Neural Networks

Réda DEHAK
http://ia.dehak.org

1

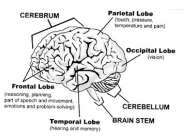
Contents

- Origins/History
- Human Brain / Biological Neuron
- Perceptron
- Neural Networks Topology
- Learning:
  - Perceptron
  - Multi Layers Neural Networks: Backpropagation Algorithm
- Conclusions

2

Origins

- Many machine learning methods inspired by biology, e.g., the (human) brain
- Neural Networks:
  - Algorithms that try to mimic the brain
  - Produce artificial systems capable of complex calculation similar to the human brain



3

History

1943 : McCulloch and Pitts proposed the McCulloch-Pitts neuron model
1949 : Hebb published his book The Organization of Behavior, in which the Hebbian learning rule was proposed.
1958 : Rosenblatt introduced the simple single layer networks now called Perceptrons.
1969 : Minsky and Papert's book Perceptrons demonstrated the limitation of single layer perceptrons, and almost the whole field went into hibernation.
1982 : Hopfield published a series of papers on Hopfield networks.
1982 : Kohonen developed the Self-Organising Maps that now bear his name.
1986 : The Back-Propagation learning algorithm for Multi-Layer Perceptrons was rediscovered and the whole field took off again.
1990s : The sub-field of Radial Basis Function Networks was developed.
2000s : The power of Neural Networks, Support Vector Machines, and Bayesian Techniques methods become apparent.
2009- : DNNs outperform other Machine Learning Methods in different competitions

4

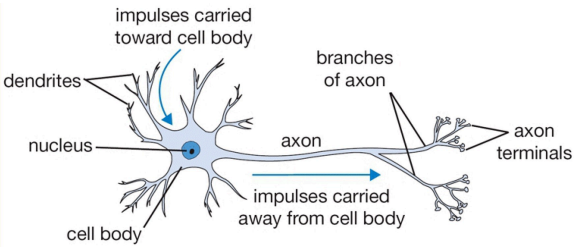
Human brain

- Our brain has ~ 10^{11} neurons, each of which communicates (is connected) to ~ 10^{4} other neurons
- The brain can fire all the neurons in a single step (parallelism).
- The human brain is extremely energy efficient, using approximately 10^{-16} joules per operation per second.
- Brains have been evolving for tens of millions of years

	Processing Elements	Element size	Energy use	Processing speed	Style of computation	Fault Tolerant	Learns	Intelligent, conscious
	10^{11} synapses	10^{-6} m	30 W	100 Hz	parallel, distributed	yes	yes	usually
	10^{8} transistors	10^{-6} m	30 W (CPU)	10^{9} Hz	serial, centralized	no	a little	not (yet)

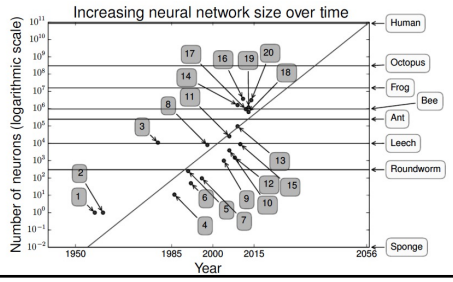
5

Biological Neuron



6

Neural Network Sizes



Réda DEHAK 7

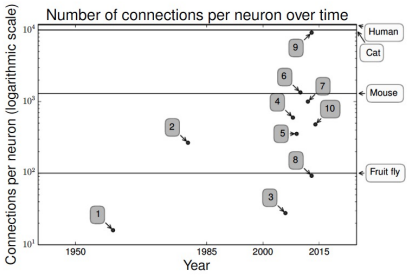
7

1. Perceptron (Rosenblatt, 1958, 1962)
2. Adaptive Linear Element (Widrow and Hoff, 1960)
3. Neocognitron (Fukushima, 1980)
4. Early backpropagation network (Rumelhart *et al.*, 1986b)
5. Recurrent neural network for speech recognition (Robinson and Fallside, 1991)
6. Multilayer perceptron for speech recognition (Bengio *et al.*, 1991)
7. Mean field sigmoid belief network (Saul *et al.*, 1996)
8. LeNet-5 (LeCun *et al.*, 1998b)
9. Echo state network (Jaeger and Haas, 2004)
10. Deep belief network (Hinton *et al.*, 2006)
11. GPU-accelerated convolutional network (Chellapilla *et al.*, 2006)
12. Deep Boltzmann machines (Salakhutdinov and Hinton, 2009a)
13. GPU-accelerated deep belief network (Raina *et al.*, 2009)
14. Unsupervised convolutional network (Jarrett *et al.*, 2009b)
15. GPU-accelerated multilayer perceptron (Ciresan *et al.*, 2010)
16. OMP-1 network (Coates and Ng, 2011)
17. Distributed autoencoder (Le *et al.*, 2012)
18. Multi-GPU convolutional network (Krizhevsky *et al.*, 2012a)
19. COTS HPC unsupervised convolutional network (Coates *et al.*, 2013)
20. GoogLeNet (Szegedy *et al.*, 2014a)

Réda DEHAK 8

8

Number of connections



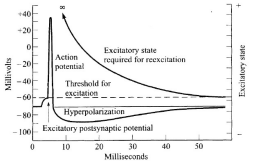
Réda DEHAK 9

9

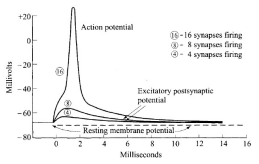
1. Adaptive Linear Element (Widrow and Hoff, 1960)
2. Neocognitron (Fukushima, 1980)
3. GPU-accelerated convolutional network (Chellapilla *et al.*, 2006)
4. Deep Boltzmann machines (Salakhutdinov and Hinton, 2009a)
5. Unsupervised convolutional network (Jarrett *et al.*, 2009b)
6. GPU-accelerated multilayer perceptron (Ciresan *et al.*, 2010)
7. Distributed autoencoder (Le *et al.*, 2012)
8. Multi-GPU convolutional network (Krizhevsky *et al.*, 2012a)
9. COTS HPC unsupervised convolutional network (Coates *et al.*, 2013)
10. GoogLeNet (Szegedy *et al.*, 2014a)

10

Biological Neuron Activation



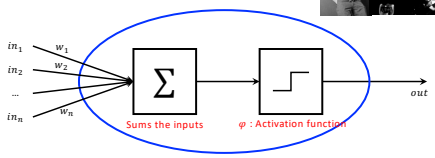
From: Principles of Neurocomputing for Science & Engineering, Ham & Kostanic, McGraw-Hill



From: Principles of Neurocomputing for Science & Engineering, Ham & Kostanic, McGraw-Hill

11

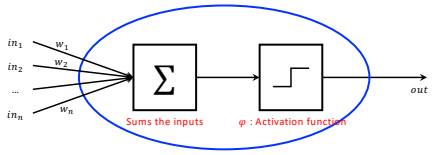
McCulloch-Pitts Neuron



- A set of synapses (i.e. connections) brings in activations from other neurons.
- A processing unit sums the inputs, and then applies a non-linear activation function (i.e. squashing/transfer/threshold function).
- An output line transmits the result to other neurons.
- Any number of McCulloch-Pitts neurons can be connected together in any way we like

12

McCulloch-Pitts Neuron

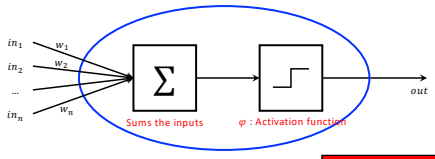


$$out = \begin{cases} 1 & if \sum_{i=1}^{n} w_i in_i \geq \theta \\ 0 & else \end{cases}$$

Réda DEHAK 13

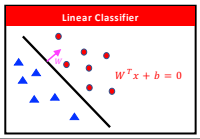
13

McCulloch-Pitts Neuron



$$out = \varphi\left(\sum_{i=1}^{n} w_i in_i - \theta\right)$$

$$out = sign\left(\sum_{i=1}^{n} w_i in_i - \theta\right)$$



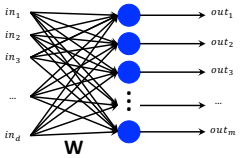
Réda DEHAK

14

Perceptron

(Rosenblatt, 1958)

- An arrangement of one layer of McCulloch-Pitts neurons is known as a Perceptron (Dense Layer).



$$out_j = sign\left(\sum_{i=1}^{d} in_i w_{ij} - \theta_j\right)$$

$$\text{Out} = sign(W^T In)$$

$$In = \begin{bmatrix} 1 \\ in_1 \\ in_2 \\ \vdots \\ in_d \end{bmatrix}$$

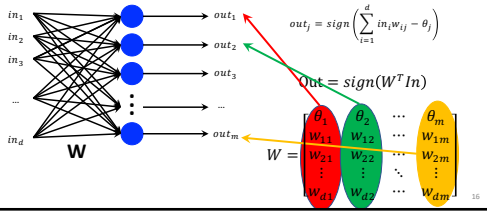
Réda DEHAK 15

15

Perceptron

(Rosenblatt, 1958)

- An arrangement of one layer of McCulloch-Pitts neurons is known as a Perceptron (Dense Layer).



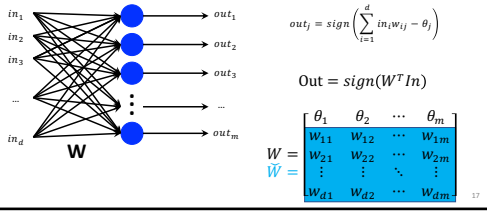
Réda DEHAK 16

16

Perceptron

(Rosenblatt, 1958)

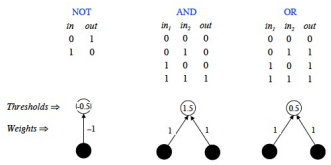
- An arrangement of one layer of McCulloch-Pitts neurons is known as a Perceptron (Dense Layer).



Réda DEHAK 17

17

Implementation of Logical operator

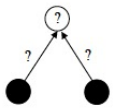


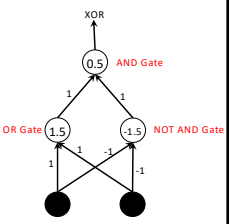
Réda DEHAK 18

18

XOR

XOR		
in_1	in_2	out
0	0	0
0	1	1
1	0	1
1	1	0





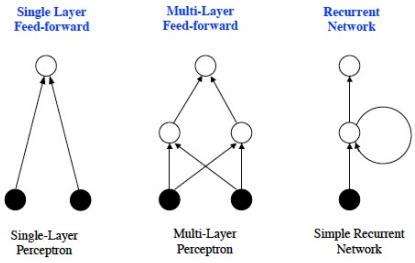
19

Architectures, Structures or Topology

- Single-Layer Feed-forward NNs: One input layer and one output layer of processing units. No feed-back connections. (For example, a simple Perceptron.)
- Multi-Layer Feed-forward NNs: One input layer, one output layer, and one or more hidden layers of processing units. No feed-back connections. The hidden layers sit in between the input and output layers, and are thus hidden from the outside world. (For example, a Multi-Layer Perceptron.)
- Recurrent NNs: Any network with at least one feed-back connection. It may, or may not, have hidden units. (For example, a Simple Recurrent Network.)

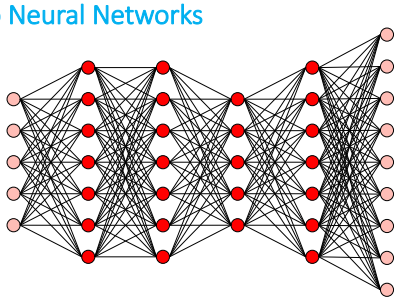
20

Examples



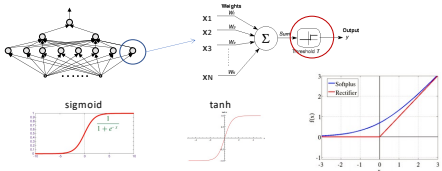
21

Deep Neural Networks



22

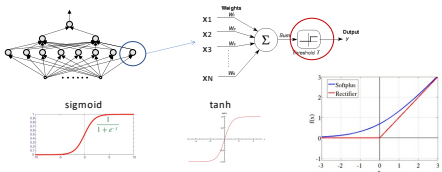
A note on activations



- Previous explanations assumed network units used a threshold function for "activation"
- In reality, we use a number of other *differentiable* functions
  - Mostly, but not always, "squashing" functions

23

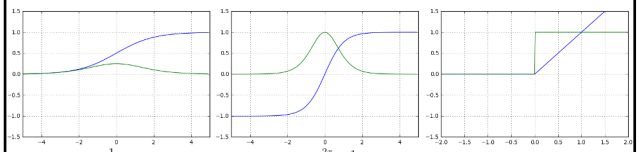
A note on activations



- Composing/learning networks using threshold "activations" is a combinatorial problem
  - Exponentially hard to find the right solution
- The smoother *differentiable* activation functions enable learning through optimization

24

Activation functions



$$\text{sigm}(x) = \frac{1}{1+e^{-x}} \qquad \tanh(x) = \frac{e^{2x}-1}{e^{2x}+1} \qquad \text{relu}(x) = \max(0,x)$$

$$\text{sigm}'(x) = \text{sigm}(x)(1-\text{sigm}(x)) \qquad \tanh'(x) = 1-\tanh(x)^2 \qquad \text{relu}'(x) = 1_{x>0}$$

Réda DEHAK 25

25

Softmax function

$$softmax(\mathbf{x}) = \frac{1}{\sum_{i=1}^{n} e^{x_i}} \cdot \begin{bmatrix} e^{x_1} \\ e^{x_2} \\ \vdots \\ e^{x_n} \end{bmatrix}$$

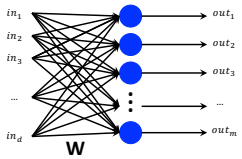
$$\frac{\partial softmax(\mathbf{x})_i}{\partial x_j} = \begin{cases} softmax(\mathbf{x})_i \cdot (1 - softmax(\mathbf{x})_i) & i = j \\ -softmax(\mathbf{x})_i \cdot softmax(\mathbf{x})_j & i \neq j \end{cases}$$

- Vector of values in $[0,1]$ that add up to 1
- $\mathrm{P}(Y = k|X = x) = \mathrm{softmax}(\mathrm{z}(x))_k$
- The pre-activation vector $\mathrm{z}(x)$ is often called "the logits"

Réda DEHAK 26

26

Learning: 1-Perceptron



$$\mathrm{Out}(in) = \varphi(W^T In)$$

Learning: Gradient Descent

Given a training set: $\{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$
$x_1, x_2, \dots, x_n$ are the input vectors
$y_1, y_2, \dots, y_n$ are the desired output (target)
Minimize MSE :

$$E = \frac{1}{N}\sum_{n=1}^{N} \|e_n\|^2$$

$$E = \frac{1}{N}\sum_{n=1}^{N} \|Out(x_n) - y_n\|^2$$

Réda DEHAK 27

27

Derivatives: Chain Rule

1. $z = u(v(x))$ with $v(x)$ is a vector

$$\frac{\partial z}{\partial x_{ij}} = \sum_k \frac{\partial z}{\partial v_k}\frac{\partial v_k}{\partial x_{ij}} = \left(\frac{\partial v}{\partial x_{ij}}\right)^T @ \frac{\partial z}{\partial v}$$

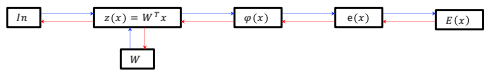
2. $z = u(v(x))$ with $v(x)$ is a matrix

$$\frac{\partial z}{\partial x_{ij}} = \sum_k \sum_l \frac{\partial z}{\partial v_{kl}}\frac{\partial v_{kl}}{\partial x_{ij}}$$

Réda DEHAK 28

28

Learning: 1-Perceptron



- $\frac{\partial E(x)}{\partial e(x)_i} = \frac{\partial (e^T e)}{\partial e(x)_i} = 2\, e(x)_i$
- $\frac{\partial e(x)_j}{\partial \varphi(x)_i} = \frac{\partial (\varphi(x)_j - y_j)}{\partial \varphi(x)_i} = \mathbb{I}_{i=j}$
- $\frac{\partial E(x)}{\partial \varphi(x)_i} = 2(\vec{\mathbb{I}}_i)^T @\, e(x) = 2e(x)_i$

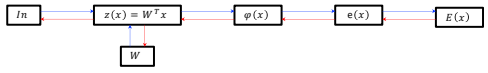
$$\frac{\partial E(x)}{\partial e(x)} = 2e(x)$$
$$\frac{\partial e(x)}{\partial \varphi(x)_i} = \vec{\mathbb{I}}_i$$
$$\frac{\partial E(x)}{\partial \varphi(x)} = 2e(x)$$

$\vec{\mathbb{I}}_i =$ (0, …, 0, 1, 0, …, 0) — i^{th} row

Réda DEHAK 29

29

Learning: 1-Perceptron



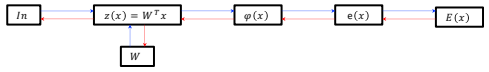
- $\frac{\partial \varphi(z(x))_j}{\partial z(x)_i} = \frac{\partial (\varphi(z(x))_j)}{\partial z(x)_i} = \varphi'(z(x)_j)\mathbb{I}_{i=j} = \varphi'(z(x))_j \,.\, \mathbb{I}_{i=j}$

$$\frac{\partial \varphi(x)}{\partial z(x)_i} = \varphi'(z(x)) \,.\, \vec{\mathbb{I}}_i$$
$$\frac{\partial E(x)}{\partial z(x)_i} = 2(\varphi'(z(x)) \,.\, \vec{\mathbb{I}}_i)^T @ e(x) = 2e(x)_i \varphi'(z(x))_i$$
$$\frac{\partial E(x)}{\partial z(x)} = 2e(x) \,.\, \varphi'(z(x))$$

Réda DEHAK 30

30

Learning: 1-Perceptron



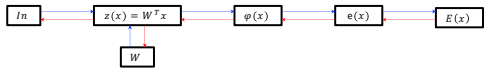
$$\frac{\partial z(x)_k}{\partial W_{ij}} = \frac{\partial (W^T In)_k}{\partial W_{ij}} = \frac{\partial \sum_{d=1}^{D} W_{dk} in_d}{\partial W_{ij}} = in_i \mathbb{I}_{k=j} \qquad \frac{\partial z(x)}{\partial W_{ij}} = in_i \vec{\mathbb{I}}_j$$

$$\frac{\partial E(x)}{\partial W_{ij}} = 2\left(in_i \vec{\mathbb{I}}_j\right)^T @ \left(e(x) . \varphi'(z(x))\right)$$

$$\frac{\partial E(x)}{\partial W_{ij}} = 2in_i \left(e(x) . \varphi'(z(x))\right)_j$$

31

Learning: 1-Perceptron



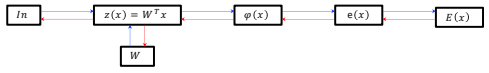
$$\frac{\partial z(x)_k}{\partial W_{ij}} = \frac{\partial (W^T In)_k}{\partial W_{ij}} = \frac{\partial \sum_{d=1}^{D} W_{dk} in_d}{\partial W_{ij}} = in_i \mathbb{I}_{k=j} \qquad \frac{\partial z(x)}{\partial W_{ij}} = in_i \vec{\mathbb{I}}_j$$

$$\frac{\partial E(x)}{\partial W_{ij}} = 2\left(in_i \vec{\mathbb{I}}_j\right)^T @ \left(e(x) . \varphi'(z(x))\right)$$

$$\frac{\partial E(x)}{\partial W} = 2In @ \left(e(x) . \varphi'(z(x))\right)^T$$

32

Learning: 1-Perceptron



$$\frac{\partial z(x)_k}{\partial W_{ij}} = \frac{\partial (W^T In)_k}{\partial W_{ij}} = \frac{\partial \sum_{d=1}^{D} W_{dk} in_d}{\partial W_{ij}} = in_i \mathbb{I}_{k=j} \qquad \frac{\partial z(x)}{\partial W_{ij}} = in_i \vec{\mathbb{I}}_j$$

$$\frac{\partial E(x)}{\partial W_{ij}} = 2\left(in_i \vec{\mathbb{I}}_j\right)^T @ \left(e(x) . \varphi'(z(x))\right)$$

$$\frac{\partial E(x)}{\partial W} = In @ \left(\frac{\partial E(x)}{\partial z(x)}\right)^T$$

33

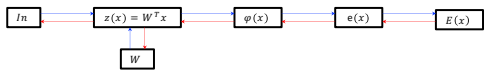
Learning Multi Mayers Neural Networks

Backpropagation Algorithm

34

Learning: 2-Multilayers Neural Networks

Backpropagation Algorithm



- $\frac{\partial z(x)_k}{\partial In_i} = \frac{\partial (W^T In)_k}{\partial In_i} = \frac{\partial \sum_{d=1}^{D} W_{dk} in_d}{\partial In_i} = \tilde{W}_{ik}$ $\qquad \frac{\partial z(x)}{\partial In_i} = (\tilde{W}_{i\bullet})^T$

$$\frac{\partial E(x)}{\partial In_i} = 2\tilde{W}_{i\bullet} @ \left(e(x) \,.\, \varphi'(z(x))\right)$$

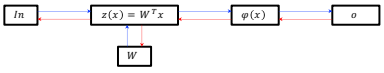
$$\frac{\partial E(x)}{\partial In} = \tilde{W} @ \frac{\partial E(x)}{\partial z(x)}$$

Réda DEHAK 35

35

Learning: 2-MultiLayers Neural Networks

Backpropagation Algorithm



- $\frac{\partial o_j}{\partial z(x)_i} = \frac{\partial \left(\varphi(z(x))\right)_j}{\partial z(x)_i} = \varphi'(z(x)_j)\mathbb{I}_{i=j} = \varphi'(z(x))_j \,.\, \mathbb{I}_{i=j}$

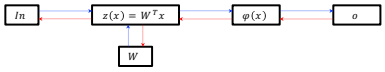
$$\frac{\partial o}{\partial z(x)_i} = \varphi'(z(x)) \,.\, \vec{\mathbb{I}}_i$$

$$\frac{\partial E(x)}{\partial z(x)_i} = (\varphi'(z(x)) \,.\, \vec{\mathbb{I}}_i)^T @ \frac{\partial E(x)}{\partial o} = \left(\frac{\partial E(x)}{\partial o}\right)_i \varphi'(z(x))_i$$

Réda DEHAK 36

36

Learning: 2-MultiLayers Neural Networks
Backpropagation Algorithm



- $\frac{\partial o_j}{\partial z(x)_i} = \frac{\partial \left(\varphi(z(x))\right)_j}{\partial z(x)_i} = \varphi'(z(x)_j)\mathbb{I}_{i=j} = \varphi'(z(x))_j . \mathbb{I}_{i=j}$

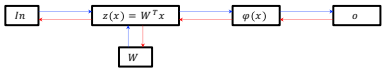
$$\frac{\partial E(x)}{\partial z(x)} = \left(\frac{\partial E(x)}{\partial o} . \varphi'(z(x))\right)$$

$$\frac{\partial E(x)}{\partial z(x)} = \left(\frac{\partial E(x)}{\partial In_+} . \varphi'(z(x))\right)$$

Réda DEHAK 37

37

Learning: 2-MultiLayers Neural Networks
Backpropagation Algorithm



$$\frac{\partial z(x)_k}{\partial w_{ij}} = \frac{\partial (W^T In)_k}{\partial w_{ij}} = \frac{\partial \sum_{d=1}^{D} W_{dk} in_d}{\partial w_{ij}} = in_i \mathbb{I}_{k=j} \qquad \frac{\partial z(x)}{\partial W_{ij}} = in_i \vec{\mathbb{I}}_j$$

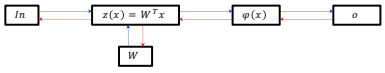
$$\frac{\partial E(x)}{\partial W_{ij}} = 2\left(in_i \vec{\mathbb{I}}_j\right)^T @ \left(\frac{\partial E(x)}{\partial z(x)}\right)$$

$$\frac{\partial E(x)}{\partial W} = In @ \left(\frac{\partial E(x)}{\partial z(x)}\right)^T$$

Réda DEHAK 38

38

Learning: 2-MultiLayers Neural Networks
Backpropagation Algorithm



- $\frac{\partial z(x)_k}{\partial \widetilde{In}_i} = \frac{\partial (W^T In)_k}{\partial \widetilde{In}_i} = \frac{\partial \sum_{d=1}^{D} W_{dk} in_d}{\partial \widetilde{In}_i} = \widetilde{W}_{ik} \qquad \frac{\partial z(x)}{\partial \widetilde{In}_i} = (\widetilde{W}_{i\bullet})^T$

$$\frac{\partial E(x)}{\partial In_i} = 2\widetilde{W}_{i\bullet} @ \left(e(x) . \varphi'(z(x))\right)$$

$$\frac{\partial E(x)}{\partial \widetilde{In}} = \widetilde{W} @ \frac{\partial E(x)}{\partial z(x)}$$

Réda DEHAK 39

39

Learning: 2-MultiLayers Neural Networks
Backpropagation Algorithm

For a Multilayer NN with L Layers

- First Step: Forward Propagation of the input
  - Compute and store the activation and the output of each layer in order from the input layer to the output layer

$$In^{(l)} = \begin{bmatrix} 1 \\ o^{(l-1)} \end{bmatrix}$$

$$z^{(l)} = W^{(l)^T} in^{(l)}$$

$$o^{(l)} = \varphi\left(z^{(l)}\right)$$

Réda DEHAK 40

40

Learning: 2-MultiLayers Neural Networks
Backpropagation Algorithm

For a Multilayers NN with L Layers

- Second Step: Backward Propagation of the gradient
  - Compute and store the gradient of each layer in order from the ouput layer to the input layer

Compute : $\frac{\partial E(X)}{\partial o^{(L)}} = 2e$ with $e = o^{(L)} - y$ in the case of *MSE*

and $\delta^{(L)} = \frac{\partial E(x)}{\partial z(x)^{(L)}} = \frac{\partial E(X)}{\partial o^{(L)}} . \varphi'\left(z(x)^{(L)}\right)$

Iterate: $\frac{\partial E(x)}{\partial W^{(l)}} = In^{(l)} @ \left(\delta^{(l)}\right)^T$

$$\delta^{(l-1)} = \frac{\partial E(x)}{\partial z(x)^{(l-1)}} = \varphi'\left(z(x)^{(l-1)}\right) . \left(\tilde{W}^{(l)} @ \delta^{(l)}\right)$$

Réda DEHAK 41

41

Loss Function

- Regression:
  - Use MSE cost function with a Identity activation function for output layer.

$$E = \frac{1}{N}\sum_{i=1}^{N} \left\| o^{(L)}(x_i) - y_i \right\|^2 \qquad \frac{\partial E(x)}{\partial o^{(L)}} = 2\left(o^{(L)} - y\right)$$

- Classification:
  - Use Cross Entropy cost function with sigmoid (in the case of binary classifier) or softmax (in the case of multiclass classifier) activation function for output layer

$$E = -\frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K} \mathbb{I}_{y_i=k} \log\left(o^{(L)}(x_i)\right)_k \qquad \frac{\partial E(x)}{\partial o^{(L)}} = -\frac{\mathbb{I}_y}{o^{(L)}}$$

Réda DEHAK 42

42

Gradient descent variants

- Three variants of gradient descent:
  - Stochastic gradient descent: uses one random sample of the training dataset to compute the gradient at each iteration
  - Batch gradient descent: uses the full trainning dataset to compute the gradient at each iteration
  - Mini-batch gradient descent: use a small random portion of the training dataset to compute the gradient at each iteration

43

Heuristic for making the Back Propagation algorithm perform better

It is more of an art than a science

- Input data should be normalized to have approx. same range: standardization or quantile normalization
- Initializing W^h and W^o:
  - Zero is a saddle point: no gradient, no learning
  - Constant init: hidden units collapse by symmetry
  - Solution: random init, ex: $W \sim \mathcal{N}(0; 0.01)$
  - Better inits:
    - Xavier Glorot and Kaming He
    - orthogonal
- Biases can (should) be initialized to zero

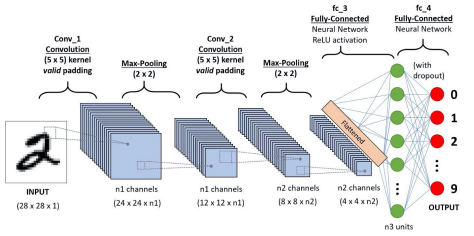
44

Conclusions:

- Neural Networks is a complex function.
- NN Learning is based on Gradient Descent and Backpropagation algorithm.
- New Regulation algorithms were proposed recently: dropout, …
- New topologies emerge recently.

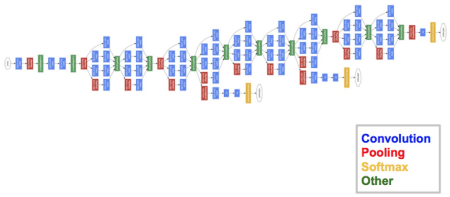
45

CNN: Convolutional Neural Networks



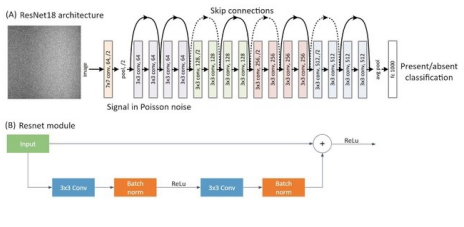
46

Googlenet/Inception(2014)



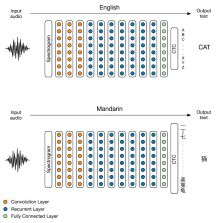
47

ResNet48



48

DL Applications: Speech to Text



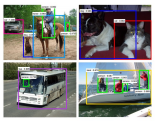
[Baidu 2014]

49

DL Applications: Vision

[Krizhevsky 2012]

[Ciresan et al. 2013]



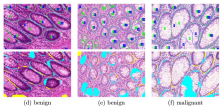
[Faster R-CNN - Ren 2015]

[NVIDIA dev blog]

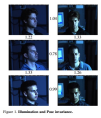
50

DL Applications: Vision

[Stanford 2017]



[Nvidia Dev Blog 2017]



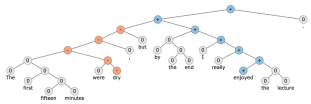
[FaceNet - Google 2015]

[Facial landmark detection CUHK 2014]

51

DL Applications: NLP

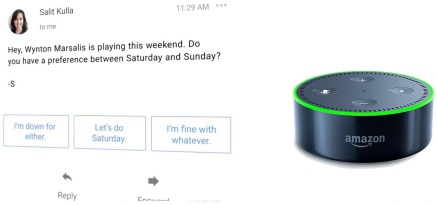
[Google Translate System - 2016]



[Socher 2015]

52

DL Applications: NLP

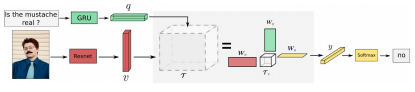


Most of chatbots claiming "AI" do not use Deep Learning (yet?)

[Google Inbox Smart Reply] [Amazon Echo / Alexa]

53

DL Applications: Vision and NLP



[VQA - Mutan 2017]

"man in black shirt is playing guitar."
"construction worker in orange safety vest is working on road."
"two young girls are playing with lego toy."
"boy is doing backflip on wakeboard."

[Karpathy 2015]

54

DL Applications: Image translation

[DeepDream 2015]

[Gatys 2015]

[Ledig 2016]

55

DL Applications: Generative Models

Sampled celebrities [Nvidia 2017]

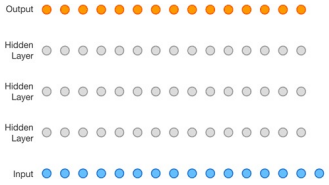
56

DL Applications: Generative Models

StackGAN v2 [Zhang 2017]

57

DL Applications: Generative Models



Sound generation with WaveNet
[DeepMind 2017]

58

DL Applications: Generative Models

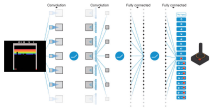
- Guess which one is generated?

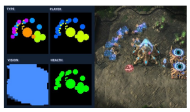
- *Tacotron 2 Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions, 2017*

59

DL Applications: Games

[Deepmind AlphaGo / Zero 2017]





[Atari Games - DeepMind 2016] [Starcraft 2 for AI research]

60