A Soft Introduction to Deep Learning

Deep Ray
Institute of Mathematics,
EPFL, Switzerland
deep.ray@epfl.ch
http://deepray.github.io

What this course is about

At the end of these lectures

- You will NOT be able to train a gaming algorithm
- You will NOT be able to design a self-driving car
- You will NOT be able to solve PDEs using AI
- You will NOT be able to predict the stock-market

What this course is about

At the end of these lectures

- You will NOT be able to train a gaming algorithm
- You will NOT be able to design a self-driving car
- You will NOT be able to solve PDEs using AI
- You will NOT be able to predict the stock-market
- You will see that Deep Learning is uncomplicated

What this course is about

At the end of these lectures

- You will NOT be able to train a gaming algorithm
- You will NOT be able to design a self-driving car
- You will NOT be able to solve PDEs using AI
- You will NOT be able to predict the stock-market
- You will see that Deep Learning is uncomplicated
- You will understand the fundamentals of training networks

What this course is about

At the end of these lectures

- You will NOT be able to train a gaming algorithm
- You will NOT be able to design a self-driving car
- You will NOT be able to solve PDEs using AI
- You will NOT be able to predict the stock-market
- You will see that Deep Learning is uncomplicated
- You will understand the fundamentals of training networks
- You will get the courage to experiment and explore

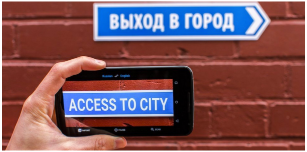
What this course is about

At the end of these lectures

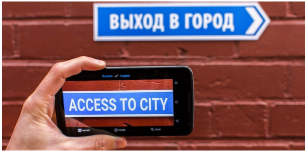
- You will NOT be able to train a gaming algorithm
- You will NOT be able to design a self-driving car
- You will NOT be able to solve PDEs using AI
- You will NOT be able to predict the stock-market
- You will see that Deep Learning is uncomplicated
- You will understand the fundamentals of training networks
- You will get the courage to experiment and explore
- You might get ideas to use Deep Learning solve your problems

Where have we seen AI in action

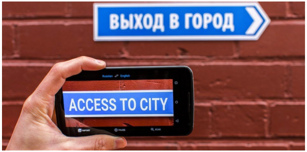
Where have we seen AI in action



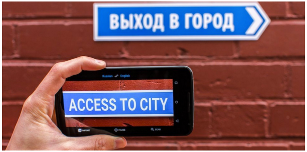
Where have we seen AI in action

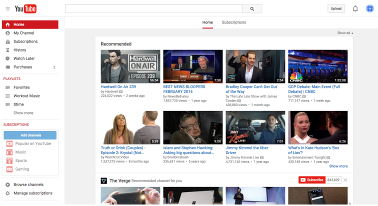


Where have we seen AI in action

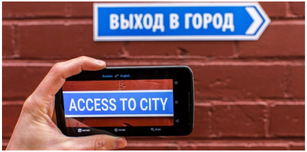


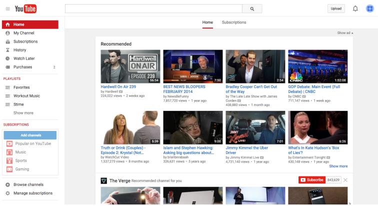
Where have we seen AI in action

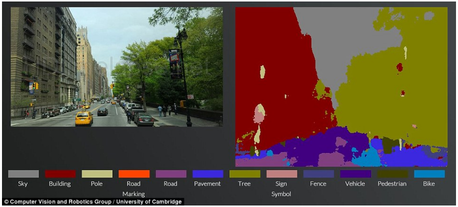




Where have we seen AI in action







How did we get here?

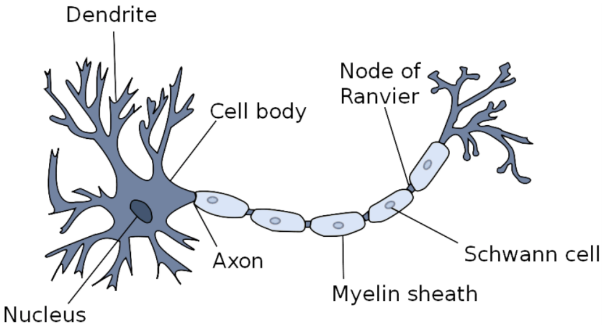
The Thresholded Logic Unit

1940	1950	1960	1970	1980	1990	2000	2010
McCulloch and Pitts (1943)							

The Thresholded Logic Unit

1940	1950	1960	1970	1980	1990	2000	2010

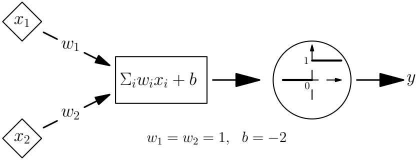
McCulloch and Pitts (1943)



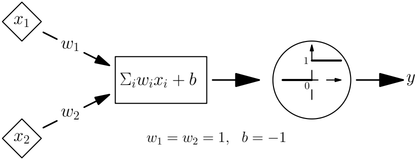
Biological neuron

1940	1950	1960	1970	1980	1990	2000	2010

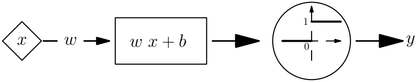
McCulloch and Pitts (1943)



AND



OR



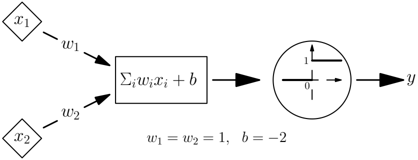
$w = -1, \quad b = 0$

NOT

The Thresholded Logic Unit

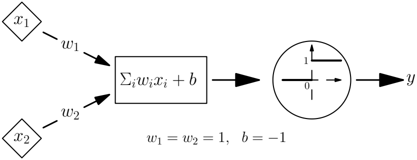
1940	1950	1960	1970	1980	1990	2000	2010

McCulloch and Pitts (1943)



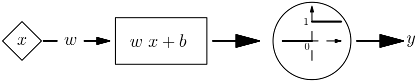
$w_1 = w_2 = 1, \quad b = -2$

AND



$w_1 = w_2 = 1, \quad b = -1$

OR



$w = -1, \quad b = 0$

NOT

Weights are adjustable but not learned!

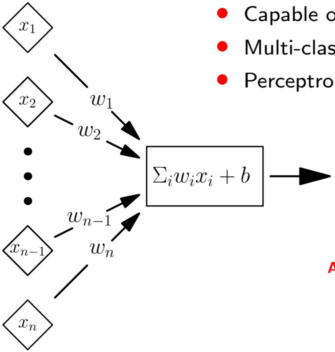
The Perceptron

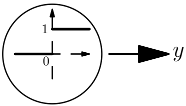
1940 1950 1960 1970 1980 1990 2000 2010

McCulloch and Pitts (1943)

Rosenblatt (1957)

- Weights are learnable
- Capable of classifying data into 2 classes
- Multi-class classification using One-vs-All
- Perceptron convergence theorem





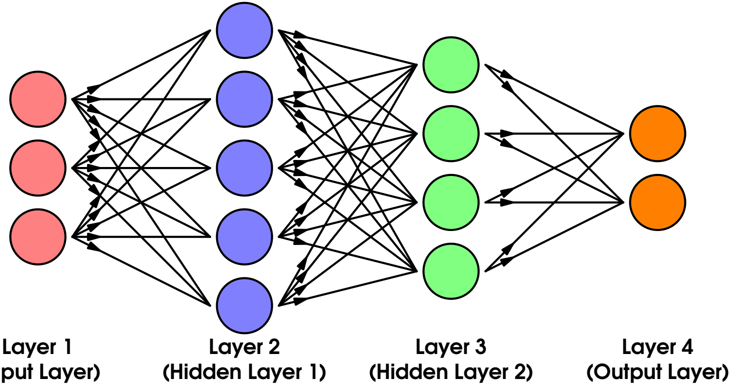
Activation function

1940	1950	1960	1970	1980	1990	2000	2010

McCulloch and Pitts (1943)

Rosenblatt (1957)

Multilayer perceptron (MLP)



The Perceptron: Criticism

1940	1950	1960	1970	1980	1990	2000	2010
McCulloch and Pitts (1943)		Rosenblatt (1957)	Minsky and Papert (1969)				

Perceptron: An Introduction to Computational Geometry

- Detailed mathematical analysis of perceptrons
- Limitations of perceptrons – for e.g. XOR problem
- Claimed issues exist for other variants

INPUT		OUTPUT
X_1	X_2	
0	0	0
0	1	1
1	0	1
1	1	0

The Perceptron: Criticism

1940	1950	1960	1970	1980	1990	2000	2010

McCulloch and Pitts (1943)

Rosenblatt (1957)

Minsky and Papert (1969)

"The perceptron has shown itself worthy of study despite (and even because of!) its severe limitations. It has many features to attract attention: its linearity; its intriguing learning theorem; its clear paradigmatic simplicity as a kind of parallel computation. There is no reason to suppose that any of these virtues carry over to the many-layered version. Nevertheless, we consider it to be an important research problem to elucidate (or reject) our intuitive judgement that the <u>extension to multilayer systems is sterile</u>."

The Perceptron: Criticism

1940	1950	1960	1970	1980	1990	2000	2010

McCulloch and Pitts (1943)

Rosenblatt (1957)

Minsky and Papert (1969)

1st A.I. Winter (1969-1986)

"The perceptron has shown itself worthy of study despite (and even because of!) its severe limitations. It has many features to attract attention: its linearity; its intriguing learning theorem; its clear paradigmatic simplicity as a kind of parallel computation. There is no reason to suppose that any of these virtues carry over to the many-layered version. Nevertheless, we consider it to be an important research problem to elucidate (or reject) our intuitive judgement that the extension to multilayer systems is sterile."

Revival with Backpropagation

1940	1950	1960	1970	1980	1990	2000	2010

McCulloch and Pitts (1943)

Rosenblatt (1957)

Minsky and Papert (1969)

1st A.I. Winter (1969-1986)

Backprop and theory (1986)

- *Learning representations by back-propagating errors* by Rumelhart, Hinton and Williams (1986)
- Other advancements by Bengio, Lecun and others ...
- Efficient evaluation of gradients
- Universal Function Approximation theorem for MLPs by Cybenko (1989)
- Theoretical investigation by Barron, Pinkus, Mhaskar ...

Revival with Backpropagation

1940	1950	1960	1970	1980	1990	2000	2010

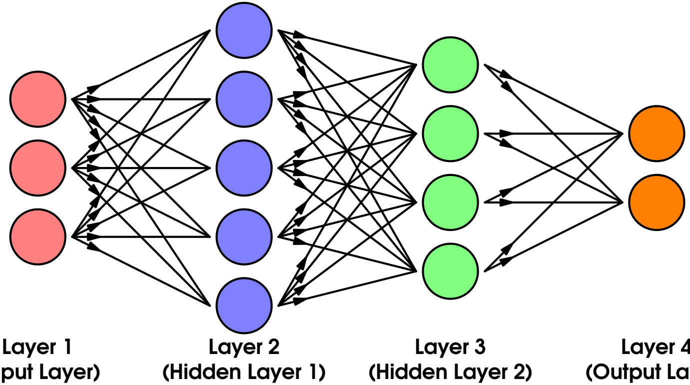
McCulloch and Pitts (1943)

Rosenblatt (1957)

Minsky and Papert (1969)

1st A.I. Winter (1969-1986)

Backprop and theory (1986)



What is backpropogation?

Second freeze and resurgence

1940	1950	1960	1970	1980	1990	2000	2010

McCulloch and Pitts (1943)

Rosenblatt (1957)

Minsky and Papert (1969)

1st A.I. Winter (1969-1986)

Backprop and theory (1986)

2nd A.I. Winter (1993-2008)

Issues with back-propagation

- Not enough labelled data
- Learning time scales badly (exponentially) with multiple layers
- Deep networks can have several local minima

Second freeze and resurgence

1940	1950	1960	1970	1980	1990	2000	2010

McCulloch and Pitts (1943)

Rosenblatt (1957)

Minsky and Papert (1969)

1st A.I. Winter (1969-1986)

Backprop and theory (1986)

2nd A.I. Winter (1990-2006)

Resurgence (2006)

Issues with back-propagation

- Not enough labelled data
- Learning time scales badly (exponentially) with multiple layers
- Deep networks can have several local minima

Resurgence

- Hinton's idea of unsupervised pre-training and Deep Belief Nets – helps with semi-supervised training
- Availability of large data sets
- GPUs and other computational advancements
- Better training algorithms

Components of MLPs

- Depth Vs width
- Data sets
- Activation functions
- Loss/cost functions
- Initialization
- Stochasticity and mini-batches
- Overfitting and underfitting
- Optimizers and learning-rate
- Hyper-parameter tuning

Depth Vs Width

- Several partial results exist about the width and depth
- Need $O(N_{inp} + N_{out} + M)$ parameters to represent a dataset of size M
- There is a gap between theory and training
- Going "deeper and narrow" gives better results than staying "shallow but wider"
- Need to find optimal structure based on application

Components of MLPs

- Depth Vs width
- Data sets
- Activation functions
- Loss/cost functions
- Initialization
- Stochasticity and mini-batches
- Overfitting and underfitting
- Optimizers and learning-rate
- Hyper-parameter tuning

Training, validation and testing

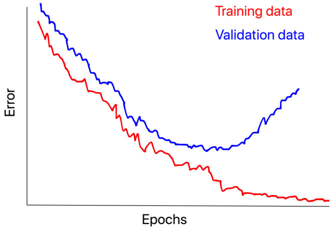
Three types of data sets:

- **Training data:** Used to optimize the weights and biases.

Training, validation and testing

Three types of data sets:

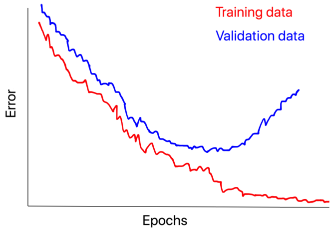
- **Training data:** Used to optimize the weights and biases.
- **Validation data:** Used while training the network for cross-validation – helps to detect *overfitting*.



Training, validation and testing

Three types of data sets:

- **Training data:** Used to optimize the weights and biases.
- **Validation data:** Used while training the network for cross-validation – helps to detect *overfitting*.

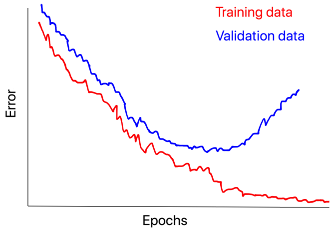


- **Test data:** Used to test final trained model – DO NOT TOUCH DURING TRAINING PHASE!

Training, validation and testing

Three types of data sets:

- **Training data:** Used to optimize the weights and biases.
- **Validation data:** Used while training the network for cross-validation – helps to detect *overfitting*.



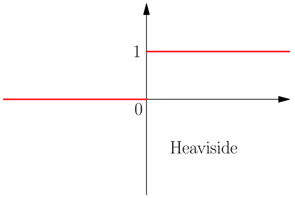
- **Test data:** Used to test final trained model – DO NOT TOUCH DURING TRAINING PHASE!

Scaling and pre-processing the input data – important for generalization.

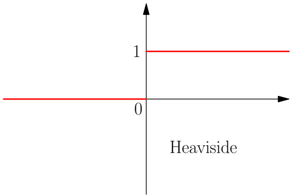
Components of MLPs

- Depth Vs width
- Data sets
- Activation functions
- Loss/cost functions
- Initialization
- Stochasticity and mini-batches
- Overfitting and underfitting
- Optimizers and learning-rate
- Hyper-parameter tuning

Activation functions

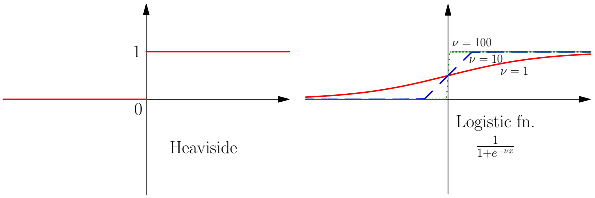


Activation functions



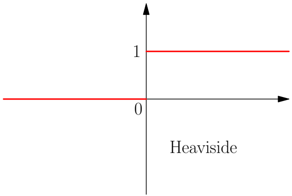
- McCullock-Pitts neuron
- Zero gradient – bad for backpropagation
- Not used anymore

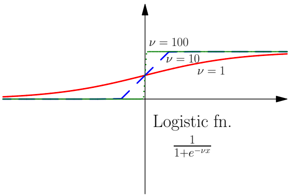
Activation functions



- McCullock-Pitts neuron
- Zero gradient – bad for backpropagation
- Not used anymore

Activation functions

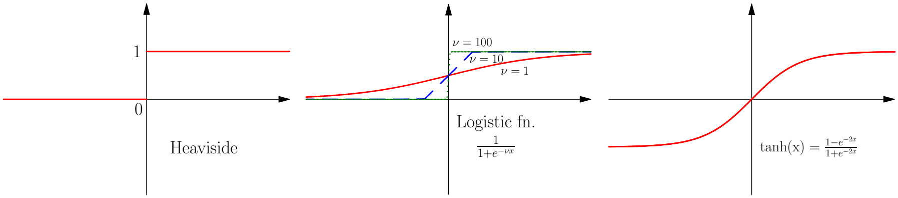




- McCullock-Pitts neuron
- Zero gradient – bad for backpropagation
- Not used anymore

- Smooth approximation to Heaviside func.
- Sigmoidal function – used in most proofs
- Good for binary classification
- Not symmetric

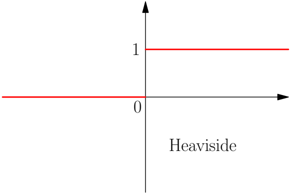
Activation functions

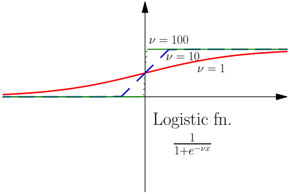


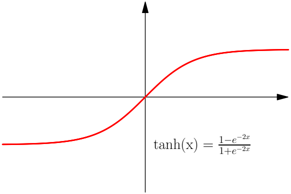
- McCullock-Pitts neuron
- Zero gradient – bad for backpropagation
- Not used anymore

- Smooth approximation to Heaviside func.
- Sigmoidal function – used in most proofs
- Good for binary classification
- Not symmetric

Activation functions



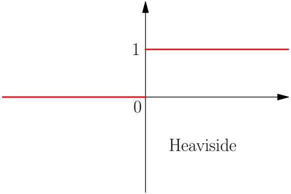




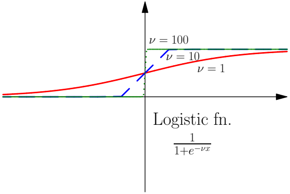
- McCullock-Pitts neuron
- Zero gradient – bad for backpropagation
- Not used anymore

- Smooth approximation to Heaviside func.
- Sigmoidal function – used in most proofs
- Good for binary classification
- Not symmetric

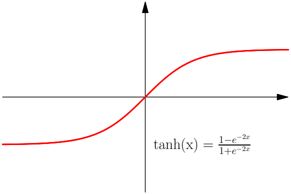
- Symmetric unlike Logisitic func.
- Smooth
- Vanishing gradients away from 0.



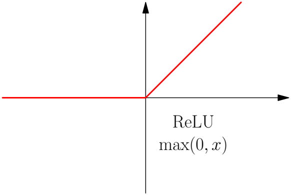
1
0
Heaviside



$\nu = 100$
$\nu = 10$
$\nu = 1$
Logistic fn.
$\frac{1}{1+e^{-\nu x}}$

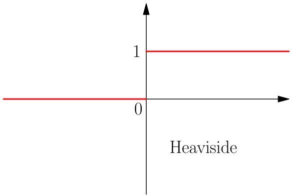


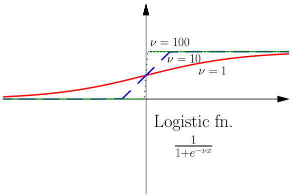
$\tanh(\mathrm{x}) = \frac{1-e^{-2x}}{1+e^{-2x}}$

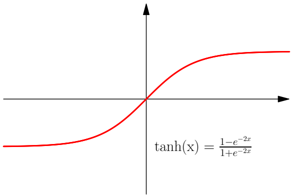


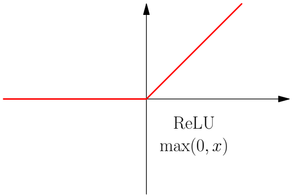
ReLU
$\max(0, x)$

Activation functions

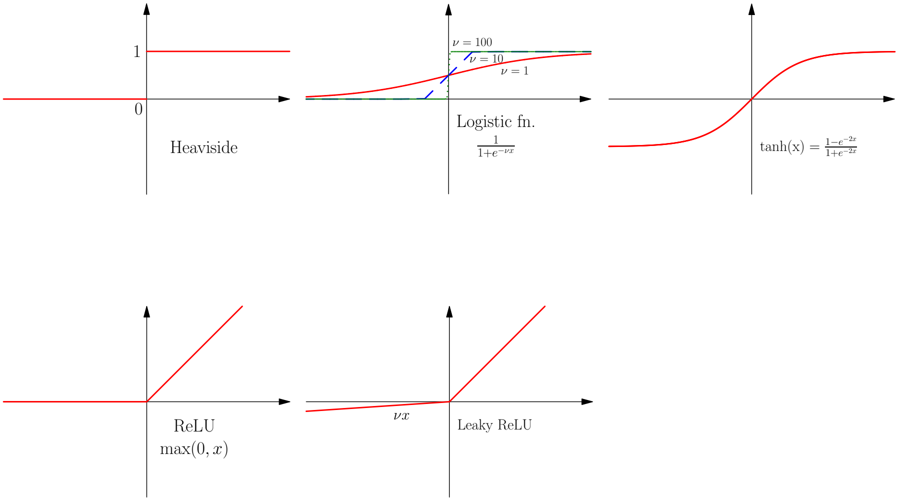




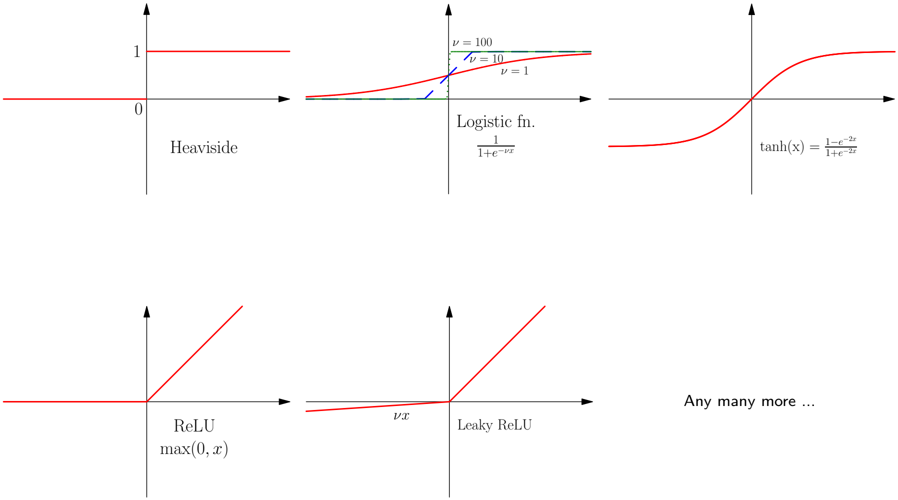




- Easy to compute
- Reduces vanishing gradient problem
- Scale invariant
- Issue of dying neurons


1
0
Heaviside
$\nu = 100$
$\nu = 10$
$\nu = 1$
Logistic fn.
$\frac{1}{1+e^{-\nu x}}$
$\tanh(x) = \frac{1-e^{-2x}}{1+e^{-2x}}$
ReLU
$\max(0, x)$
νx
Leaky ReLU

Activation functions



Any many more ...

Components of MLPs

- Depth Vs width
- Data sets
- Activation functions
- Loss/cost functions
- Initialization
- Stochasticity and mini-batches
- Overfitting and underfitting
- Optimizers and learning-rate
- Hyper-parameter tuning

Loss/cost function

Regression problem

- Mean squared error
- Mean L1 error
- Mean absolute error
- ...

Loss/cost function

Regression problem

- Mean squared error
- Mean L1 error
- Mean absolute error
- ...

Classification problem

- Use softmax output function

$$\hat{Y}^{(k)} = \frac{e^{\hat{Y}^{(k)}}}{\sum_j e^{\hat{Y}^{(j)}}} \quad \in \quad [0,1] \quad \longrightarrow \quad \text{probabilities/classification}$$

- Cross-entropy loss function

$$C = -\sum_{i=1}^{\mathsf{M}} \sum_j Y_i^{(j)} \log\left(\hat{Y}_i^{(j)}\right)$$

Components of MLPs

- Depth Vs width
- Data sets
- Activation functions
- Loss/cost functions
- Initialization
- Stochasticity and mini-batches
- Overfitting and underfitting
- Optimizers and learning-rate
- Hyper-parameter tuning

Initialization

Before starting the training algorithm, the weights (w) and biases b need to be initialized

Initialization

Before starting the training algorithm, the weights (w) and biases b need to be initialized

- Do not initialize w to be zero – leads to linear model

Initialization

Before starting the training algorithm, the weights (w) and biases b need to be initialized

- Do not initialize w to be zero – leads to linear model
- Initialize randomly using normal distribution etc
- Exploding gradients – avoided using heuristic scaling depending on activation function – minimize variance of the weights
  - For ReLU

$$w^l = \text{nrand}(N_l \ , \ N_{l-1}).\sqrt{\frac{2}{N_{l-1}}}$$

  - For tanh (Xavier initialization)

$$w^l = \text{nrand}(N_l \ , \ N_{l-1}).\sqrt{\frac{1}{N_{l-1}}}$$

Components of MLPs

- Depth Vs width
- Data sets
- Activation functions
- Loss/cost functions
- Initialization
- Stochasticity and mini-batches
- Overfitting and underfitting
- Optimizers and learning-rate
- Hyper-parameter tuning

Stochasticity and mini-batches

Training $\longrightarrow$ M samples

Randomly shuffle data $\longrightarrow$ introduces stochasticity – speeds-up convergence

Choose m samples at a time to take one optimization step

Stochasticity and mini-batches

Training $\longrightarrow$ M samples

Randomly shuffle data $\longrightarrow$ introduces stochasticity – speeds-up convergence

Choose m samples at a time to take one optimization step

$m = M \longrightarrow$ Batch optimization
$m = 1 \;\longrightarrow$ Stochastic optimization

Stochasticity and mini-batches

Training $\longrightarrow$ M samples

Randomly shuffle data $\longrightarrow$ introduces stochasticity – speeds-up convergence

Choose m samples at a time to take one optimization step

$m = M$ $\longrightarrow$ Batch optimization too long per iteration
$m = 1$ $\longrightarrow$ Stochastic optimization lose speed-up via vectorization

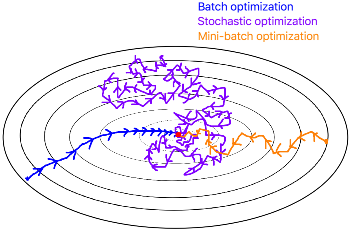
Stochasticity and mini-batches

Training $\longrightarrow$ M samples

Randomly shuffle data $\longrightarrow$ introduces stochasticity – speeds-up convergence

Choose m samples at a time to take one optimization step

$m = M$ $\longrightarrow$ Batch optimization too long per iteration
$m = 1$ $\longrightarrow$ Stochastic optimization lose speed-up via vectorization
$1 < m < M$ $\longrightarrow$ Mini-batch optimization a good comprimise



Stochasticity and mini-batches

Training $\longrightarrow$ M samples

Randomly shuffle data $\longrightarrow$ introduces stochasticity – speeds-up convergence

Choose m samples at a time to take one optimization step

$m = M$ $\longrightarrow$ Batch optimization too long per iteration
$m = 1$ $\longrightarrow$ Stochastic optimization lose speed-up via vectorization
$1 < m < M$ $\longrightarrow$ Mini-batch optimization a good comprimise

Typically choose $m \sim 32$ for large data sets.

Stochasticity and mini-batches

Training $\longrightarrow M$ samples

Randomly shuffle data $\longrightarrow$ introduces stochasticity – speeds-up convergence

Choose m samples at a time to take one optimization step

$m = M \longrightarrow$ Batch optimization too long per iteration
$m = 1 \longrightarrow$ Stochastic optimization lose speed-up via vectorization
$1 < m < M \longrightarrow$ Mini-batch optimization a good comprimise

Typically choose $m \sim 32$ for large data sets.

We complete **1 epoch** when we have finished striding over the whole dataset – approx. M/m optimizations steps.

- Depth Vs width
- Data sets
- Activation functions
- Loss/cost functions
- Initialization
- Stochasticity and mini-batches
- Overfitting and underfitting
- Optimizers and learning-rate
- Hyper-parameter tuning

Overfitting and underfitting

Training Accuracy	99 %	90 %	89 %
Validation Accuracy	89 %	89 %	78 %
	High Variance or Overfitting	High Bias or Underfitting	High Bias and High Variance

Overfitting and underfitting

Training Accuracy	99 %	90 %	89 %
Validation Accuracy	89 %	89 %	78 %
	High Variance or Overfitting	High Bias or Underfitting	High Bias and High Variance

Aim: Have small bias and variance

Overfitting and underfitting

Training Accuracy	99 %	90 %	89 %
Validation Accuracy	89 %	89 %	78 %
	High Variance or Overfitting	High Bias or Underfitting	High Bias and High Variance

Aim: Have small bias and variance

High Bias:
(depends on training accuracy)

- Try a bigger network
- Train longer
- Change architecture
- Taking more data won't help!

Overfitting and underfitting

Training Accuracy	99 %	90 %	89 %
Validation Accuracy	89 %	89 %	78 %
	High Variance or Overfitting	High Bias or Underfitting	High Bias and High Variance

Aim: Have small bias and variance

High Bias:
(depends on training accuracy)

- Try a bigger network
- Train longer
- Change architecture
- Taking more data won't help!

High Variance:
(depends on validation accuracy)

- Take a bigger training set
- Regularization
- Change architecture

Overfitting and underfitting

Training Accuracy	99 %	90 %	89 %
Validation Accuracy	89 %	89 %	78 %
	High Variance or Overfitting	High Bias or Underfitting	High Bias and High Variance

Aim: Have small bias and variance

High Bias:
(depends on training accuracy)

- Try a bigger network
- Train longer
- Change architecture
- Taking more data won't help!

High Variance:
(depends on validation accuracy)

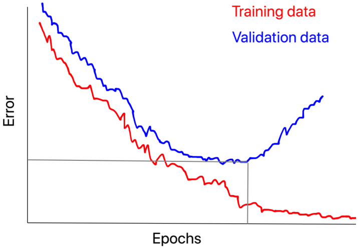
- Take a bigger training set
- **Regularization**
- Change architecture

Regularization

Improves generalization error

Improves generalization error

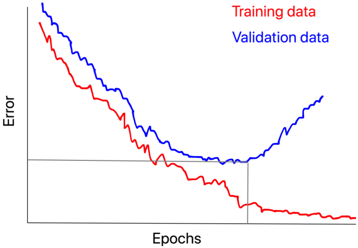
- Early stopping of training after M succesive increases in validation error

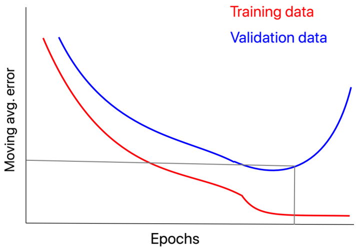


Regularization

Improves generalization error

- Early stopping of training after M succesive increases in validation error





$$\overline{E_n} = \frac{1}{n}\sum_{i=1}^{n} E_i$$

Regularization

Improves generalization error

- Early stopping of training after M succesive increases in validation error – not recommended

Regularization

Improves generalization error

- Early stopping of training after M succesive increases in validation error – not recommended
- Penalizing the cost function

$$\widetilde{C}(Y, \hat{Y}; W, b) = C(Y, \hat{Y}; W, b) + \alpha\Omega(W)$$

Regularization

Improves generalization error

- Early stopping of training after M succesive increases in validation error – not recommended
- Penalizing the cost function

$$\widetilde{C}(Y, \hat{Y}; W, b) = C(Y, \hat{Y}; W, b) + \alpha\Omega(W)$$

b not regularized as

- b easier to fit
- W model variable interactions
- regularizing b can cause severe underfitting

Regularization

Improves generalization error

- Early stopping of training after M succesive increases in validation error – not recommended
- Penalizing the cost function

$$\widetilde{C}(Y, \hat{Y}; W, b) = C(Y, \hat{Y}; W, b) + \alpha\Omega(W)$$

  - $\Omega(W) = \|W\|_2^2 \longrightarrow$ drives weights closer to zero
  - $\Omega(W) = \|W\|_1 \longrightarrow$ induces sparsity

Regularization

Improves generalization error

- Early stopping of training after M succesive increases in validation error – not recommended
- Penalizing the cost function

$$\widetilde{C}(Y, \hat{Y}; W, b) = C(Y, \hat{Y}; W, b) + \alpha\Omega(W)$$

  - $\Omega(W) = \|W\|_2^2 \longrightarrow$ drives weights closer to zero
  - $\Omega(W) = \|W\|_1 \longrightarrow$ induces sparsity
- Data augmentation – when meaningful geometric information is available

Regularization

Improves generalization error

- Early stopping of training after M succesive increases in validation error – not recommended
- Penalizing the cost function

$$\widetilde{C}(Y, \hat{Y}; W, b) = C(Y, \hat{Y}; W, b) + \alpha\Omega(W)$$

  - $\Omega(W) = \|W\|_2^2 \longrightarrow$ drives weights closer to zero
  - $\Omega(W) = \|W\|_1 \longrightarrow$ induces sparsity
- Data augmentation – when meaningful geometric information is available
- Dropout – randomly kill hidden neuron outputs while training (only)

Components of MLPs

- Depth Vs width
- Data sets
- Activation functions
- Loss/cost functions
- Initialization
- Stochasticity and mini-batches
- Overfitting and underfitting
- Optimizers and learning-rate
- Hyper-parameter tuning

Optimizers and learning-rate

Several optimizers are available (a nice summary available here)

Adam optimizer: Adaptive moment estimation

Optimizers and learning-rate

Several optimizers are available (a nice summary available here)

Adam optimizer: Adaptive moment estimation

Estimate the first and second moments of gradient g_t

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t, \qquad v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$

Optimizers and learning-rate

Several optimizers are available (a nice summary available here)

Adam optimizer: Adaptive moment estimation

Estimate the first and second moments of gradient g_t

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t, \qquad v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$

Apply bias-correction

$$\hat{m}_t = \frac{m_t}{1-\beta_1}, \qquad \hat{v}_t = \frac{v_t}{1-\beta_2}$$

Several optimizers are available (a nice summary available here)

Adam optimizer: Adaptive moment estimation

Estimate the first and second moments of gradient g_t

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t, \qquad v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$

Apply bias-correction

$$\hat{m}_t = \frac{m_t}{1-\beta_1}, \qquad \hat{v}_t = \frac{v_t}{1-\beta_2}$$

Update the trainable parameter θ

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$$

Several optimizers are available (a nice summary available here)

Adam optimizer: Adaptive moment estimation

Estimate the first and second moments of gradient g_t

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t, \qquad v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$

Apply bias-correction

$$\hat{m}_t = \frac{m_t}{1-\beta_1}, \qquad \hat{v}_t = \frac{v_t}{1-\beta_2}$$

Update the trainable parameter θ

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$$

Recommended values:

$$\beta_1 = 0.9, \quad \beta_2 = 0.999, \quad \epsilon = 10^{-8}$$

Optimizers and learning-rate

Several optimizers are available (a nice summary available here)

Adam optimizer: Adaptive moment estimation

Estimate the first and second moments of gradient g_t

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t, \qquad v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$

Apply bias-correction

$$\hat{m}_t = \frac{m_t}{1-\beta_1}, \qquad \hat{v}_t = \frac{v_t}{1-\beta_2}$$

Update the trainable parameter θ

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$$

Recommended values:

$$\beta_1 = 0.9, \quad \beta_2 = 0.999, \quad \epsilon = 10^{-8}$$

Learning-rate η needs to be chosen. Can be adapted if needed

$$\eta_t = \frac{\eta_0}{1+\gamma t}, \quad \text{or} \quad \eta_t = 0.9^t \eta_0, \quad \text{or} \quad \ldots$$

Components of MLPs

- Depth Vs width
- Data sets
- Activation functions
- Loss/cost functions
- Initialization
- Stochasticity and mini-batches
- Overfitting and underfitting
- Optimizers and learning-rate
- Hyper-parameter tuning

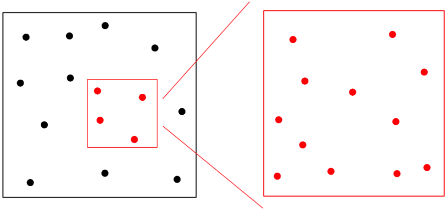
Hyper-parameter tuning: where the most time is spent!

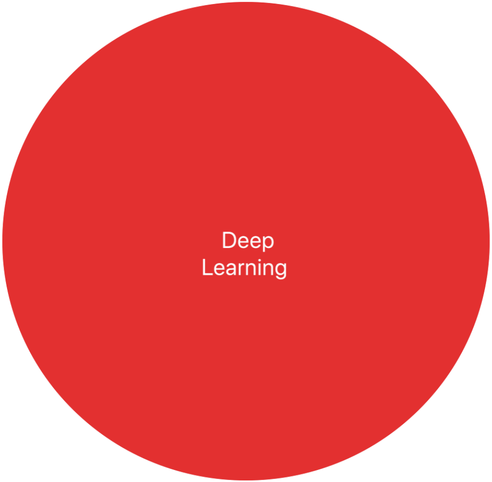
- Width and depth
- Learning-rate
- Choice of regularization and associated parameter
- Activation function and associated parameters
- Mini-batch size
- Number of training epochs
- Number of retrains (with different initialiations)

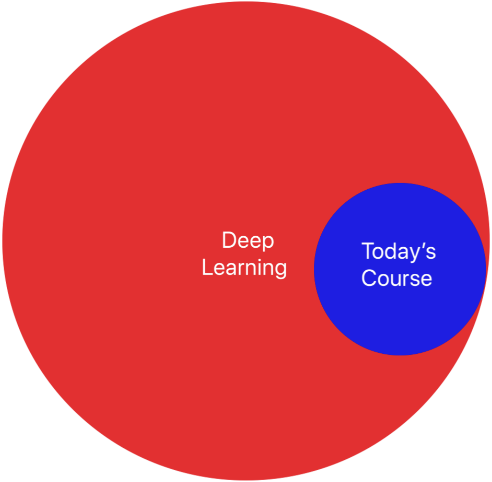
Hyper-parameter tuning: where the most time is spent!

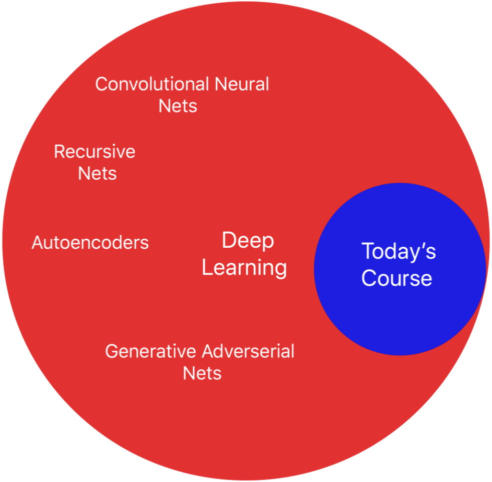
- Width and depth
- Learning-rate
- Choice of regularization and associated parameter
- Activation function and associated parameters
- Mini-batch size
- Number of training epochs
- Number of retrains (with different initialiations)

Several strategies proposed. For instance random coarse to fine search




Deep
Learning


Deep
Learning
Today's
Course


Convolutional Neural Nets
Recursive Nets
Autoencoders
Deep Learning
Today's Course
Generative Adverserial Nets