Artificial Intelligence

Module 8: Learning from Example

Dr. Chandra Prakash

(Slides adapted from StuartJ. Russell, B Ravindran, Mausam, Dan Klein and Pieter Abbeel, Partha P Chakrabarti, Saikishor Jangiti)

Module 8: Learning from Example

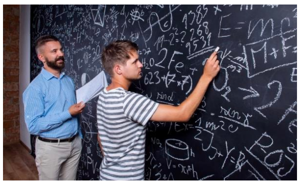
- PART 8.1 : Supervised learning : Introduction
- PART 8.2 : Naive Bayes,
- PART 8.3 : Discriminative Learning
  - Perceptron, Neural Network
- PART 8.4 : Introduction to Deep Learning

What is Learning?

- Learning is an important area in AI, perhaps more so than planning.
  - Problems are hard -- harder than planning.
  - Recognised Solutions are not as common as planning.
  - A goal of AI is to enable computers that can be taught rather than programmed.
- Learning is an area of AI that focusses on processes of self-improvement.
- Information processes that improve their performance or enlarge their knowledge bases are said to learn.
- Why is it hard?
  - Intelligence implies that an organism / machine must be able to adapt to new situations.
  - It must be able to learn to do new things.
  - This requires knowledge acquisition, inference, updating/refinement of knowledge base, acquisition of heuristics, applying faster searches, etc.

How can we learn?

- Your View ??

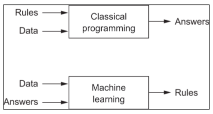


How can we learn?

- **Skill refinement**
  - one can learn by practicing, e.g playing the piano.
- **Knowledge acquisition**
  - one can learn by experience and by storing the experience in a knowledge base.
  - Example - rote learning (process of memorizing information based on repetition).
- **Problem Solving**
  - Solve a problem, learn from this experience. Next time see similar problem, solve it more efficiently.
  - not usually involve gathering new knowledge but may involve reorganisation of data or remembering how to achieve to solution.
- **Taking advice**
  - Similar to rote learning although the knowledge that is input may need to be transformed (or operationalised) in order to be used effectively.
- **Induction**
  - One can learn from examples. Humans often classify things in the world without knowing explicit rules. Usually involves a teacher or trainer to aid the classification.
- **Discovery**
  - learns knowledge without the aid of a teacher.
- **Analogy**
  - If a system can recognise similarities in information already stored then it may be able to transfer some knowledge to improve to solution of the task in hand.

Machine Learning

- Up until now: how use a model to make optimal decisions
- Machine learning: how to acquire a model from data / experience
  - Learning parameters (e.g. probabilities)
  - Learning structure (e.g. BN graphs)
  - Learning hidden concepts (e.g. clustering, neural nets)
- Today: model-based classification



Rules
Data
Classical programming
Answers
Data
Answers
Machine learning
Rules

Why to use Machine learning Techniques

Difference between

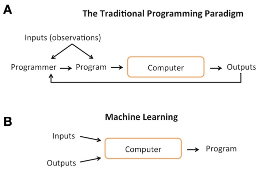
- Algorithm and Machine learning techniques

Learning :

- Ability to improve behaviour based on experience

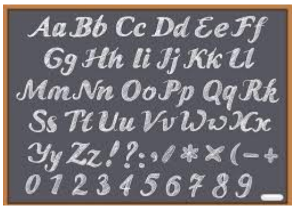
Fundamental law in ML

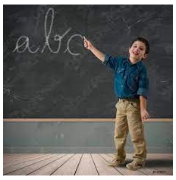
- Learn/build models for data
- Model used for prediction, decision making or solving tasks



Recall : How we learn

- Our Learning

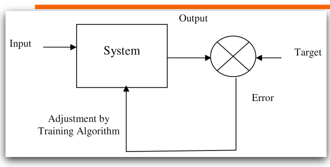


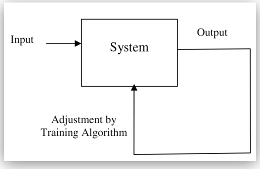


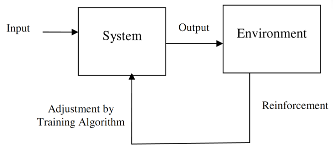
- Teacher

- Feedback

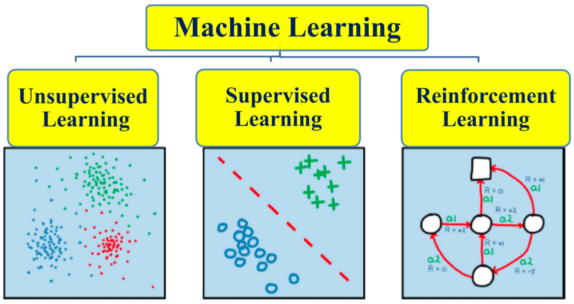
Learning based on Feedback







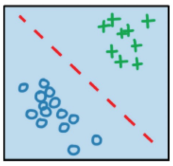
Three broad categories of ML



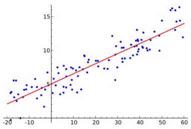
Machine Learning

- **Unsupervised Learning**
- **Supervised Learning**
- **Reinforcement Learning**

Supervised learning



Classification



Regression

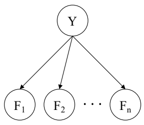
Model-Based Classification

- Model-based approach
  - Build a model (e.g. Bayes' net) where b the output label and input features are random variables
  - Instantiate any observed features
  - Query for the distribution of the label conditioned on the features

- Challenges
  - What structure should the BN have?
  - How should we learn its parameters?

Naïve Bayes for Digits

- Naïve Bayes: Assume all features are independent effects of the label
- Simple digit recognition version:
  - One feature (variable) F_{ij} for each grid position <i,j>
  - Feature values are on / off, based on whether intensity is more or less than 0.5 in underlying image
  - Each input maps to a feature vector, e.g.



$\rightarrow \langle F_{0,0}=0 \;\; F_{0,1}=0 \;\; F_{0,2}=1 \;\; F_{0,3}=1 \;\; F_{0,4}=0 \;\; \ldots F_{15,15}=0 \rangle$

  - Here: lots of features, each is binary valued

- Naïve Bayes model: $P(Y|F_{0,0}\ldots F_{15,15}) \propto P(Y)\prod_{i,j} P(F_{i,j}|Y)$
- What do we need to learn?

Naïve Bayes

- Bayes classification

$$P(Y\,|\,\mathbf{X}) \propto P(\mathbf{X}\,|\,Y)P(Y) = P(X_1,\cdots,X_n\,|\,Y)P(Y)$$

Difficulty: learning the joint probability $P(X_1,\cdots,X_n\,|\,C)$

- Naïve Bayes classification

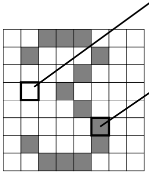
Assume all input features are conditionally independent!

$$\begin{aligned} P(X_1,X_2,\cdots,X_n\,|\,Y) &= P(X_1\,|\,X_2,\cdots,X_n,Y)P(X_2,\cdots,X_n\,|\,Y) \\ &= P(X_1\,|\,Y)P(X_2,\cdots,X_n\,|\,Y) \\ &= P(X_1\,|\,Y)P(X_2\,|\,Y)\cdots P(X_n\,|\,Y) \end{aligned}$$

Example 1: Conditional Probabilities

$P(Y)$

1	0.1
2	0.1
3	0.1
4	0.1
5	0.1
6	0.1
7	0.1
8	0.1
9	0.1
0	0.1



$P(F_{3,1}=on|Y)$

1	0.01
2	0.05
3	0.05
4	0.30
5	0.80
6	0.90
7	0.05
8	0.60
9	0.50
0	0.80

$P(F_{5,5}=on|Y)$

1	0.05
2	0.01
3	0.90
4	0.80
5	0.90
6	0.90
7	0.25
8	0.85
9	0.60
0	0.80

Naïve Bayes for Text (A Spam Filter)

- Bag-of-words Naïve Bayes:
  - Features: W_i is the word at position i
  - As before: predict label conditioned on feature variables (spam vs. ham)
  - As before: assume features are conditionally independent given label
  - New: each W_i is identically distributed
- Generative model: $P(Y,W_1\ldots W_n) = P(Y)\prod_i P(W_i|Y)$

Word at position i, not ith word in the dictionary!

- "Tied" distributions and bag-of-words
  - Usually, each variable gets its own conditional probability distribution P(F|Y)
  - In a bag-of-words model
    - Each position is identically distributed
    - All positions share the same conditional probs P(W|Y)
    - Why make this assumption?
  - Called "bag-of-words" because model is insensitive to word order or reordering

Example 2: Spam Filtering

- Model: $P(Y,W_1\ldots W_n) = P(Y)\prod_i P(W_i|Y)$
- What are the parameters?

$P(Y)$

ham	0.66
spam	0.33

$P(W|\text{spam})$

the	0.0156
to	0.0153
and	0.0115
of	0.0095
you	0.0093
a	0.0086
with:	0.0080
from:	0.0075
...	

$P(W|\text{ham})$

the	0.0210
to	0.0133
of	0.0119
2002:	0.0110
with:	0.0108
from:	0.0107
and	0.0105
a	0.0100
...	

- Where do these tables come from?

Spam Example

Word	P(w|spam)	P(w|ham)	Tot Spam	Tot Ham
(prior)	0.33333	0.66666	-1.1	-0.4

P(spam | w) = 98.9

Example 3 : Play Tennis

PlayTennis: training examples

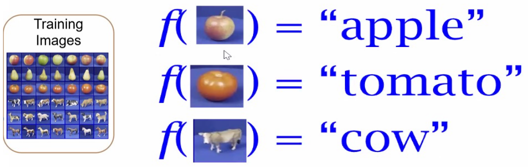
Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Find the output ??

x=(Outlook=Sunny, Temperature=Cool, Humidity=High, Wind=Strong)

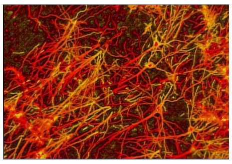
Learning from Example

- Apply a prediction function to a feature representation of the image to get the desired output:



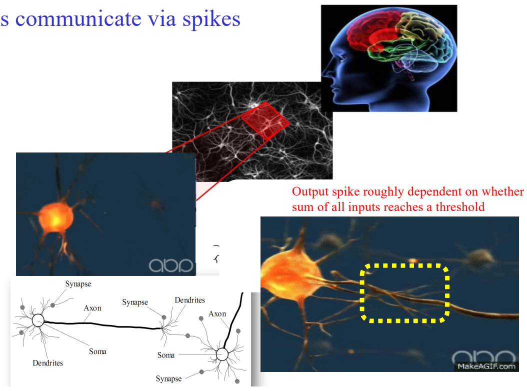
Slide credit: L. Lazebnik

Real Brain Interconnections



Human brain contains a massively interconnected net of 10^{10} -10^{11} (10 billion) neurons (cortical cells)

Neurons communicate via spikes



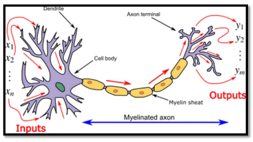
Output spike roughly dependent on whether sum of all inputs reaches a threshold

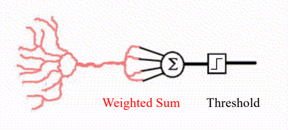
Neural Network: Introduction

- A neural network can be defined as a model of reasoning based on the human brain.
- Inspired by the human brain.
- Some NNs are models of biological neural networks
- Human brain contains a massively interconnected net of 10^{10}-10^{11} (10 billion) neurons (cortical cells)
- 60 trillion connections, synapses, between them
  - Massive parallelism – large number of simple processing units
  - Connectionism – highly interconnected
  - Associative distributed memory
    - Pattern and strength of synaptic connections

Artificial Neural Network

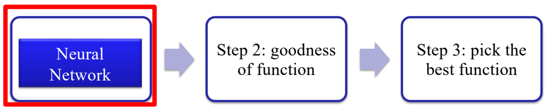
- Consists of a number of very simple processors, also called **neurons**,
  - Analogous to the biological neurons in the brain.
- Neurons are connected by weighted links passing signals from one neuron to another.
- The output signal is transmitted through the neuron’s outgoing connection.
- The outgoing connection splits into a number of branches that transmit the same signal.
  - The outgoing branches terminate at the incoming connections of other neurons in the network.





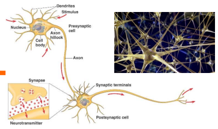
Source: Wikipedia

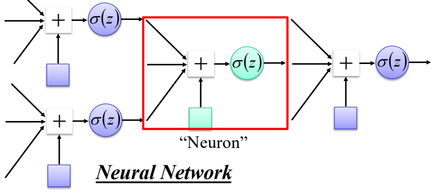
Three Steps for Machine Learning



Machine Learning is so simple ……

Neural Network

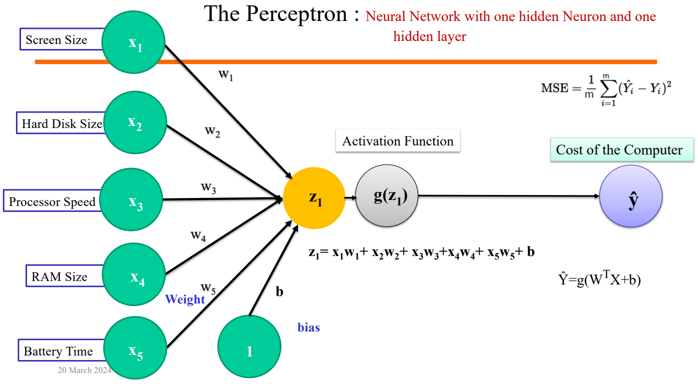




Different connection leads to different network structures

Network parameter θ: all the weights and biases in the "neurons"

The Perceptron : Neural Network with one hidden Neuron and one hidden layer



$$MSE = \frac{1}{m}\sum_{i=1}^{m}(\hat{Y}_i - Y_i)^2$$

$$z_1 = x_1w_1 + x_2w_2 + x_3w_3 + x_4w_4 + x_5w_5 + b$$

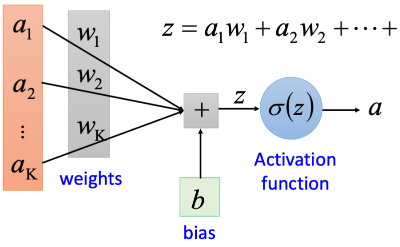
$$\hat{Y} = g(W^TX + b)$$

20 March 2024

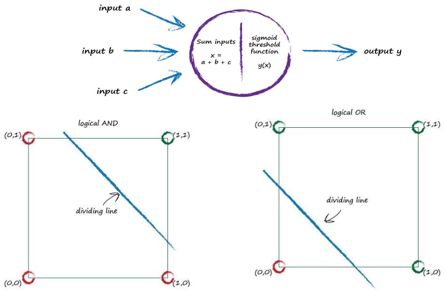
Perceptron : When we have more than one input?

Neuron $f: R^K \rightarrow R$

$$z = a_1w_1 + a_2w_2 + \cdots + a_Kw_K + b$$

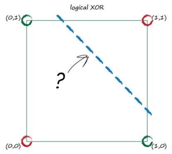


Perceptron : Single Neuron- Linear Separability



Single Neuron and Linear Inseparability

Input A	Input B	XOR
0	0	0
0	1	1
1	0	1
1	1	0



Perceptron with threshold units fails if classification task is not linearly separable

- Example: XOR
- Can a perceptron separate the 1 outputs from the 0 outputs?
- No single line can separate the "yes" (1) outputs from the "no" (0) outputs!

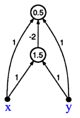
Minsky and Papert's book showing such negative results put a damper on neural networks research for over a decade!

Some problems can't be solved with just a single simple linear classifier.

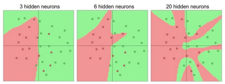
You can use multiple nodes working together to solve many of these problems.

How do we deal with linear inseparability?

- Idea 1: Multilayer Perceptrons
  - Removes limitations of single-layer networks
    - Can solve XOR
  - Example: Two-layer perceptron that computes XOR
  - Output is +1 if and only if x + y – 2Θ (x + y – 1.5) – 0.5 > 0



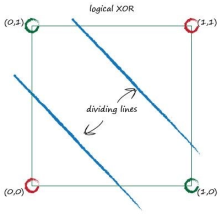
- Idea 2: Activation functions
  - Non-linearities needed to learn complex (non-linear) representations of data, otherwise the NN would be just a linear function

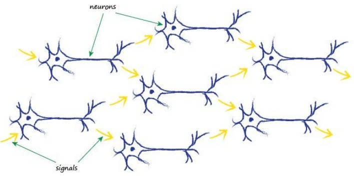


http://cs231n.github.io/assets/nn1/layer_sizes.jpeg

  - More layers and neurons can approximate more complex functions

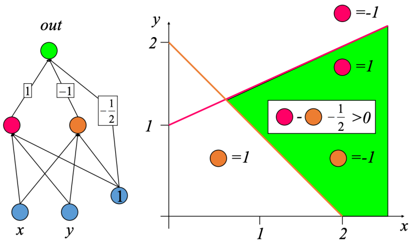
A solution: Multi-layer Perceptron (MLP)



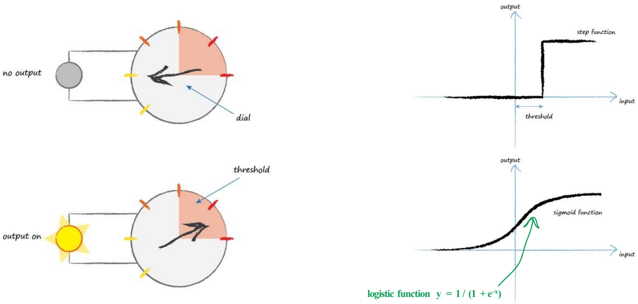


…Use more than one node!

Multilayer Perceptron: What does it do?

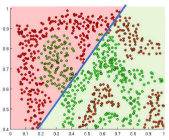


Activation Function/ Threshold

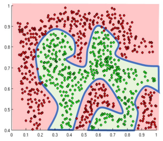


Need of Activation functions

- The purpose of activation functions is to introduce non-linearities into the network
- A neural network without an activation function is essentially just a linear regression model.



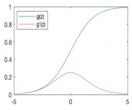
Linear Activation functions produce linear decisions no matter the network size



Non-linearities allow us to approximate arbitrarily complex functions

Popular Activation Functions

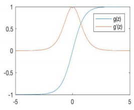
Sigmoid Function



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

$$g'(z) = g(z)(1-g(z))$$

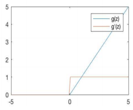
Hyperbolic Tangent



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

$$g'(z) = 1-g(z)^2$$

Rectified Linear Unit (ReLU)

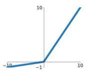


$$g(z) = \max(0, z)$$

$$g'(z) = \begin{cases} 1, & z > 0 \\ 0, & \text{otherwise} \end{cases}$$

Leaky ReLU

$\max(0.1x, x)$

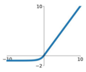


Maxout

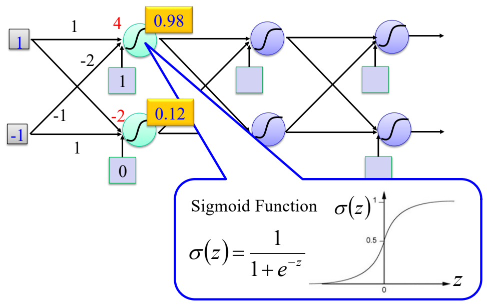
$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU

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



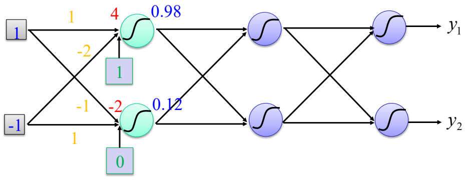
Fully Connect Feedforward Network



Sigmoid Function

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

Matrix Operation



$$\sigma(\underbrace{\begin{bmatrix}1 & -2\\ -1 & 1\end{bmatrix}\begin{bmatrix}1\\ -1\end{bmatrix}+\begin{bmatrix}1\\ 0\end{bmatrix}}_{\begin{bmatrix}4\\ -2\end{bmatrix}}) = \begin{bmatrix}0.98\\ 0.12\end{bmatrix}$$

Matrix Multiplication

weights

incoming signals

$$\begin{pmatrix} w_{1,1} & w_{2,1}\\ w_{1,2} & w_{2,2}\end{pmatrix}\begin{pmatrix} input\_1\\ input\_2\end{pmatrix} = \begin{pmatrix} (input\_1 * w_{1,1}) + (input\_2 * w_{2,1})\\ (input\_1 * w_{1,2}) + (input\_2 * w_{2,2})\end{pmatrix}$$

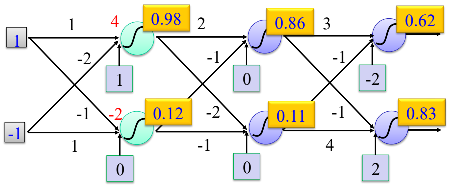
W·I= X

dot product

1. The many feedforward calculations can be expressed **concisely** as **matrix multiplication**, no matter what shape the network.

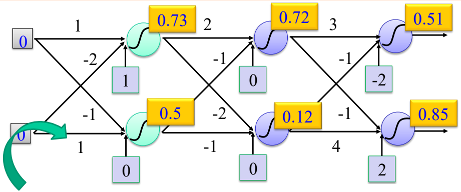
2. Some programming languages can do matrix multiplication really **efficiently** and **quickly**.

Fully Connect Feedforward Network



$$f\left(\begin{bmatrix}1\\ -1\end{bmatrix}\right) = \begin{bmatrix}0.62\\ 0.83\end{bmatrix}$$

Fully Connect Feedforward Network

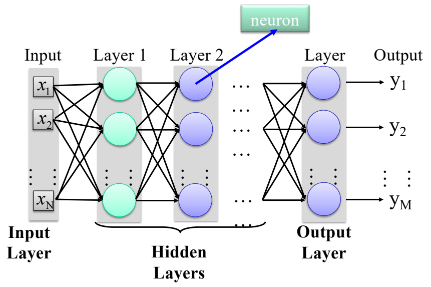


This is a function.
Input vector, output vector

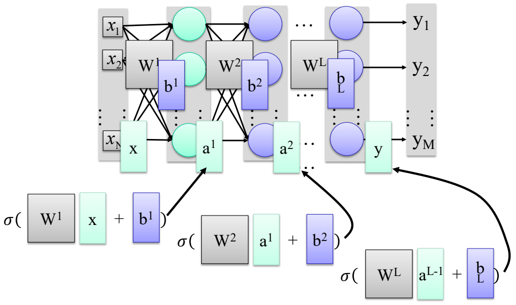
$$f\left(\begin{bmatrix}1\\ -1\end{bmatrix}\right) = \begin{bmatrix}0.62\\ 0.83\end{bmatrix} \quad f\left(\begin{bmatrix}0\\ 0\end{bmatrix}\right) = \begin{bmatrix}0.51\\ 0.85\end{bmatrix}$$

Given network structure, define ***a function set***
Different parameters define different function

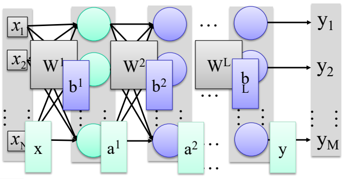
Fully Connect Feedforward Network



Neural Network



Neural Network



$y = f(x)$

Using parallel computing techniques to speed up matrix operation

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

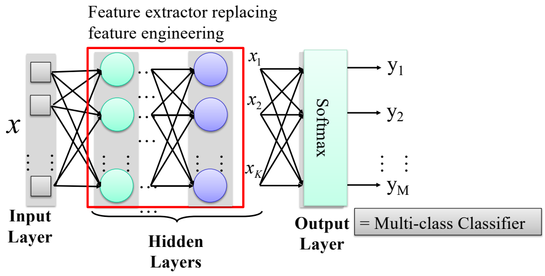
x

Features

- The features are the elements of your input vectors.
- The number of features is equal to the number of nodes in the input layer of the network

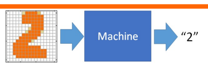
Category	**Features**
Housing Prices	No. of Rooms, House Area, Air Pollution, Distance from facilities, Economic Index city, Security Ranking etc.
Spam Detection	presence or absence of certain email headers, the email structure, the language, the frequency of specific terms, the grammatical correctness of the text etc.
Speech Recognition	noise ratios, length of sounds, relative power of sounds, filter matches
Cancer Detection	Clump thickness, Uniformity of cell size, Uniformity of cell shape, Marginal adhesion, Single epithelial cell size, Number of bare nuclei, Bland chromatin, Number of normal nuclei, Mitosis etc.
Cyber Attacks	IP address, Timings, Location, Type of communication, traffic details etc.
Video Recommendations	Text matches, Ranking of the video, Interest overlap, history of seen videos, browsing patterns etc.
Image Classification	Pixel values, Curves, Edges etc.

Output Layer

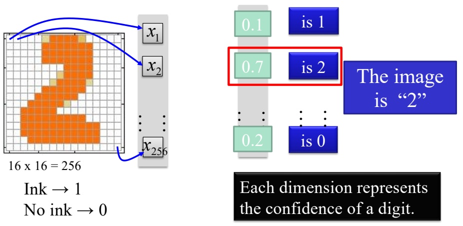


Example Application

Input



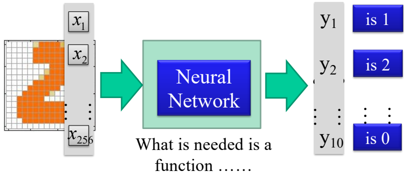
Output



Each dimension represents the confidence of a digit.

Example Application

- Handwriting Digit Recognition



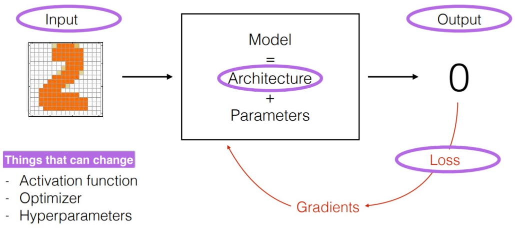
What is needed is a function ……

Input:
256-dim vector

output:
10-dim vector

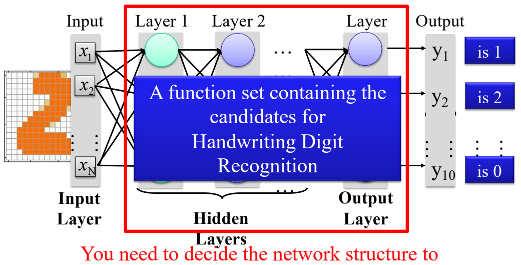
Learning Process

- A model is defined by its architecture and its parameters
  - The labelling strategy matters to successfully train your models. For example, if you're training a 10-class (0,1,2,...9) classifier under the constraint of one digit per picture, you might use one-hot vectors to label your data.



Things that can change
- Activation function
- Optimizer
- Hyperparameters

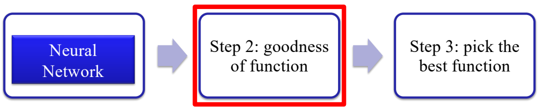
Example Application



A function set containing the candidates for Handwriting Digit Recognition

You need to decide the network structure to let a good function in your function set.

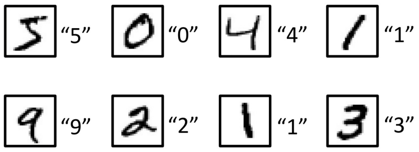
Three Steps for Machine Learning



Machine Learning is so simple ……

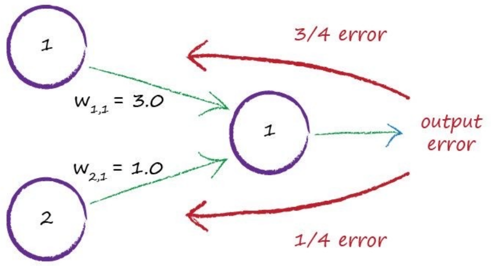
Training Data

- Preparing training data: images and their labels



Using the training data to find the network parameters.

Network Error



Matrices Again

$$error_{hidden} = \begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{pmatrix} \cdot \begin{pmatrix} e_1 \\ e_2 \end{pmatrix}$$

$$error_{hidden} = W^T_{hidden\_output} \cdot error_{output}$$

1. Remember we use the **error** to guide how we refine a model's parameter - link weights.
2. The error at the output nodes is easy - the difference between the **desired** and **actual** outputs.
3. The error at internal nodes isn't obvious. A **heuristic** approach is to split it in **proportion** to the link weights.
4. … and back propagating the error can be expressed as a **matrix** multiplication too!

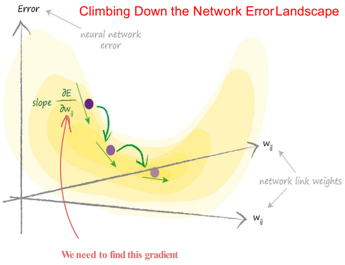
Error Gradient

$\mathrm{E} = (\text{desired} - \text{actual})^2$

school level calculus (chain rule)

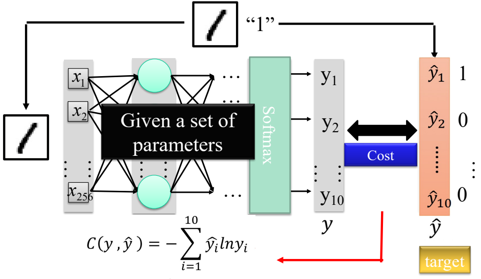
$\mathrm{dE}/\mathrm{dw}_{ij} = -\ e_j \cdot o_j \cdot (1 - o_j) \cdot o_i$

previous node


Climbing Down the Network ErrorLandscape
Error
neural network error
slope $\frac{\partial E}{\partial w_{ij}}$
w_{ij}
network link weights
w_{ij}
We need to find this gradient

Cost / Loss

Given a set of network parameters θ, each example has a cost value.

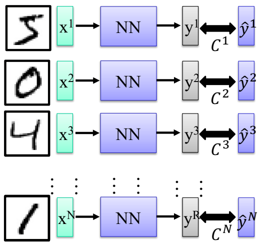


$$C(y, \hat{y}) = -\sum_{i=1}^{10} \hat{y}_i \ln y_i$$

Cost can be Euclidean distance or cross entropy of the network output and target

Total Loss

For all training data …



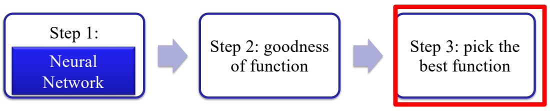
Total Loss:

$$L = \sum_{n=1}^{N} C^n$$

Find ***a function in function set*** that minimizes total loss L

Find ***the network parameters θ^**** that minimize total loss L

Three Steps for Machine Learning



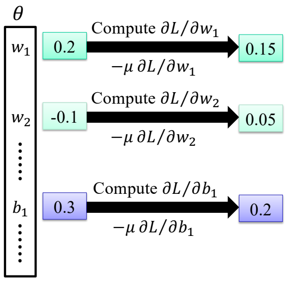
Step 1: Neural Network

Step 2: goodness of function

Step 3: pick the best function

Machine Learning is so simple ……

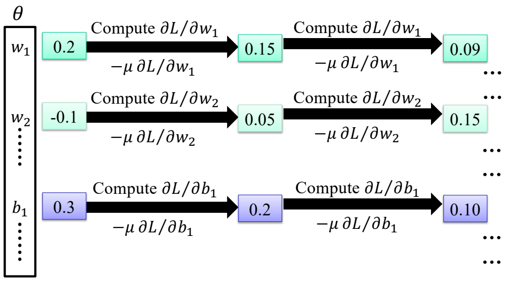
Gradient Descent


θ
w_1
0.2
Compute $\partial L/\partial w_1$
$-\mu\,\partial L/\partial w_1$
0.15
w_2
-0.1
Compute $\partial L/\partial w_2$
$-\mu\,\partial L/\partial w_2$
0.05
b_1
0.3
Compute $\partial L/\partial b_1$
$-\mu\,\partial L/\partial b_1$
0.2

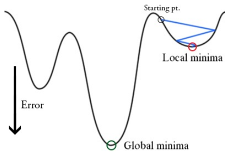
$$\nabla L = \begin{bmatrix} \frac{\partial L}{\partial w_1} \\ \frac{\partial L}{\partial w_2} \\ \vdots \\ \frac{\partial L}{\partial b_1} \\ \vdots \end{bmatrix}$$

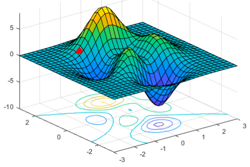
gradient

Gradient Descent


θ
w_1
0.2
Compute $\partial L/\partial w_1$
$-\mu\,\partial L/\partial w_1$
0.15
Compute $\partial L/\partial w_1$
$-\mu\,\partial L/\partial w_1$
0.09
w_2
-0.1
Compute $\partial L/\partial w_2$
$-\mu\,\partial L/\partial w_2$
0.05
Compute $\partial L/\partial w_2$
$-\mu\,\partial L/\partial w_2$
0.15
b_1
0.3
Compute $\partial L/\partial b_1$
$-\mu\,\partial L/\partial b_1$
0.2
Compute $\partial L/\partial b_1$
$-\mu\,\partial L/\partial b_1$
0.10

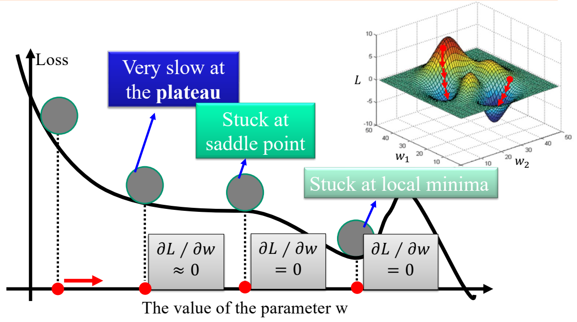
Local Minima





- NN can get stuck in local minima for small networks.
- For most large networks (many weights) local minima rarely occurs.
- It is unlikely that you are in a minima in every dimension simultaneously.

Besides local minima



Very slow at the **plateau**

Stuck at saddle point

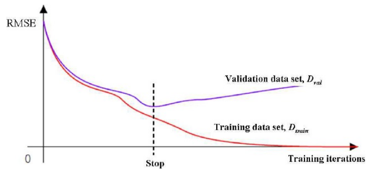
Stuck at local minima

$\partial L / \partial w \approx 0$

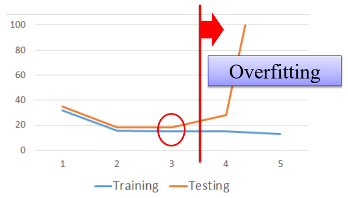
$\partial L / \partial w = 0$

$\partial L / \partial w = 0$

Overfitting in ANNs



Model Selection



Overfitting

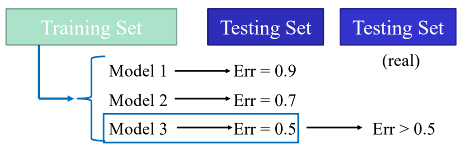
	Training	Testing
1	31.9	35.0
2	15.4	18.4
3	15.3	18.1
4	14.9	28.2
5	12.8	232.1

A more complex model does not always lead to better performance on ***testing data***.

This is ***Overfitting***. → Select suitable model

Model Selection

- There is usually a trade-off between bias and variance.
- Select a model that balances two kinds of error to minimize total error
- What you should NOT do:



Training Set

Testing Set

Testing Set (real)

Model 1 → Err = 0.9

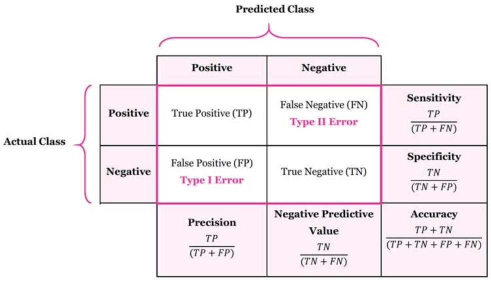
Model 2 → Err = 0.7

Model 3 → Err = 0.5 → Err > 0.5

Experimental Evaluation for ML Models

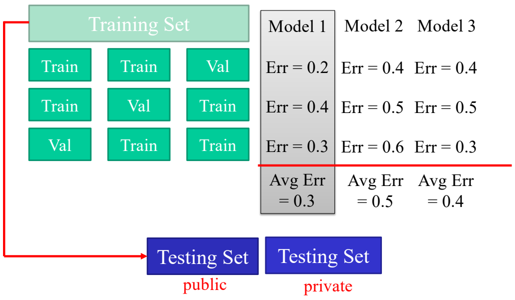
- Evaluating the performance of learning systems is important because:
  - Learning systems are usually designed to predict the class of future unlabeled data points.
- Typical choices for Performance Evaluation:
  - Error
  - Accuracy
  - Precision/Recall
- Typical choices for Sampling Methods:
  - Train/Test Sets
  - K-Fold Cross-validation

Confusion Matrix



		Predicted Class		
		Positive	**Negative**	
Actual Class	**Positive**	True Positive (TP)	False Negative (FN) **Type II Error**	**Sensitivity** $\frac{TP}{(TP+FN)}$
	Negative	False Positive (FP) **Type I Error**	True Negative (TN)	**Specificity** $\frac{TN}{(TN+FP)}$
		Precision $\frac{TP}{(TP+FP)}$	**Negative Predictive Value** $\frac{TN}{(TN+FN)}$	**Accuracy** $\frac{TP+TN}{(TP+TN+FP+FN)}$

N-fold Cross Validation



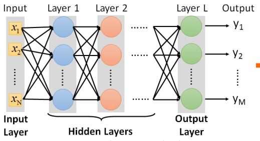
Training Set			Model 1	Model 2	Model 3
Train	Train	Val	Err = 0.2	Err = 0.4	Err = 0.4
Train	Val	Train	Err = 0.4	Err = 0.5	Err = 0.5
Val	Train	Train	Err = 0.3	Err = 0.6	Err = 0.3
			Avg Err = 0.3	Avg Err = 0.5	Avg Err = 0.4

Testing Set

public

Testing Set

private



- Q: How many layers? How many neurons for each layer?

Trial and Error + Intuition

- Q: Can the structure be automatically determined?
  - E.g. Evolutionary Artificial Neural Networks
- Q: Can we design the network structure?

Convolutional Neural Network (CNN)

NEURAL NETWORK ARCHITECTURES

EXAMPLE : SOLVE NN BY HAND

Back Propagation Neural Network

1. Determine the architecture

- **how many input and output neurons; what output encoding**
- **hidden neurons and layers**

2. Initialize all weights and biases to small random values, typically $\in$[-1,1]

3. Repeat until termination criterion satisfied:

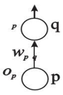
- **Present a training example and propagate it through the network *(forward pass)***
- **Calculate the actual output**
- **Adapt weights starting from the output layer and working backwards *(backward pass)***

$w_{pq}(t+1) = w_{pq}(t) + \Delta w_{pq}$ $w_{pq}(t)$ **- weight from node *p* to node *q* at time**

$\Delta w_{pq} = \eta \cdot \delta_q \cdot o_p$ **- weight change**

$\delta_i = (d_i - o_i) \cdot o_i \cdot (1-o_i)$ **- for output neuron *i***

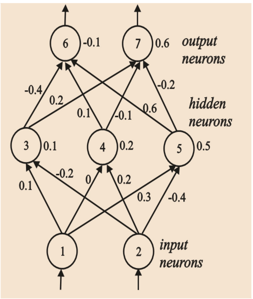
$\delta_j = o_j \cdot (1-o_j) \cdot \sum_i w_{ji} \cdot \delta_i$ **- for hidden neuron *j* (the sum is over the *i* nodes in the layer above the node *j*)**



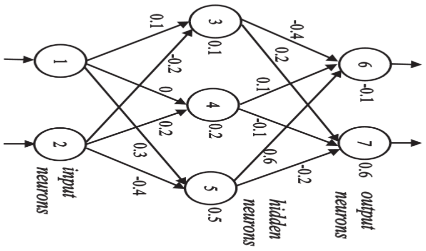
Back Propagation Neural Network

Consider an (2 classes, 2 dim. input data) ,

The training set is :
ex.1: 0.6 0.1 | class 1 (banana)
ex.2: 0.2 0.3 | class 2 (orange)



Back Propagation Neural Network



Back Propagation Neural Network

Neural Network architecture Questions

•How many inputs?

•How many hidden neurons?

•How many output neurons?

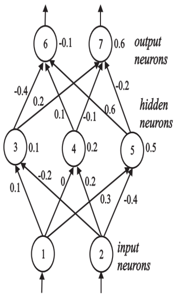
•What encoding of the outputs?

•Initial weights and learning rate

•What should be the stopping Criteria

Let's η=0.1 and
the weights are set as in the picture.

<u>what will be the final weight after 1st,2nd iteration.</u>

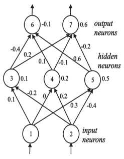


Inputs Encoding

- *The training set is :*
- *ex.1: 0.6 0.1 | class 1 (banana)*
- *ex.2: 0.2 0.3 | class 2 (orange)*

- How many inputs?
  - Same as no of variables
  - Number of samples
- <u>How to encode the inputs for nominal attributes?</u>
- Example - nominal attribute A with values none, some and many
- Local encoding
  - use a single input neuron and use an appropriate number of distinct values to correspond to the attribute values, e.g. none=0, some=0.5 and many=1

- Distributed (binary) encoding
  - use one neuron for each attribute value which is on or off (i.e. 1 or 0) depending on the correct value

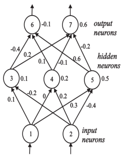


Network architecture

- ***<u>How many hidden neurons?</u>***
  - *Heuristic :*
    - *N=(input + output) /2*

- ***<u>How many output neurons?</u>***

- ***<u>What encoding of the outputs?</u>***
  - *Local encoding*
    - *1 output neuron (<0.2 –class 1, > 0.8 – class 2, in between- ambiguous class(class 3)*
  - *Distributed (binary, 1-of-n) encoding*
    - *No of outputs= number of classes*
    - *10 for class 1*
    - *01 for class 0*

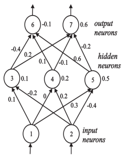


Back Propagation Neural Network

- Motivation for choosing binary over local encoding

- Provides more degree of freedom to represent the target function (n times as many weights available)

- The difference between the output with highest value and the second highest can be used as a measure how confident the prediction is (close values => ambiguous classification)



Stopping Criteria

- The **<u>stopping criteria</u>** is checked at the end of each epoch:

  - The error (**<u>mean absolute or mean square</u>**) at the end of an epoch is below a threshold
    - All training examples are propagated and the mean (absolute or square) error is calculated
    - The threshold is determined heuristically – e.g. 0.3

  - Maximum number of epochs is reached

  - Early stopping using a validation set (TTS)

• It typically takes hundreds or thousands of epochs for an NN to converge

Back Propagation Neural Network

1. Forward pass for ex. 1 - calculate the outputs o_6 and o_7

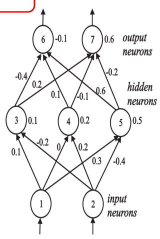
o_1=0.6, o_2=0.1, target output 1 0, i.e. class 1

- Activations of the hidden units:

$net_3= o_1 *w_{13}+ o_2*w_{23}+b_3=0.6*0.1+0.1*(-0.2)+0.1=0.14$

$o_3=1/(1+e^{-net3}) =0.53$

- **Calculate**
- net_4 , o_4 ??
- net_5, o_5 ??
- net_6, o_6 ??
- net_7, o_7??



Back Propagation Neural Network

1. Forward pass for ex. 1 - calculate the outputs o_6 and o_7

o_1=0.6, o_2=0.1, target output 1 0, i.e. class 1

- Activations of the hidden units:

$net_3= o_1 *w_{13}+ o_2*w_{23}+b_3=0.6*0.1+0.1*(-0.2)+0.1=0.14$

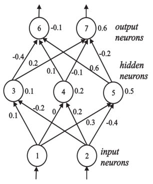
$o_3=1/(1+e^{-net3}) =0.53$

$net_4= o_1 *w_{14}+ o_2*w_{24}+b_4=0.6*0+0.1*0.2+0.2=0.22$

$o_4=1/(1+e^{-net4}) =0.55$

$net_5= o_1 *w_{15}+ o_2*w_{25}+b_5=0.6*0.3+0.1*(-0.4)+0.5=0.64$

$o_5=1/(1+e^{-net5}) =0.65$



- Activations of the output units:

$net_6= o_3 *w_{36}+ o_4*w_{46}+ o_5*w_{56} +b_6=0.53*(-0.4)+0.55*0.1+0.65*0.6-0.1=0.13$

$o_6=1/(1+e^{-net6}) =0.53$

$net_7= o_3 *w_{37}+ o_4*w_{47}+ o_5*w_{57} +b_7=0.53*0.2+0.55*(-0.1)+0.65*(-0.2)+0.6=0.52$

$o_7=1/(1+e^{-net7}) =0.63$

Back Propagation Neural Network

- 2. Backward pass for ex. 1
  - Calculate the output errors δ_6 and δ_7 (note that d_6=1, d_7=0 for class 1)

$\delta_6 = (d_6-o_6) * o_6 * (1-o_6)$

$\delta_7 = (d_7-o_7) * o_7 * (1-o_7)$

- Calculate the new weights between the hidden and output units (η=0.1)

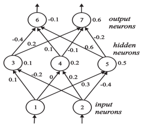
$\Delta w_{36}= \eta * \delta_6 * o_3$

$w_{36}^{new} = w_{36}^{old} + \Delta w_{36}$

$\Delta w_{37}= \eta * \delta_7 * o_3$

$w_{37}^{new} = w_{37}^{old} + \Delta w_{37}$

Similarly for w_{46}^{new} , w_{47}^{new} , w_{56}^{new} and w_{57}^{new}



For the biases b_6 and b_7 (remember: biases are weights with input 1):

$\Delta b_6= \eta * \delta_6 * 1$

$b_6^{new} = b_6^{old} + \Delta b_6$

Similarly for b_7

Back Propagation Neural Network

- 2. Backward pass for ex. 1
  - Calculate the output errors δ_6 and δ_7 (note that d_6=1, d_7=0 for class 1)

$\delta_6 = (d_6-o_6) * o_6 * (1-o_6)=(1-0.53)*0.53*(1-0.53)=0.12$

$\delta_7 = (d_7-o_7) * o_7 * (1-o_7)=(0-0.63)*0.63*(1-0.63)=-0.15$

- Calculate the new weights between the hidden and output units (η=0.1)

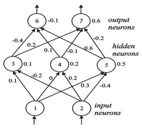
$\Delta w_{36}= \eta * \delta_6 * o_3 = 0.1*0.12*0.53=0.006$

$w_{36}^{new} = w_{36}^{old} + \Delta w_{36} = -0.4+0.006=-0.394$

$\Delta w_{37}= \eta * \delta_7 * o_3 = 0.1*-0.15*0.53=-0.008$

$w_{37}^{new} = w_{37}^{old} + \Delta w_{37} = 0.2-0.008=-0.19$

Similarly for w_{46}^{new} , w_{47}^{new} , w_{56}^{new} and w_{57}^{new}



For the biases b_6 and b_7 (remember: biases are weights with input 1):

$\Delta b_6= \eta * \delta_6 * 1 = 0.1*0.12=0.012$

$b_6^{new} = b_6^{old} + \Delta b_6 = -0.1+0.012=-0.012$

Similarly for b_7

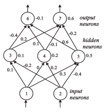
Back Propagation Neural Network

- Calculate the errors of the hidden units δ_3, δ_4 and δ_5

$\delta_3 = o_3 * (1-o_3) * (w_{36}* \delta_6 +w37 * \delta_7) =$

$= 0.53*(1-0.53)(-0.4*0.12+0.2*(-0.15))=-0.019$

Similarly for δ_4 and δ_5



- Calculate the new weights between the input and hidden units (η=0.1)

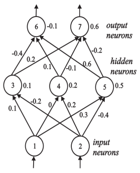
$\Delta w_{13}= \eta * \delta_3 * o_1 = 0.1*(-0.019)*0.6=-0.0011$

$w_{13}^{new} = w_{13}^{old} + \Delta w_{13} = 0.1-0.0011=0.0989$

Similarly for w_{23}^{new} , w_{14}^{new} , w_{24}^{new} , w_{15}^{new} and w_{25}^{new}; b_3, b_4 and b_6

Back Propagation Neural Network

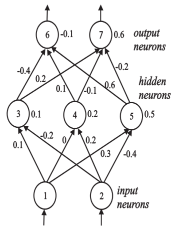
3. Repeat the same procedure for the other training examples
   - Forward pass for ex. 2…backward pass for ex.2…
   - Forward pass for ex. 3…backward pass for ex. 3…
   - …
   - Note: it's better to apply input examples in random order

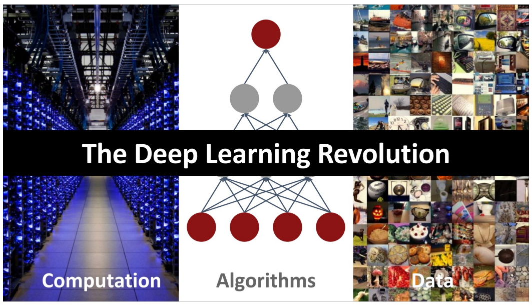


Back Propagation Neural Network

4. At the end of the epoch – check if the stopping criteria is satisfied:

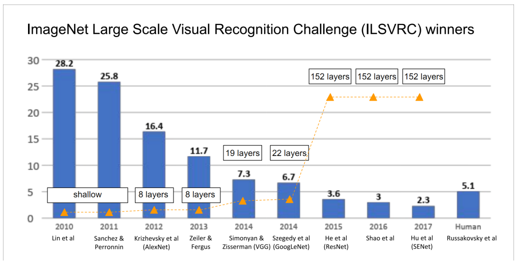
- **if yes: stop training**
- **if not, continue training:**
  - **epoch++**
  - **go to step 1**





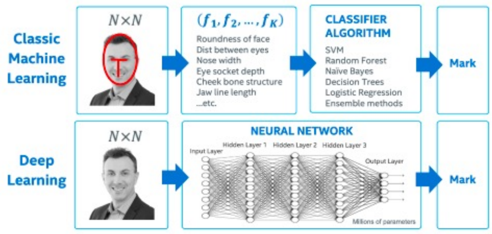
The Deep Learning Revolution

Vision and Deep Learning



ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

Traditional ML vs Deep Learning



Classic Machine Learning

$N \times N$

$(f_1, f_2, \dots, f_K)$

Roundness of face
Dist between eyes
Nose width
Eye socket depth
Cheek bone structure
Jaw line length
...etc.

CLASSIFIER ALGORITHM

SVM
Random Forest
Naïve Bayes
Decision Trees
Logistic Regression
Ensemble methods

Mark

Deep Learning

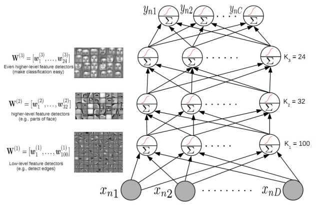
$N \times N$

NEURAL NETWORK

Mark

Neural Networks: The Features Learned

- Deep neural networks are good at detecting features at multiple layers of abstraction
- The connection weights between layers can be thought of as feature detectors or filters


y_{n1}
y_{n2}
y_{nC}
$\mathbf{W}^{(3)} = [\boldsymbol{w}_1^{(3)}, \dots, \boldsymbol{w}_{24}^{(3)}]$
Even higher-level feature detectors (make classification easy)
$K_3 = 24$
$\mathbf{W}^{(2)} = [\boldsymbol{w}_1^{(2)}, \dots, \boldsymbol{w}_{32}^{(2)}]$
higher-level feature detectors (e.g., parts of face)
$K_2 = 32$
$\mathbf{W}^{(1)} = [\boldsymbol{w}_1^{(1)}, \dots, \boldsymbol{w}_{100}^{(1)}]$
Low-level feature detectors (e.g., detect edges)
$K_1 = 100$
x_{n1}
x_{n2}
x_{nD}

- Lowest layer weights detect generic features, higher level weights detect more specific features
- Features learned in one layer are composed of features learned in the layer below

With Deep Neural Network

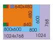
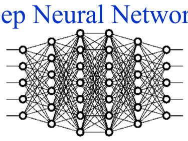


Image with
28 x 28 x 3
pixels



Number of weights in the first hidden layer will be 2352

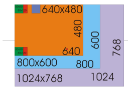
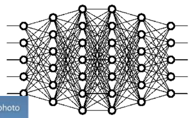


Image with
200 x 200 x 3
pixels

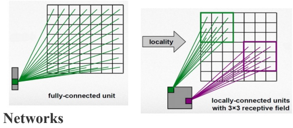


Number of weights in the first hidden layer will be 120,000

» We have Larger size image in todays era

Feature Extraction Using Convolution

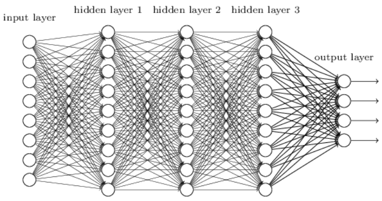
- **Fully Connected Networks**
  - Fully connect all the hidden units to all the input units.
  - With small images it was computationally feasible to learn features on the entire image.
  - But, with larger images learning features it is very computationally expensive



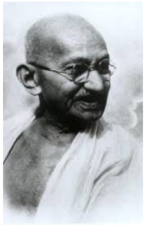
- **Locally Connected Networks**
  - Restrict the connections between the hidden units and the input units,
    - Allow each hidden unit to connect to only a small subset of the input units.
    - Each hidden unit connects to only a small contiguous region of pixels.
  - This idea of having locally connected networks also draws inspiration from how the early visual system is wired up in biology.
    - Neurons in the visual cortex have localized receptive fields (i.e., they respond only to stimuli in a certain location).

ANN: Too many parameters

- **We know it is good to learn a small model.**
- **From this fully connected model, do really need all the edges?**
- **Can some of these be shared?**

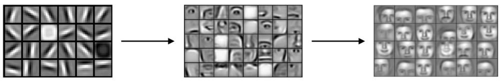


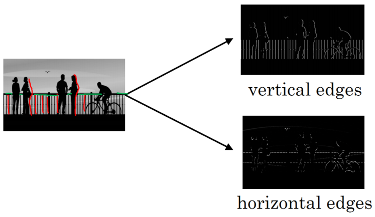
Can you recognize ??



learned features over small patches sampled randomly from the larger image

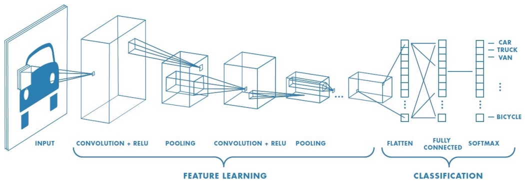
Computer Vision Problem





Neural network with many convolutional layers

- Natural images have the property of being "'stationary"', meaning that the statistics of one part of the image are the same as any other part.
- This suggests that the features that we learn at one part of the image can also be applied to other parts of the image, and we can use the same features at all locations.

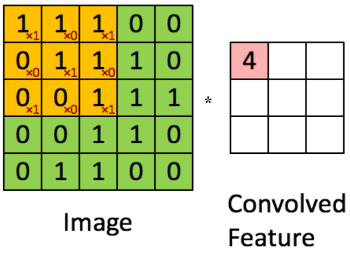


- 3 Layer in a convolutional network:
  - Convolution (CONV) - Pooling (POOL)
  - Fully connected (FC)

Convolution

- Convolution is a pointwise multiplication of two functions to produce a third function.
- Primary purpose of convolution in CNN is to extract features from the input image.
- Matrix formed by sliding the filter over the image and computing the dot product is called the 'Convolved Feature' or 'Activation Map' or the 'Feature Map'.

Convolution Example: Vertical Edge Detection



1 ×1	1 ×0	1 ×1	0	0
0 ×0	1 ×1	1 ×0	1	0
0 ×1	0 ×0	1 ×1	1	1
0	0	1	1	0
0	1	1	0	0

*

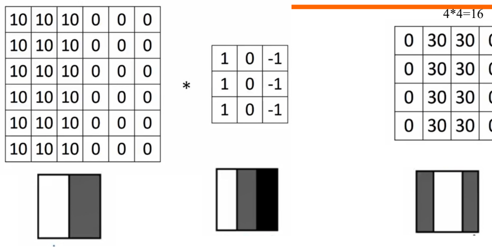
4		

Image

Convolved Feature

Detecting Vertical edges

6*6 = 36



10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

*

1	0	-1
1	0	-1
1	0	-1

4*4=16

0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0

- In case of ANN # parameter to train = 36*16 = 576
- In case of CNN # parameter to train = 9

Filter Weights

1	1	1
0	0	0
-1	-1	-1

Horizontal Filter

1	0	-1
2	0	-2
1	0	-1

Sobel Filter

3	0	-3
10	0	- 10
3	0	-3

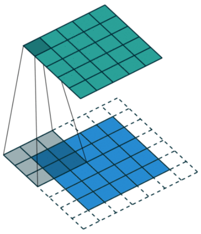
Schorr Filter

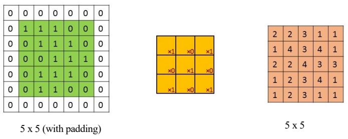
w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

Convolutional Neural Networks automatically estimates the weights of the filter

Padding

- **Two problems**
  - Shrinking output
  - Through away info from edges: Pixel at the corner are used much lesser then the pixel at middle
- Padding is used to preserve the original dimensions of the input
- Zeros are added to outside of the input
- Number of zero layers depend upon the size of the kernel





0	0	0	0	0	0	0
0	1	1	1	0	0	0
0	0	1	1	1	0	0
0	0	0	1	1	1	0
0	0	1	1	1	0	0
0	0	1	1	0	0	0
0	0	0	0	0	0	0

5 x 5 (with padding)

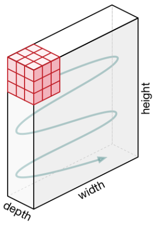
×1	×0	×1
×0	×1	×0
×1	×0	×1

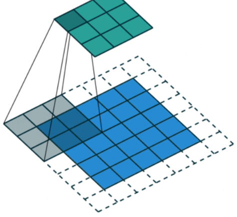
2	2	3	1	1
1	4	3	4	1
2	2	4	3	3
1	2	3	4	1
1	2	3	1	1

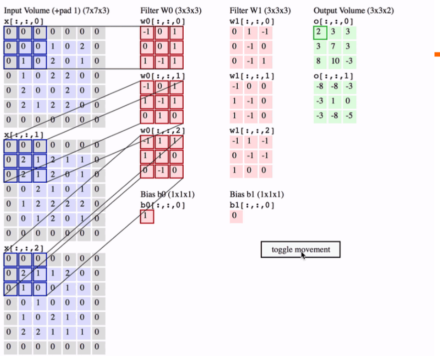
5 x 5

Strided Convolutions

- Stride defines the number of nodes a filter moves between two consecutive convolution operations
- Likewise, we have a stride to define the same when applying pooling
  - When the stride is 1 then we move the filters to 1 pixel at a time.
  - When the stride is 2 then we move the filters to 2 pixels at a time and so on.







Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	0	0	1	0	2	0
0	1	0	2	0	1	0
0	1	0	2	2	0	0
0	2	0	0	2	0	0
0	2	1	2	2	0	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	2	1	2	1	1	0
0	2	1	2	0	1	0
0	0	2	1	0	1	0
0	1	2	2	2	2	0
0	0	1	2	0	1	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	2	1	1	2	0	0
0	1	0	0	1	0	0
0	0	1	0	0	0	0
0	1	0	2	1	0	0
0	2	2	1	1	1	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

w0[:,:,0]

-1	0	1
0	0	1
1	-1	1

w0[:,:,1]

-1	0	1
1	-1	1
0	1	0

w0[:,:,2]

-1	1	1
1	1	0
0	-1	0

Bias b0 (1x1x1)

b0[:,:,0]

1

Filter W1 (3x3x3)

w1[:,:,0]

0	1	-1
0	-1	0
0	-1	1

w1[:,:,1]

-1	0	0
1	-1	0
1	-1	0

w1[:,:,2]

-1	1	-1
0	-1	-1
1	0	0

Bias b1 (1x1x1)

b1[:,:,0]

0

Output Volume (3x3x2)

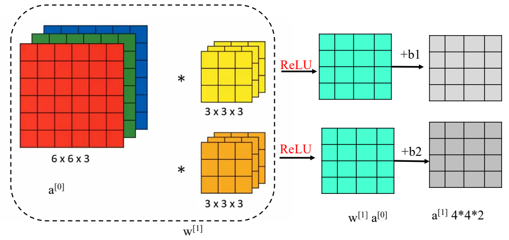
o[:,:,0]

2	3	3
3	7	3
8	10	-3

o[:,:,1]

-8	-8	-3
-3	1	0
-3	-8	-5

One layer of a convolutional network



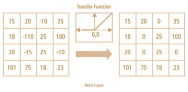
$w^{[1]}$

2 filters means two units here

$z^{[1]} = w^{[1]} a^{[0]} + b^{[1]}$
$a^{[1]} = g(z^{[1]})$

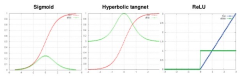
Non Linearity (ReLU)

- ReLU stands for Rectified Linear Unit for a non-linear operation. The output is $f(x) = \max(0,x)$.
- Why ReLU is important :
  - ReLU's purpose is to introduce non-linearity in our ConvNet.
  - Result should be non-negative linear.
- Other non linear functions
  - tanh or
  - sigmoid



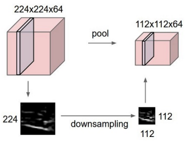
15	20	-10	35
18	-110	25	100
20	-15	25	-10
101	75	18	23

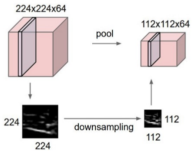
15	20	0	35
18	0	25	100
20	0	25	0
101	75	18	23

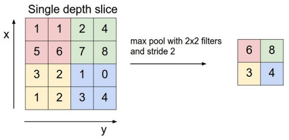


Pooling [Downsampling] Layer

- Used to downsample the representation-size after convolution step
- Pooling layers section would reduce the number of parameters when the images are too large.
- Spatial pooling also called subsampling or downsampling which reduces the dimensionality of each map but retains important information.
- Ensures robustness against minor rotations, shifts, corruptions in the image
- Popular approaches:
  - Max-pooling , Average Pooling, Sum Pooling, etc



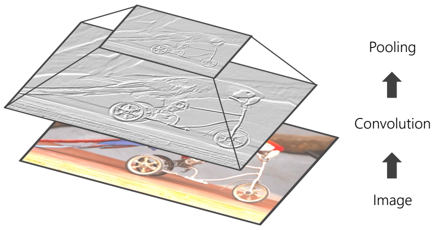




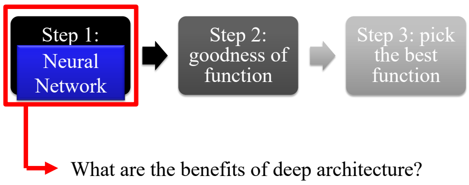
1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

6	8
3	4

A Basic Module of the CNN



Concluding Remarks



What are the benefits of deep architecture?

Deeper is Better?

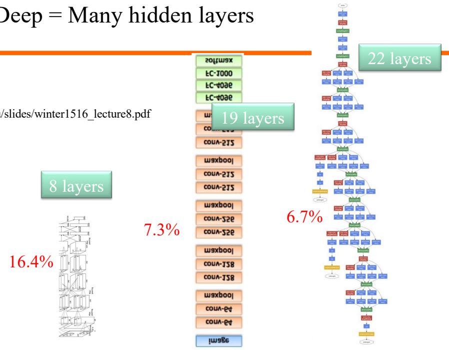
Layer X Size	Word Error Rate (%)
1 X 2k	24.2
2 X 2k	20.4
3 X 2k	18.4
4 X 2k	17.8
5 X 2k	17.2
7 X 2k	17.1

Not surprised, more parameters, better performance

Seide, Frank, Gang Li, and Dong Yu. "Conversational Speech Transcription Using Context-Dependent Deep Neural Networks." *Interspeech*. 2011.

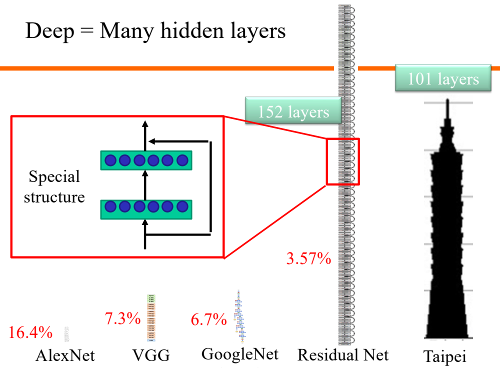
Deep = Many hidden layers

http://cs231n.stanford.edu/slides/winter1516_lecture8.pdf



AlexNet (2012) VGG (2014) GoogleNet (2014)

Deep = Many hidden layers



AlexNet (2012) VGG (2014) GoogleNet (2014) Residual Net (2015) Taipei 101

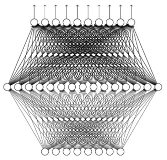
Universality Theorem

Any continuous function f

$$f : R^N \rightarrow R^M$$

Can be realized by a network with one hidden layer

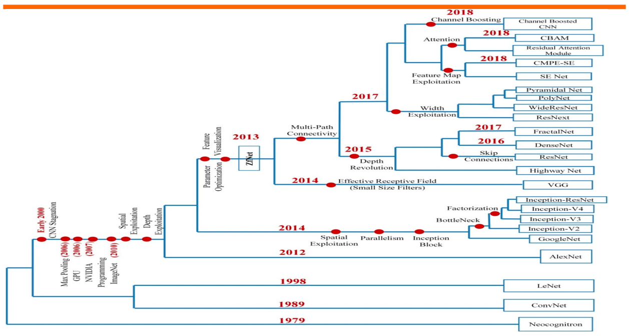
(given **enough** hidden neurons)



Reference for the reason: http://neuralnetworksanddeeplearning.com/chap4.html

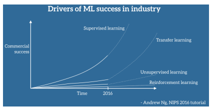
Why "Deep" neural network not "Fat" neural network?

Famous CNN Models



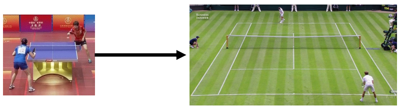
"You need a lot of a data if you want to train/use CNNs"

Eg: Cycle riding



When to Use Transfer Learning?

- Situation where what has been learned in one setting is exploited to improve generalization in another setting.



- Task A and B have the same input x
- You have a lot more data for Task A than Task B.
- Low level features from A could be helpful for learning B.

Transfer Learning Process

1. Select a pre-trained model
   - https://keras.io/api/applications/
2. Classify your problem according to the Size-Similarity Matrix
3. Fine-tune model.

WARNING :

Don’t use this until you know the maths in the background

Challenge: Real-World Applications

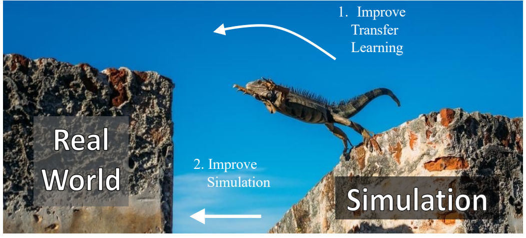
Reminder:

Supervised learning:
teach by example

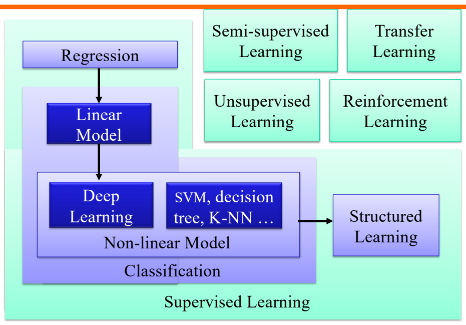
Reinforcement learning:
teach by experience

Open **Challenges**. Two Options:

1. Real world observation + one-shot trial & error
2. Realistic simulation + transfer learning



Learning Map



Next

- Module 9: AI Applications & Ethics
  - PART 9.1 : Computer Vision and Robotics
  - PART 9.2 : Natural language understanding
  - PART 9.3 : AI in Healthcare
  - PART 9.4 : Ethics of AI