CS456: Machine Learning

Artificial Neural Networks & Deep learning

Jakramate Bootkrajang

Department of Computer Science
Chiang Mai University

Objectives

- To understand the philosophy behind neural networks classifier
- To understand how to train neural network models
- To understand the effect of important neural networks parameters
- To understand what deep neural networks are

Outlines

- A brief history of neural network
- Multilayer neural network
- Backpropagation
- Deep neural network
  - Convolutional neural networks (CNNs)
  - Long-Short Term Memory (LSTM)

Artificial Neural Networks

Facts of Human Brain

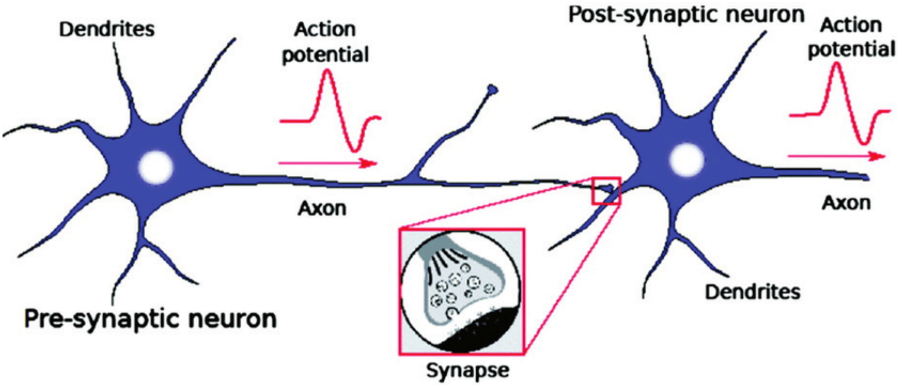
(complex, nonlinear and parallel computer)

- The brain contains about 10^{10} (100 billion) basic units called neurons
- Each neuron connected to about 10^4 other neurons
- Weight: birth 0.3 kg, adult ~1.5 kg
- Power consumption 20-40W (~20% of body consumption)
- Signal propagation speed inside the axon ~90m/s in ~170,000 Km of axon length for adult male
- Firing frequency of a neuron ~250 – 2000Hz
- Operating temperature: 37±2°C
- Sleep requirement: average 7.5 hours (adult)

Intel Pentium 4 1.5GHz

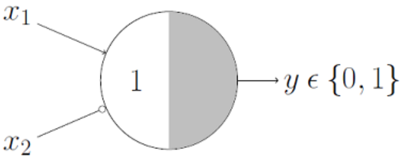
Number of transistors	$4.2\text{x}10^7$
Power consumption	up to 55 Watts
Weight	0.1 kg cartridge w/o fans, 0.3 kg with fan/heatsink
Maximum firing frequency	1.5 GHz
Normal operating temperature	15-85°C
Sleep requirement	0 (if not overheated/ overclocked)
Processing of complex stimuli	if can be done, takes a long time

A (biological) neuron



1940s: Artificial neuron

Warrent McCulloch (neuroscientist) and Walter Pitts (logician) modelled a logic unit after the theories of how neuron works



$$x_1 \ AND \ !x_2{}^*$$

Figure: https://towardsdatascience.com/mcculloch-pitts-model-5fdf65ac5dd1

1950s: Perceptron

F.Rosenblatt's perceptron with learnable weights

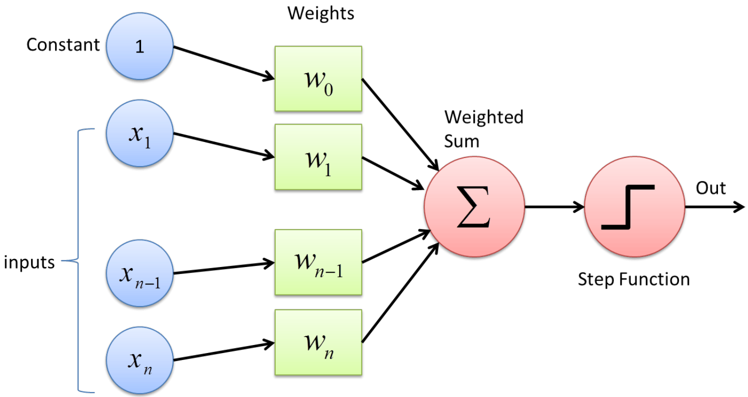
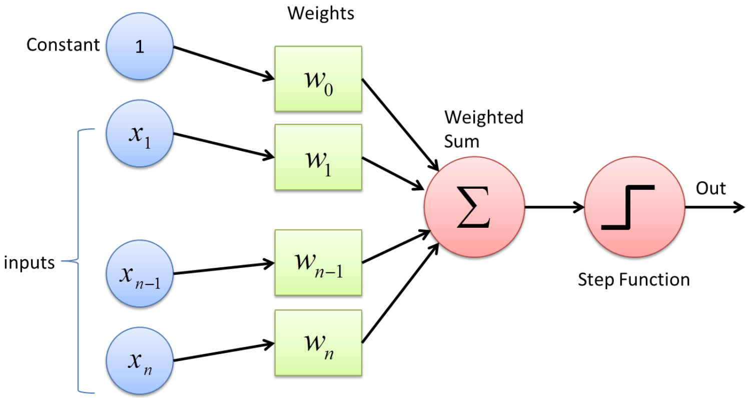


Figure: https://towardsdatascience.com/what-the-hell-is-perceptron-626217814f53

1950s: Perceptron

Rosenblatt's perceptron acts as linear function of inputs, $\mathbf{w}^T\mathbf{x}$



1970s: XOR problem

M.Minsky showed that the perceptron is not suitable for non-linear problems

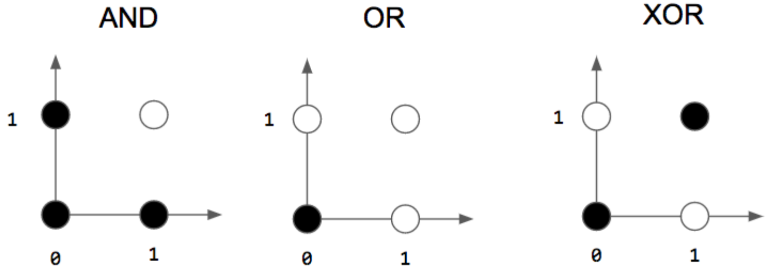


Figure: `https://www.rawanyat.com/data-science/2018/1/25/classification-via-perceptrons`

1980s: AI winter

No significant developments

1990s: Multilayer perceptron

D.Rumelhart, G.Hinton put forward multilayer perceptron with non-linear differentiable activation function and backpropagation algorithm to train the model

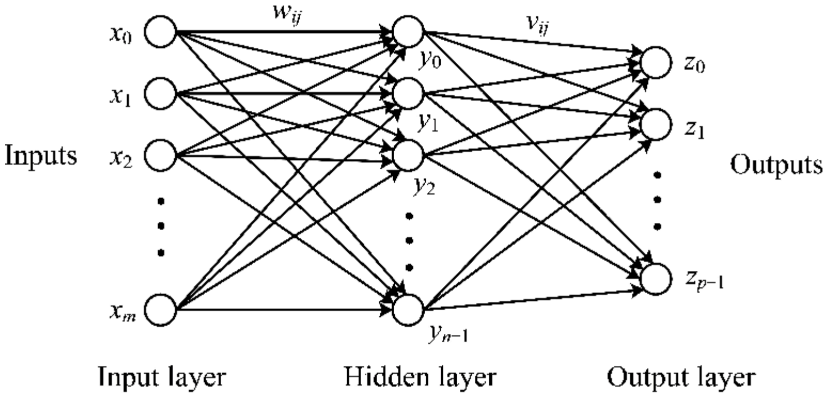


Figure: Shao, Changpeng. "A Quantum Model for Multilayer Perceptron." (2018).

2010s: Deep learning

G.Hinton, Y.Lecun increased the depth of MLP and proposed ways to deal with learning millions of parameters (weight sharing, pretraining)

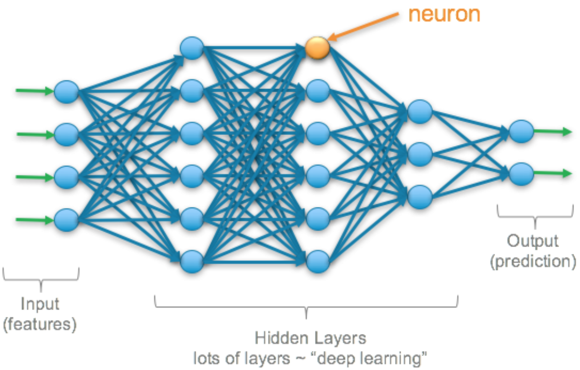
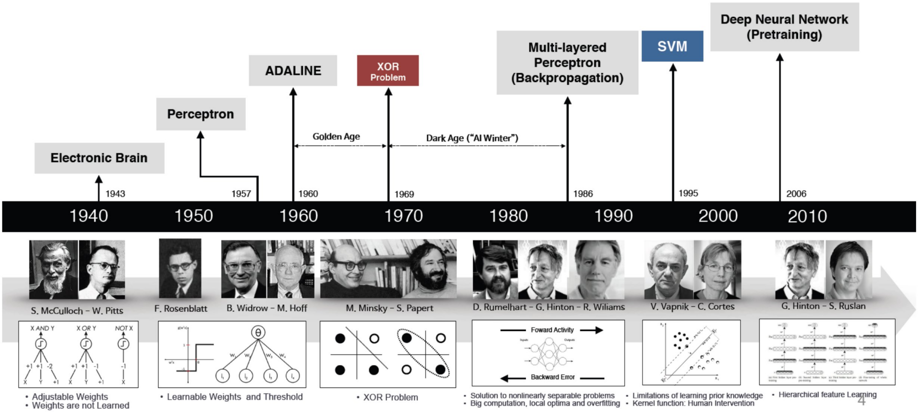


Figure: `https://mc.ai/deep-learning-overview-of-neurons-and-activation-functions/`

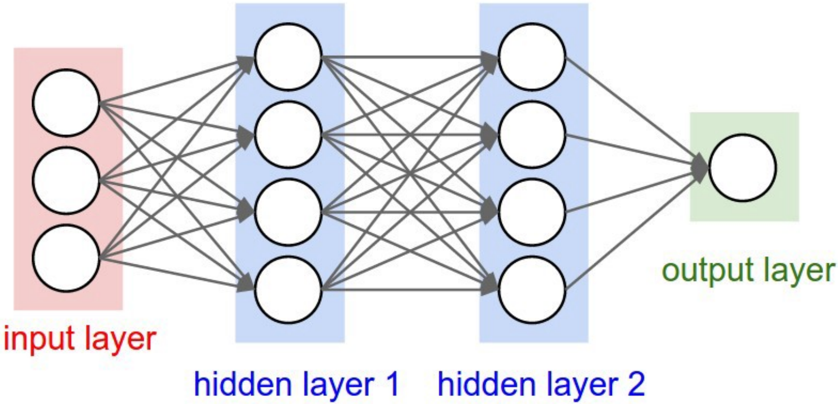
The timeline



Neural network

- A bio-inspired (directed) weighted graph where vertices (nodes) are organised into sets, and any two sets of vertices may form bi-partite graph.
- Set of vertices is referred to as a layer
- Typically, there are three types of layer,
  - input layer
  - hidden layer
  - output layers
- The edges represent the weights

Neural network layers



Note: Perceptron is neural network with no hidden layer

Input layer

- An interface where data enters the network
- Usually, the size of the layer (number of nodes) equals to the dimension of data

Hidden layer

- Hidden layer adds capacity to the network
- The more the hidden layers, the higher the capacity
  - able to represent complicated non-linear function
- The number of hidden layers is referred to as depth of network
  - while the number of neuron in one hidden layer is referred to as width of layer

Output layer

- A layer where prediction is made
- The number of output nodes depends on dimension of y, the target
  - regression: (usually) 1 output node
  - binary classification: (usually) 1 output node
  - multiclass classification: k output node, (k is the number of classes)

Node and its activation function

- A node in neural network
  - receives inputs,
  - aggregates the results, (often using simple weighted sum)
  - passes it through node's activation function
  - sends the output as an input to nodes in the next layer

Activation function

- Mimic behaviour of synapes in human brain
- Only send out information only if the input is strong enough

Various Activation functions

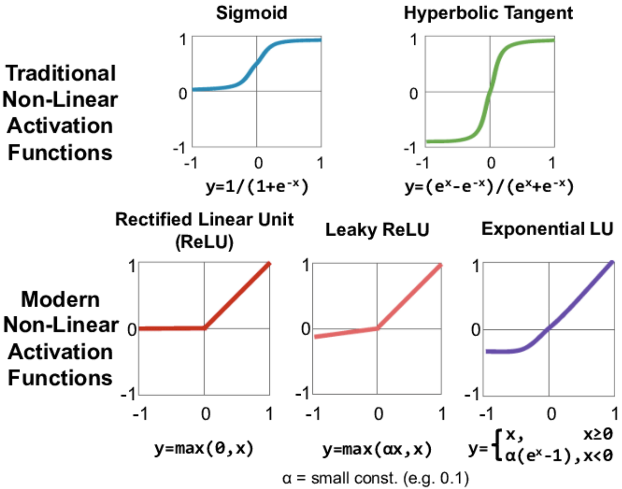


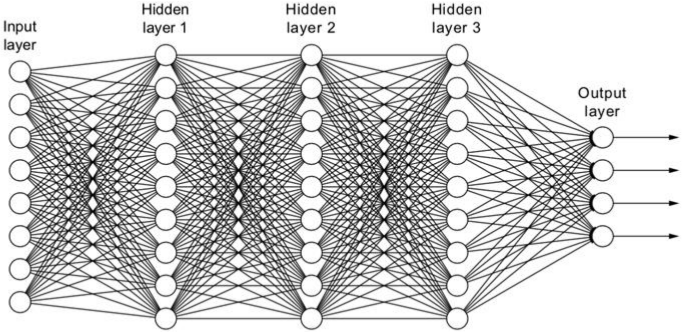
Figure: V.Sze et.al.:Efficient Processing of Deep Neural Networks: A Tutorial and Survey

Types of NNs

- Fully connected NN
- Recurrent NN
- Lateral NN
- Partially connected NN

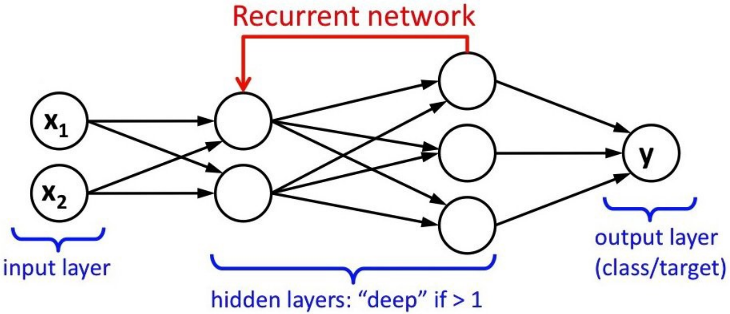
Fully connected neural network

A node from previous layer is connected to all nodes in the next layer



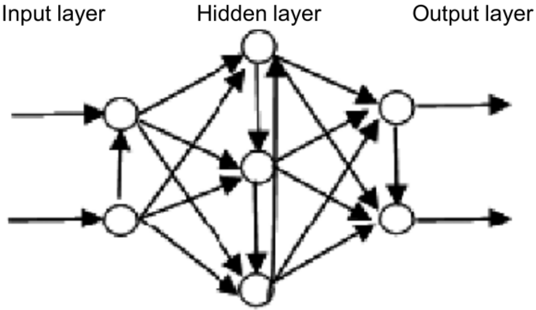
Recurrent neural network

A node from the next layer may be connected to nodes in the previous layer



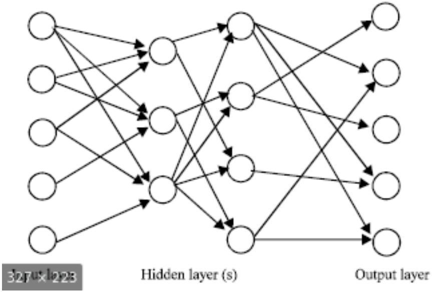
Lateral networks

There exists connections between nodes in the same layer



Partially connected NN

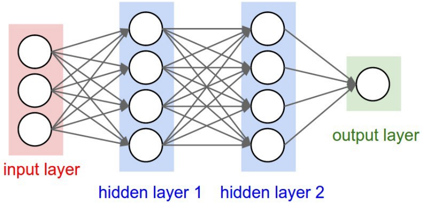
A sub-set of all possible connections of fully connected network. More similar to human brain.



How to train NNs ?

- Supervised learning (current focus)
- Unsupervised learning
  - e.g, for training a self-organising map, autoencoder
- Reinforcement learning
  - e.g, for training Deep Q Network (DQN)

Feedforward and backpropagation (supervised learning)



- Data flow from left to right to generate output (feed forward)
- The generated output will be compared with desired target so that we can measure error (this is a form of supervised learning)
- The error will back propagate in the reverse order from right to left and are used to update weights along the way

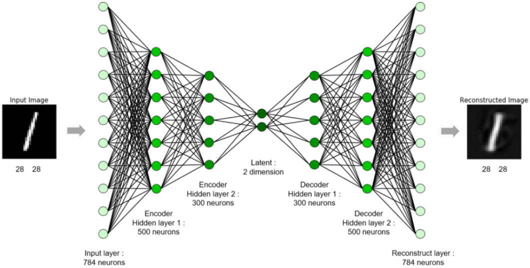
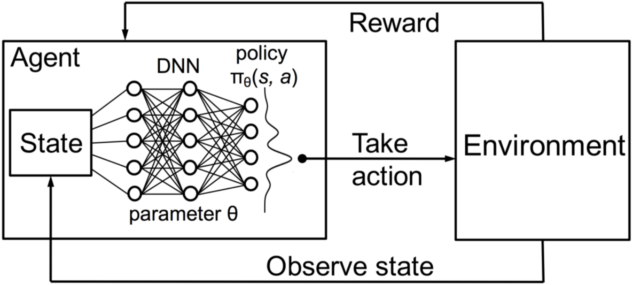


Figure: `https://mc.ai/auto-encoder-in-biology/`

- Goal: to find the compact representation of inputs
- Usually trained using backpropagation with the input itself as target

Reinforcement learning

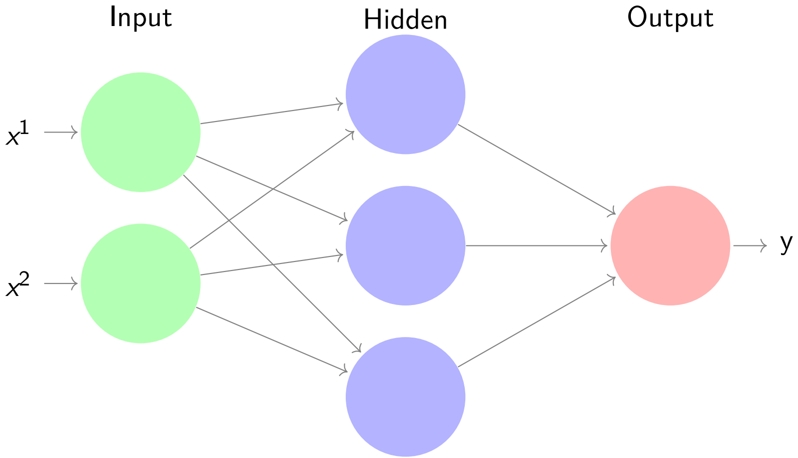


https://java.works-hub.com/learn/using-deep-q-learning-in-fifa-18-to-perfect-the-art-of-free-kicks-9bf7f

Backpropagation

Let's do a simple forward run

Given $(\mathbf{x}, y) = ([0.5, 0.1], 1)$,



Measuring the error

- For regression: Squared error
  - $\sum_{i=1}^{N} \frac{1}{2}(y - \hat{y})^2$
- For binary classification: binary cross entropy (neg.log.likelihood)
  - $-\sum_{i=1}^{N} \Big(\delta(y_i = 0)\log(P_0) + \delta(y_i = 1)\log(P_1)\Big)$ for $y \in \{0, 1\}$
  - P_1 is represented by the sigmoid function $\frac{1}{1+e^{-\mathbf{w}^T\mathbf{x}}}$
- For multi-class classification: cross entropy
  - $-\sum_{i=1}^{N}\sum_{k=1}^{K} \delta(y_i = k)\log(P_k)$ for $y \in \{1, \ldots, K\}$, $P_k = \frac{e^{-\mathbf{w}_k^T\mathbf{x}}}{\sum_k e^{-\mathbf{w}_k^T\mathbf{x}}}$
  - P_k is represented by the softmax function
- These are called loss function denoted $\mathcal{L}$

Minimising the error

- Minimising the error can be done by minimising the loss function
- Standard gradient descent method can be employed
- The optimisation plan
  1. Decide the loss function for our problem
  2. Find the derivative of the loss w.r.t w_{ij} for all i, j
  3. Update $w_{ij}^{new} = w_{ij}^{old} - \eta \nabla_{w_{ij}} \mathcal{L}$

Problem with the plan

- It is quite inefficient, (repeated gradient calculation)
- Therefore, not scalable to deep networks
- Backpropagation solves this inefficiency by
  - Computing the derivative of w_{ij} in terms of derivative of weight in the next layer w.r.t error
  - instead of finding derivative of w_{ij} w.r.t the final error

Chain rule

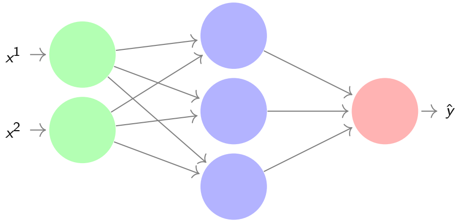
Useful for finding derivative of a composite function

$$\frac{\partial f(g(h(x)))}{\partial x} = \frac{\partial f(g(h(x)))}{\partial g(h(x))} \frac{\partial g(h(x))}{\partial x}$$
$$= \frac{\partial f(g(h(x)))}{\partial g(h(x))} \frac{\partial g(h(x))}{\partial h(x)} \frac{\partial h(x)}{\partial x} \quad (1)$$

Note: a neural network can be viewed as a big composite function

Let's find the update rules (1/3)

Assume $\mathcal{L} = \sum_{i=1}^{N} \frac{1}{2}(y_i - \hat{y}_i)^2$, for node j, $o_j = \sigma(net_j) = \sigma(\sum_{i=1}^{k} w_{kj} o_k)$



$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial w_{ij}} &= \frac{\partial \mathcal{L}}{\partial o_j} \frac{\partial o_j}{\partial w_{ij}} \\
&= \frac{\partial \mathcal{L}}{\partial o_j} \frac{\partial o_j}{\partial net_j} \frac{\partial net_j}{\partial w_{ij}} \qquad (2)
\end{aligned}$$

$$\frac{\partial net_j}{\partial w_{ij}} = \frac{\partial \sum_{i=1}^{k} w_{kj} o_k}{\partial w_{ij}} = o_i \qquad (3)$$

$$\frac{\partial o_j}{\partial net_j} = \frac{\partial \sigma(net_j)}{\partial net_j} = \sigma(net_j)(1 - \sigma(net_j)) \qquad (4)$$

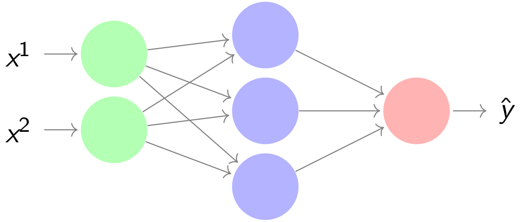
Let's find the update rules (2/3)

Two cases for $\frac{\partial \mathcal{L}}{\partial o_j}$

1. node j is the output node, we know that $\hat{y}_j = o_j$

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial o_j} &= \frac{\partial \sum_{i=1}^{N} \frac{1}{2}(y_i - \hat{y}_i)^2}{\partial \hat{y}_j} \\ &= \end{aligned} \tag{5}$$

Let's find the update rules (3/3)



2. node j is intemediate node, o_j acts as input to nodes in the next layer

[observation] o_j affects $\mathcal{L}$ via net_k for all k in the next layer

$$
\begin{aligned}
\frac{\partial \mathcal{L}}{\partial o_j} &= \sum_k \left(\frac{\partial \mathcal{L}}{\partial net_k} \frac{\partial net_k}{\partial o_j} \right) \\
&= \sum_k \left(\frac{\partial \mathcal{L}}{\partial o_k} \frac{\partial o_k}{\partial net_k} \frac{\partial net_k}{\partial o_j} \right) = \sum_k \left(\frac{\partial \mathcal{L}}{\partial o_k} \sigma(net_k)(1 - \sigma(net_k)) w_{jk} \right) \quad (6)
\end{aligned}
$$

Recursion !! derivative w.r.t to o_j depends on derivative w.r.t o_k

Backpropagation algorithm

Starting from output layer

- Calculate weight gradients

$$\frac{\partial \mathcal{L}}{\partial w_{ij}} = \frac{\partial \mathcal{L}}{\partial o_j}\frac{\partial o_j}{\partial net_j}\frac{\partial net_j}{\partial w_{ij}} = \delta_j o_i \quad (7)$$

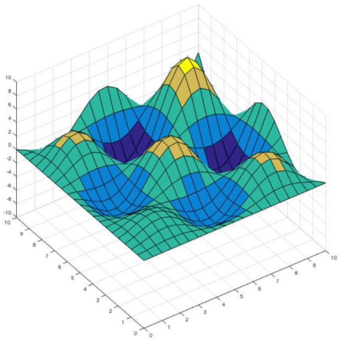
- if node i is output node
  $\delta_j = -(y - \hat{y})\sigma(net_j)(1 - \sigma(net_j)) = -(y - \hat{y})o_j(1 - o_j)$

- if node i is intermediate node $\delta_j = \left(\sum_l (w_{jl}\delta_l)\right) o_j(1 - o_j)$

- Update $w_{ij}^{new} = w_{ij}^{new} - \eta \frac{\partial \mathcal{L}}{\partial w_{ij}} = \eta \delta_j o_i$

Solution landscape

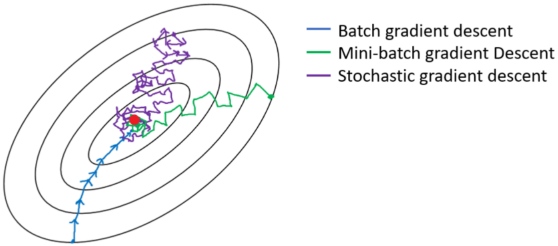
Objective function of MLP with hidden layers is non-convex (there are multiple local optima)



Gradient descent may get stuck in local optima

Stocastic Gradient descent (SDG)

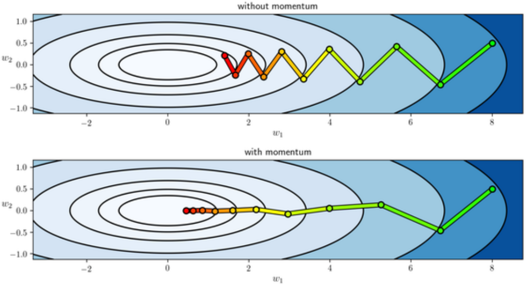
Compute gradient based on small batch of data



Might help escape local optima

Momentum

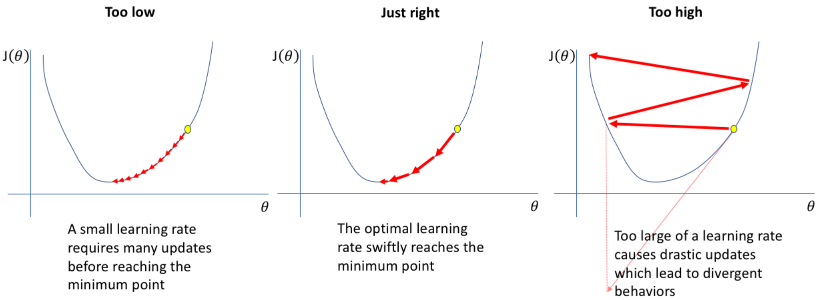
Just like in Physics, momentum helps moving object maintains direction



Momentum helps stochastic gradient descent reaches target quicker

Learning rate and rate decay

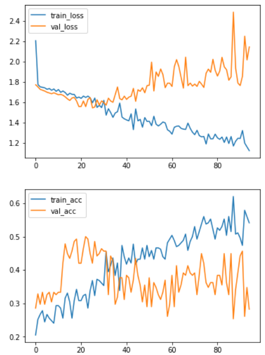
Assumption: when near stationary point avoid jumpting too much as it might overshoot the target



Helps stay focus to the target

Overfitting

The network was left learning for too long. It memorises trainining data but cannot generalise



Remedies: early stopping, dropout, regularisation

Convolutional Neural Network

NN for visual related tasks

- One of the most popular deep neural network models
- Found its use in various visual recognition tasks

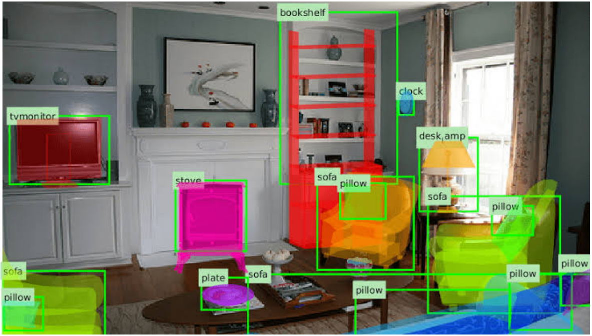


Figure: `https://neurohive.io/en/datasets/new-datasets-for-3d-object-recognition/`

Typical computer vision pipeline

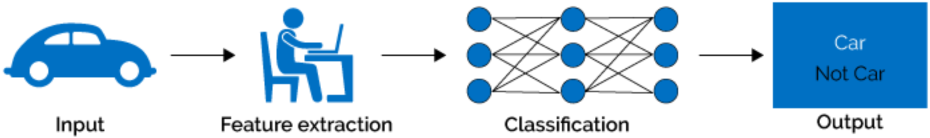


Figure: https://mc.ai/cnn-application-on-structured-data-automated-feature-extraction/

- Composed of two steps
  - Feature extraction (requires domain knowledge)
  - Classification step

Deep NN based computer vision pipeline

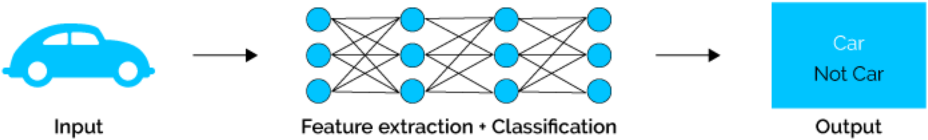


Figure: https://mc.ai/cnn-application-on-structured-data-automated-feature-extraction/

- Train an end-2-end convolutional neural network
  - No explicit feature extraction (Features are learned)
  - Classification step

Convolutional Neural Network

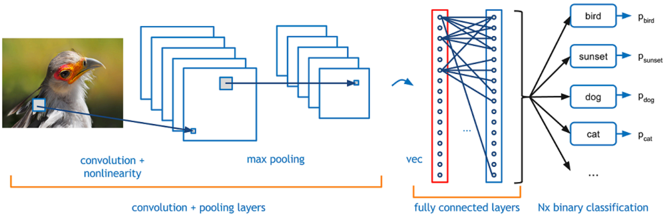


Figure: https://adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks/

A deep neural network composed of Convolutional Layers for (trainable) feature extraction part and Fully-connected NN for classification part

Anatomy of CNN

- Convolutional layer
- Pooling layer
- Fully-connected layer

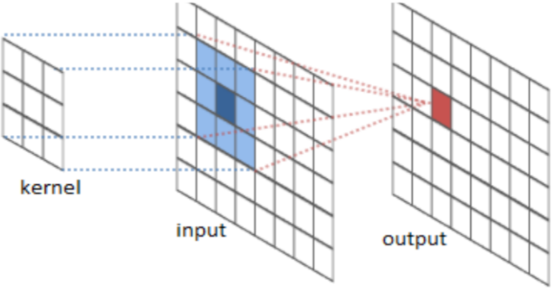
Convolution operation motivation

Why ?

- would like to detect some patterns within the input image

How ?

- define the patterns in a kernel/filter and perform pattern matching
  - the output from this operation should indicate how likely the patterns are found at the specific area



Filter: image processing vs CNN

- Filter contains pattern which we want to detect from input image
- In image processing filter is often predefined
- In CNN filter is learned from the data

Sobel edge detection kernel

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

Horizontal

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

Vertical

Kernels used in the Sobel edge detection

CNN kernels learned from ImageNet

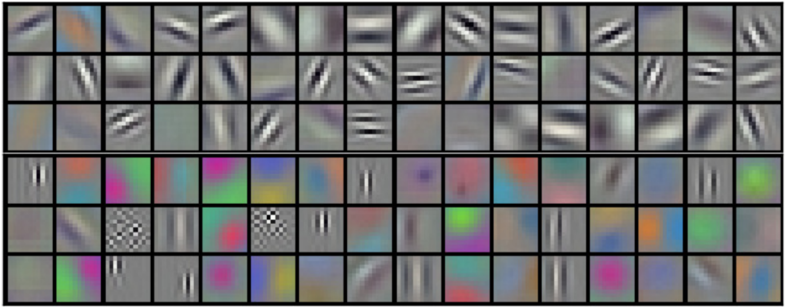
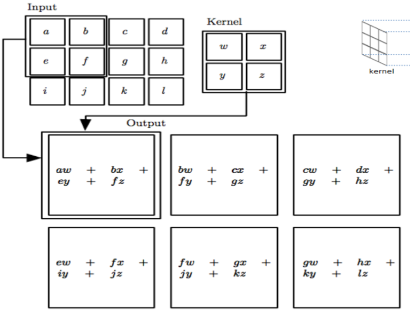


Figure: Krizhevsky et al. "ImageNet Classification with Deep Convolutional Neural Networks"

How the output is calculated ? (1/2)

Cross correlation operation (widely used despite the name)

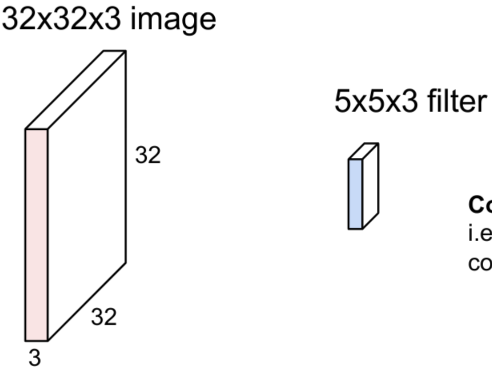
Compute the sum of element-wise products



How the output is calculated ? (2/2)

Convolution operation (flipping kernel from right to left and from top to buttom and perform cross-correlation)

Convolutional layer in CNN

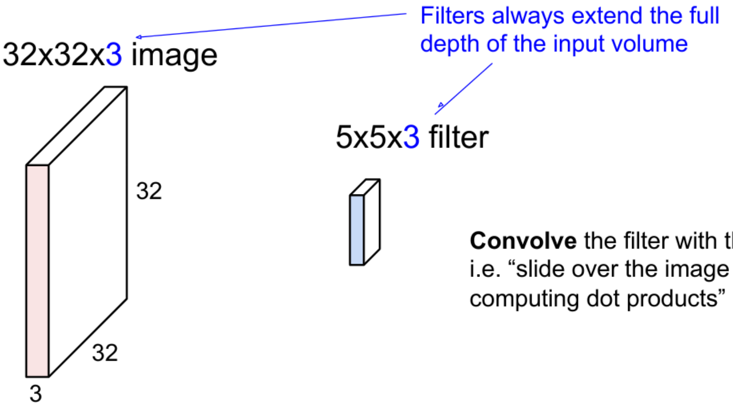


Convolve the filter with the image i.e. "slide over the image spatially, computing dot products"

slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

- Flat filter can be seen as a matrix. However, filter can have depth resulting in mathematical object called tensor

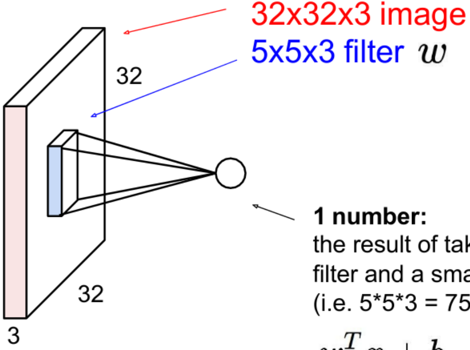
Convolutional layer in CNN



Convolve the filter with the image i.e. "slide over the image spatially, computing dot products"

slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

Convolutional layer in CNN

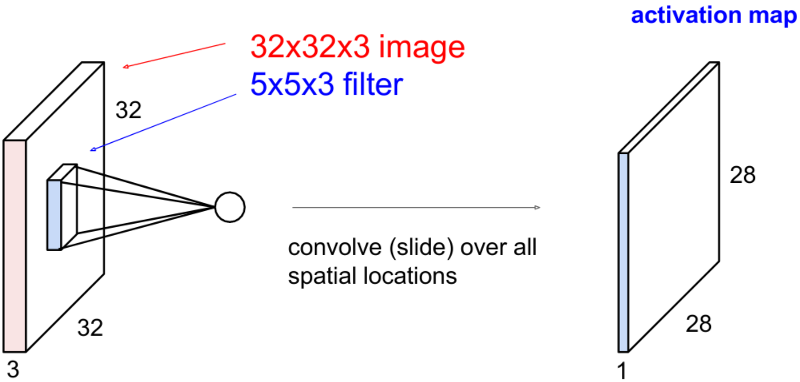


1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image (i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

Convolutional layer in CNN



slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

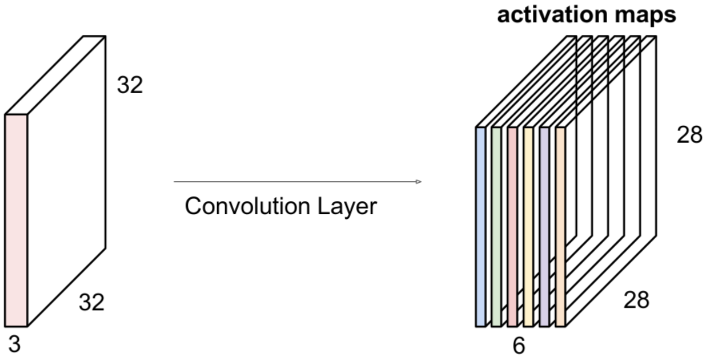
Convolutional layer in CNN

consider a second, green filter

slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

Convolutional layer in CNN

For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:

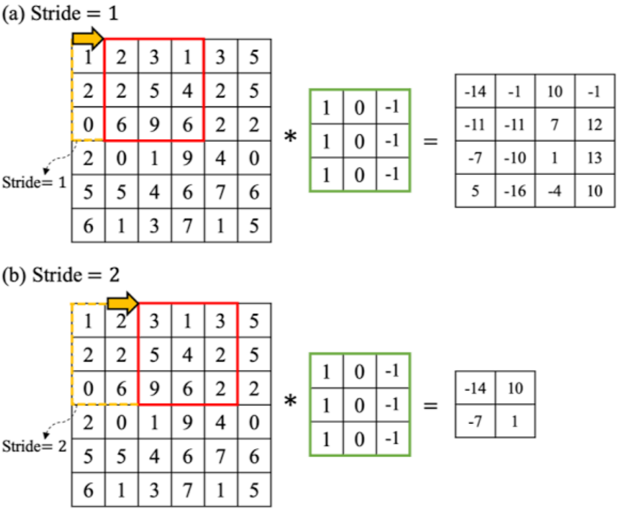


We stack these up to get a "new image" of size 28x28x6!

slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

Stride

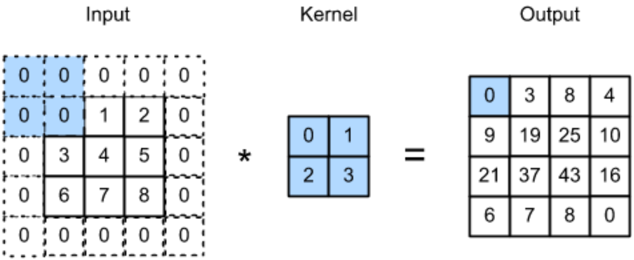
The number of row and column to skip when sliding the kernel



Created by brilliantcode.net

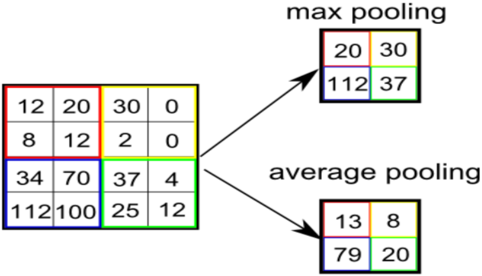
Padding

Input augmentation so that kernel output can be computed



Pooling layer

- Perform compression on input using window of size p.
- Making the representation invariant to small translation



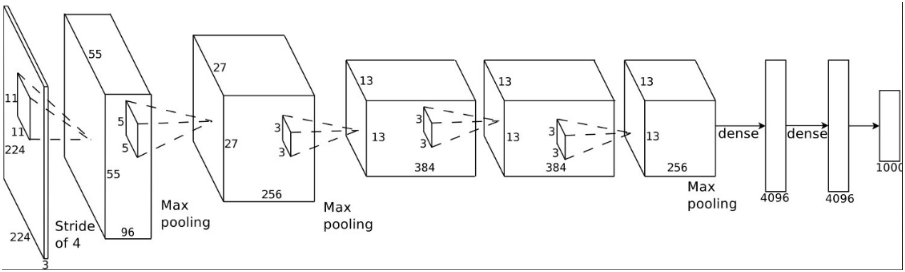
Fully-connected layer

- The final output from convolutional layer can be seen as feature vector
- The features can be fed into fully-connected NN for classification
- Some researchers have successfully used the features to train other classifiers (SVM, LR)

Transfer learning

- Many of the recognition tasks share important low-level features
- Trained convolutional layer of existing network can be used as it is (freezing the weights)
- Only train the fully-connected NN to match our current task
- Note: tasks similarity dictates the success of transfer learning (but how to measure ?)

Case study: Alexnet



- Max-pooling layers follow first, second, and fifth convolutional layers
- The number of neurons in each layer is given by 253440, 186624, 64896, 64896, 43264, 4096, 4096, 1000

Recurrent Neural Network

Motivation

- In some situations, input instances are not completely independent with some temporal dependencies.
- Examples
  - text generation: "I was born in France, I fluently speak ..."
  - weather forecast: tomorrow's weather may depend on todays' and yesterday's
- CNN or NN cannot model such temporal dependencies

Recurrent Neural Network (RNN)

An RNN is neural network where the outputs of the node can flow back into the node at the next time step

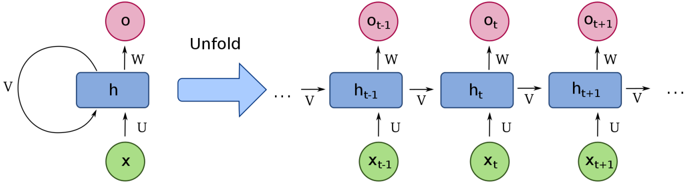
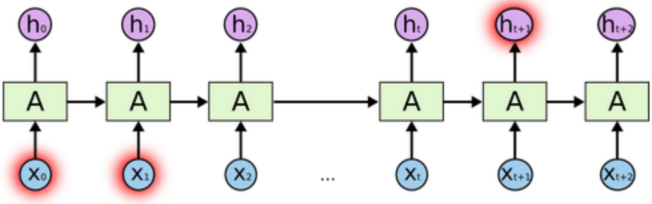


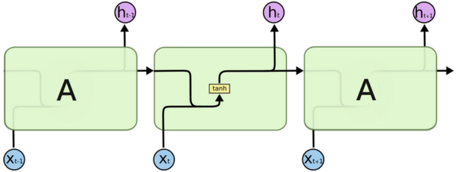
Figure: (left) typical RNN diagram, (right) diagram unfolded through time

Issue with RNN

- RNN works well but sometimes *out of context information* persists in the memory for too long

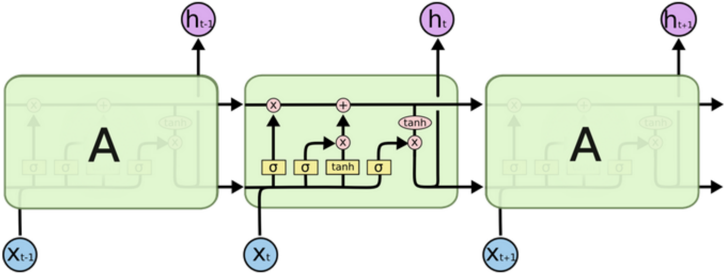


- Here, x_0, x_1 still have some influences on output h_{t+1}



Long Short Term Memory (LSTM)

- LSTM tries to solve this problem with *forgetting* mechanism

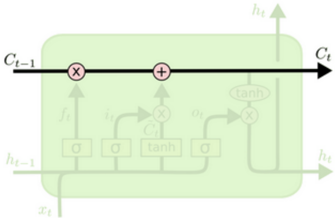


- Based heavily on `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`

Investigating LSTM components

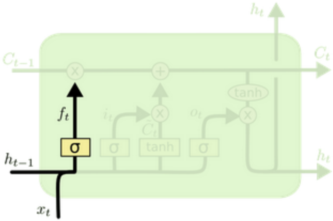
- Memory lane (cell state)
- Forgetting gate layer
- Input gate layer
- output layer

Memory lane



- It runs throught the whole chain
- Information can be removed (via X mark)
- or added to memory (via + mark)
- C_t sometimes called cell state

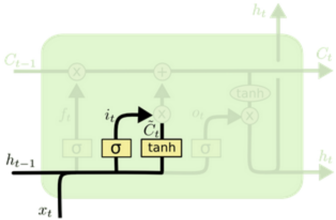
Forgetting gate layer



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

- f_t is between 0 and 1
- $f_t = 0$ indicates that C_{t-1} should be wiped out
- $f_t = 1$ retains fully the state C_{t-1}

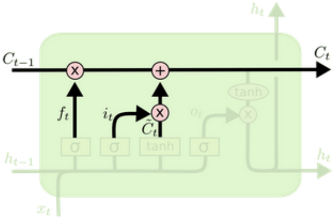
Input gate layer



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

- i_t tells which component of C_t to update
- $\tilde{C}_t$ computes the update values

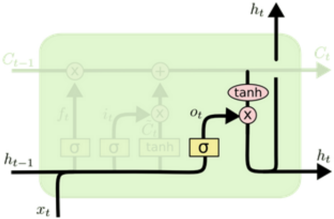
Input gate layer



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

- the new cell state equals the sum of old memories with new memories

Output layer



$$o_t = \sigma\left(W_o \left[h_{t-1}, x_t\right] + b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

- Output layer produces prediction as well as sends info back into the node in the next time step

Python code for LSTM

```python
def LSTMCELL(prev_ct, prev_ht, input):
    combine = prev_ht + input
    ft = forget_layer(combine)
    candidate = candidate_layer(combine)
    it = input_layer(combine)
    Ct = prev_ct * ft + candidate * it
    ot = output_layer(combine)
    ht = ot * tanh(Ct)
    return ht, Ct



ct = [0, 0, 0]
ht = [0, 0, 0]

for input in inputs:
    ct, ht = LSTMCELL(ct, ht, input)
```

Figure: https://towardsdatascience.com/
illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44

Applications of LSTM

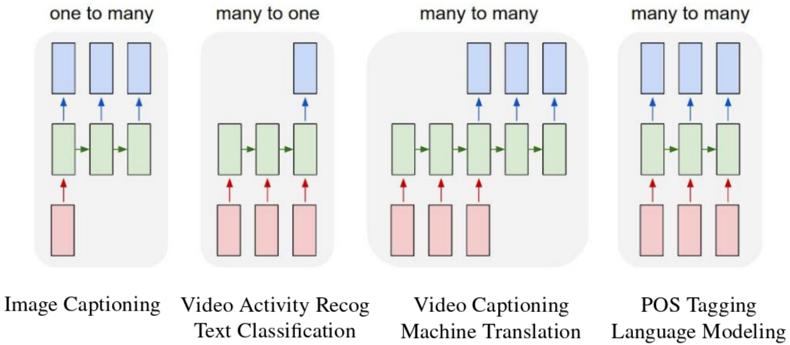


Figure: by Raymond J. Mooney

Wait! How to train LSTM ?

backpropagation

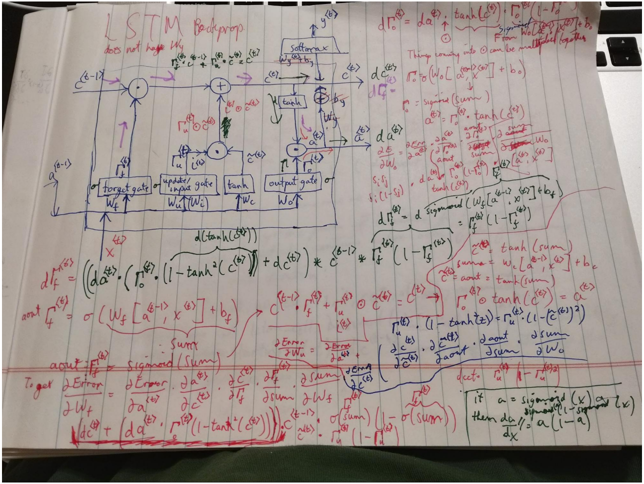


Figure: https://www.kdnuggets.com/2019/05/understanding-backpropagation-applied-lstm.html

Objectives: revisited

- To understand the philosophy behind neural networks classifier
- To understand how to train neural network models
- To understand the effect of important neural networks parameters
- To understand what deep neural networks are
    - CNN
    - LSTM

Reading list

- https://www.math.univ-toulouse.fr/~besse/Wikistat/pdf/st-m-hdstat-rnn-deep-learning.pdf
- https://project.inria.fr/deeplearning/files/2016/05/session3.pdf
- https://colah.github.io/posts/2015-08-Understanding-LSTMs/