



Session 1

Machine learning and Neural network

General machine learning
Neurons
Evaluation and performance
Limitation of traditional NN

1

1



Course resources

- Stanford class cs231n by Fei-Fei Li, A. Karpathy and J. Johnson: <http://cs231n.stanford.edu/slides/2024>
- [Mitchell, Tom](#) (1997). *[Machine Learning](#)*. New York: McGraw Hill. [ISBN 0-07-042807-7](#). [OCLC 36417892](#).
- Drive into deep learning: <https://d2l.ai/index.html>

2

2

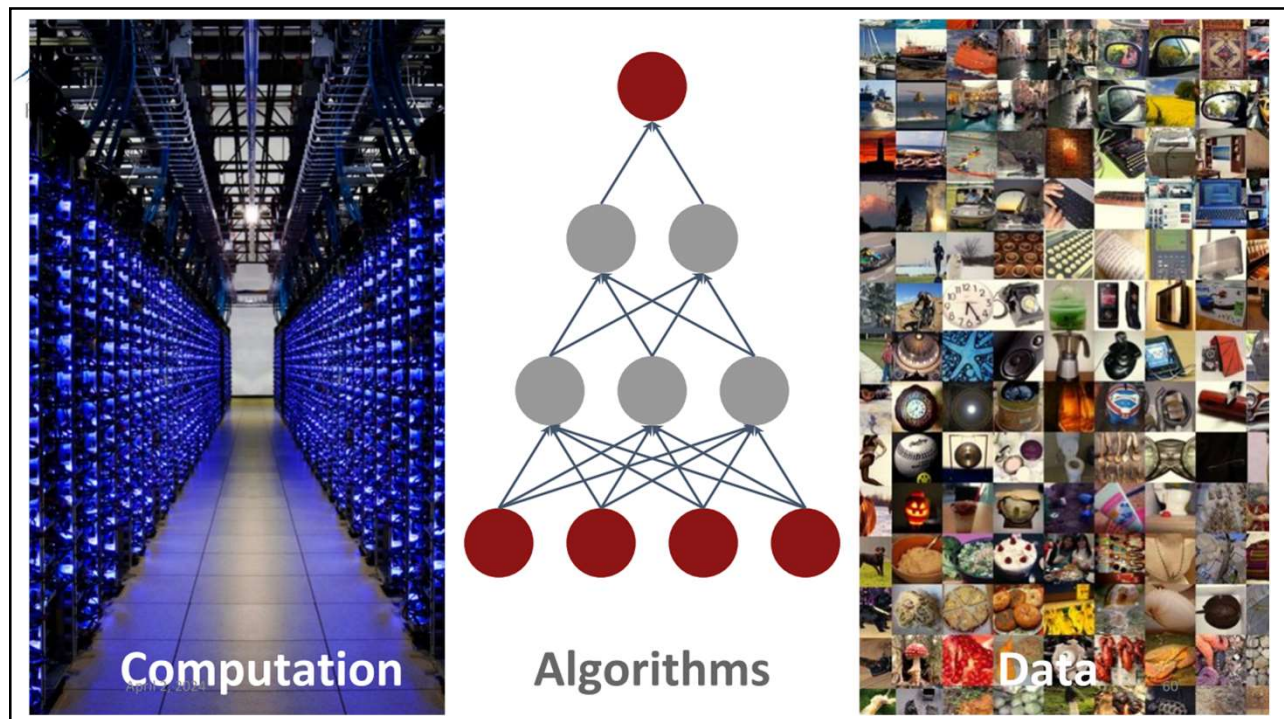


PSL Outline

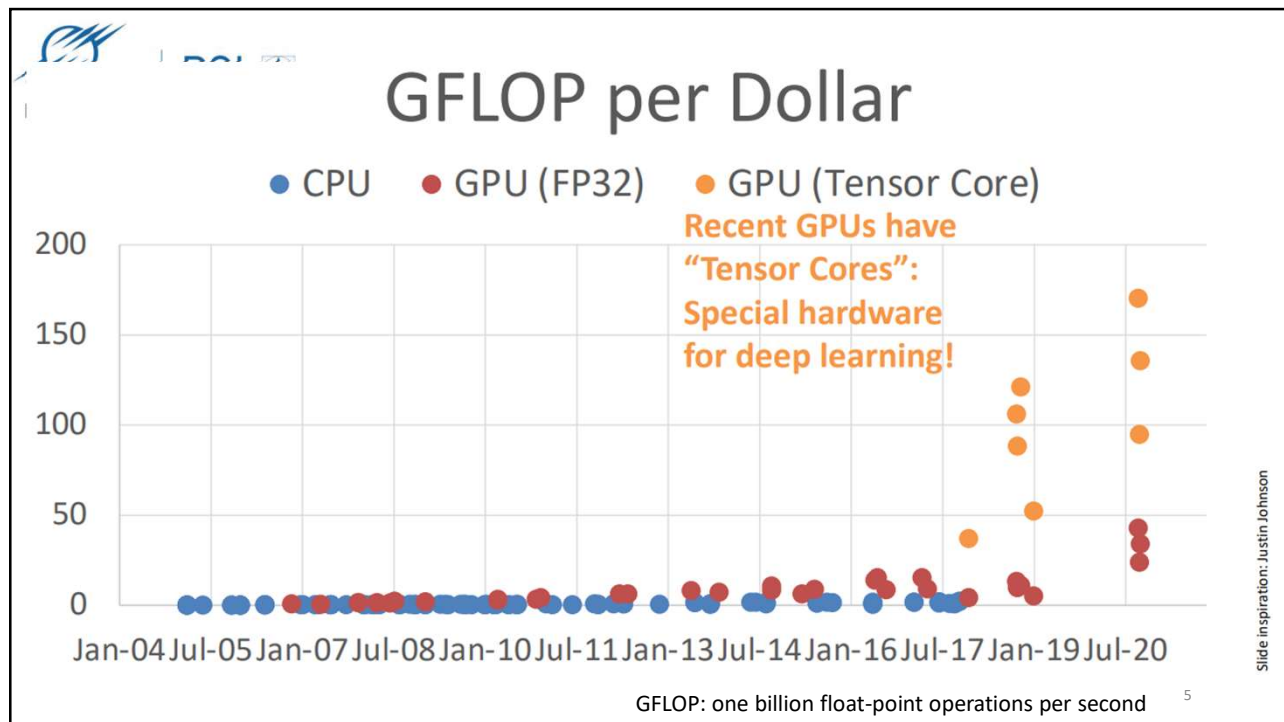
- General machine learning
- Perceptron
- Artificial neural network
- Evaluation and performance
- Limitation of traditional ANN

3

3



4



5

PSL

Overview of Artificial Intelligent

- **Machine learning (ML)** is the study of computer algorithms that can improve automatically through the experience and by the use of data [Mitchell, Tom, 1997]. Popular algorithms are:
 - Linear classifiers
 - K-nearest Neighbors
 - Boosted stumps
 - Support Vector machines (SVMs)
- **Artificial neural networks (ANNs)**, usually simply called **neural networks (NNs)**, are computing systems inspired by the biological neural networks that constitute animal brains.
 - Concerning on the design of structure, connection, flow of the signal

6



Machine learning basic

- Deep learning is a specific kind of machine learning.
- Learning algorithms: A computer program is said to learn from experience **E** with respect to some class of tasks **T** and performance measure **P**, if its performance at tasks in T, as measured by P, improves with Experience E (Mitchell, 1997)
- Two types of learning functions
 - Parametric models: fixed number of parameters
 - Linear regression, naïve Bayes, logistic regression, neural network
 - Nonparametric models: flexible parameters needed. They may lead to a better model shows as no assumptions are being made about the primary function.
 - Decision tree, SVM, kNN

7

7



The tasks, T

- Classification
- Regression
- Transcription: an image of text \Leftrightarrow return the text, an audio waveform \Leftrightarrow a sequence of characters
- Translation
- Structured output: pixel-wise segmentation of images
- Anomaly detection: credit card fraud detection
- Synthesis and sampling: automatically generate landscapes of a video games
- Density estimation

8

8



The experience, E

- ML algorithms can be broadly categorized as unsupervised or supervised by what kind of experience
- Unsupervised: learn useful properties of the structure of the given dataset. Representation learning → probability distribution $p(x)$
- Supervised learning: given a dataset containing features and each example (x) is associated with a label or target (y). → conditional probability distribution $p(y|x)$
- Self-supervised learning (SSL): a model is trained on a task using the data itself to generate supervisory signals, rather than relying on externally-provided labels
 - Autoassociative SSL: autoencoder
 - Contrastive SSL: learn to minimize the distance between positive example and maximize the distance between positive and negative example
- Semi-supervised learning: use only a small portion of the labeled data
- Reinforcement learning: interact with environment and gets feedback to determine the next action

9

9



The performance, P

- A loss function tells how good our current classifier is
- Given a dataset of examples $\{(x_i, y_i)\}_{i=1}^N$ where x_i is image and y_i is label, N is the total number of examples. Our goal is to find w that minimizes the cost function L (an average of loss L_i over examples)

$$L = \frac{1}{N} \sum_i L_i(f(x_i, w), y_i)$$

- Assume f is a linear function, how to design loss function so the machine can predict the category of the input image?

Softmax(cross-entropy): probability of one object over all the others.

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

10

10



Loss function

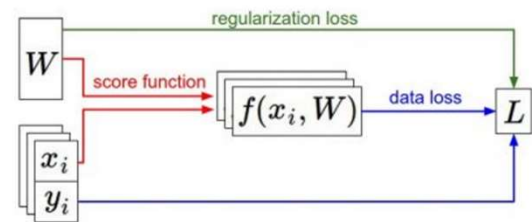
- We have some dataset of (x, y)
- We have a score function:
- We have a loss function:

$$s = f(x; W) = Wx$$

$$L_i = -\log\left(\frac{e^{sy_i}}{\sum_j e^{sj}}\right) \text{ Softmax}$$

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1) \text{ SVM}$$

$$L = \frac{1}{N} \sum_{i=1}^N L_i + R(W) \text{ Full loss}$$



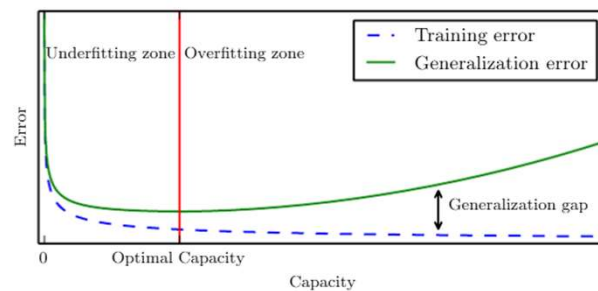
11

11



Others

- Generalization: the ability on perform well on previously unobserved inputs
- Underfitting: can not obtain a sufficiently low error on the training set
- Overfitting: the gap between the training error and test error is too large
- Capacity: ability to fit a wide variety of functions. For example, add polynomials to the hypothesis space for linear regression instead of only linear functions.
 - Change the number of input features: $f(x) = b + \sum_{i=1}^n w_i x^i$
 - Representational capacity: family of functions when varying the parameters



12

12



Regularization

- The goal of the machine learning is not to seek a universal learning algorithm or the absolute best learning algorithm → looking for data-generating distributions
- Regularization is any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error.

$$L(W) = \frac{1}{N} \sum_i L_i(f(x_i, W), y_i) + \lambda R(W)$$

- λ is a hyperparameter that controls the level of regularization
- R functions can be the following:
 - L2 regularization: $R(W) = \sum_k \sum_l W_{k,l}^2$
 - L1 regularization: $R(W) = \sum_k \sum_l |W_{k,l}|$
 - Elastic net (L1+L2): $R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$

More complex:

Dropout
Batch normalization
Stochastic depth, fractional pooling,
etc

13

13



Practice

- Given $x=[1,1,1,1]$, $w1=[1,0,0,0]$, and $w2=[0.25,0.25,0.25,0.25]$, which of $w1$ or $w2$ will the L2 regularization prefer?

14

14

Neural Network

- With the understanding of how our human brains works (although not completely), the most success “imitate” human brain modules are:



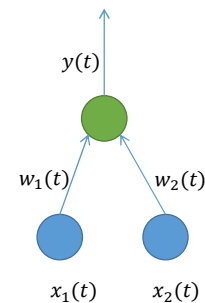
15

Introduction to perceptron

- The first artificial neural network was invented in 1943 by Warren McCulloch and Walter Pitts
- The perceptron is a binary and linear classifier for supervised learning and was simulated on an IBM 704 (Rosenblatt, 1957).
- Far from being deep, it has a single layer and has limited modeling capacity.
- They were initially designed to mathematically model the processing of information by biological neural networks found in the cortex of mammals.

$$z = b + \sum_i x_i w_i$$

$$y = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{otherwise} \end{cases}$$



16

16



Perceptron learning

- Initialize bias b and weights w
- Pick a training case from $D = \{(x_1, d_1), \dots, (x_s, d_s)\}$ where x_i is the n -dimensional input vector and d_i is the desired output value.
- Use any policy to ensure that every training case is picked up.
 - If the output unit is correct, leave its weights alone.
 - If the output unit incorrectly outputs a zero, add the input vector to the weight vector. $w = w + x * (d - y)$
 - If the output unit incorrectly outputs a 1, subtract the input vector from the weight vector. $w = w + x * (d - y)$
- This is guaranteed to find a set of weights that gets the right answer for all the training cases if any such set exists.

17



A geometrical view of perceptron

- Weight space has one dimension per weight
- A point in the space represents a particular setting of all the weights.
- Assuming that we have eliminated the threshold, each training case can be represented as a hyperplane through the origin.
- The weights must lie on one side of this hyper-plane to get the answer correct
- The plane goes through the origin and is perpendicular to the input vector.
- On one side of the plane the output is wrong because the scalar product of the weight vector with the input vector has the wrong sign.

18

MINES ParisTech | PSL Weight space

$$z = b + \sum_i x_i w_i$$

$$y = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

What does the geometrical relationship of the vector v_1 and vector v_2 when:

- (1) $v_1 \cdot v_2 > 0$
- (2) $v_1 \cdot v_2 = 0$
- (3) $v_1 \cdot v_2 < 0$

19

MINES ParisTech | PSL A geometric view of what binary threshold neurons cannot do

- Imagine “data-space” in which the axes correspond to components of an input vector.
 - Each input vector is a point in this space.
 - A weight vector defines a plane in data-space.
 - The weight plane is perpendicular to the weight vector and misses the origin by a distance equal to the threshold.

The positive and negative cases cannot be separated by a plane

20



Modern perceptron

- In the modern sense, the perceptron is a threshold function

$$f(x) = h(w \cdot x + b)$$

Where h is the Heaviside step-function, w is a vector of real-valued weights.

- Equivalently, we can augment input $x = [x, 1]$ and weight $w = [w, b]$ to form an easier math:

$$f(x) = h(w \cdot x)$$

- In the context of neural network, a perceptron is an artificial neuron using the Heaviside step function as the activation function. It is the simplest feedforward neural network.

21

21



Learning with hidden units

- Networks without hidden units are very limited in the input-output mappings they can learn to model.
 - More layers of linear units do not help. Its still linear.
 - Fixed output non-linearities are not enough.
- We need multiple layers of adaptive, non-linear hidden units. But how can we train such nets?
 - We need an efficient way of adapting all the weights, not just the last layer. This is hard.
 - Learning the weights going into hidden units is equivalent to learning features.
 - This is difficult because nobody is telling us directly what the hidden units should do.

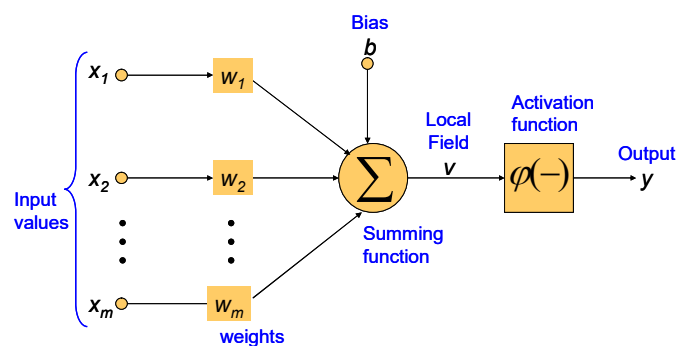
22

- A NN is specified by:
 - An architecture: a set of neurons and links connecting neurons. Each link has a weight,
 - A neuron model: the information processing unit of the NN,
 - A learning algorithm: used for training the NN by modifying the weights in order to solve the particular learning task correctly on the training examples.
- Topology
 - The topology of an ANN consist of the following information
 - The number of neurons (input, hidden, and output)
 - The number of layers (input, hidden, and output)
 - The connection types

23


23

- From a mathematical perspective, the processing of information within a neuron involves the following two distinct mathematical operations
 - Somatic operation
 - Synaptic operation



24

24




Identity	Sigmoid	TanH	ArcTan
ReLU	Leaky ReLU	Randomized ReLU	Parametric ReLU
Binary	Exponential Linear Unit	Soft Sign	Inverse Square Root Unit (ISRU)
Inverse Square Root Linear	Square Non-Linearity	Bipolar ReLU	Soft Plus

Types of neurons

- Common type of neurons (activation functions)
 - Linear neurons
 - Binary threshold neurons
 - Rectified Linear neuron
 - Sigmoid neurons

25



Activation

Modern approaches (ReLU)

- Rectified Linear Unit (ReLU) is proposed by Hahnloser et al. in 2000

$$ReLU(x) = \max(0, x)$$

$$ReLU'(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases}$$

- Pros:**
 - Less time and space complexity, because of sparsity, and compared to the sigmoid, it does not evolve the exponential operation, which are more costly.
 - Avoids the vanishing gradient problem.
- Cons:**
 - Introduces the *dead relu* problem, where components of the network are most likely never updated to a new value. This can sometimes also be a pro.
 - ReLU does not avoid the exploding gradient problem

26

26



Activation

Modern approaches (ELU)

- Exponential Linear Unit (ELU) is proposed by D.A. Clevert et al. 2016

$$ELU(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha(e^x - 1) & \text{if } x \leq 0 \end{cases}$$

$$ELU'(x) = \begin{cases} 1 & \text{if } x > 0 \\ \alpha e^x & \text{if } x \leq 0 \end{cases}$$

α is around 0.1 to 0.3

- Pros
 - Avoids the *dead relu* problem.
 - Produces negative outputs, which helps the network nudge weights and biases in the right directions.
 - Produce activations instead of letting them be zero, when calculating the gradient.
- Cons
 - Introduces longer computation time, because of the exponential operation included
 - Does not avoid the exploding gradient problem
 - The neural network does not learn the alpha value

27

27



Activation

Modern approaches (Leaky ReLU)

- Leaky Rectified Linear Unit is proposed by AL Maas et al. 2013

$$LReLU(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{if } x \leq 0 \end{cases}$$

$$LReLU'(x) = \begin{cases} 1 & \text{if } x > 0 \\ \alpha & \text{if } x \leq 0 \end{cases}$$

α is around 0.1 to 0.3

- Pros
 - Like the ELU, we avoid the *dead relu* problem, since we allow a small gradient, when computing the derivative.
 - Faster to compute than ELU, because no exponential operation is included
- Cons
 - Does not avoid the exploding gradient problem
 - The neural network does not learn the alpha value
 - Becomes a linear function, when it is differentiated, whereas ELU is partly linear and nonlinear.

28

28



Activation

Modern approaches (GELU)

- Gaussian Error Linear Unit. An activation function used in the most recent Transformers – Google's BERT and OpenAI's GPT-2. It is proposed by Hendrycks, Dan; Gimpel, Kevin (2016)

$$GELU(x) = 0.5x \left(1 + \tanh \left(\sqrt{\frac{2}{\pi}} (x + 0.044715x^3) \right) \right)$$

$$\begin{aligned} GELU'(x) &= 0.5 \tanh(0.0356774x^3 + 0.797885x) \\ &\quad + (0.535161x^3 + 0.398942x) \operatorname{sech}^2(0.0356774x^3 + 0.797885x) + 0.5 \end{aligned}$$

- Pros
 - Seems to be state-of-the-art in NLP, specifically Transformer models – i.e. it performs best
 - Avoids vanishing gradients problem
- Cons
 - Fairly new in practical use, although introduced in 2016.

Good resource to understand how you choose/test activation functions:
<https://mlfromscratch.com/activation-functions-explained/#relu>

29

29





Architectures

- Prior to looking into the general ANNs architectures, two fundamental elements of constructing any ANNs architecture are given first:
 - Layers
 - Connections
- Here we introduce two popular ones first:
 - Feed-forward neural network
 - Recurrent neural network
 - Attention (Transformer)

30

30

 |  Why we care about the architecture?

Deep learning descends from **connectionism**:

Wiring of computational networks plays key role in building intelligent machines

Structures that define the wiring:

- Architecture - connections fixed in training (e.g. operation types) ← Focus of architecture design
- Parameters - connections updated in training (e.g. kernels learned via SGD/backprop)

Background

We inhabit a **resource-limited environment**. We have limited supplies of:



Energy Computation Memory Time

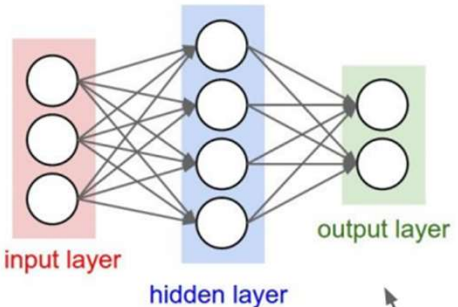
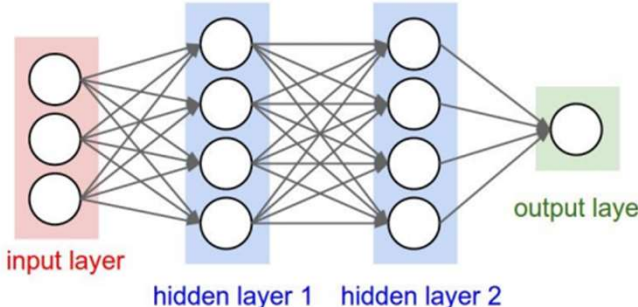
Typically, we want architectures with:

Greatest task performance (e.g. accuracy) Acceptable resource burden ← Changes over time!

Goals

31

 |  Networks architecture

“2-layer Neural Net”, or
“1-hidden-layer Neural Net” “Fully-connected” layers

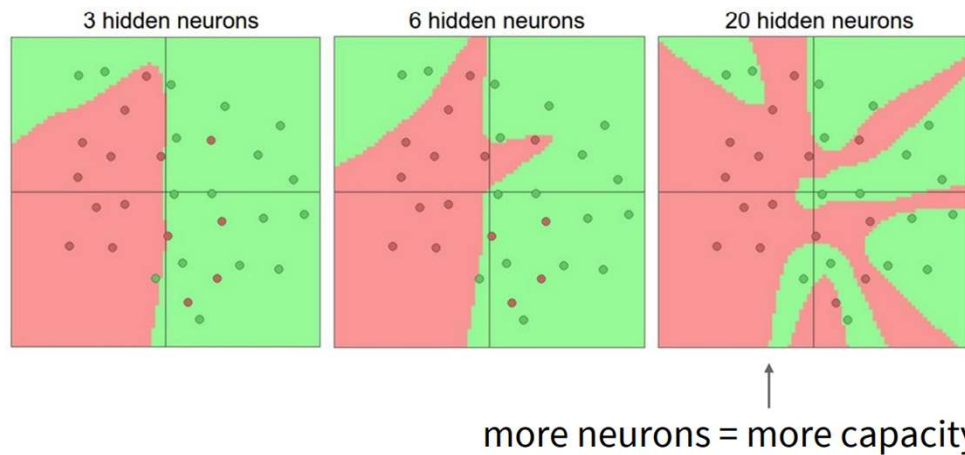
“3-layer Neural Net”, or
“2-hidden-layer Neural Net”

*Also called Multi-layer perceptron (MLP)

32

32

Setting the number of layers and their sizes



33

33

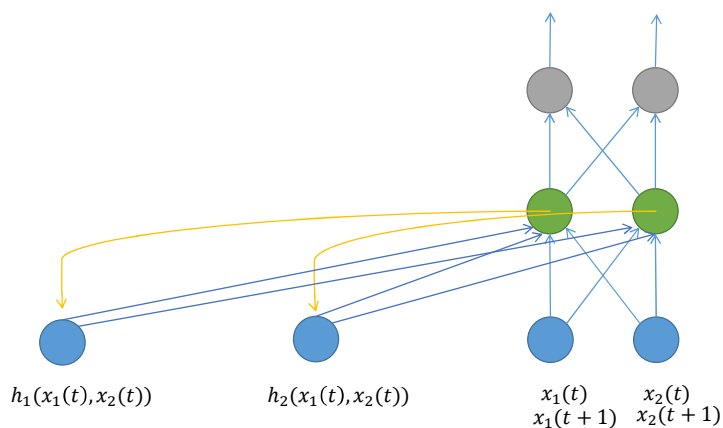
- These have directed cycles in their connection graph.
 - That means you can sometimes get back to where you started by following the arrows.
- They can have complicated dynamics and this can make them very difficult to train.
 - There is a lot of interest at present in finding efficient ways of training recurrent nets.
- They are more biologically realistic.

34

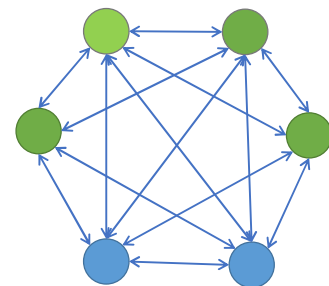
- Recurrent neural networks are a very natural way to model sequential data:
 - They are equivalent to very deep nets with one hidden layer per time slice.
 - Except that they use the same weights at every time slice and they get input at every time slice.
- They can remember information in their hidden state for a long time.
 - But its very hard to train them to use this potential.

35

- Recurrent net architectures



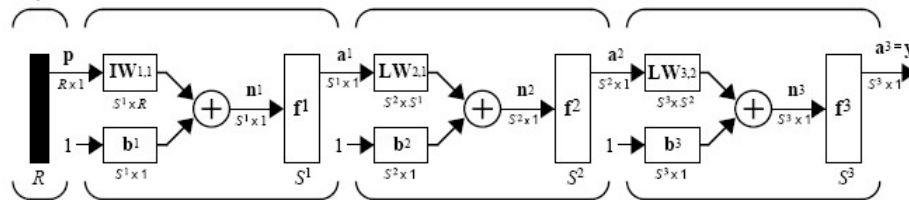
Hopfield network



36

36

- Given all the parameters in the following figure, how do you calculate the output of this feed-forward neural network?



- How many layers you have:

37

37

- Encoder: learns useful representation of input
- Decoder: decodes encoded representation and combines with other input to predict output
- Three popular variants:
 - Encoder-Only: useful for learning representations → BERT
 - Decoder-Only: useful for generation tasks → GPT-3
 - Encoder-Decoder: useful for sequence-to-sequence task → language translation

More discussion in RNN session

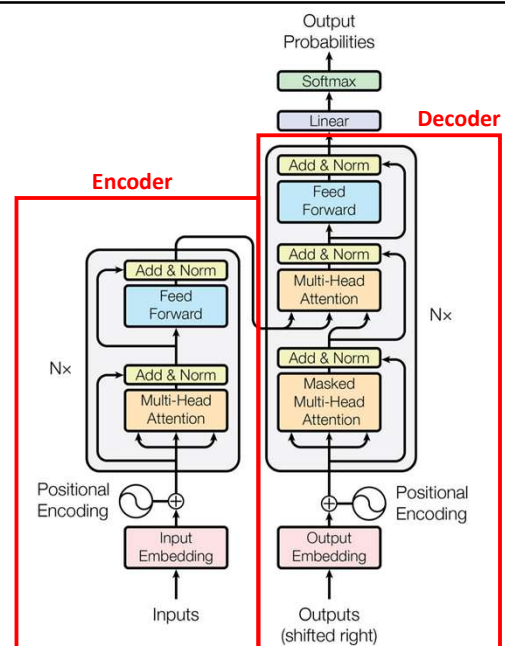


Figure 1: The Transformer - model architecture.

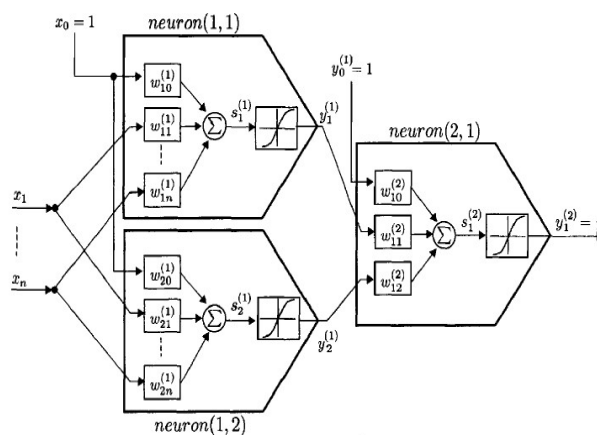
38

Methodology for supervised training

39

39

Multiple neurons



- The derivatives of the logit, z , with respect to the inputs and the weights are very simple:

$$z = b + \sum_i x_i w_i$$

$$\frac{\partial z}{\partial w_i} = x_i$$

$$\frac{\partial z}{\partial x_i} = w_i$$

- The derivative of the output with respect to the logit is simple if you express it in terms of the output:

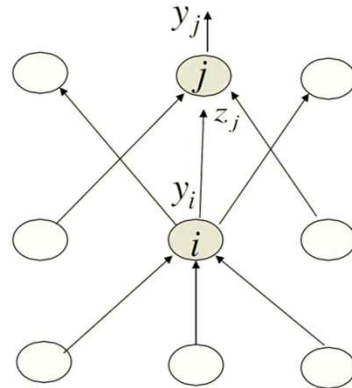
$$y = \frac{1}{1 + e^{-z}}$$

$$\frac{dy}{dz} = y(1 - y)$$

40

40

MINES ParisTech | PSL Backpropagating



$$\frac{\partial E}{\partial z_j} = \frac{dy_j}{dz_j} \frac{\partial E}{\partial y_j} = y_j (1 - y_j) \frac{\partial E}{\partial y_j}$$

$$\frac{\partial E}{\partial y_i} = \sum_j \frac{dz_j}{dy_i} \frac{\partial E}{\partial z_j} = \sum_j w_{ij} \frac{\partial E}{\partial z_j}$$

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial z_j}{\partial w_{ij}} \frac{\partial E}{\partial z_j} = y_i \frac{\partial E}{\partial z_j}$$

41



(Bad) Idea: Derive $\nabla_W L$ on paper

$$s = f(x; W) = Wx$$

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

$$= \sum_{j \neq y_i} \max(0, W_{j,:} \cdot x + W_{y_i,:} \cdot x + 1)$$

$$L = \frac{1}{N} \sum_{i=1}^N L_i + \lambda \sum_k W_k^2$$

$$= \frac{1}{N} \sum_{i=1}^N \sum_{j \neq y_i} \max(0, W_{j,:} \cdot x + W_{y_i,:} \cdot x + 1) + \lambda \sum_k W_k^2$$

$$\nabla_W L = \nabla_W \left(\frac{1}{N} \sum_{i=1}^N \sum_{j \neq y_i} \max(0, W_{j,:} \cdot x + W_{y_i,:} \cdot x + 1) + \lambda \sum_k W_k^2 \right)$$

Problem: Very tedious: Lots of matrix calculus, need lots of paper

Problem: What if we want to change loss? E.g. use softmax instead of SVM? Need to re-derive from scratch = (

Problem: Not feasible for very complex models!

42

42

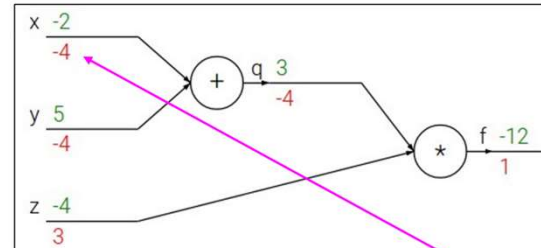
A simple case

- $f(x, y, z) = (x + y)z$
- E.g. $x=-2, y=5, z=-4$

$$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Chain rule:

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x}$$

Upstream
gradient

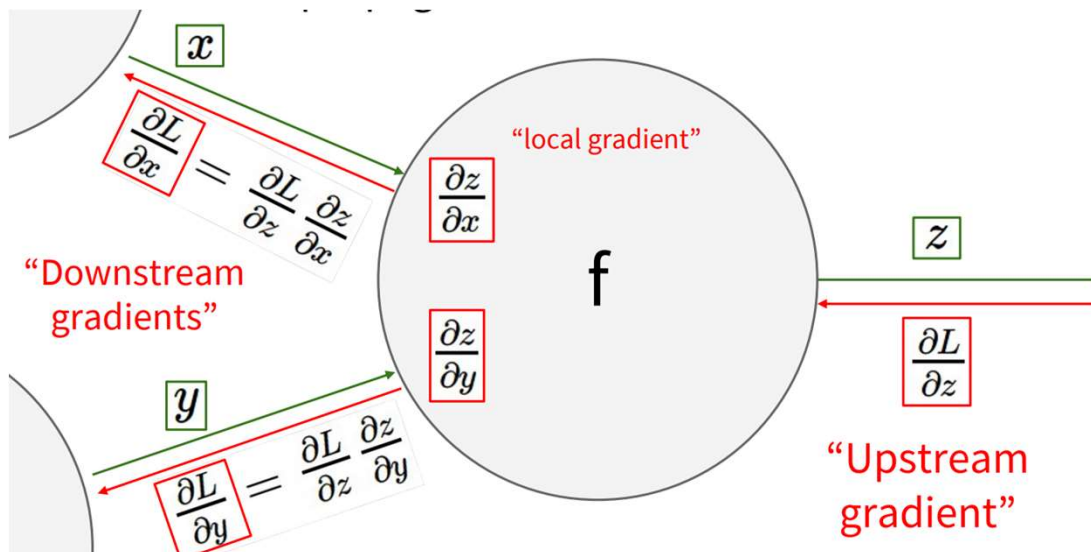
Local
gradient

$$\frac{\partial f}{\partial x}$$

43

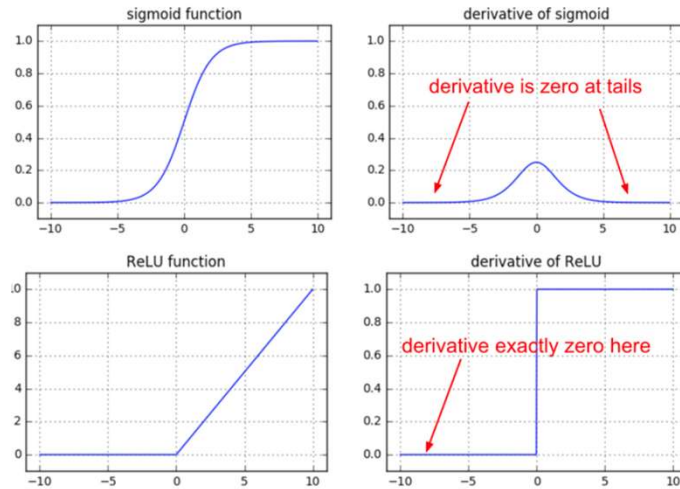
43

Computational graph



44

- Vanishing gradients on non-linear functions:



- Dead ReLU:

45

- Q1: Draw the computational graph of function $f = \text{sigmoid}(w_0x_0 + w_1x_1 + w_2)$. Given $[w_0, x_0, w_1, x_1, w_2] = [2.0, -1.0, -3.0, -2.0, -3]$, find the gradients in this graph with global gradient=1. Note: $\text{sigmoid}(1)=0.73$

46



Training set vs Test set

- Space of possible input values are usually infinite, and training set is only a FINITE subset
- Zero error on all training examples \neq good results on whole space of possible inputs (generalization error \neq empirical error...)
- Need to collect enough and representative examples (very critical in deep neural network)
- Essential to keep aside a subset of examples that shall be used only as TEST SET for estimating final generalization (when training finished)
- Need also to use some “validation set” independent from training set, in order to tune all hyper-parameters (layer sizes, number of iterations, etc...)

47

47



Optimize hyper-parameters by Validation

- To avoid over-fitting and maximize generalization, absolutely essential to use some VALIDATION estimation, for optimizing training hyper-parameters (and stopping criterion):
 - either use a separate validation dataset (random split of data into Training-set + Validation-set)
 - or use CROSS-VALIDATION:
 - Repeat k times: train on $(k-1)/k$ proportion of data + estimate error on remaining $1/k$ portion
 - Average the k error estimations



3-fold cross-validation:

- Train on $S1 \cup S2$ then estimate err_{S3} error on $S3$
- Train on $S1 \cup S3$ then estimate err_{S2} error on $S2$
- Train on $S2 \cup S3$ then estimate err_{S1} error on $S1$
- Average validation error: $(err_{S1} + err_{S2} + err_{S3})/3$

48

48

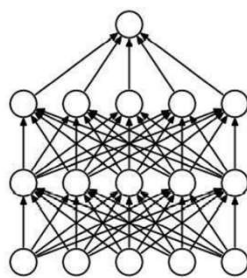
Some Neural Networks training “tricks”

- Importance of input normalization
 - zero mean, unit variance
- Importance of weights initialization
 - random but SMALL and prop. to $1/\sqrt{\text{nb Inputs}}$
- Decreasing (or adaptive) learning rate
- Importance of training set size
 - If a Neural Net has a LARGE number of free parameters
→ train it with a sufficiently large training-set!
- Avoid overfitting by Early Stopping of training iterations
- Avoid overfitting by use of L1 or L2 regularization
- For ConvNet (or network with huge amount of training parameters)
 - Dropout technique to avoid overfitting
 - Batch normalization

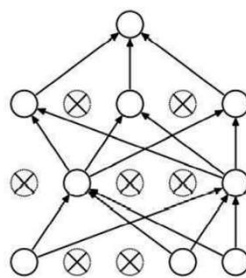
49

49

Dropout



(a) Standard Neural Net



(b) After applying dropout.

Dropout is proposed by Hinton et al., 2012, as a regularizer which randomly sets half of the activations to the fully connected layers to zero during training

At each training stage, individual nodes can be temporarily "dropped out" of the net with probability p (usually ~ 0.5), or re-installed with last values of weights

Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov (2012). Improving neural networks by preventing co-adaptation of feature detectors

50

50

Batch normalization (BN)

- Each layer of a NN has inputs with a corresponding distribution, which is affected during the training process by the randomness in the parameters initialization and input data → internal covariate shift
- It is believed that NN with BN can use higher learning rate without vanishing or exploding gradients.
- It seems to have a regularizing effect and thus unnecessary to use dropout after.

[Szegedy, Christian (2015). "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift"]

Given four trainable parameters $\mu_B, \sigma_B, \gamma, \beta$ (mean, variance, arbitrary scale, arbitrary bias, respectively). Use min-batch (B) with size m, input d-dimensional $x = [x^{(1)}, \dots, x^{(d)}]$

Step1: $\mu_B = \frac{1}{m} \sum_{i=1}^m x_i$ and $\sigma_B^2 = \frac{1}{m} \sum_{i=1}^m (x_i - \mu_B)^2$

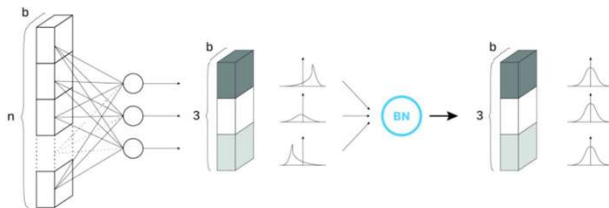
Step2: normalized input

$$\hat{x}_i^{(k)} = \frac{x_i^{(k)} - \mu_B^{(k)}}{\sqrt{\sigma_B^{(k)^2} + \epsilon}}, \text{ where } k \in [1, d], \text{ and } i \in [1, m]$$

Step3: Output of current layer

$$y_i^{(k)} = \gamma^{(k)} \hat{x}_i^{(k)} + \beta^{(k)}$$

Step4: backpropagation with chain rule



51

51

(Optional) BN limitation

- BN has shown great results and has been used heavily in tons of SOTA methods. However, BN has the following concerns
 - Very expensive computational costs
 - Introduces a lot of extra hyper-parameters that need further fine-tuning
 - Causes a lot of implementation errors in distributed training
 - Performs poorly on small batch sizes, which are used often in training larger models
- Normalized free networks (NFNets, 2021) reaches ImageNet top1 accuracy 86.5% and 8.7 times faster than EfficientNET
 - without the use of BN
 - New idea of Adaptive Gradient Clipping (AGC) → Gradient clipping with a way to automatically calculate the threshold hyperparameter based on the premise:
 - *the unit-wise ratio of the norm of the gradients to the norm of the weights of a layer provides a simple measure of how much a single gradient descent step will change the original weights.*
 - Sharpness-Aware Minimization (SAM) [Foret et al. 2012]

Brock, A., De, S., Smith, S. L., & Simonyan, K. (2021). *High-Performance Large-Scale Image Recognition Without Normalization*.

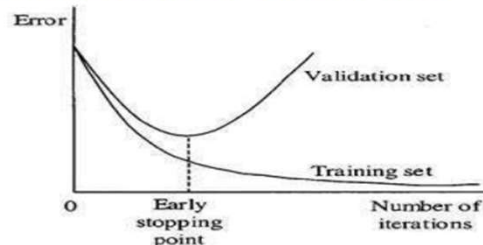
Foret, P., Kleiner, A., Mobahi, H., and Neyshabur, B. Sharpness-aware minimization for efficiently improving generalization. In 9th International Conference on Learning Representations, ICLR, 2021

52

52

Early stopping

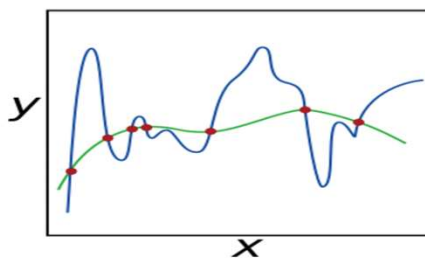
- For Neural Networks, a first method to avoid overfitting is to STOP LEARNING iterations as soon as the validation_error stops decreasing
- Generally, not a good idea to decide the number of iterations beforehand. Better to ALWAYS USE EARLY STOPPING

ALWAYS USE EARLY STOPPING

53

53

Regularization penalty



Trying to fit too many free parameters with not enough information can lead to overfitting

Regularization = penalizing too complex models

Often done by adding a special term to cost function

For neural network, the regularization term is just norm L2 or L1 of vector of all weights:

$$K = \sum_m (\text{loss}(Y_m, D_m)) + \beta \sum_{ij} |W_{ij}|^p \quad \text{with } p=2 \text{ (L2) or } p=1 \text{ (L1)}$$

→ name **“Weight decay”**

54

54



Stochastic Gradient Descent (SGD)

$$L(W) = \frac{1}{N} \sum_{i=1}^N L_i(x_i, y_i, W) + \lambda R(W)$$

$$\nabla_W L(W) = \frac{1}{N} \sum_{i=1}^N \nabla_W L_i(x_i, y_i, W) + \lambda \nabla_W R(W)$$

Full sum expensive
when N is large!

Approximate sum
using a **minibatch** of
examples
32 / 64 / 128 common

```
# Vanilla Minibatch Gradient Descent
```

```
while True:
```

```
    data_batch = sample_training_data(data, 256) # sample 256 examples
```

```
    weights_grad = evaluate_gradient(loss_fun, data_batch, weights)
```

```
    weights += - step_size * weights_grad # perform parameter update
```

55

55

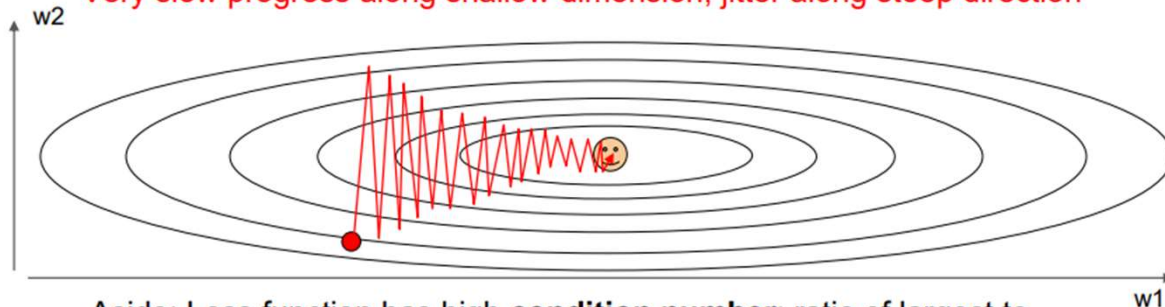


PSL Problems with SGD

What if loss changes quickly in one direction and slowly in another?

What does gradient descent do?

Very slow progress along shallow dimension, jitter along steep direction

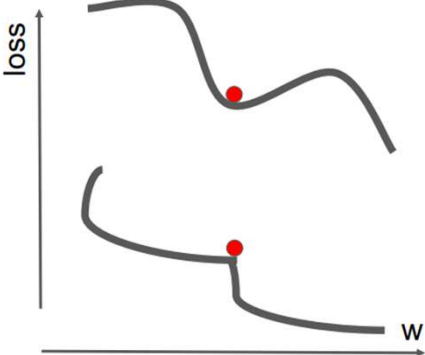


Aside: Loss function has high **condition number**: ratio of largest to smallest singular value of the Hessian matrix is large

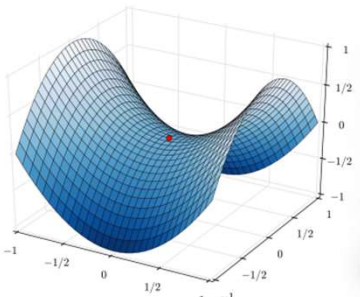
56

56

MINES ParisTech | PSL Saddle point and local minima



- Both will have zero gradient and gradient descent gets stuck
- Saddle points much more common in high dimension
- $f(x, y) = x^2 - y^2$



57

MINES ParisTech | PSL SGD+Momentum

continue moving in the general direction as the previous iterations

SGD

$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

```
while True:
    dx = compute_gradient(x)
    x -= learning_rate * dx
```

SGD+Momentum

$$v_{t+1} = \rho v_t + \nabla f(x_t)$$

$$x_{t+1} = x_t - \alpha v_{t+1}$$

```
vx = 0
while True:
    dx = compute_gradient(x)
    vx = rho * vx + dx
    x -= learning_rate * vx
```

- Build up “velocity” as a running mean of gradients
- Rho gives “friction”; typically rho=0.9 or 0.99

Sutskever et al, “On the importance of initialization and momentum in deep learning”, ICML 2013

58

58

SGD +
Momentum

```
vx = 0
while True:
    dx = compute_gradient(x)
    vx = rho * vx + dx
    x -= learning_rate * vx
```

Adds element-wise scaling of the gradient based on the historical sum of squares in each dimension (with decay)

RMSProp

```
grad_squared = 0
while True:
    dx = compute_gradient(x)
    grad_squared = decay_rate * grad_squared + (1 - decay_rate) * dx * dx
    x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

59

59

```
first_moment = 0
second_moment = 0
for t in range(1, num_iterations):
    dx = compute_gradient(x)
```

```
first_moment = beta1 * first_moment + (1 - beta1) * dx
```

Momentum

```
second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
```

```
first_unbias = first_moment / (1 - beta1 ** t)
```

Bias correction

```
second_unbias = second_moment / (1 - beta2 ** t)
```

```
x -= learning_rate * first_unbias / (np.sqrt(second_unbias) + 1e-7))
```

AdaGrad / RMSProp

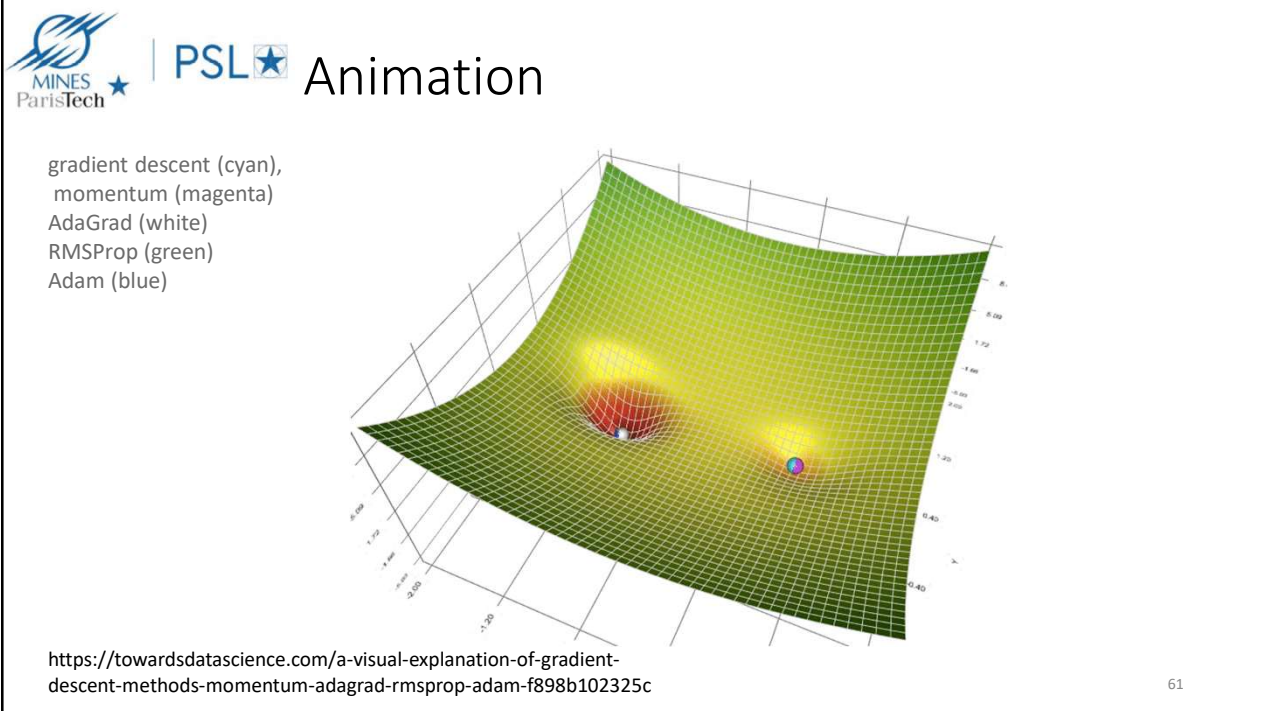
Bias correction for the fact that first and second moment estimates start at zero

Adam with **beta1 = 0.9**, **beta2 = 0.999**, and **learning_rate = 1e-3 or 5e-4** is a great starting point for many models!

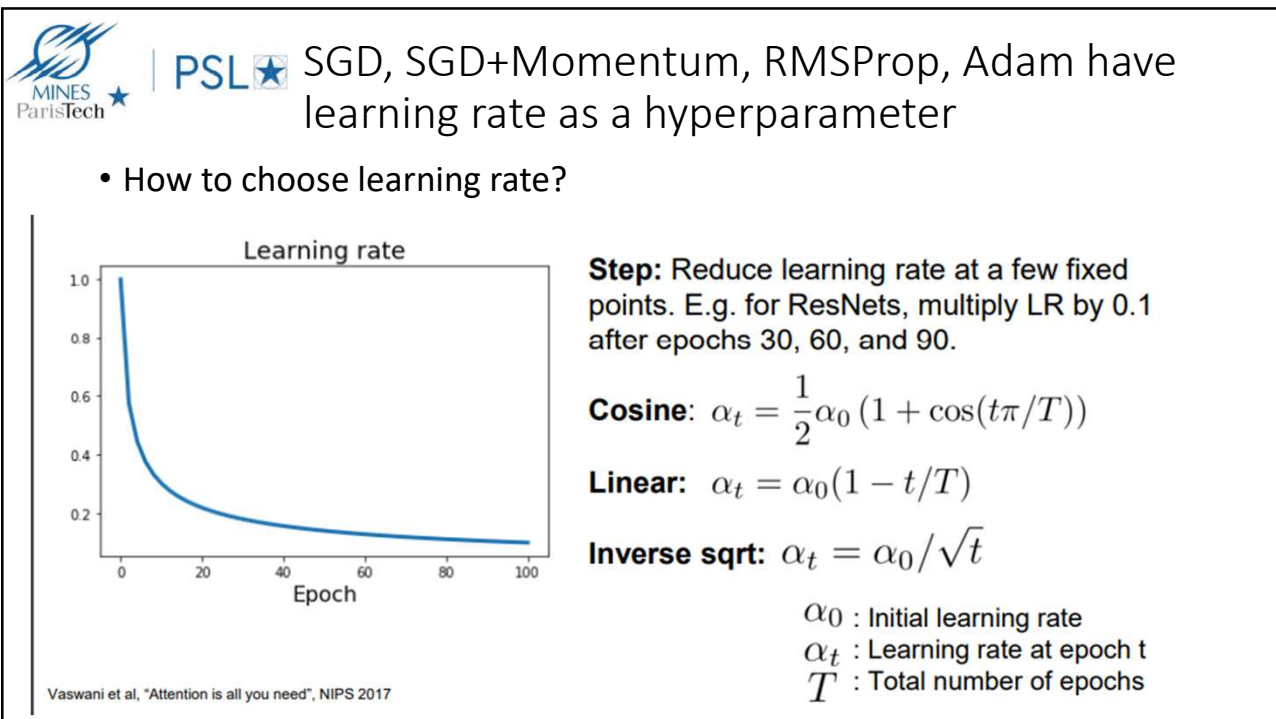
Gingma and Ba, "Adam: A method for stochastic optimization", ICLR 2015

60

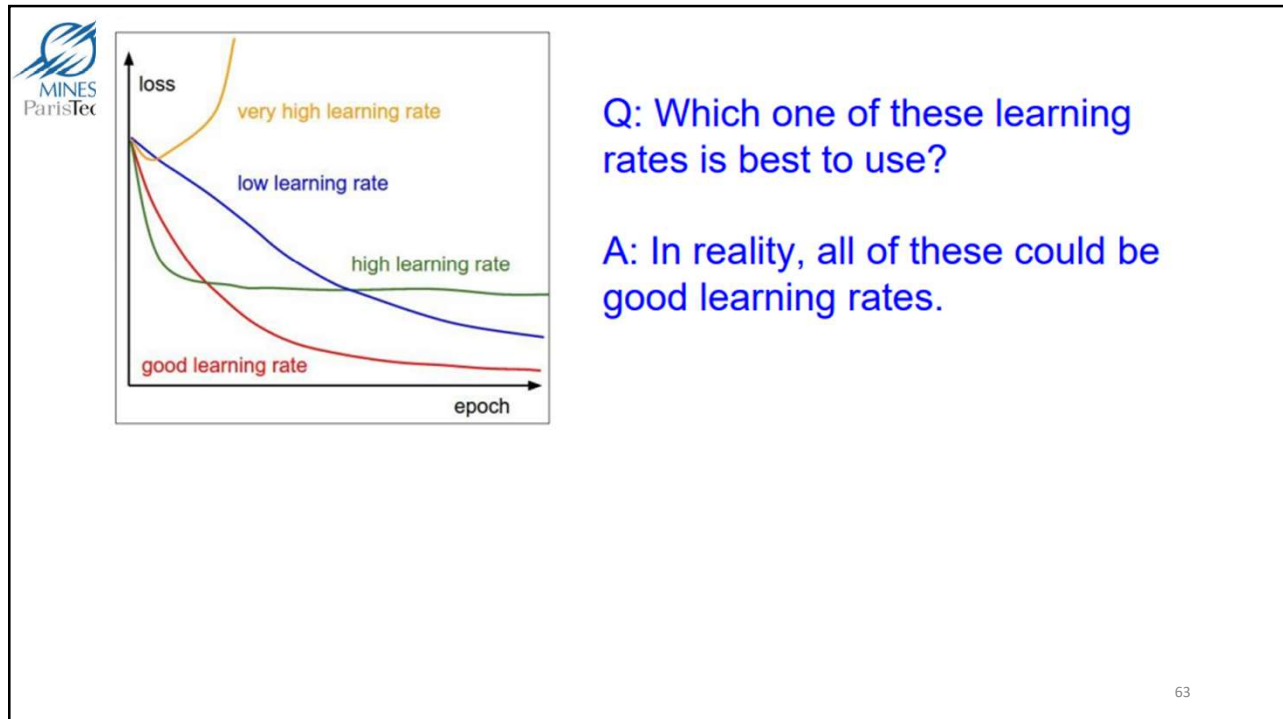
60



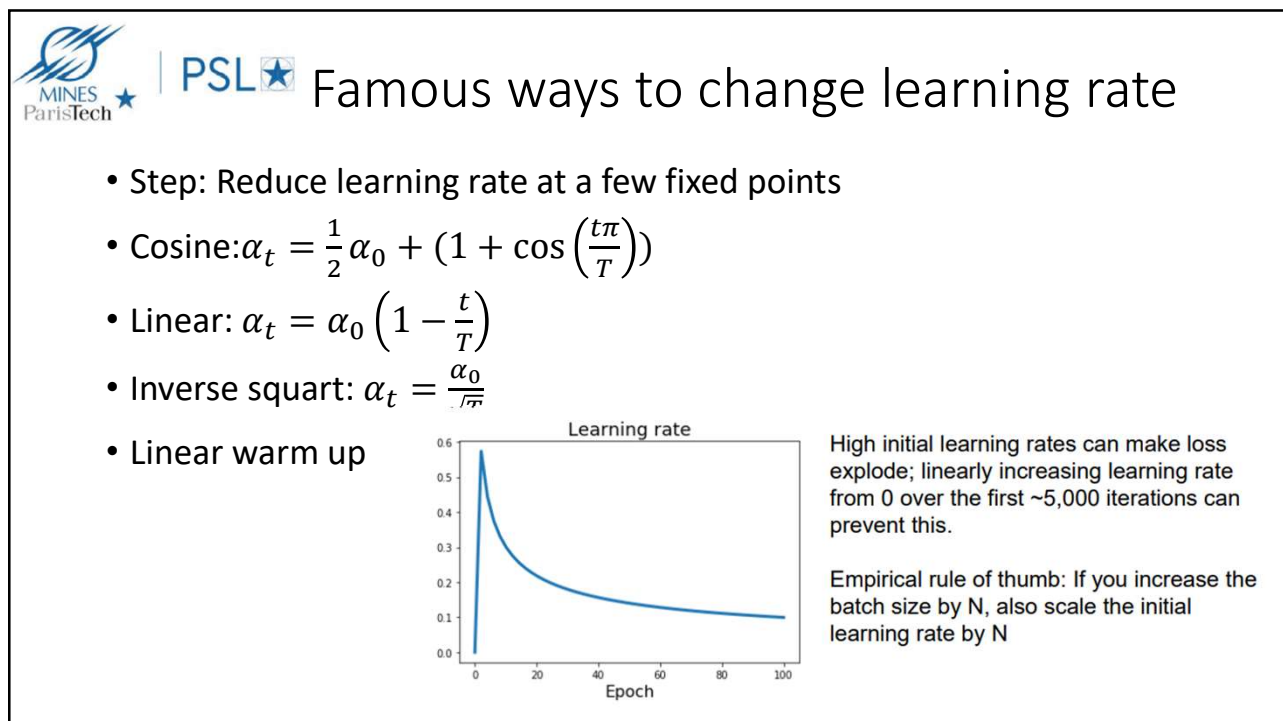
61



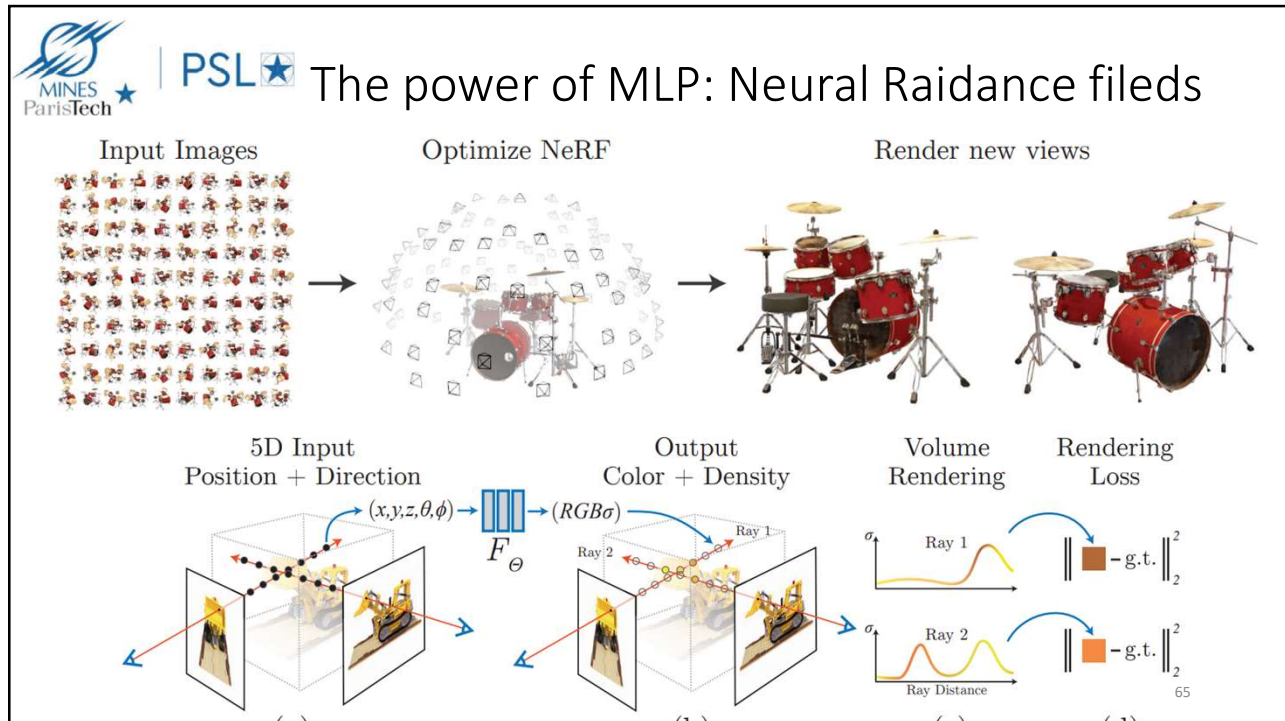
62



63



64



65



66

66