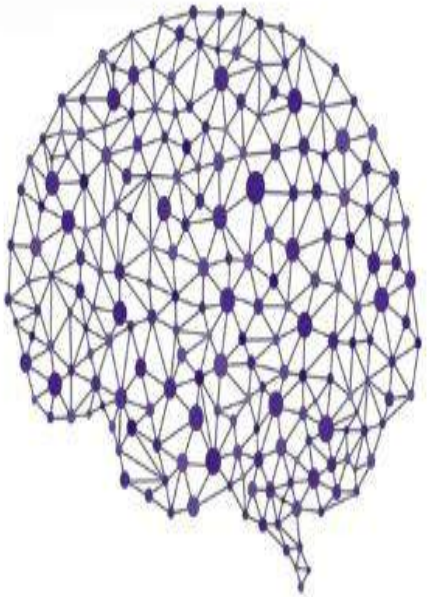
Prof AlyFahmy: Prof Aly Aly Fahmy is the former Dean of Faculty of Computers & Information – Cairo University. His research interest is in Artificial Intelligence Topics such as natural language processing, data and text mining, and information retrieval. Prof Aly Fahmy has a number of publications. He obtained B.Sc in June 1972,

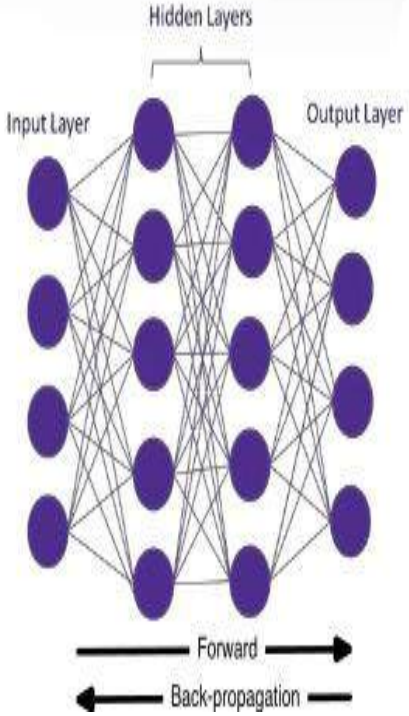
Computer Engineering Department Military Technical College (M.T.C) Excellent with Honor Grade. DPL - Diploma: General Purpose Simulation, June 1973, Computer Department, Military Technical College (M.T.C), M.Sc in Logical Database Systems 1976, Computer Department, E.N.S.A.E, Toulouse, France. Ph.D in Artificial Intelligence Control of Automatic Deductions for Logic Based Systems 1979, Computer Department, The Centre of Research and Studies, Toulouse, France(C.E.R.T) under the supervision of H. Gallaire (Ex Vice President and Chief Technical Officer of Xerox Corporation) and J.M. Nicolas.

Dr. Hanaa Bayomi: Currently working as a lecturer, Faculty of Computers and Information, Cairo University. Her research interest is in Artificial Intelligent, Machine Learning, Information Extraction, Opinion Mining, and Natural Language Processing. She is currently investigating the impact of deep learning in different fields Medical, Natural Language Processing and security. She obtained PhD degree from Faculty of Computers and Information, Cairo University, Egypt in the field of opinion mining. She received her BSc and Master degrees from Faculty of Computers and Information, Cairo University, Egypt.

Dr. Nour El-Deen Mahmoud received his B.Sc, M.Sc and Ph.D degree in 2006, 2009 and 2013 respectively, all from Cairo University, Faculty of Computers and Information, Information Technology Department, Cairo, Egypt. He had a Professional M.Sc in Cloud Computing in 2018. Currently, he is an assistant professor in faculty of computers and information, Cairo University. He had a professional experience in designing and developing IT projects that include web technologies and chatbots for Egyptian Universities and for several Industrial IT companies. For more information, https://scholar.cu.edu.eg/nourmahmoud

Eng. Sarah Hassan, Currently working as teaching assistant- Faculty of Computers and Information - Cairo University. Her research interests are machine learning, deep learning and natural language processing. She is currently applying for master degree with research on short answer clustering and automatic scoring based on paragraph embedding. She graduated in 2010 from Faculty of Computers and Information, Cairo University, Computer Science Department.





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 improves its performance on a given task with experience (i.e. examples, data).
- 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?

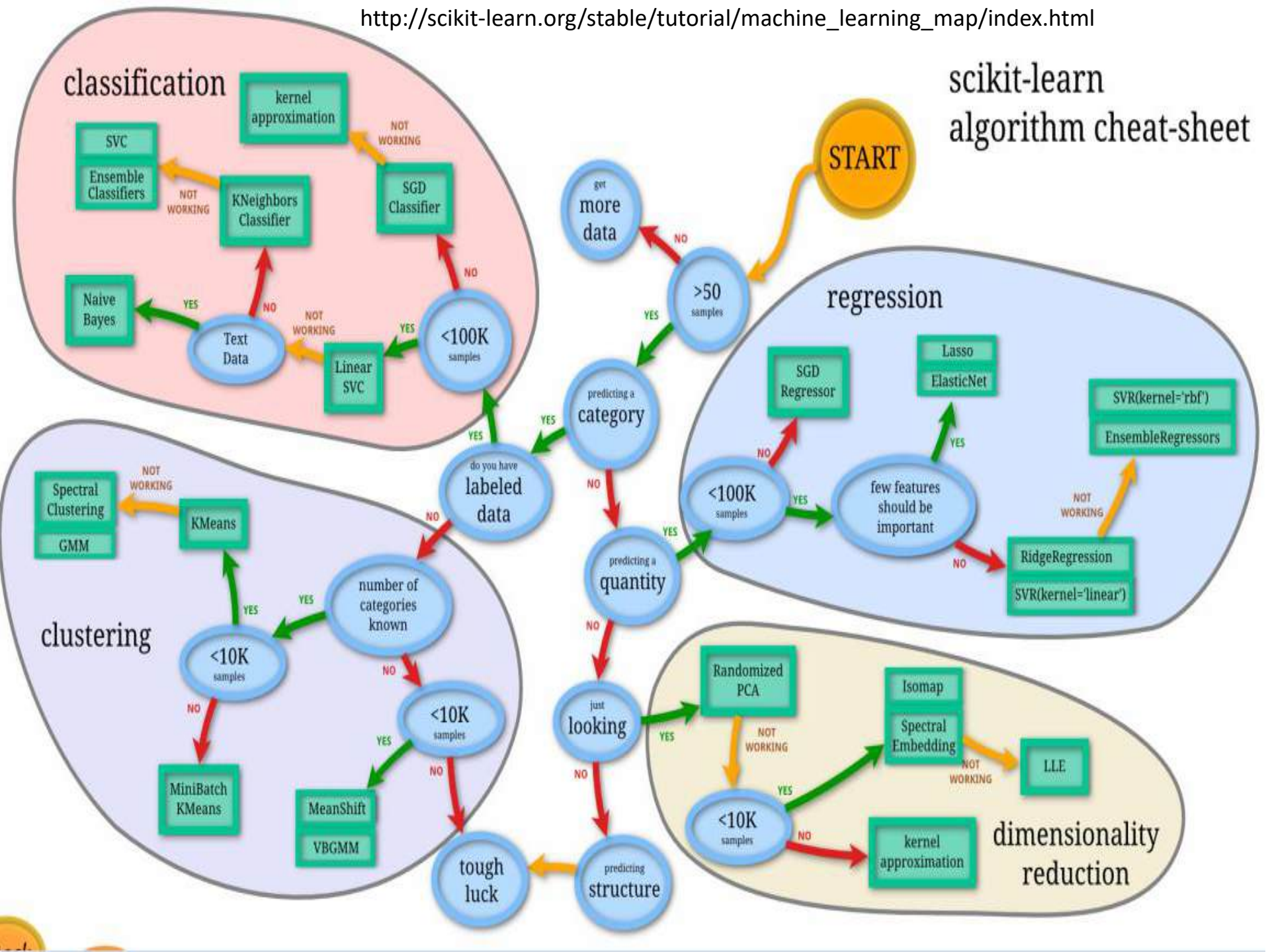
Machine Learning types

- 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

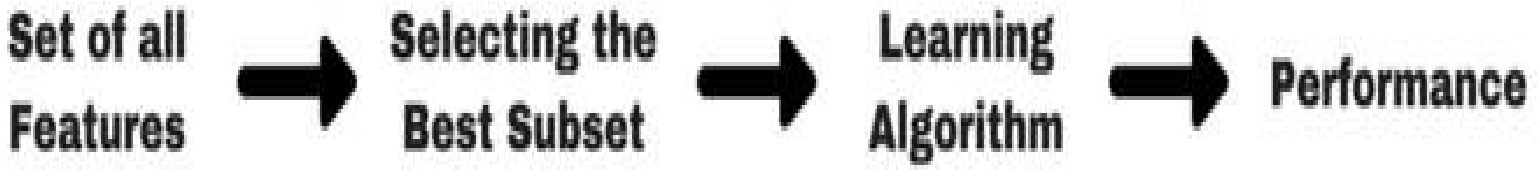
https://archive.ics.uci.edu/ml/datasets.html
UCI Machine Learning Repository: Data Sets

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

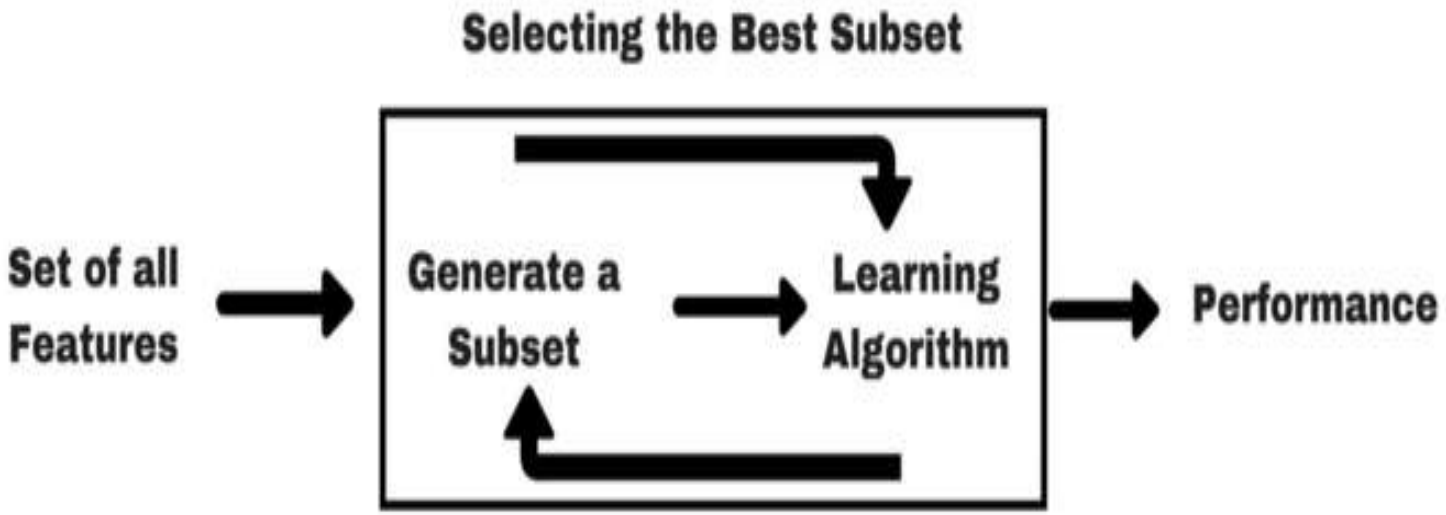
- Filter methods are generally used as a *preprocessing step*. The selection of features is independent of any machine learning algorithms.
- Two steps (score-and-filter approach)
  1. assess each feature individually for its potential in discriminating among classes in the data
  2. features falling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



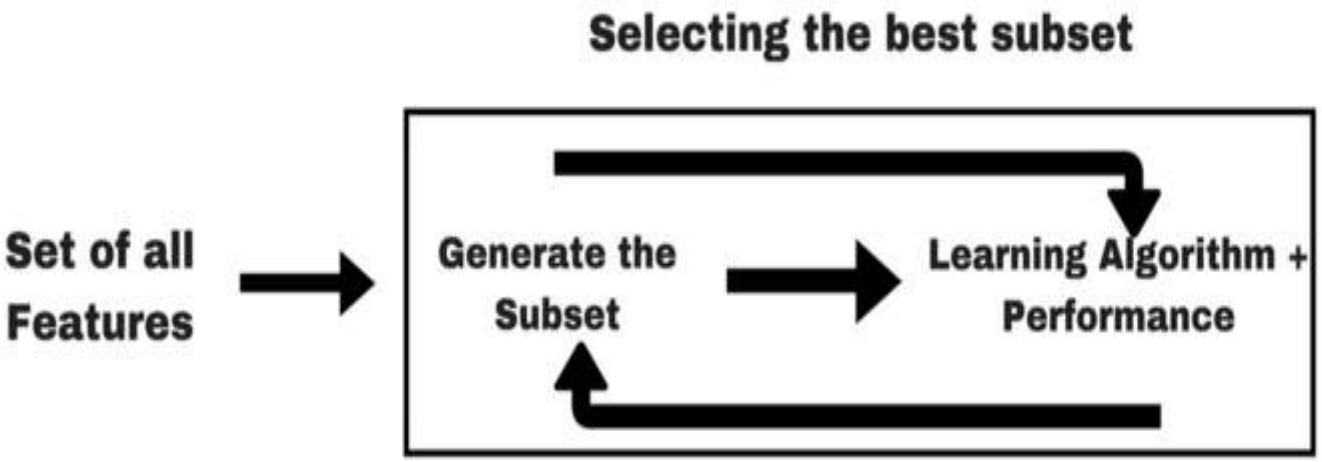
Wrappers

- 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.

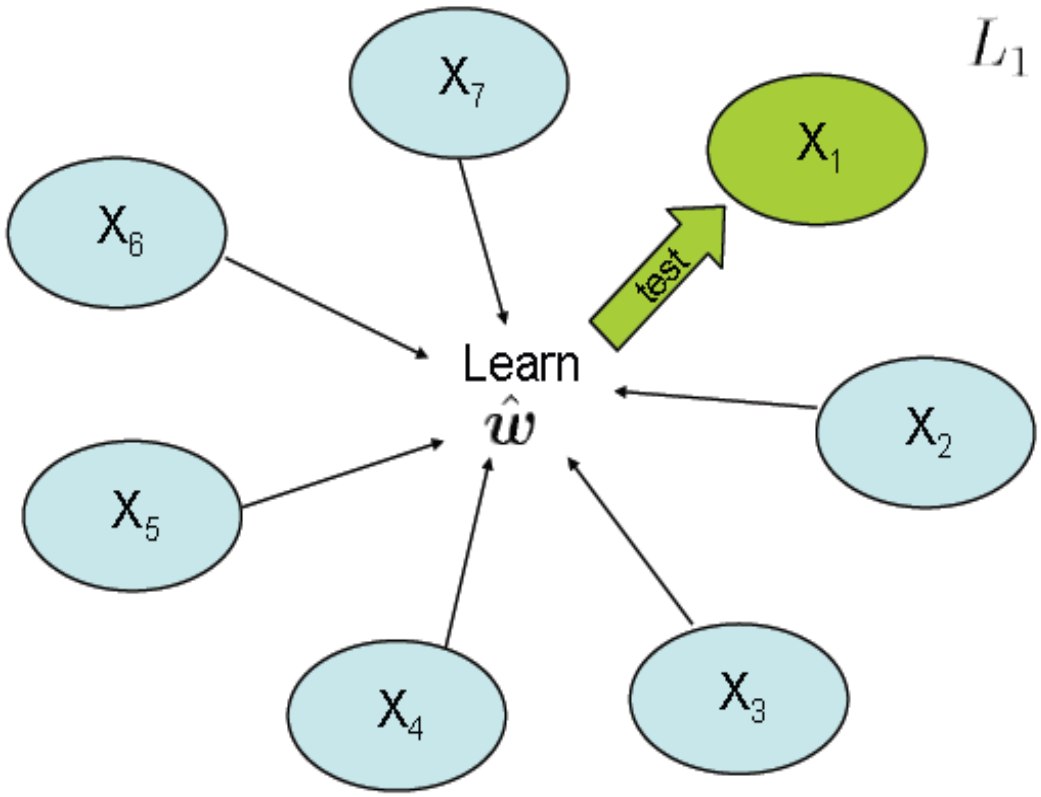
Lasso regression performs L1 regularization which adds penalty equivalent to absolute value of the magnitude of coefficients.

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

Choosing the best model

K-fold cross validation

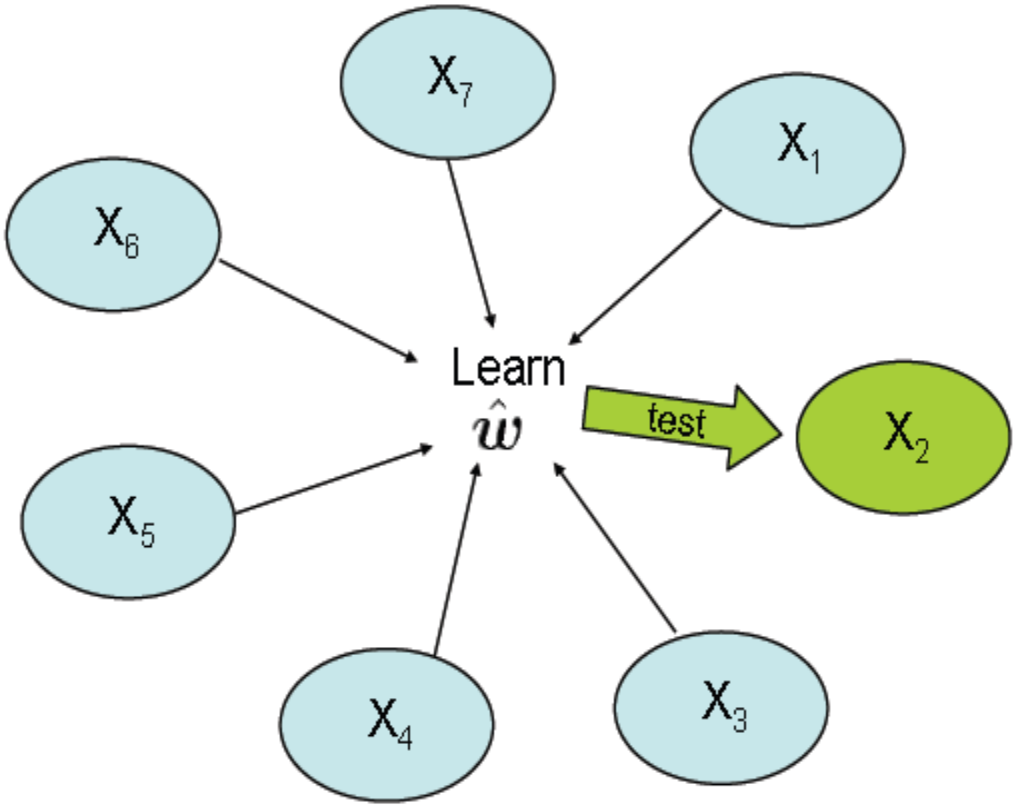
- A technique for estimating test error
- Uses *all* of the data to validate
- Divide data into K groups $\{X_1, X_2, \ldots, X_K\}$.
- Use each group as a validation set, then average all validation errors



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

K-fold cross validation

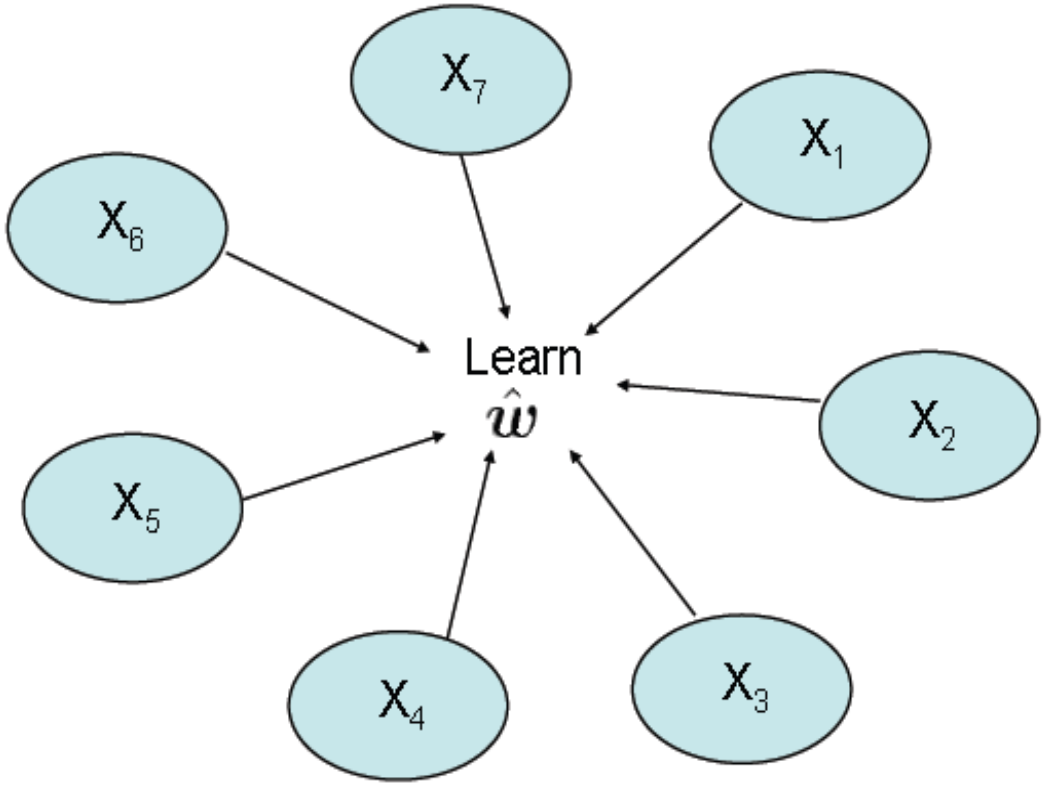
- A technique for estimating test error
- Uses *all* of the data to validate
- Divide data into K groups $\{X_1, X_2, \ldots, X_K\}$.
- Use each group as a validation set, then average all validation errors



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

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.*

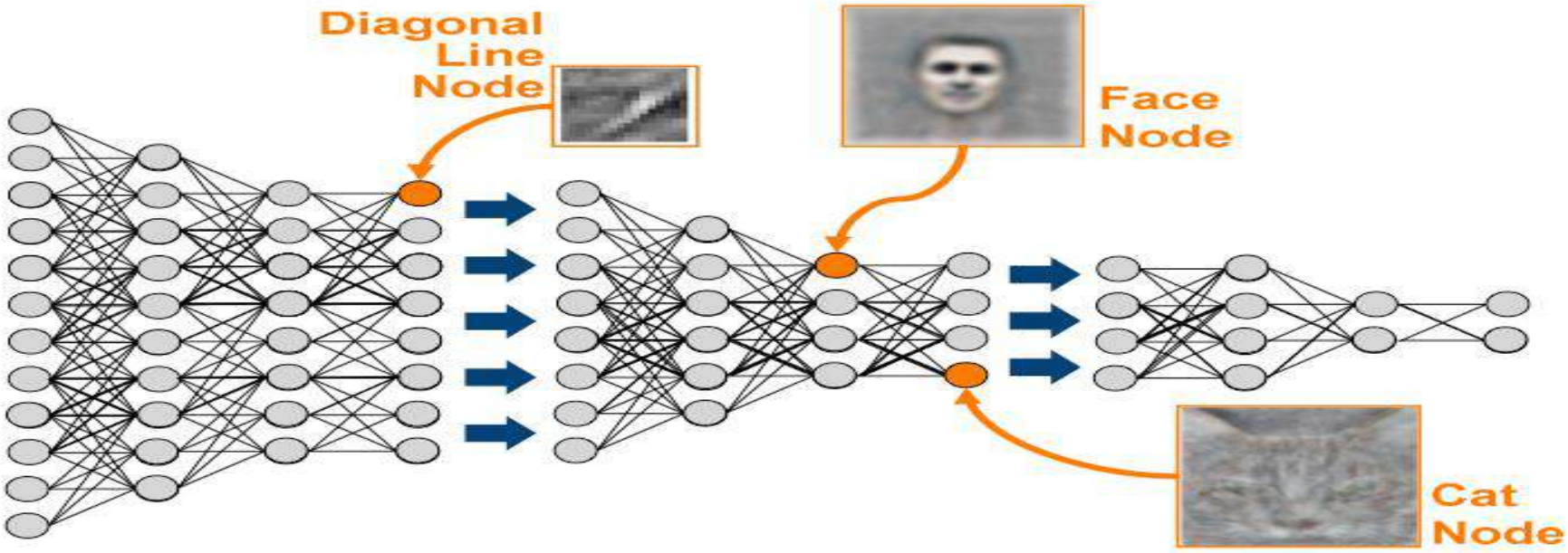
- Learning deep (many layered) neural networks

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

Deep Learning definition

E.g. Classify a cat:

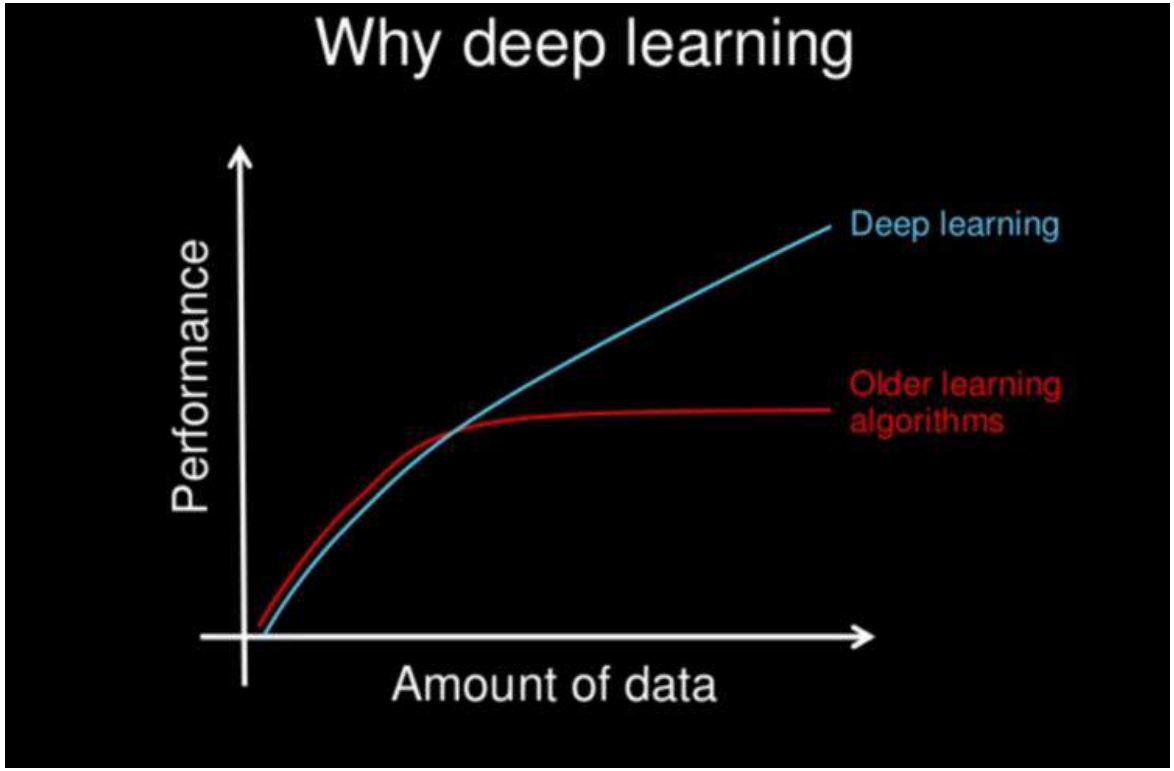
- Bottom Layers: Edge detectors, curves, corners straight lines
- Middle Layers: Fur patterns, eyes, ears
- Higher Layers: Body, head, legs
- Top Layer: Cat or Dog



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.

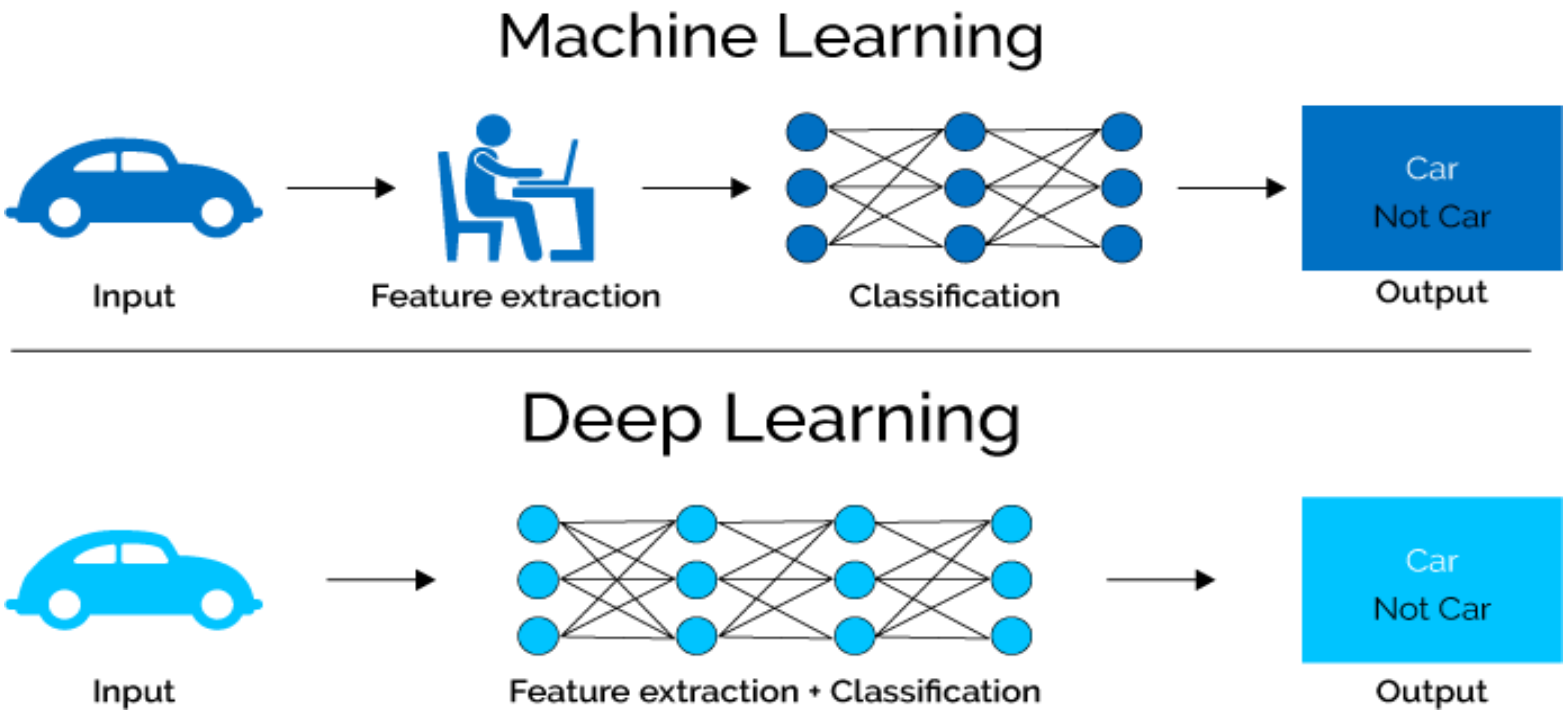
3- Execution time

- 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.

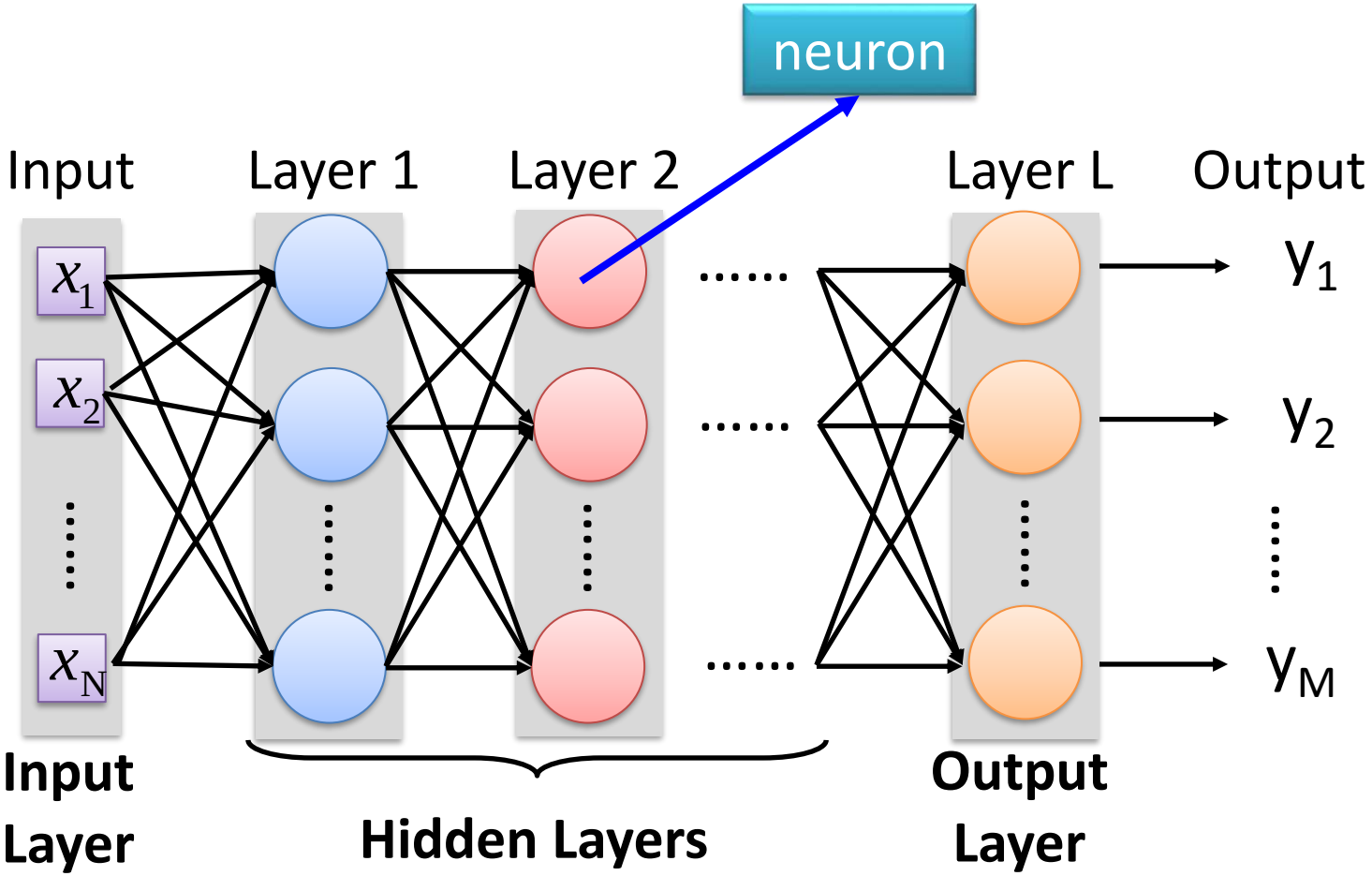
Machine Learning VS Deep Learning

4- Feature engineering

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



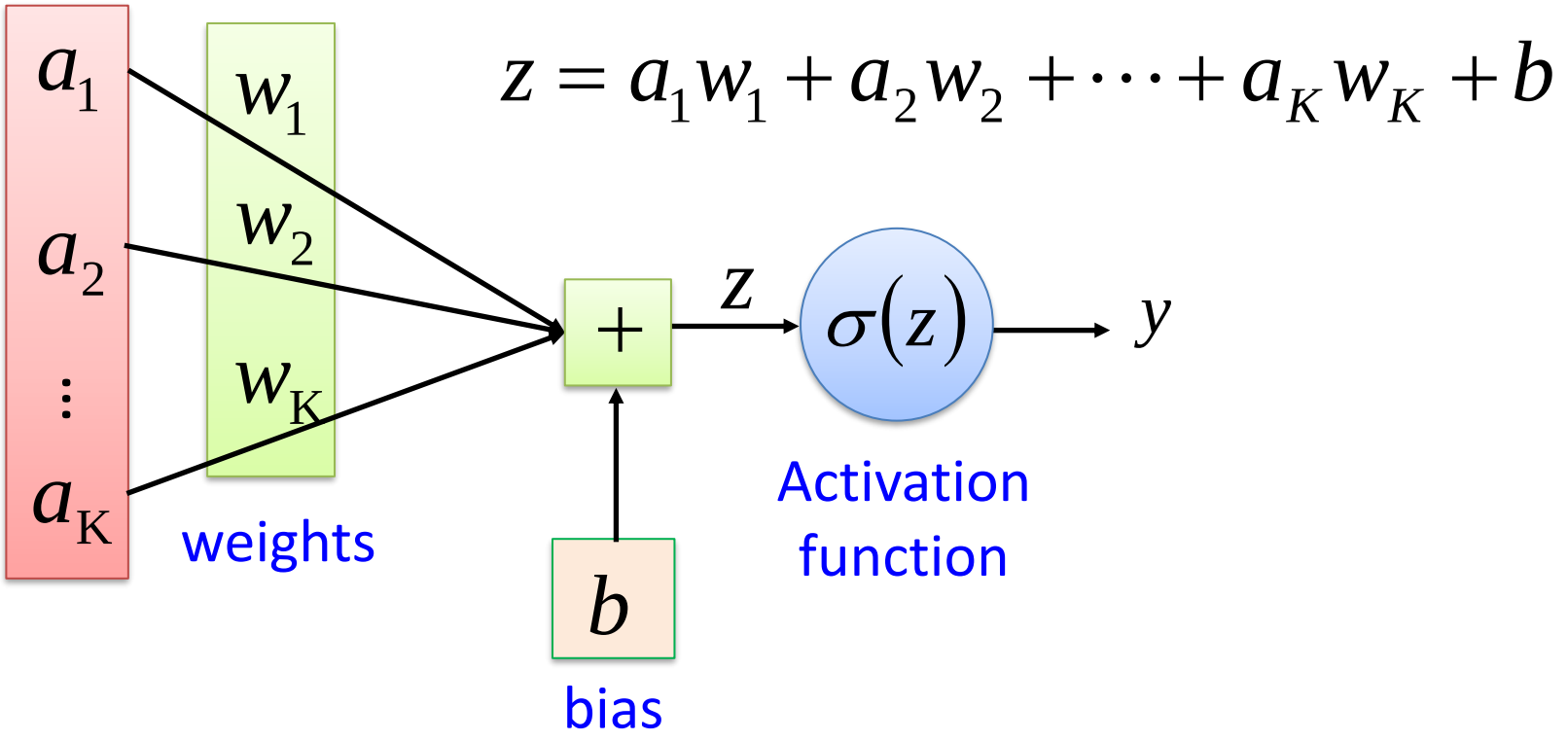
Element of Neural Network



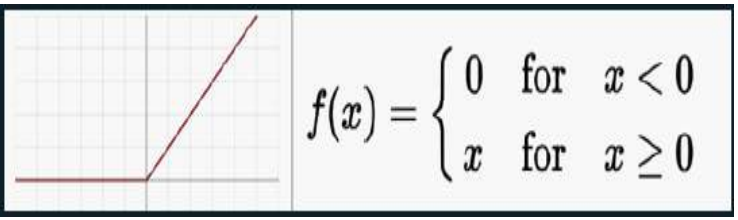
Deep means many hidden layers

Neural Network

Neuron $f: R^K \rightarrow R$

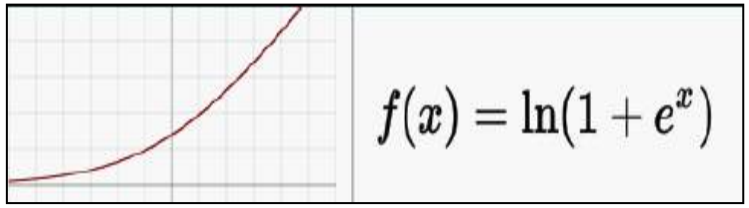


Activation Function types



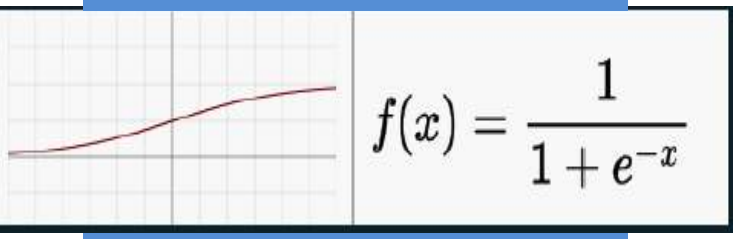
$$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases}$$

ReLU



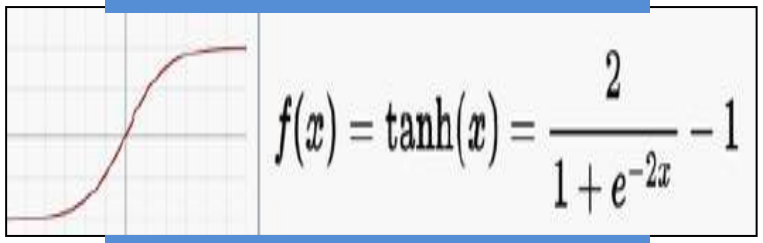
$$f(x) = \ln(1 + e^{x})$$

Softplus



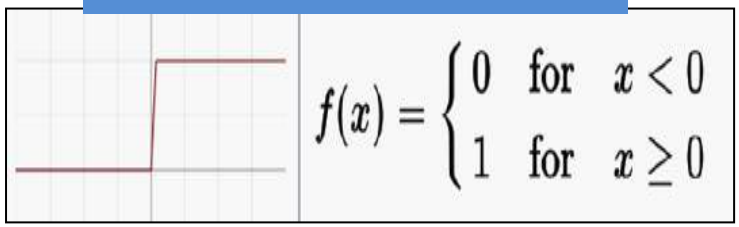
$$f(x) = \frac{1}{1 + e^{-x}}$$

Sigmoid/logistic



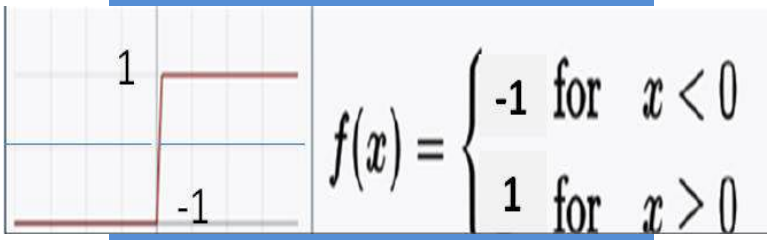
$$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$$

Tanh



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

Binary



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

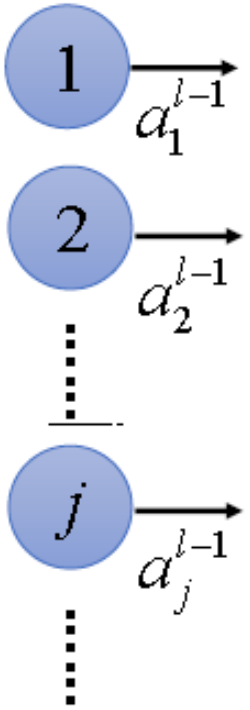
Signum

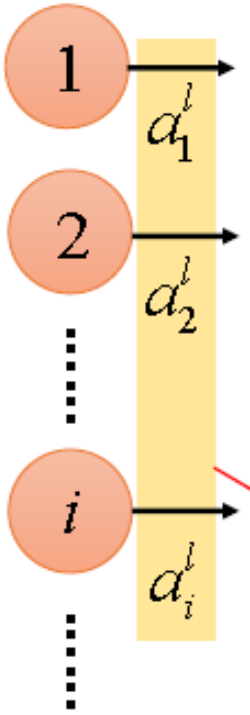
$$f_i(\vec{x}) = \frac{e^{x_i}}{\sum_{j=1}^{J} e^{x_j}} \quad \text{for } i = 1, \ldots, J$$

Softmax

Fully Connected Layer

Vanilla





Output of a neuron:

a_i^l → Layer l

a_i^l → Neuron i

Output of one layer:

a^l : a vector

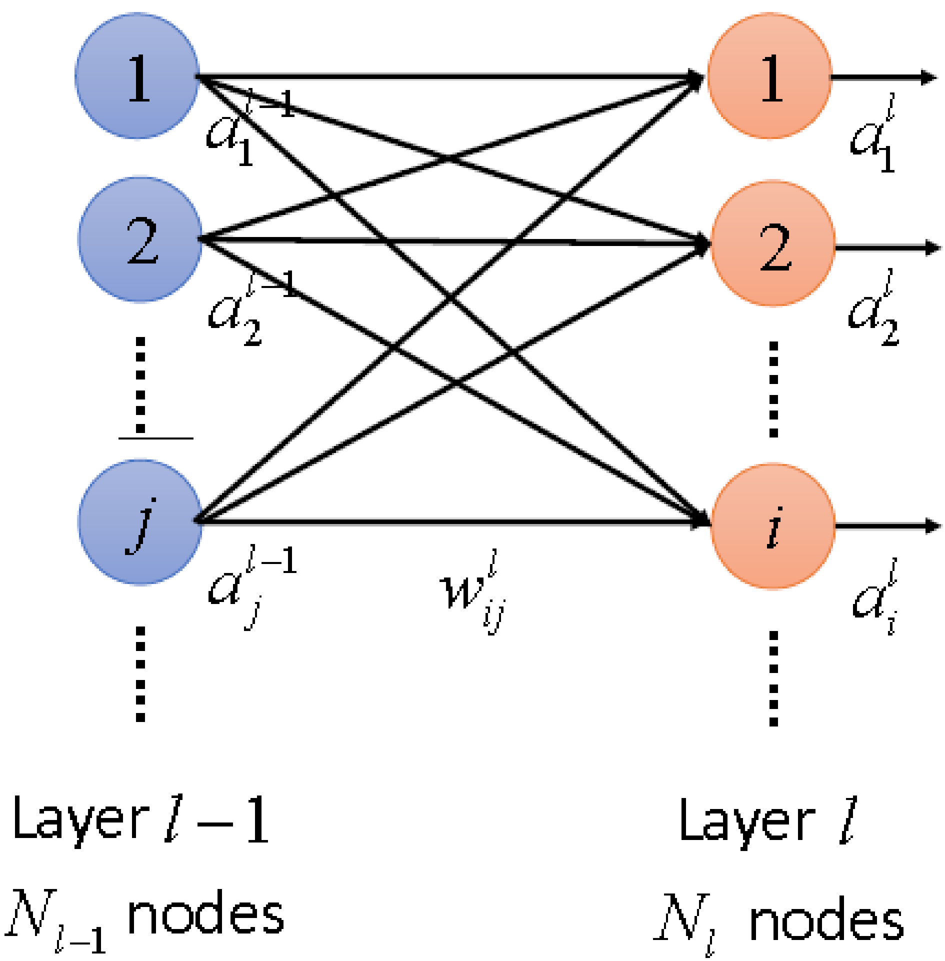
Layer $l-1$

N_{l-1} nodes

Layer l

N_l nodes

Fully Connected Layer



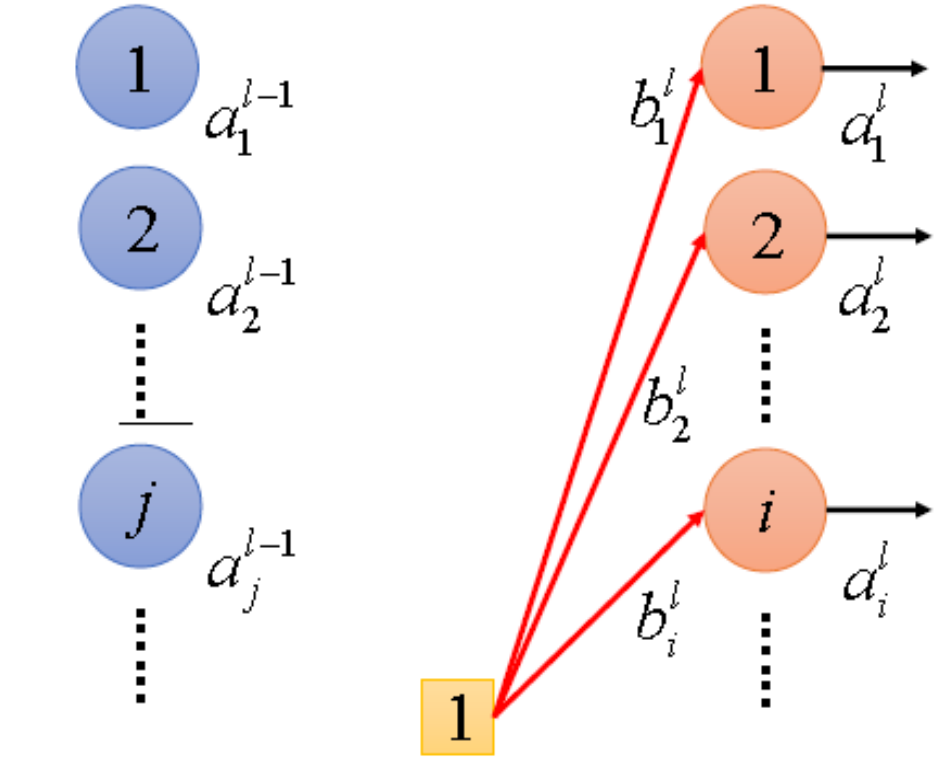
w_{ij}^{l} → Layer $l-1$ to Layer l

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

$$W^{l} = \underbrace{\begin{bmatrix} w_{11}^{l} & w_{12}^{l} & \cdots \\ w_{21}^{l} & w_{22}^{l} & \\ \vdots & & \ddots \end{bmatrix}}_{}$$

Columns: N_{l-1}; Rows: N_{l}

Fully Connected Layer

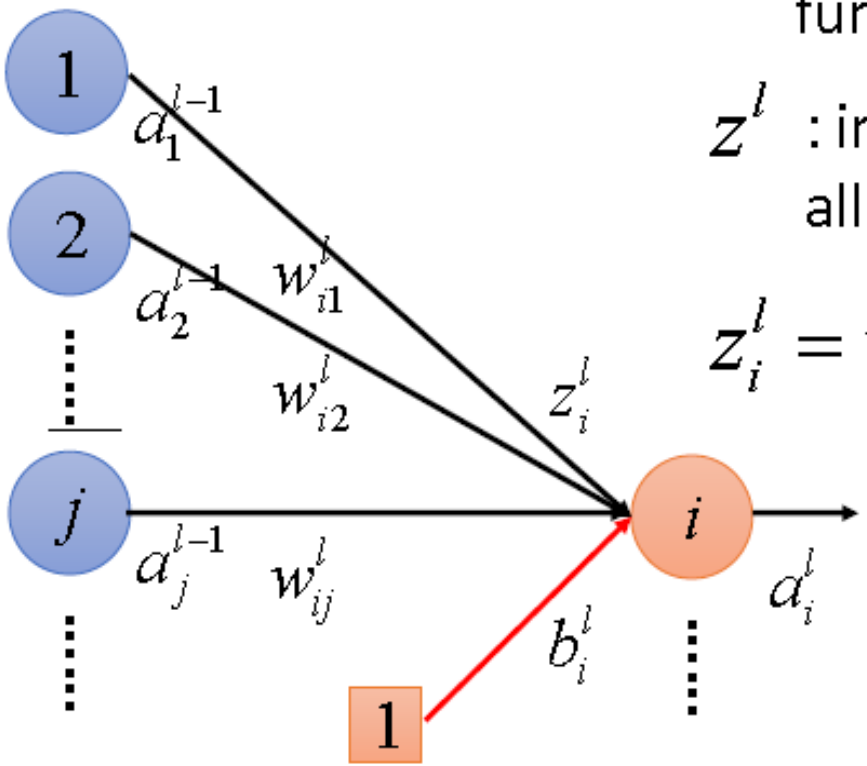


b_i^l : bias for neuron i at layer l

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

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

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

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

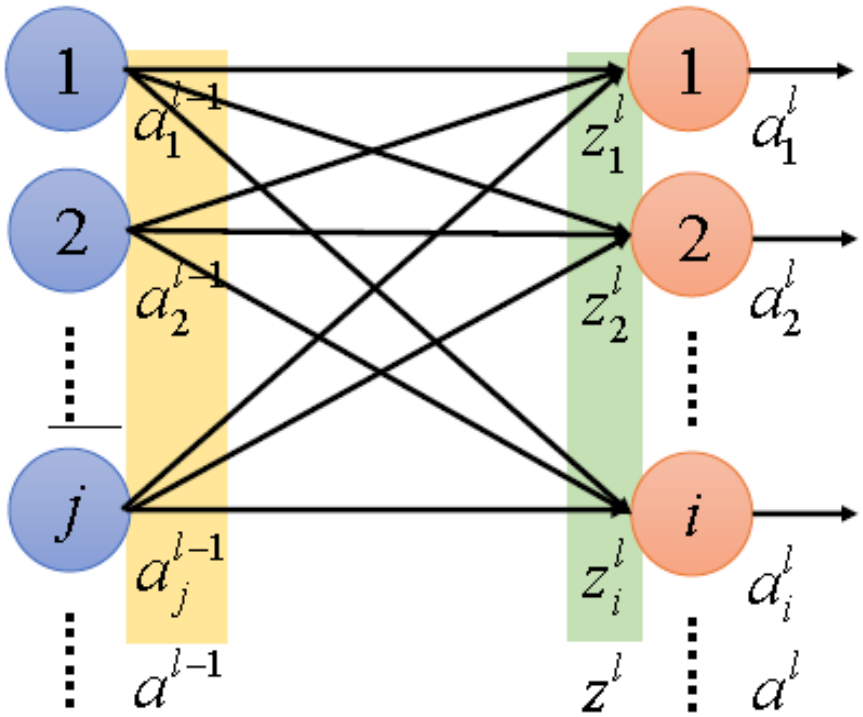
Layer $l-1$

N_{l-1} nodes

Layer l

N_l nodes

Relations between Layer Outputs



Layer $l-1$

N_{l-1} nodes

Layer l

N_l nodes

$$z_1^l = w_{11}^l a_1^{l-1} + w_{12}^l a_2^{l-1} + \cdots + b_1^l$$

$$z_2^l = w_{21}^l a_1^{l-1} + w_{22}^l a_2^{l-1} + \cdots + b_2^l$$

$$\vdots$$

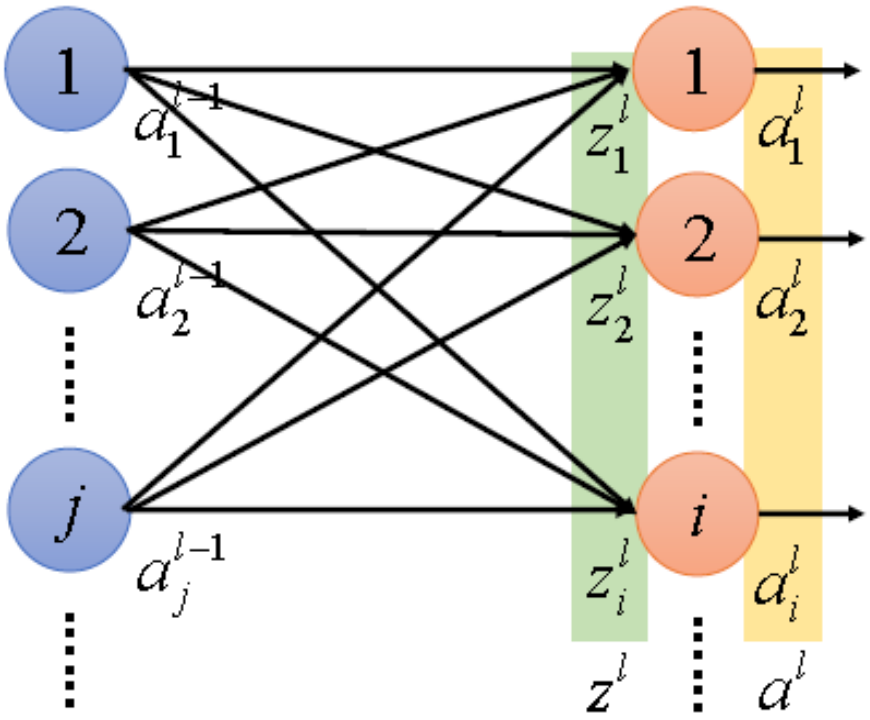
$$z_i^l = w_{i1}^l a_1^{l-1} + w_{i2}^l a_2^{l-1} + \cdots + b_i^l$$

$$\vdots$$

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

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

Relations between Layer Outputs



Layer $l-1$

N_{l-1} nodes

Layer l

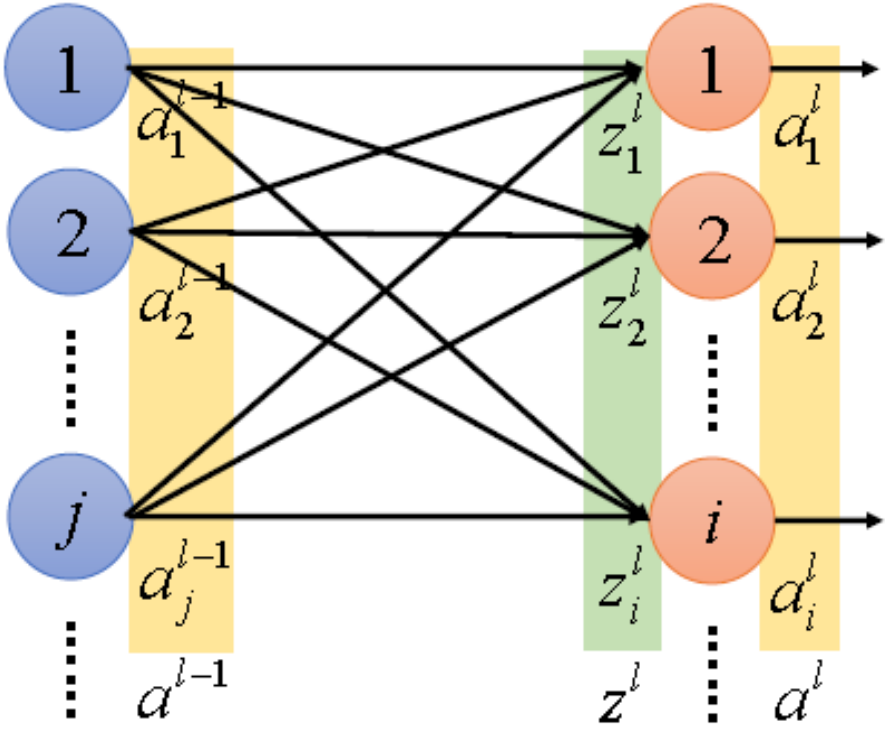
N_l nodes

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

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

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

Relations between Layer Outputs



Layer $l-1$

N_{l-1} nodes

Layer l

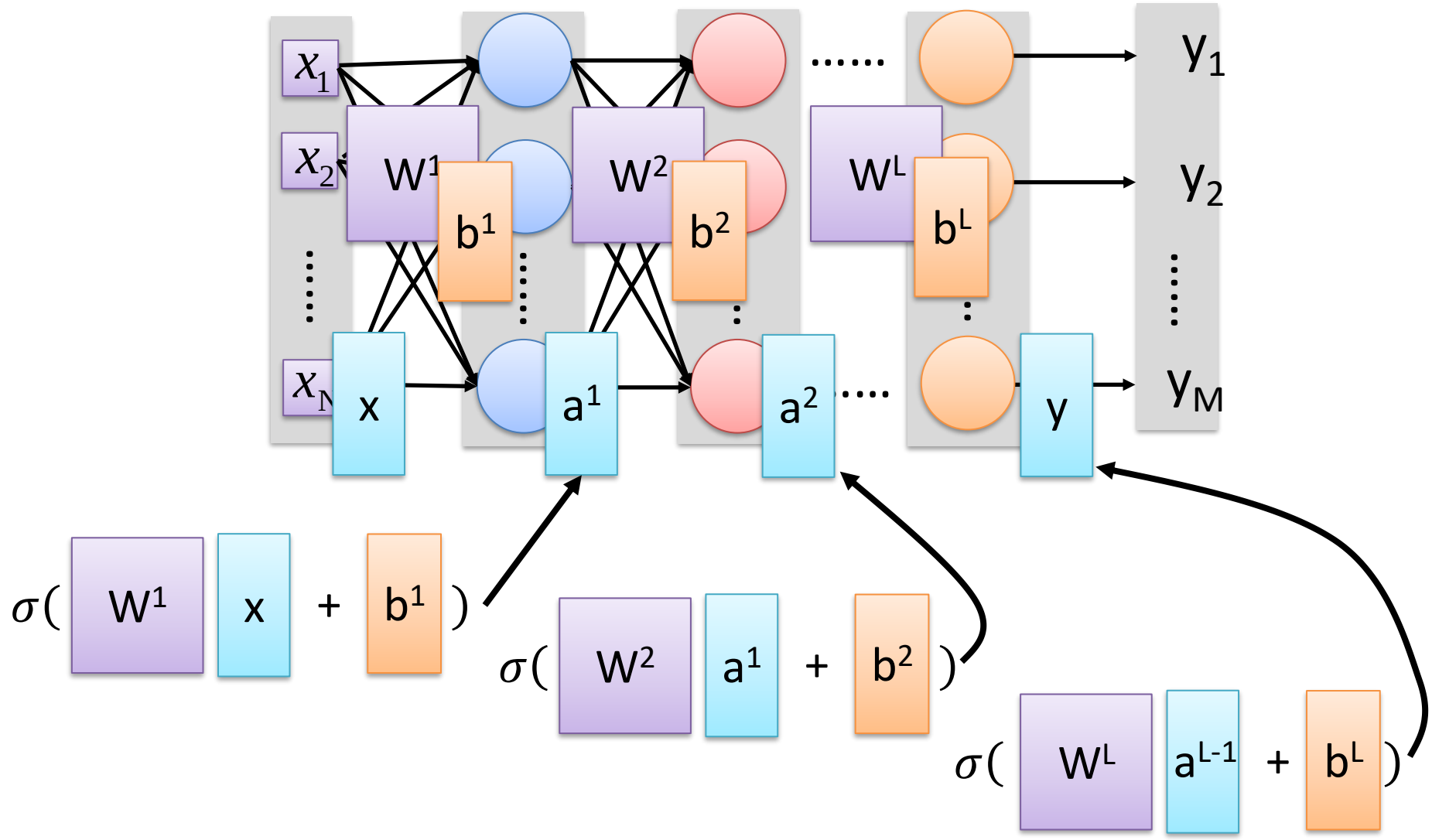
N_l nodes

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

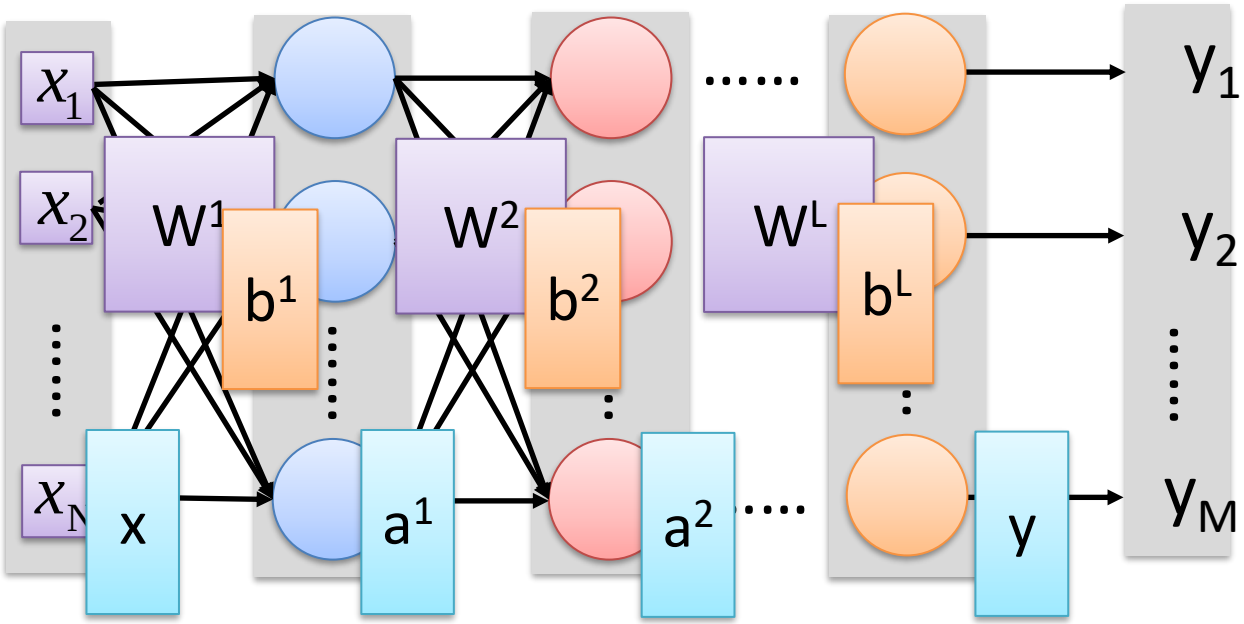
$$a^l = \sigma\left(z^l\right)$$

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

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

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

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?

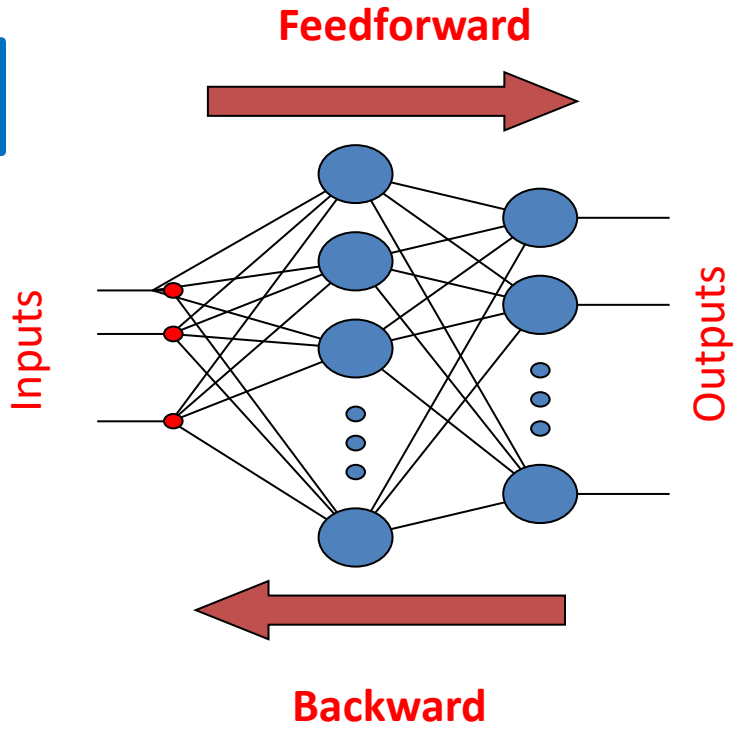
Q. How increasing or decreasing weights affects the prediction error?

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**

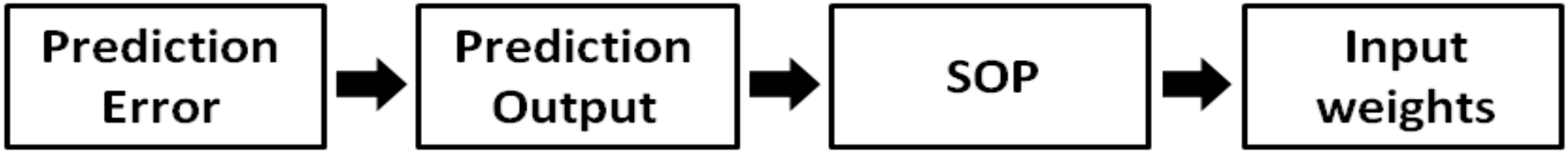
Regarding 5th step: Weights Adaptation

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 $\frac{\partial E}{\partial W}$



$$E = \frac{1}{2}(d - y)^2 \qquad f(s) = \frac{1}{1+e^{-s}} \qquad s = \sum_j^m x_i\, w_{ji} + b_i \qquad w_1, w_2$$

d (desired output) Const
y (predicted output)

s (Sum Of Product SOP)

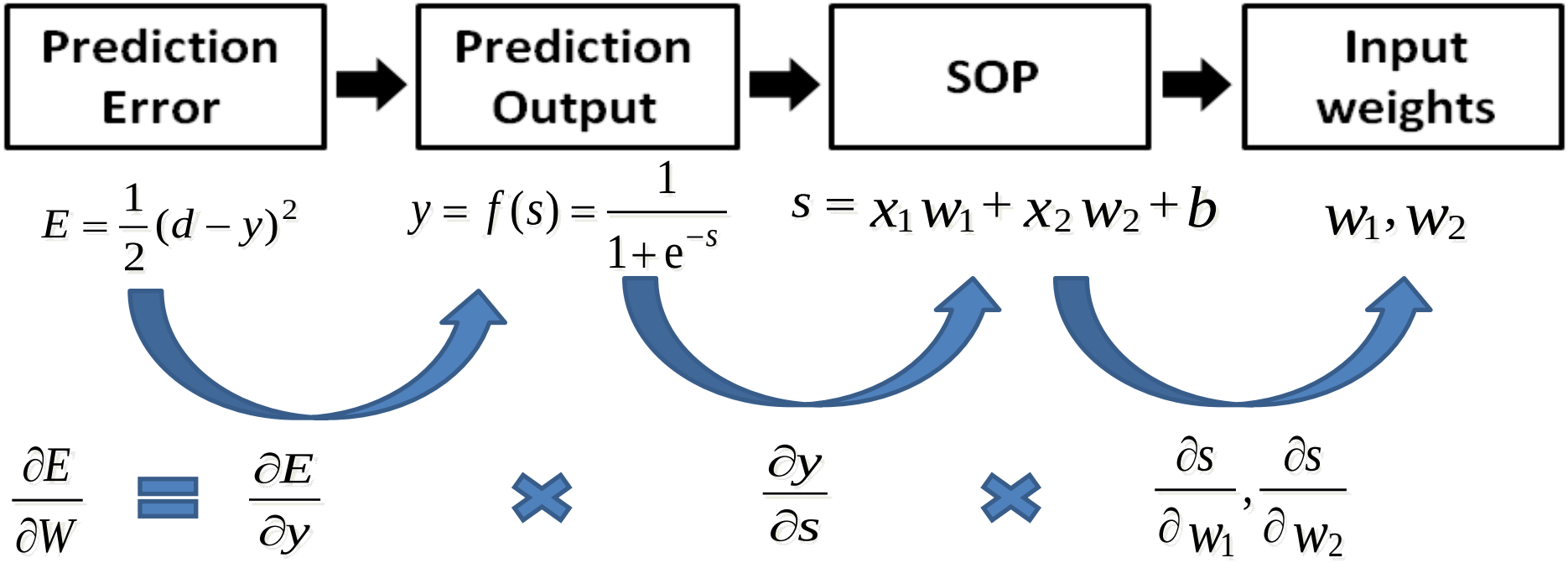
$$E = \frac{1}{2}\left(d - \frac{1}{e^{-\sum_j^n x_i w_{ij} + b_i}}\right)^2$$

Regarding 5th step: Weights Adaptation

second method: Back propagation

Chain Rule

- **Weight derivative**



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

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

Regarding 5th step: Weights Adaptation

second method: Back propagation

- **Weight derivative**

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

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

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

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

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

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.

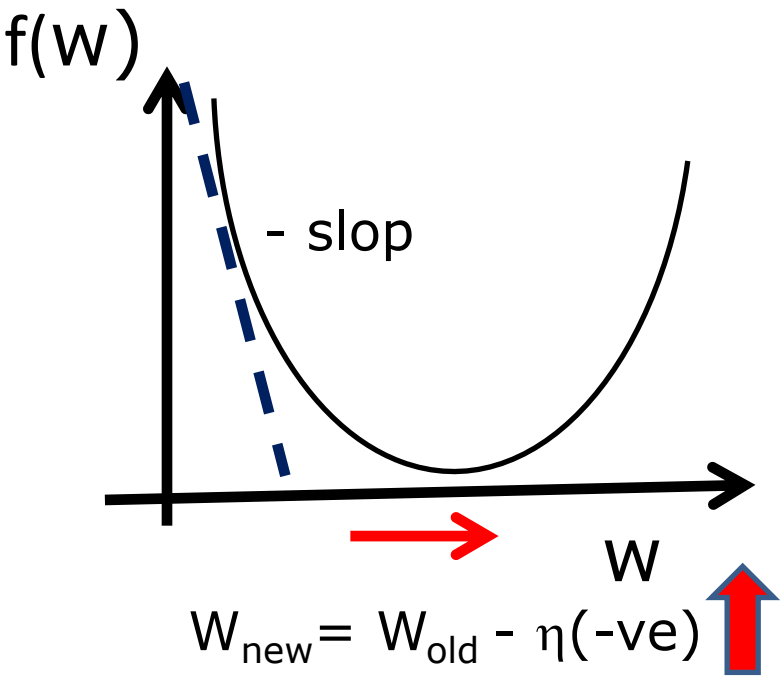
Regarding 5th step: Weights Adaptation

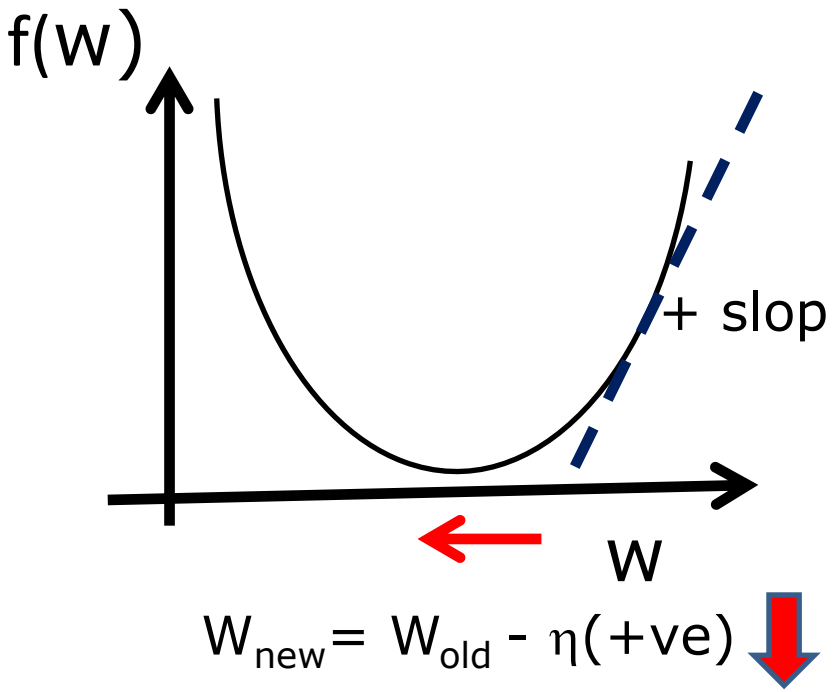
second method: Back propagation

- **Update the Weights**

  In order to update the weights , use the Gradient Descent

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





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

cat dog .....

Fully Connected Feedforward network

Convolution

Max Pooling

Convolution

Max Pooling

Can repeat many times

Flatten

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

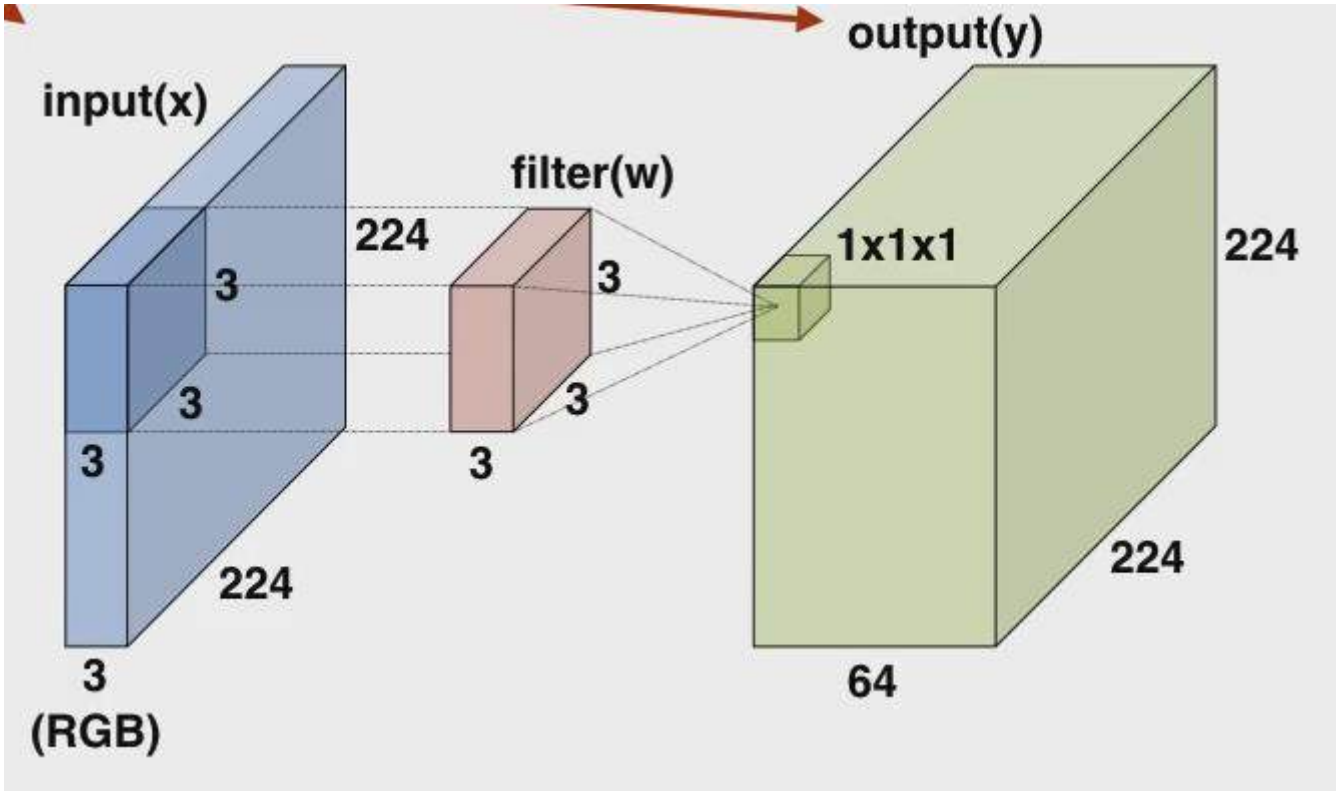
Max Pooling

Can repeat many times

Flatten

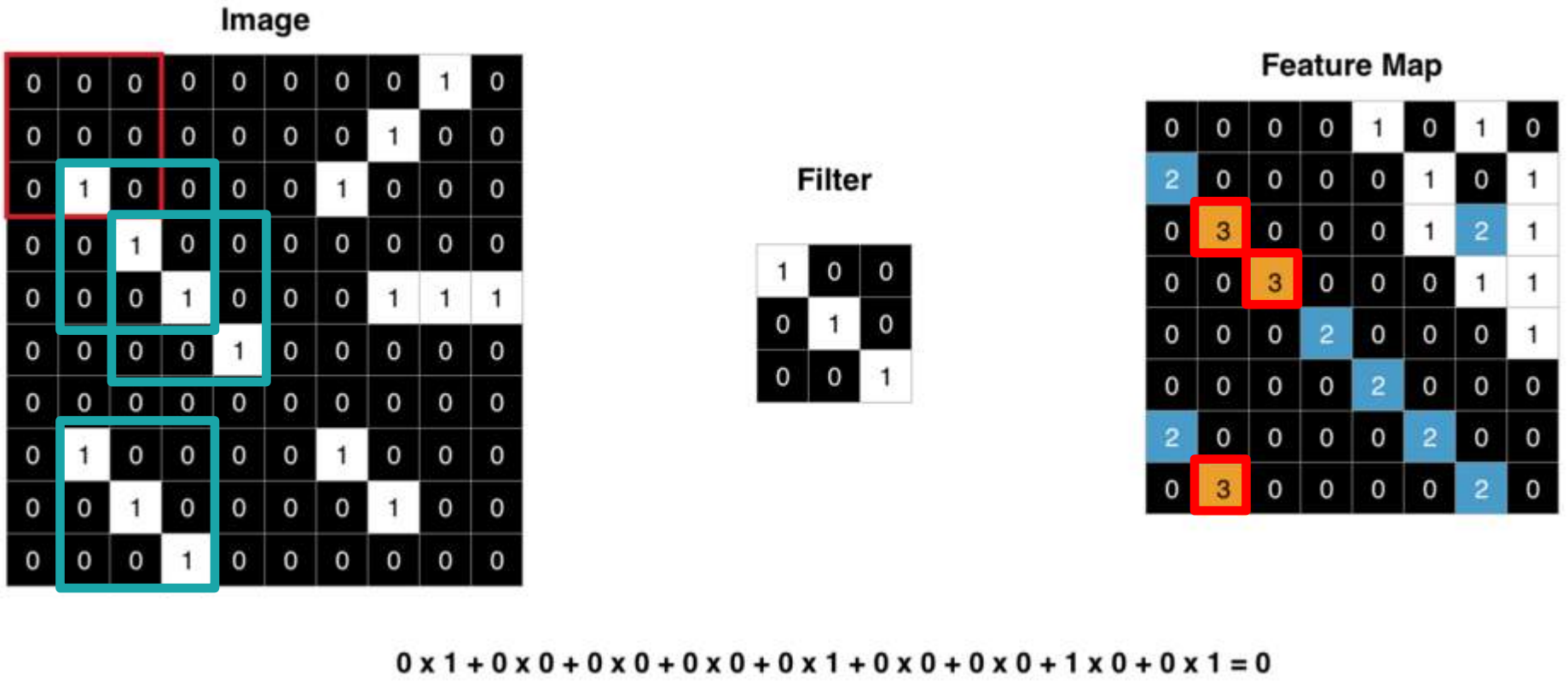
1- Convolution layer

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.

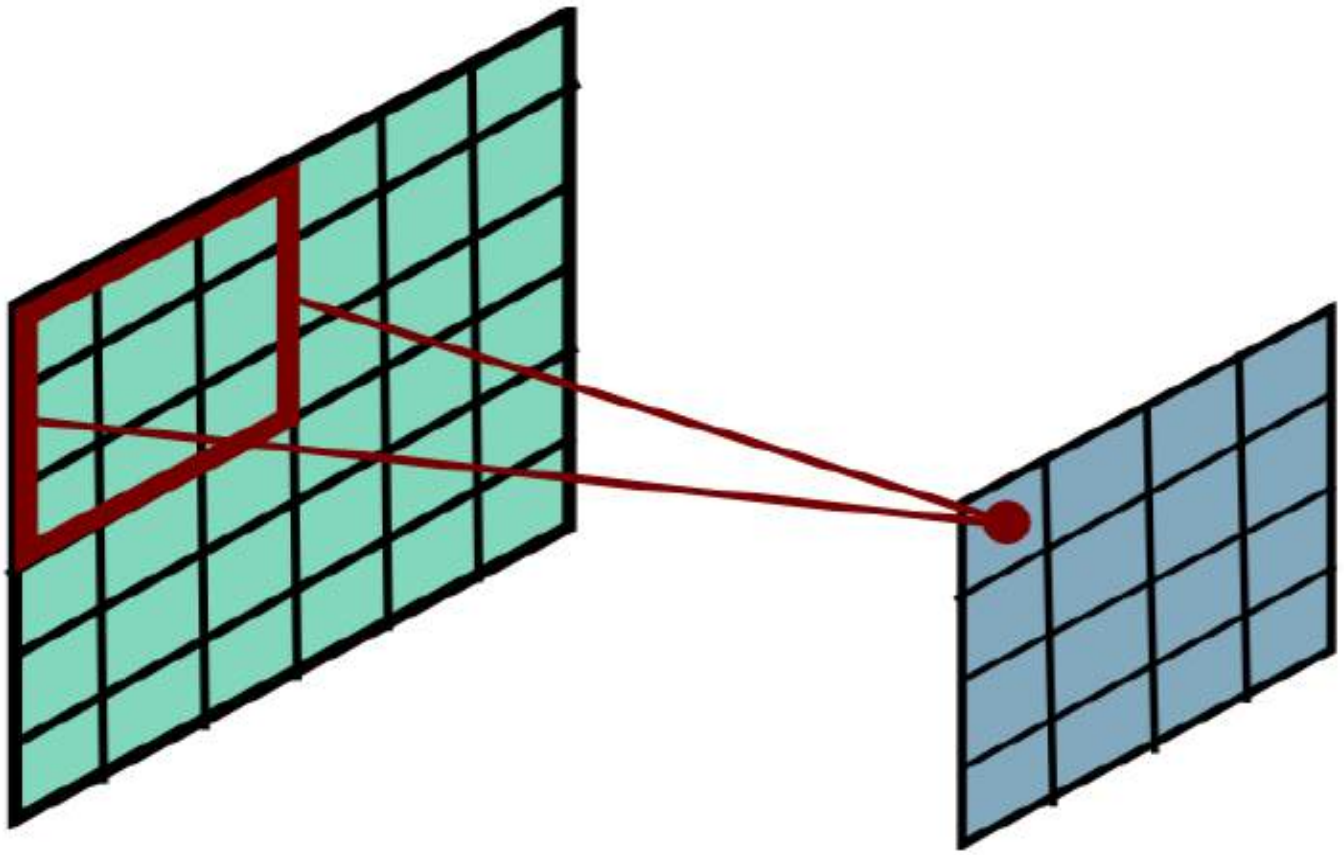


Convolution layer

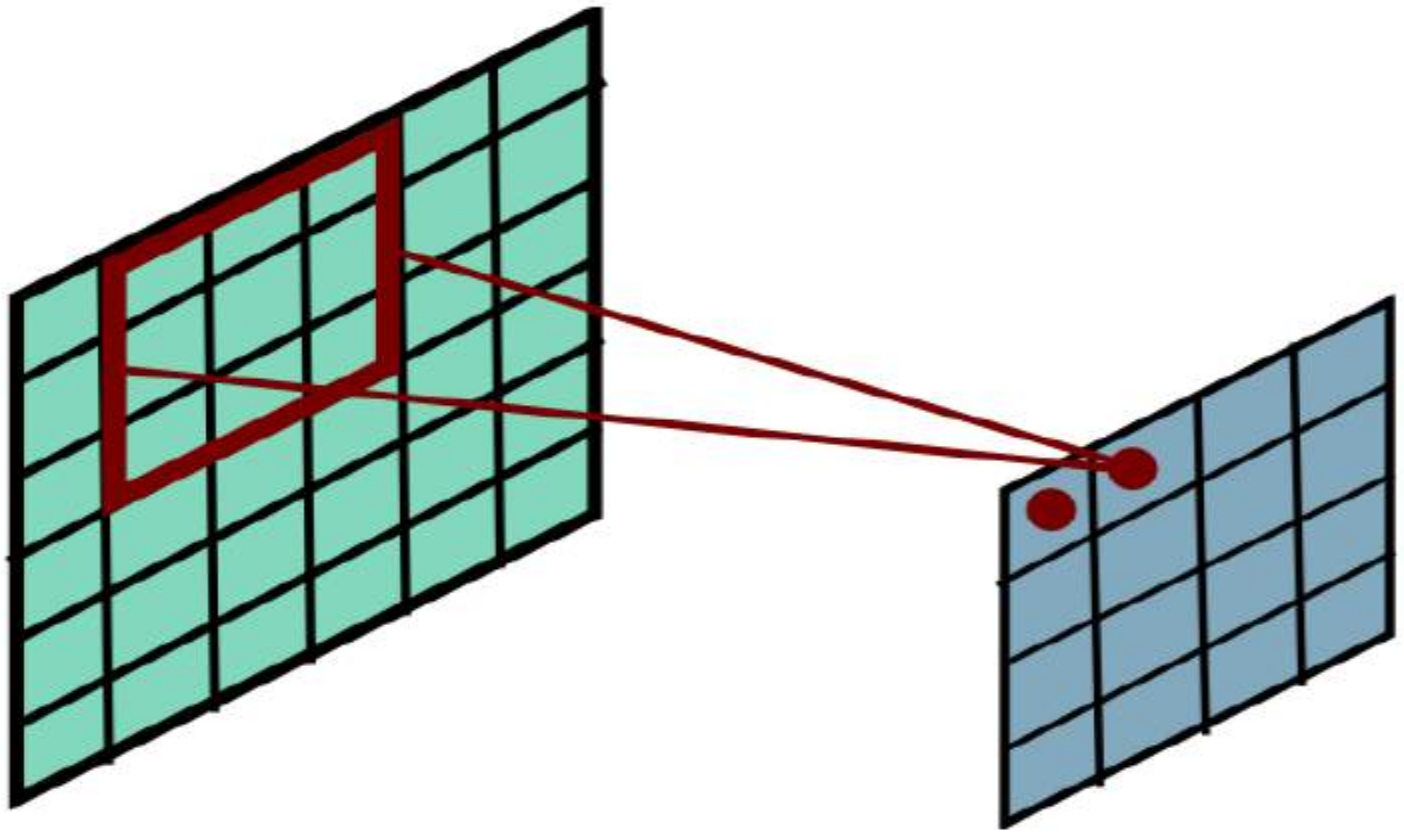
The convolution layer is always the first step in a convnet. Let's say we have a 10 x 10 pixel image, here represented by a 10 x 10 x 1 matrix of numbers:



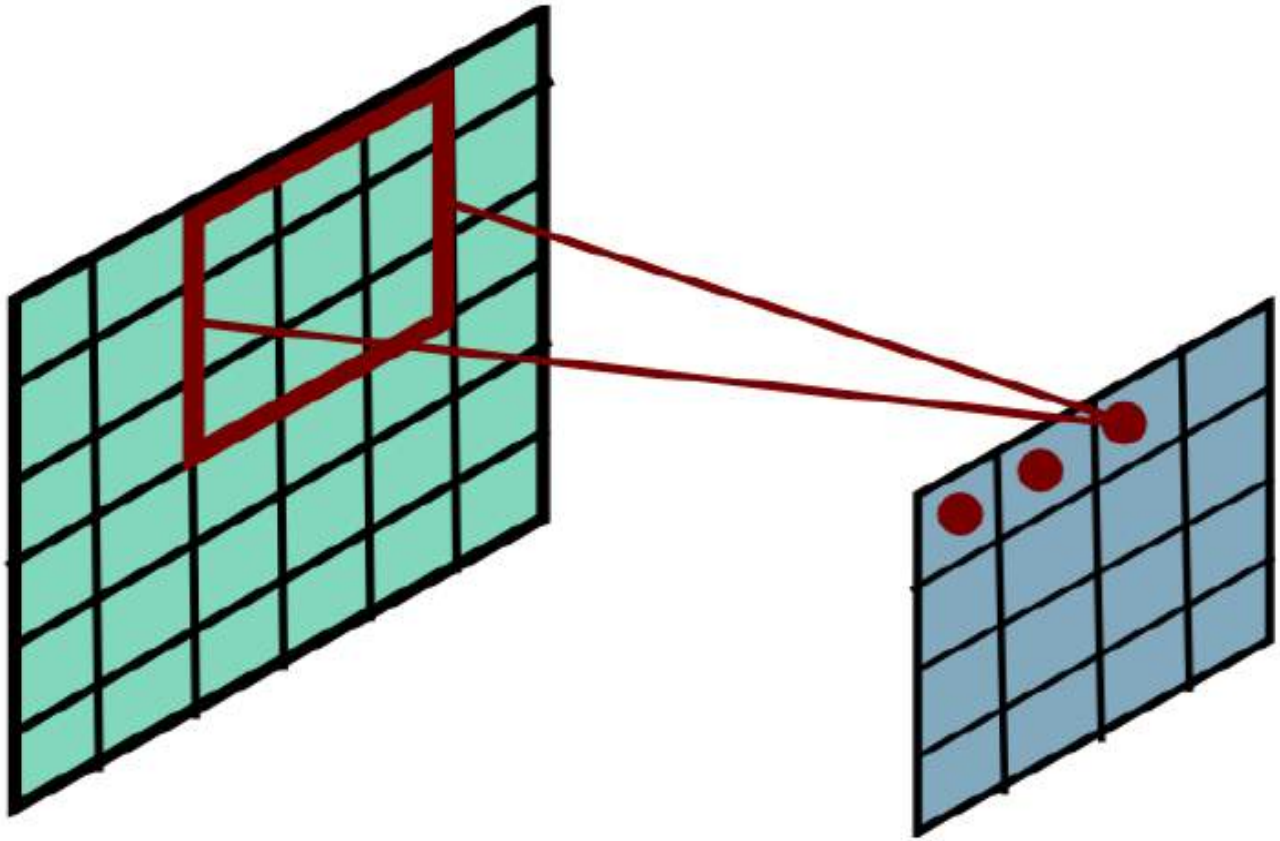
Convolutional Layer



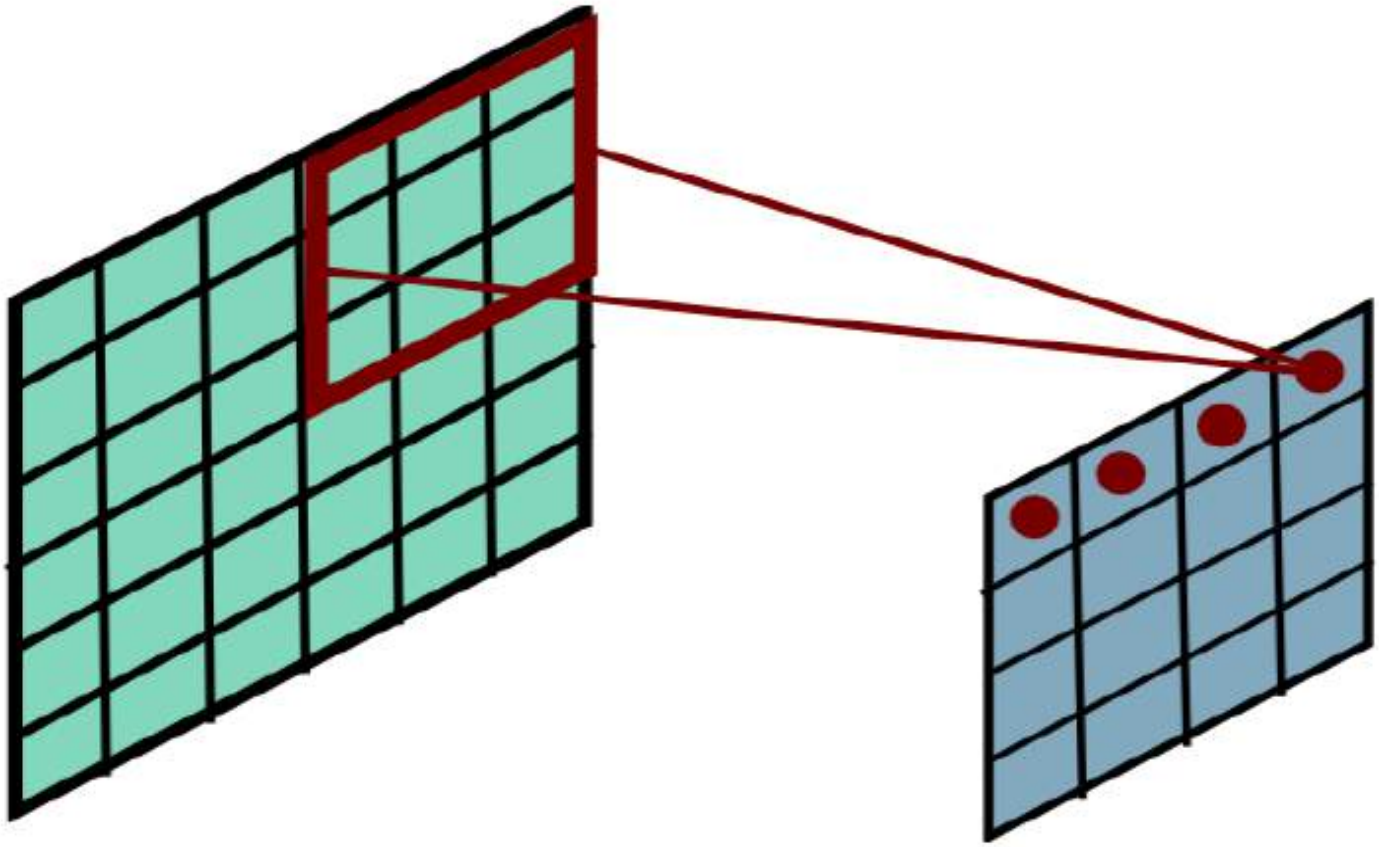
Convolutional Layer



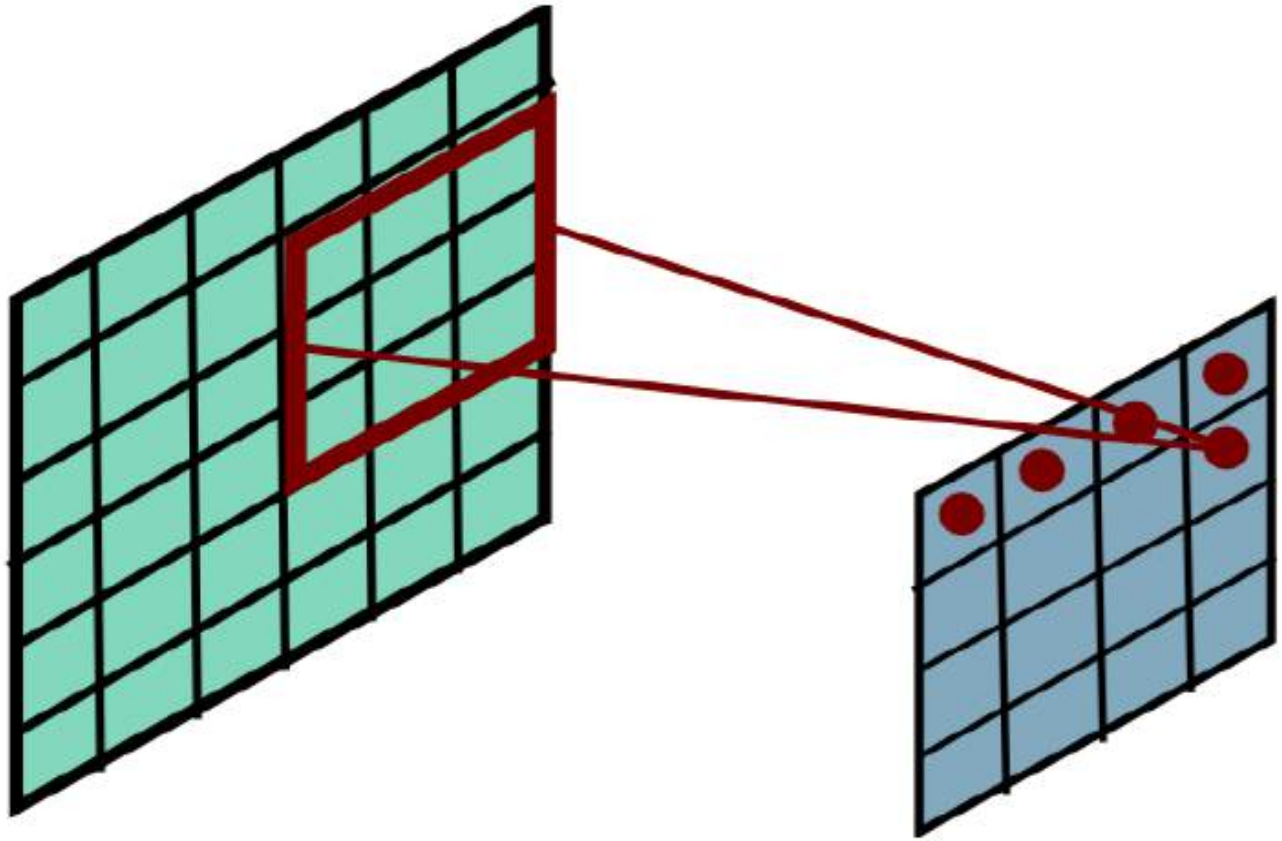
Convolutional Layer



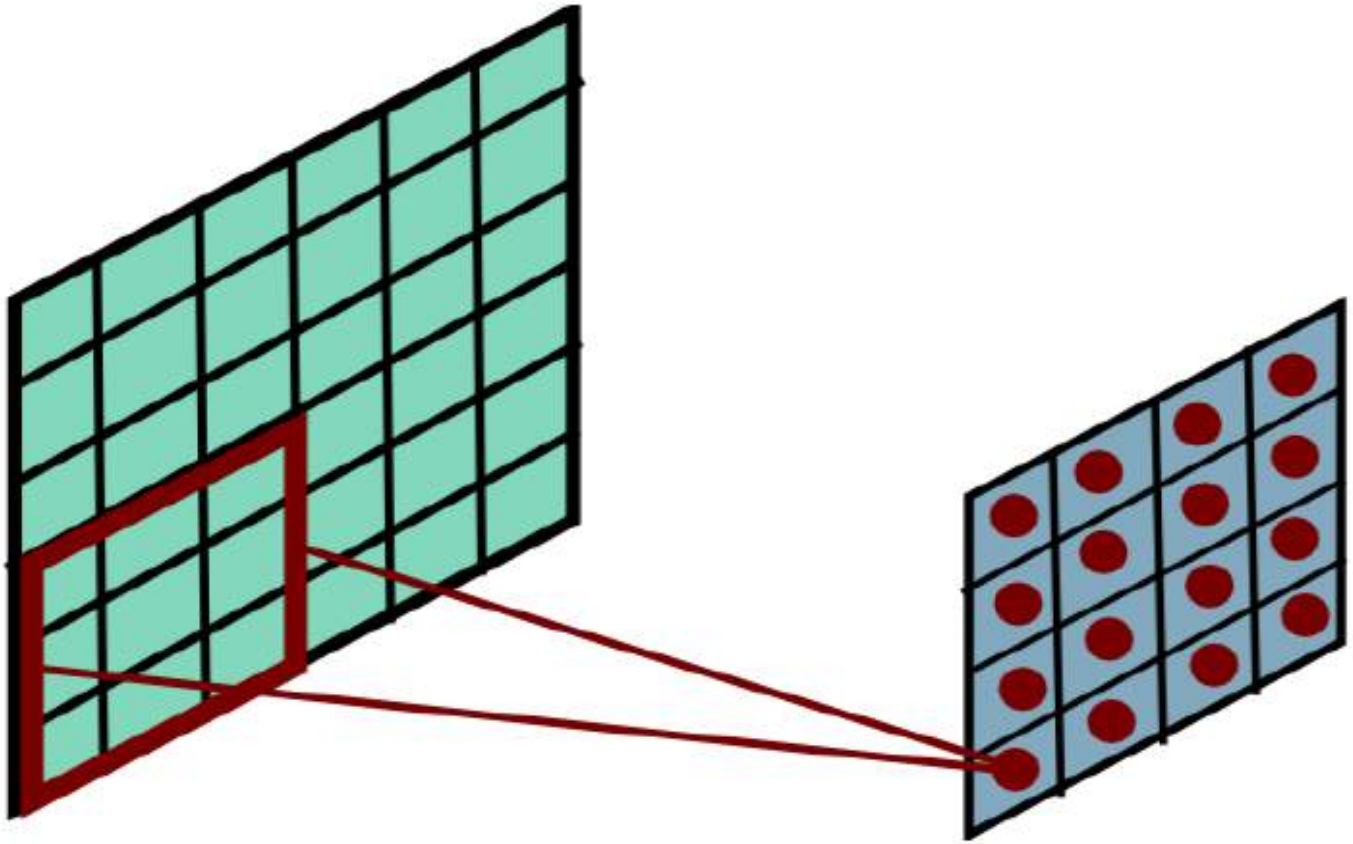
Convolutional Layer



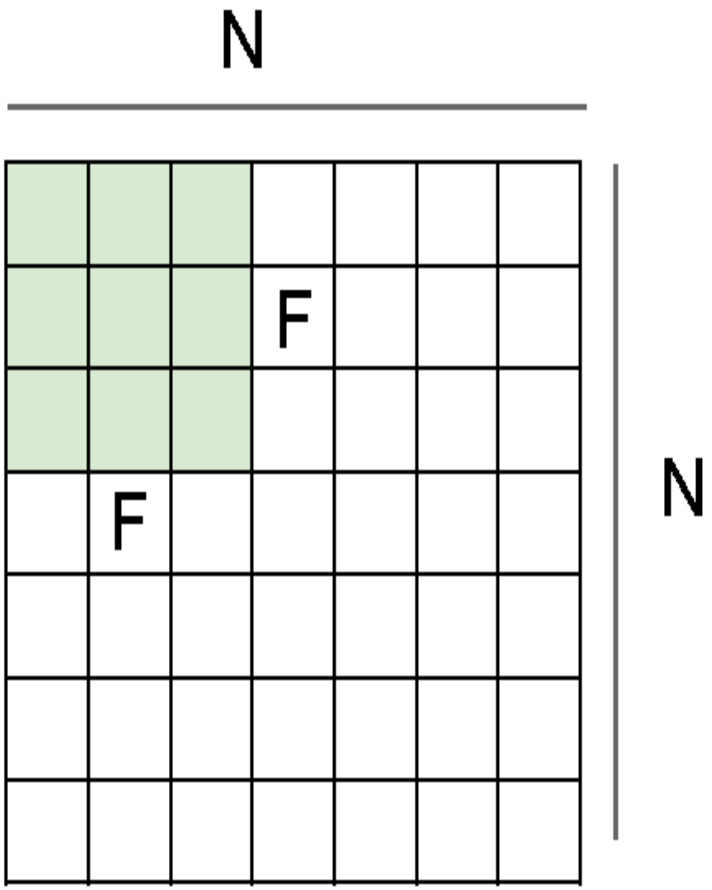
Convolutional Layer



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 :\

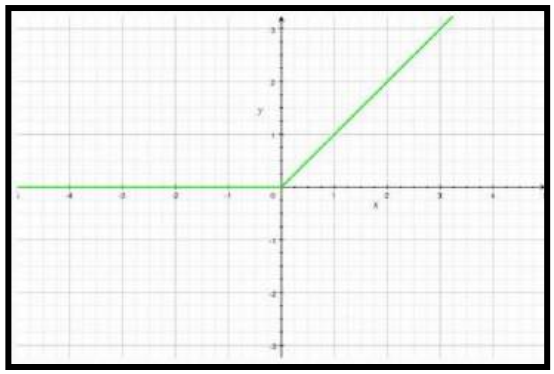
2- ReLU Layer $f(x) = \max(0,x)$

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

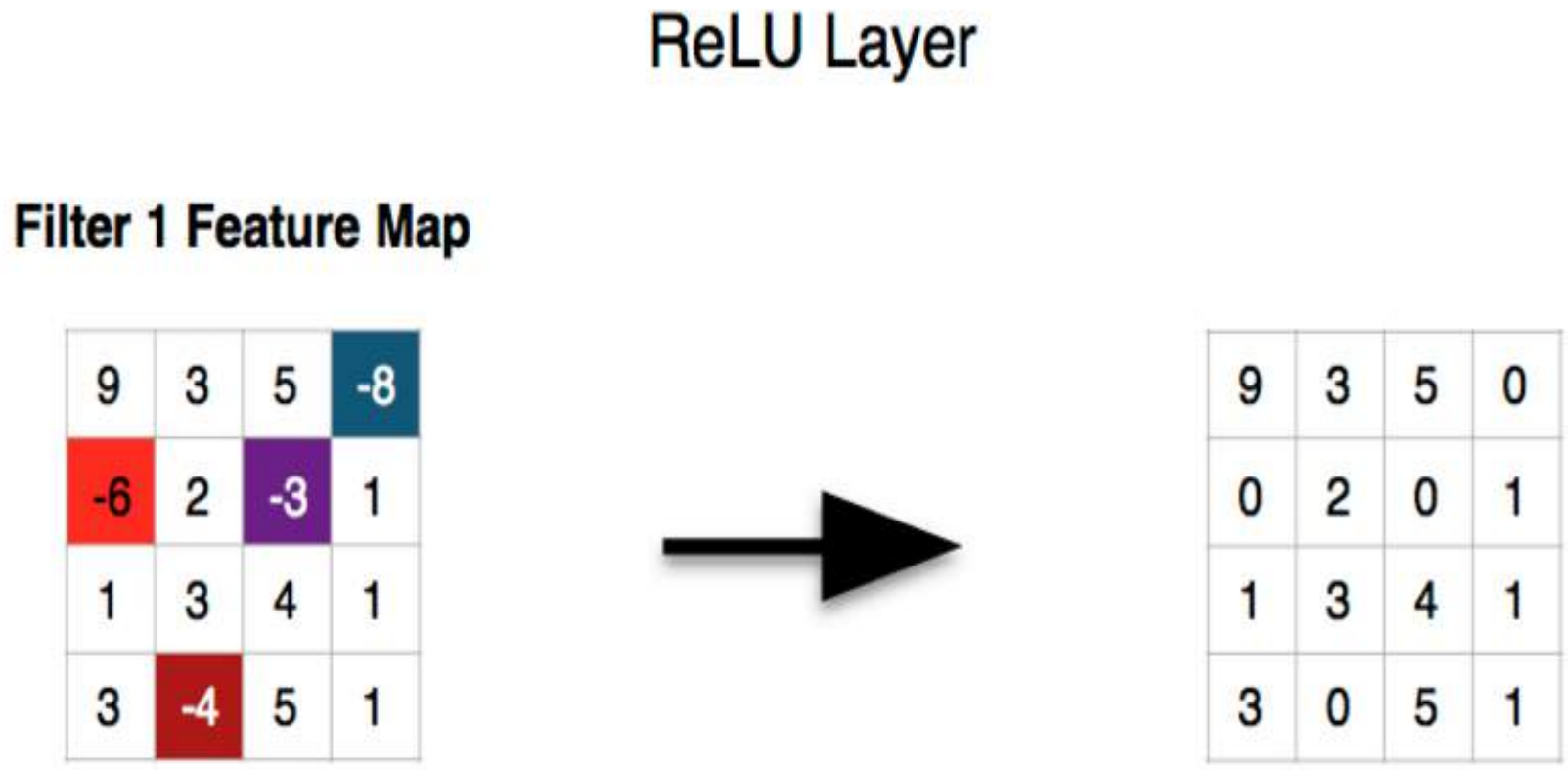
• The addition of the ReLU layer allows the neural network to account for non-linear relationships, i.e. the ReLU layer allows the convnet to account for situations in which the relationship between the pixel value inputs and the convnet output is not linear.

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

• 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)

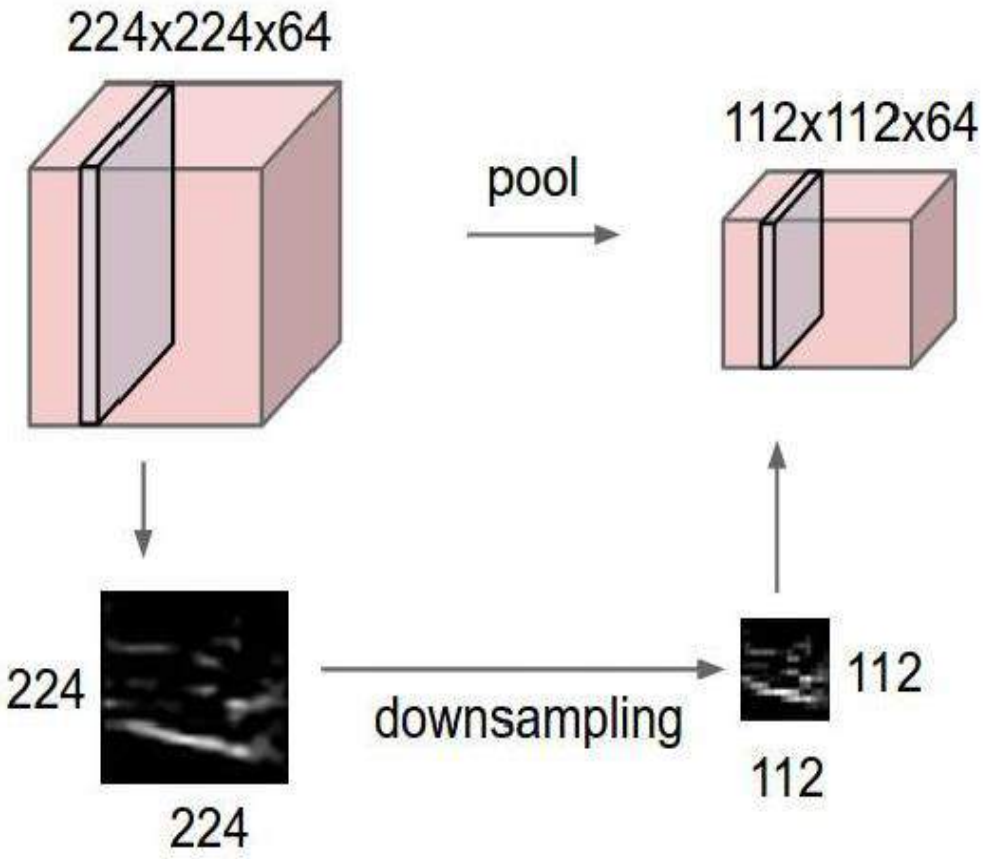


Other functions such as tanh or the sigmoid function can be used to add non-linearity to the network, but ReLU generally works better in practice.

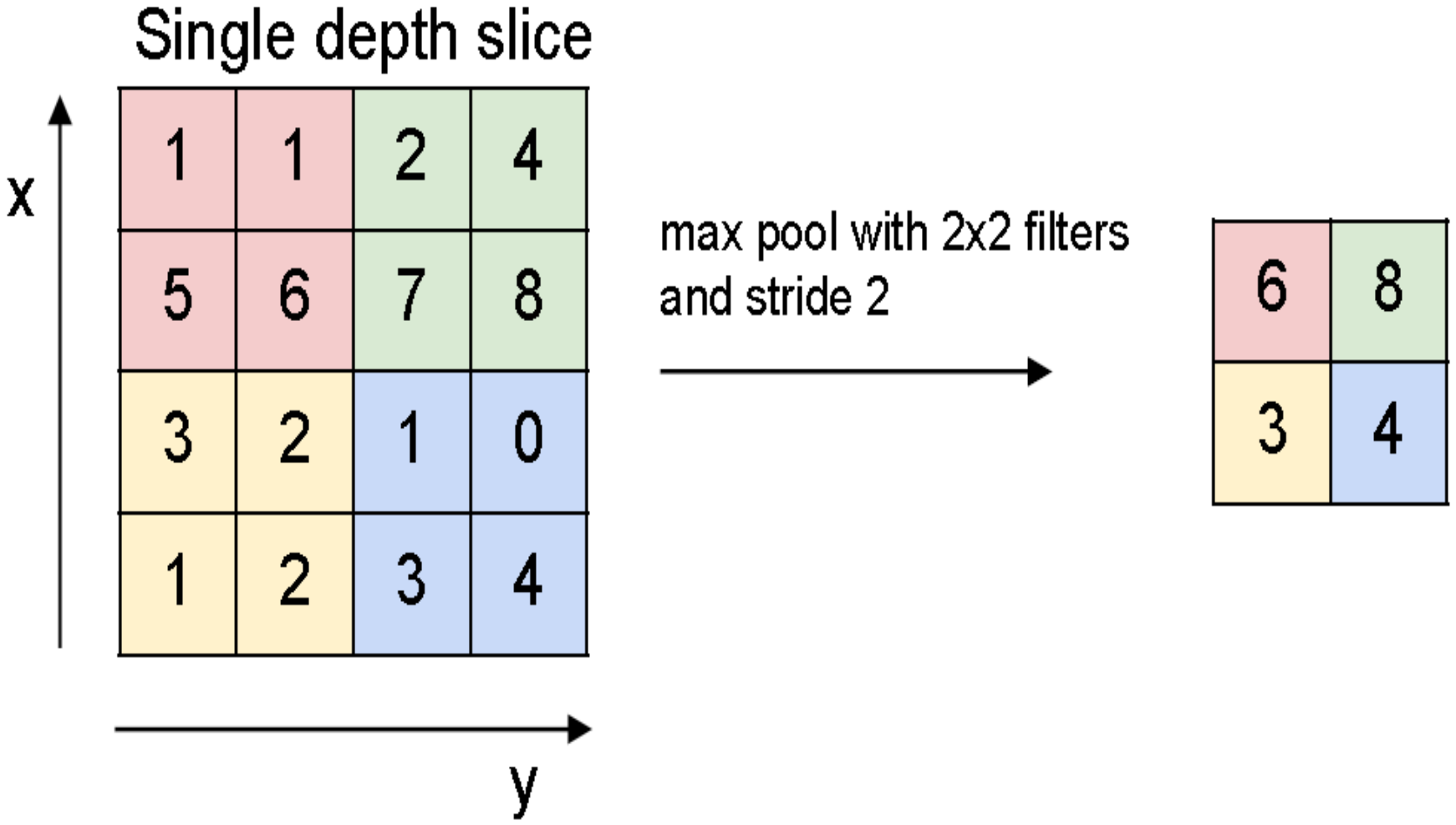
3- Pooling layer

• 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.

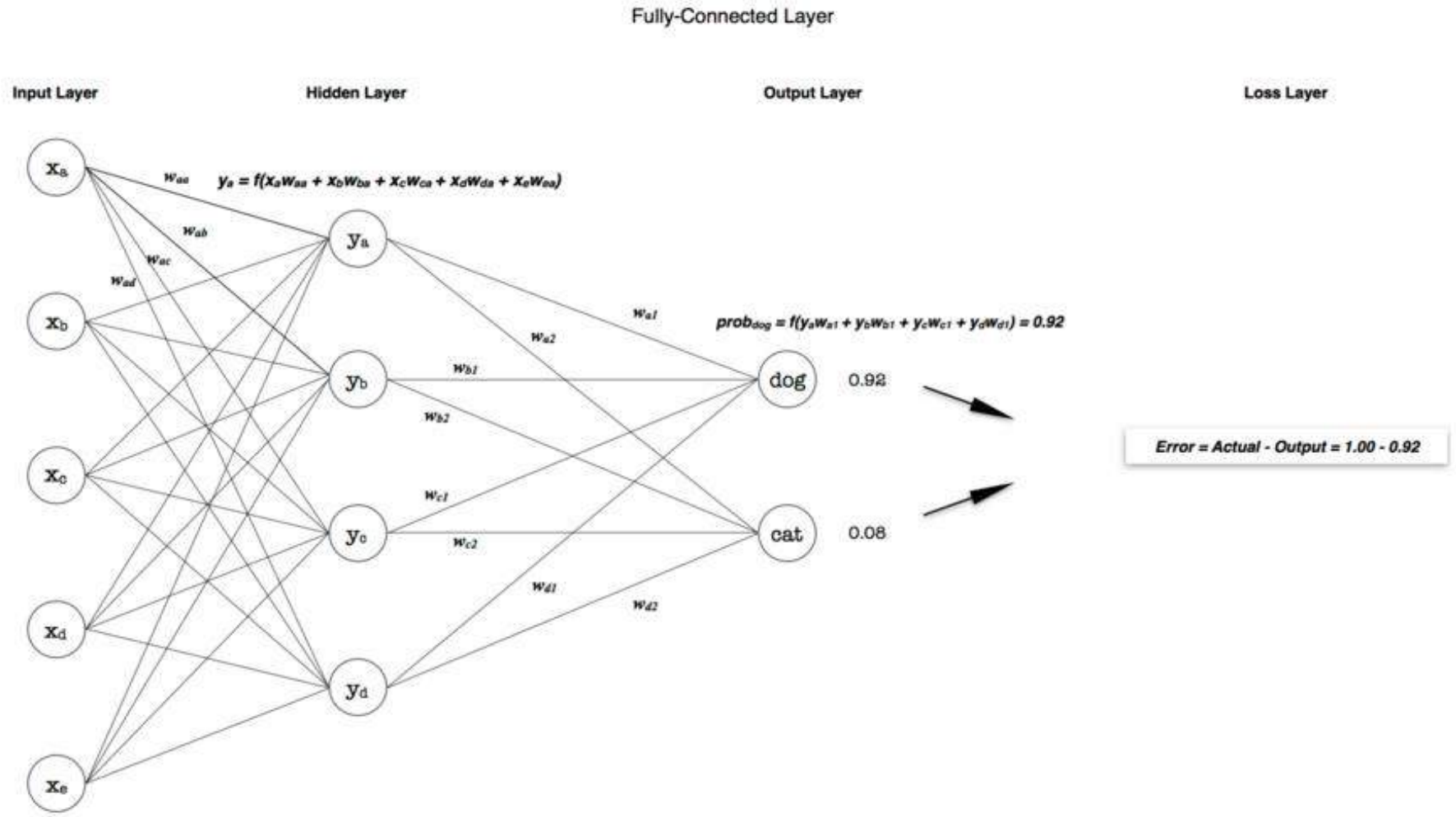


MAX POOLING

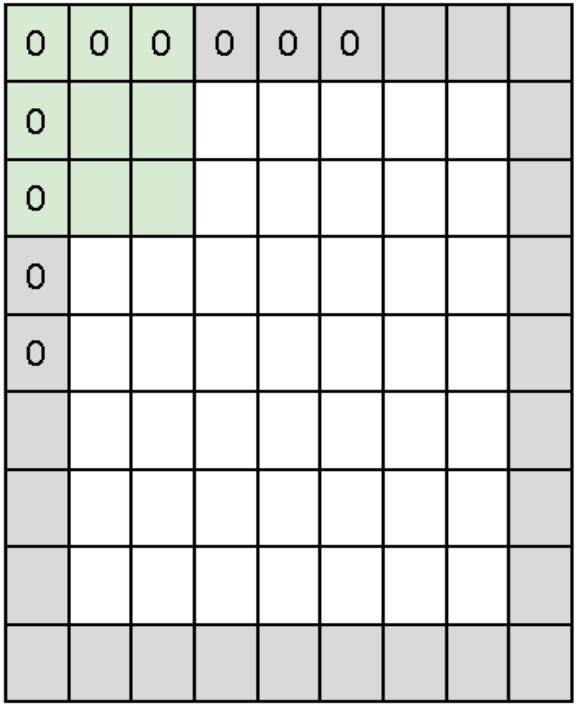

Single depth slice
x
y
1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
max pool with 2x2 filters
and stride 2
6 8
3 4

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!

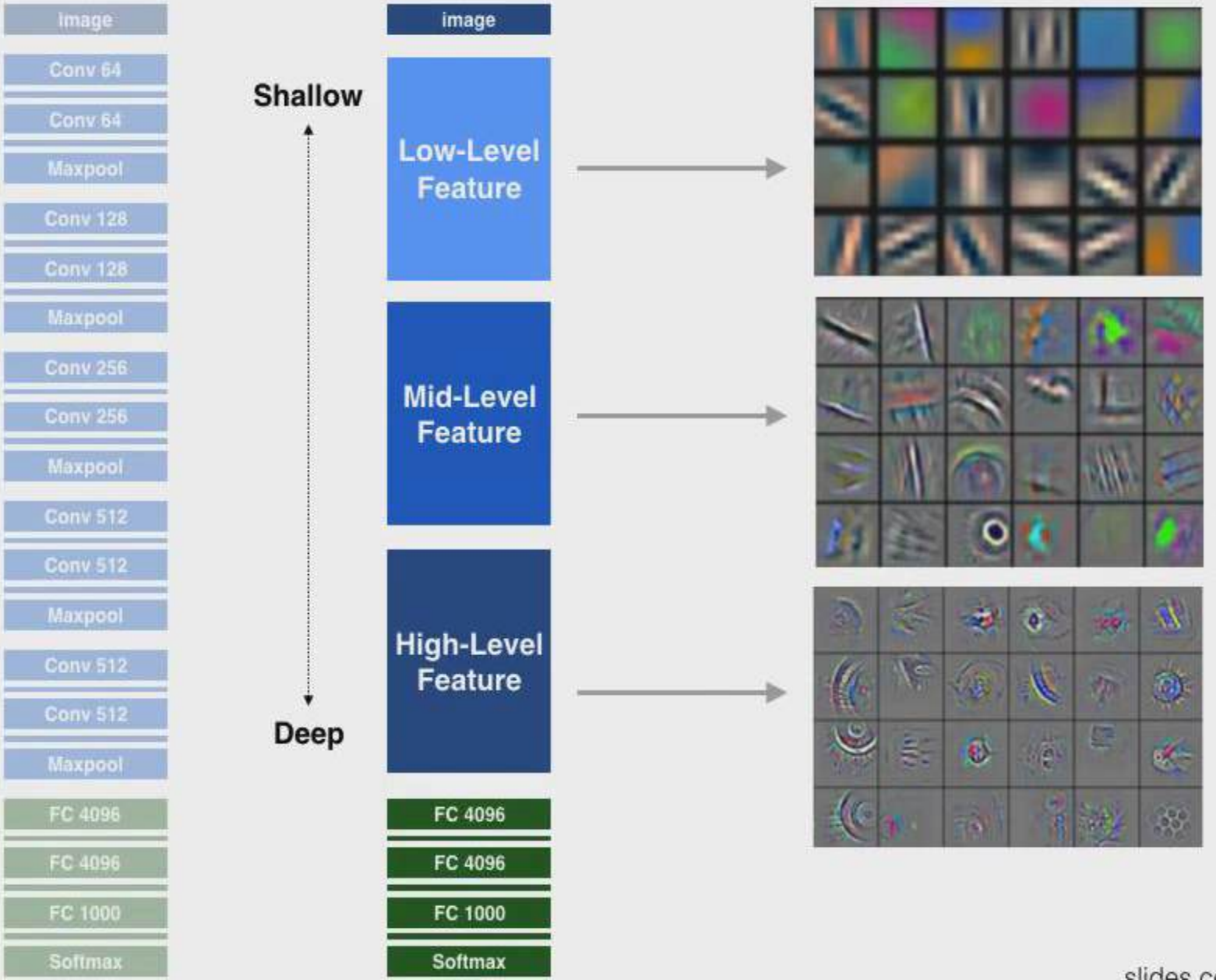
The formula for calculating the output size for any given conv layer is

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

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?

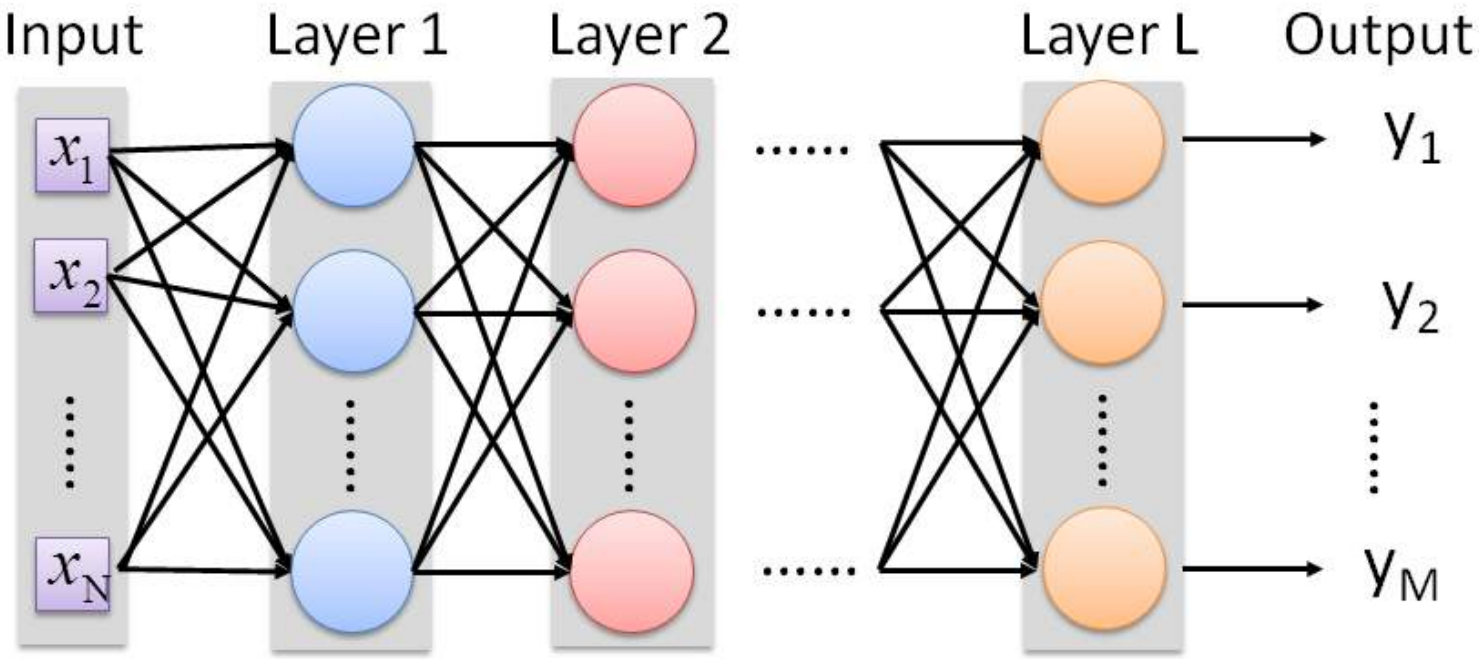


Recurrent Neural Network RNN

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

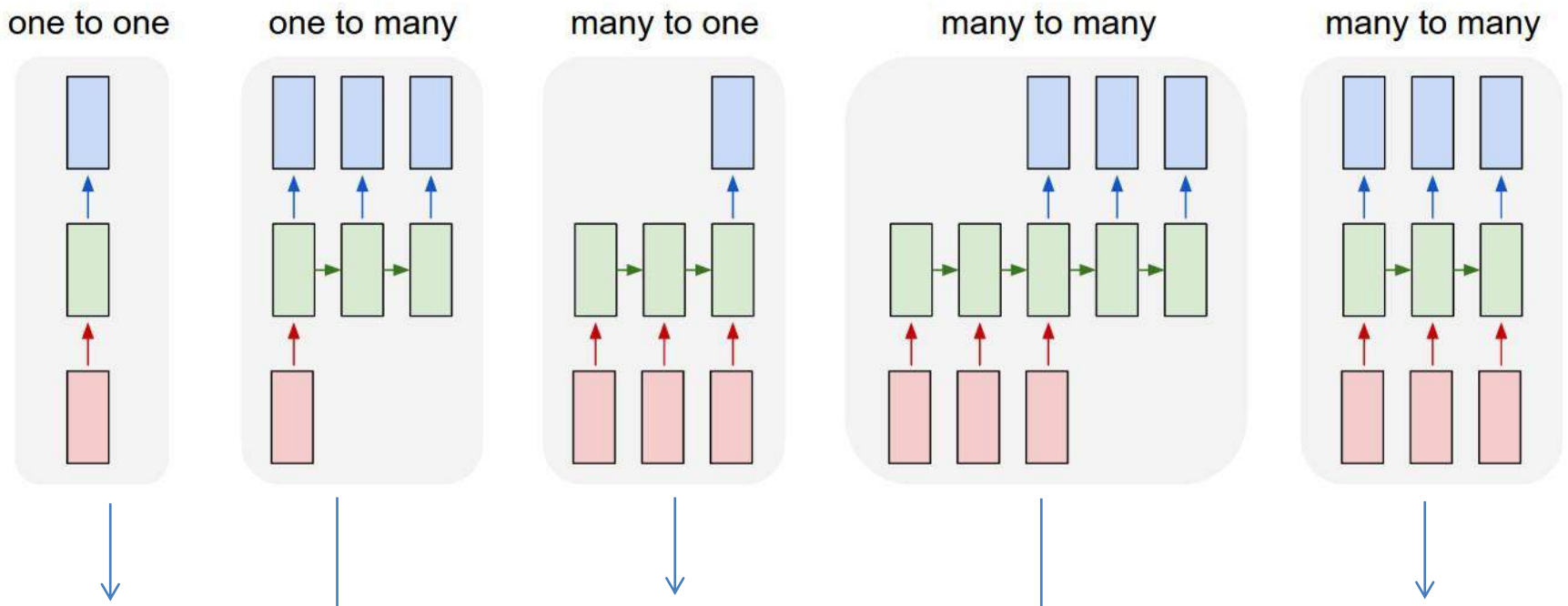


Image classification

Image captioning

Sentiment analysis

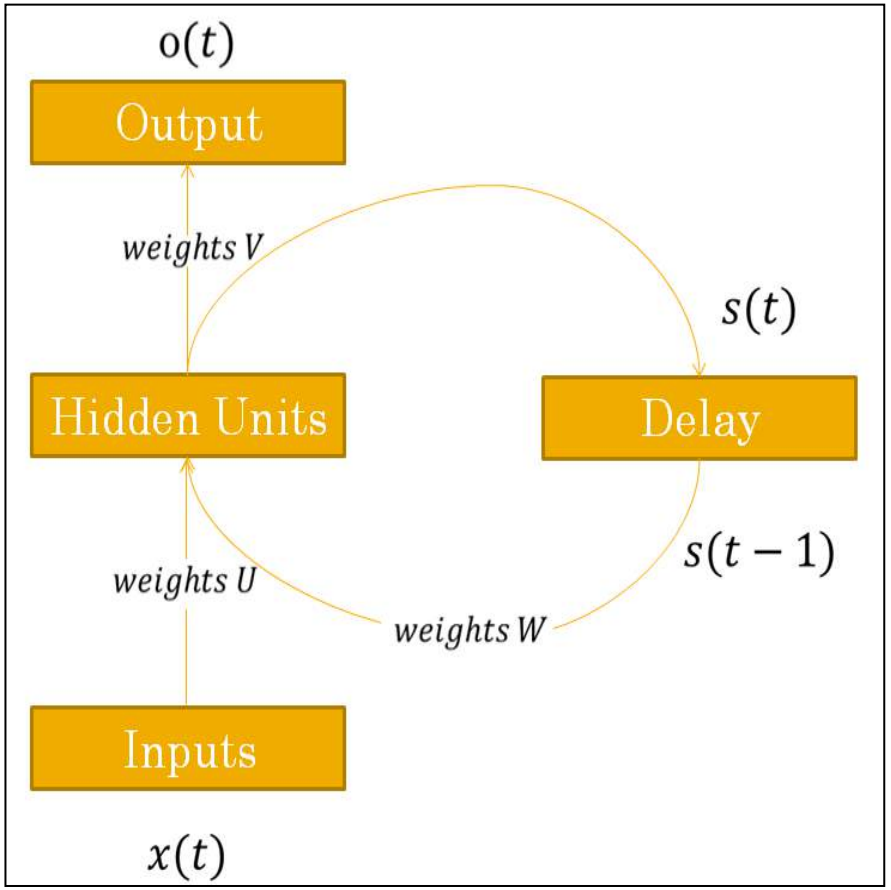
Machine translation

Synced sequence(video classification)

RNN architecture

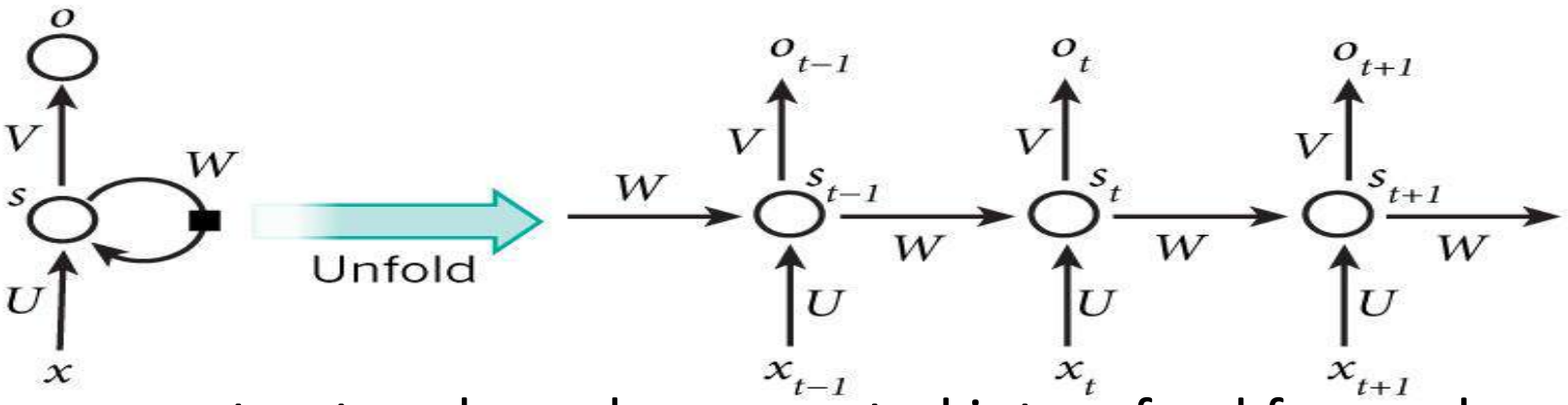
- RNNs are called *recurrent* because they perform the same task for every element of a sequence, with the output being depended on the previous computations (memory).

- 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:

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

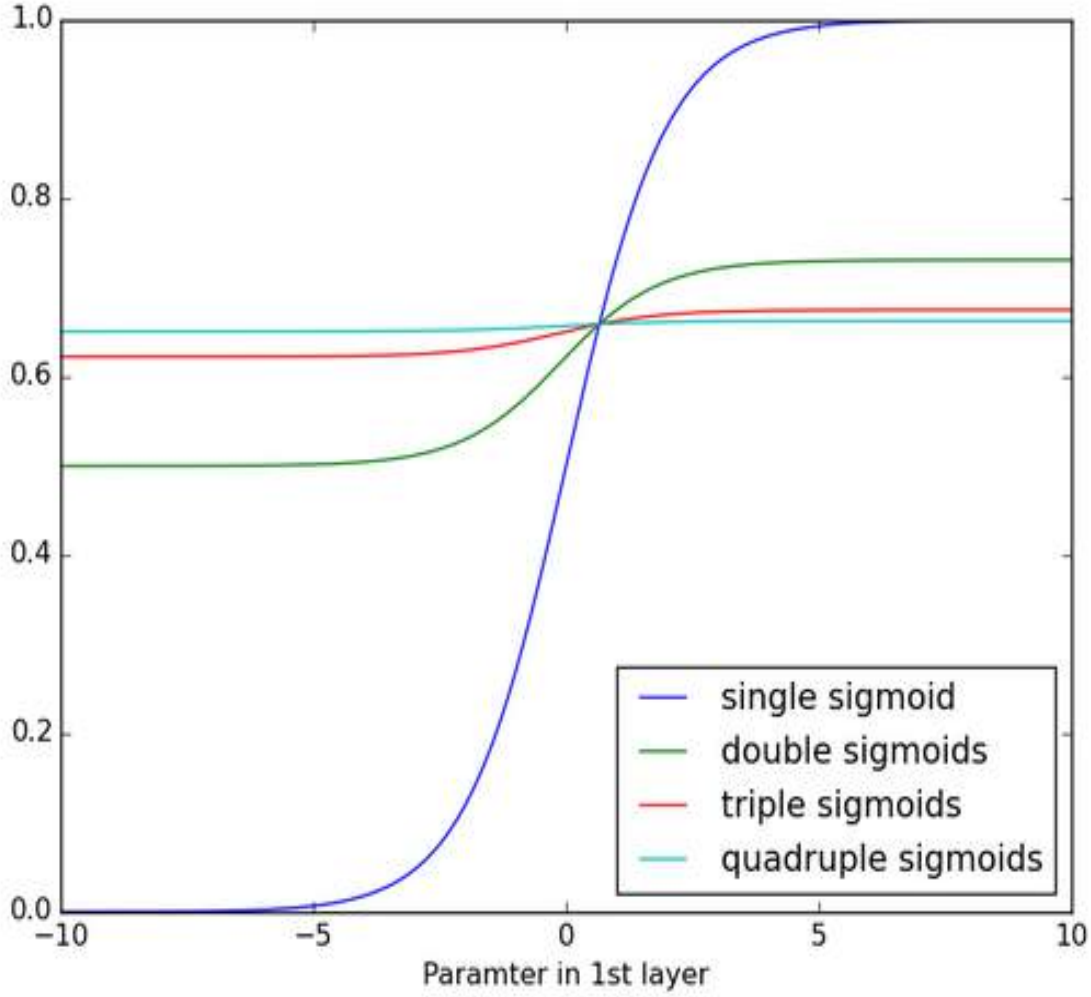
  - The network input to the output unit at time t:

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

  - The output of the network at time t is:

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

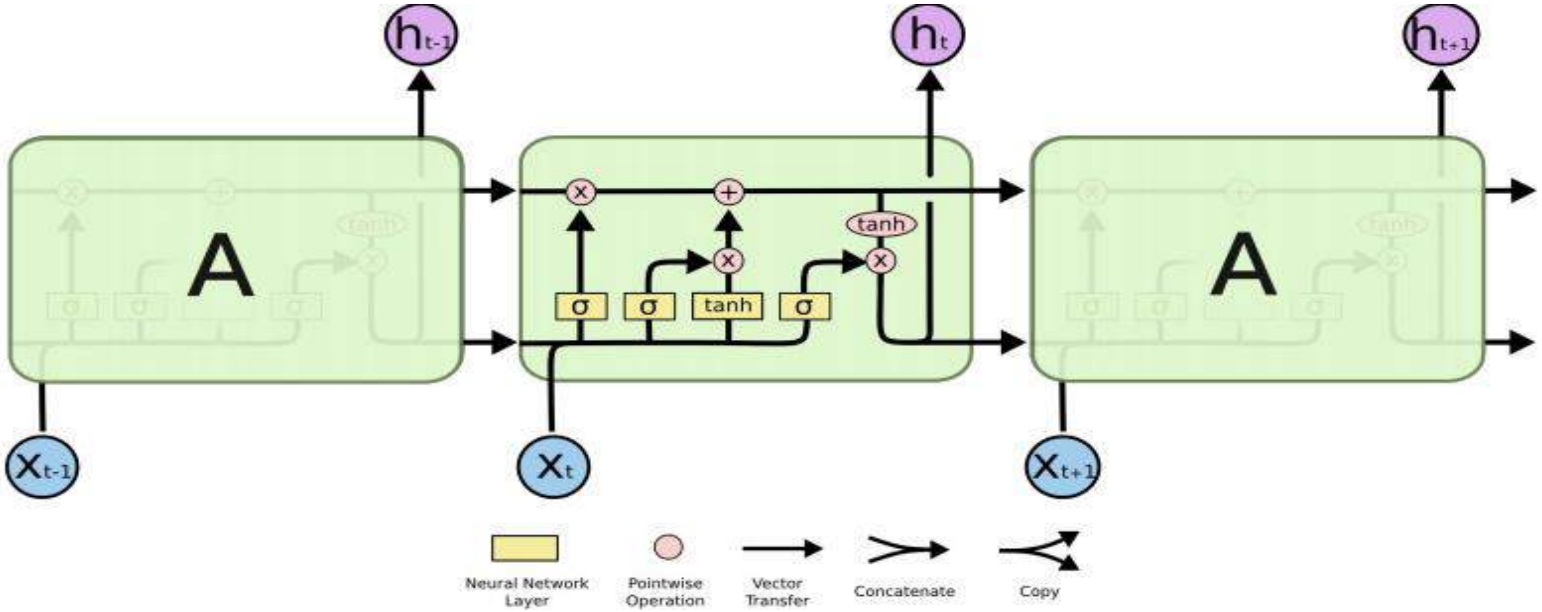
Vanishing Gradients



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.

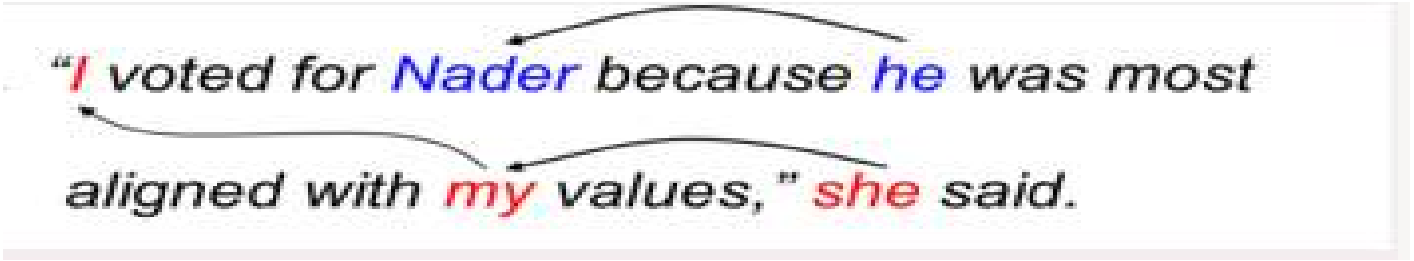
There are two methods

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.

$$f(x_1, x_2, \ldots, x_n \mid \theta) = f(x_1 \mid \theta) \times f(x_2 \mid \theta) \times \cdots \times f(x_n \mid \theta),$$

$$L(\theta; x_1, \ldots, x_n) = f(x_1, x_2, \ldots, x_n \mid \theta) = \prod_{i=1}^{n} f(x_i \mid \theta)$$

$$\ln L(\theta; x_1, \ldots, x_n) = \sum_{i=1}^{n} \ln f(x_i \mid \theta),$$

$$\hat{\ell}(\theta \mid x) = \frac{1}{n} \sum_{i=1}^{n} \ln f(x_i \mid \theta),$$

$$\left\{\hat{\theta}_{\text{ML}}\right\} \subseteq \left\{\arg\max_{\theta \in \Theta} \hat{\ell}(\theta \mid x_1, \ldots, x_n)\right\}.$$

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.

$$H(p, q) = \mathrm{E}_p[\mathrm{I}_q] = \mathrm{E}_p\left[\log \frac{1}{q(x_i)}\right]$$

$$H(p, q) = \sum_{x_i} p(x_i) \log \frac{1}{q(x_i)}$$

$$H(p, q) = -\sum_{x} p(x) \log q(x).$$

$$L_{CE} = -\frac{1}{N} \sum_{n=1}^{N} \left[y_n \log \hat{y}_n + (1 - y_n) \log (1 - \hat{y}_n) \right]$$

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

$$L_e(y, t) = \exp(\frac{1}{2} \sum_{k} (y_k - t_k)^2)$$

Hellinger Distance

$$H^2(P, Q) = \frac{1}{2} \int \left(\sqrt{\frac{dP}{d\lambda}} - \sqrt{\frac{dQ}{d\lambda}} \right)^2 d\lambda,$$

It is used to quantify the similarity between two probability distributions. It is a type of f-divergence.

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 Divergence

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.

$$D_{\mathrm{KL}}(P \| Q) = \sum_{i} P(i) \log \frac{P(i)}{Q(i)}$$

Discrete

$$D_{\mathrm{KL}}(P \| Q) = \int_{-\infty}^{\infty} p \log \frac{p}{q} \, dy.$$

Continuous

Itakura–Saito distance

is a measure of the difference between an original spectrum P(ω) and an approximation P̂(ω) of that spectrum. Although it is not a perceptual measure, it is intended to reflect perceptual (dis)similarity.

$$D_{\mathrm{IS}}(P(\omega), \hat{P}(\omega)) = \frac{1}{2\pi} \int_{-\pi}^{\pi} \left| \frac{P(\omega)}{\hat{P}(\omega)} - \log \frac{P(\omega)}{\hat{P}(\omega)} - 1 \right| d\omega$$

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.

$$R(w) = \sum_{i=1}^{n} \|W\|_{2,1}$$

Nuclear norm regularization

$$R(w) = \|\sigma(W)\|_1 \text{ where } \sigma(W) \text{ is the eigenvalues in the singular value decomposition of } W.$$

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.

$$R(W) = \sum_{i=1}^{n} \|w_i - \frac{1}{n} \sum_{i=1}^{n} w_i\|_2$$

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.

$$R(W) = \|W - \frac{1}{n} \sum_{i=1}^{n} w_i\|_F + \|W - \frac{1}{|C_k|} \sum_{i \in C_k} w_i\|_F \text{, where } C_k \text{ is a cluster of tasks}$$

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.

$$R(w) = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \|w_i - w_j\| A_{ij} \text{, for a given symmetric similarity matrix } A$$

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

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

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.

$$\frac{\partial u}{\partial v} = W^2 J$$

Intuition for backpropagation

Another Example (Circuits)

Simple Example (Circuits)

Simple Example (Flowgraphs)

Learning Rate

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.

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.

$$\theta^{new} = \theta^{old} - \alpha \nabla_\theta J_t(\theta)$$

Simplest recipe: keep it fixed and use the same for all parameters.

Tricks

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 c_0 and τ and t is iteration numbers.

$$\alpha = \frac{c_0 \tau}{\max(t, \tau)}$$

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.

$$\theta^{new} = \theta^{old} - \alpha \nabla_\theta J_t(\theta)$$

Stochastic Gradient Descent (SGD)

Gradient descent uses total gradient over all examples per update, SGD updates after only 1 or few examples:

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

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.

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.

Initialization with Small Random Numbers

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.

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.

Calibrating the Variances

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.

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)

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

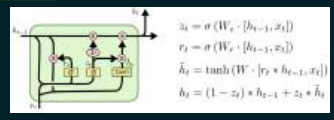
Single-Layer Perceptron

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

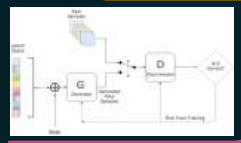


$$\tilde{z}_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Long short-term memory - It is a type of recurrent (RNN), allowing data to flow both forwards and backwards within the network.

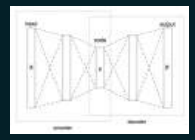
An LSTM is well-suited to learn from experience to classify, process and predict time series given time lags of unknown size and bound between important events. Relative insensitivity to gap length gives an advantage to LSTM over alternative RNNs, hidden Markov models and other sequence learning methods 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)

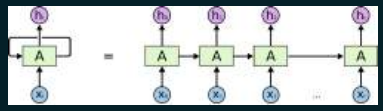
Pooling

Convolution

Subsampling

They have applications in image and video recognition, recommender systems and natural language processing.

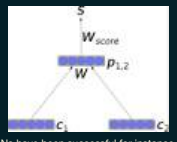
RNNs (Recurrent)



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

$$f'(\theta) \approx \frac{J(\theta^{(i+1)}) - J(\theta^{(i-1)})}{2\varepsilon} \qquad \theta^{(i+1)} = \theta + \varepsilon \times e_i$$

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

$$v = \mu v - \alpha \nabla_\theta J_t(\theta)$$
$$\theta^{new} = \theta^{old} + v$$

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 $g_{t,i} = \frac{\partial}{\partial \theta_i} J_t(\theta)$, 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 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

Packages

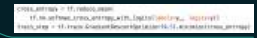
tf

Main Steps

1. Create the Model

2. Define Target

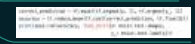
3. Define Loss function and Optimizer



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

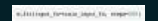
2. Define your Layers, or use a prebuilt model

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

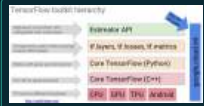


Using the fit function, on the input_fn. Notice that the feature columns are fed to the model as arguments.

5. Predict and Evaluate

Using the eval_input_fn defined previously.

Tensorflow



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.

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.