Deep Learning Basics

Datta Lab presentation @ Princeton University

2023-07-07

Martin Lellep
martin.lellep@gmail.com

Purpose of this talk

Teach you the very basic idea behind
Deep Learning (DL)

Course materials are available!

http://lellep.xyz/blog/datta-lab-ml-course.html

Questions

1) Who has **heard** of neural networks?
2) Who has **used** neural networks in some way?

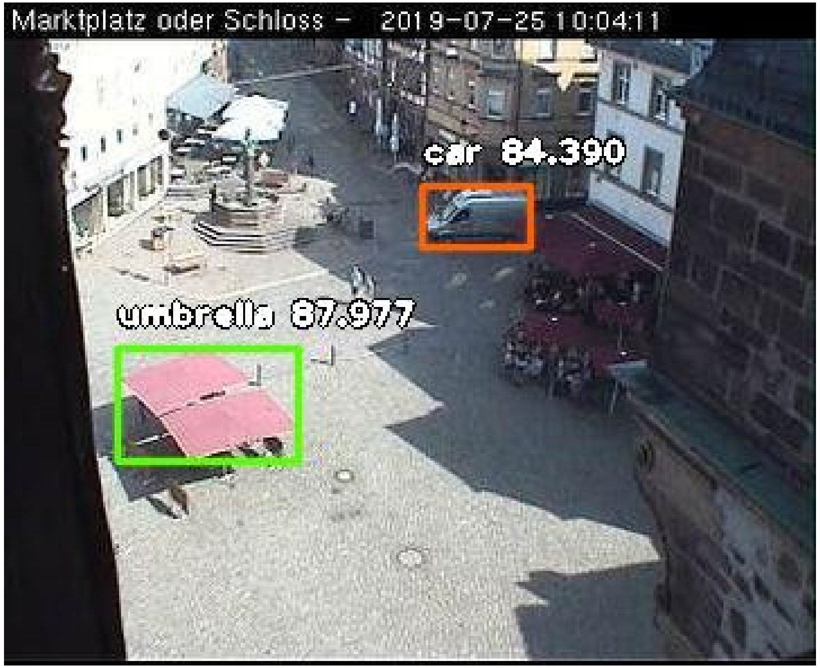
Why are Neural Networks cool?

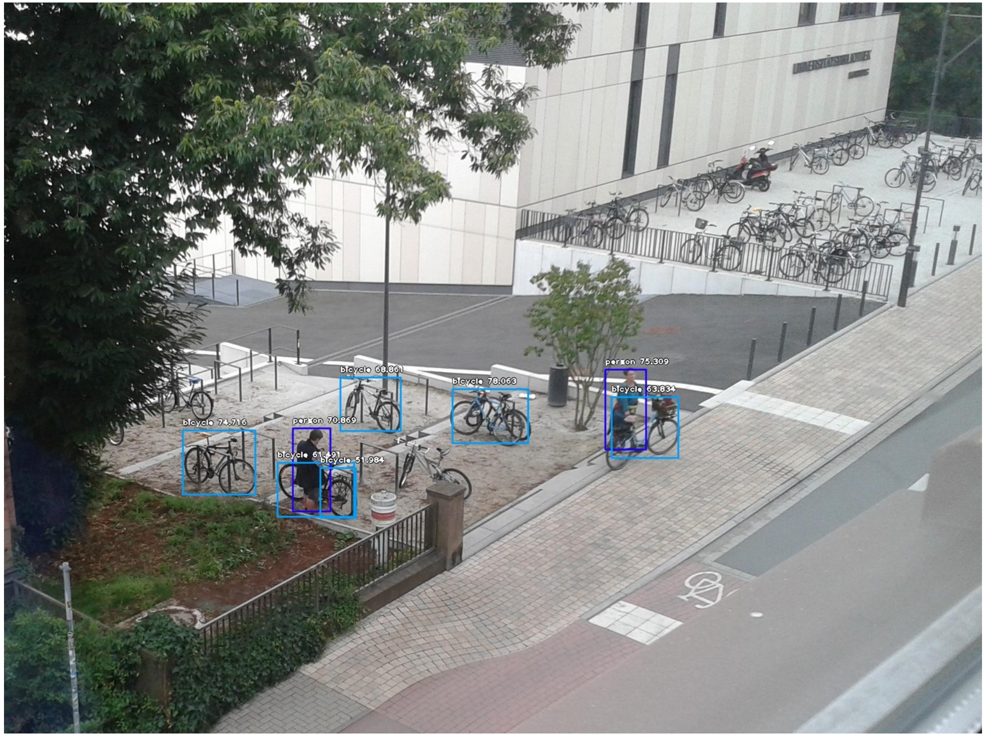
- Ever used translator? → Neural network
- Ever Amazon suggestions? → Neural network
- Ever used subtitles on YouTube? → Neural network

Conclusion: Very powerful tools!

➡ They must be useful for Physics, too!

Why are Neural Networks cool?





https://www.marburg.de/portal/seiten/webcam-blick-aufs-landgrafenschloss-und-den-marktplatz-900001035-23001.html

Object detection applied to view on Marburg (=city in Germany

Agenda

- Introduction
- Neural network (NN) structure
- Stochastic gradient descent (SGD)
- Back propagation (backprop)
- Frameworks
- Deep Learning (DL) examples

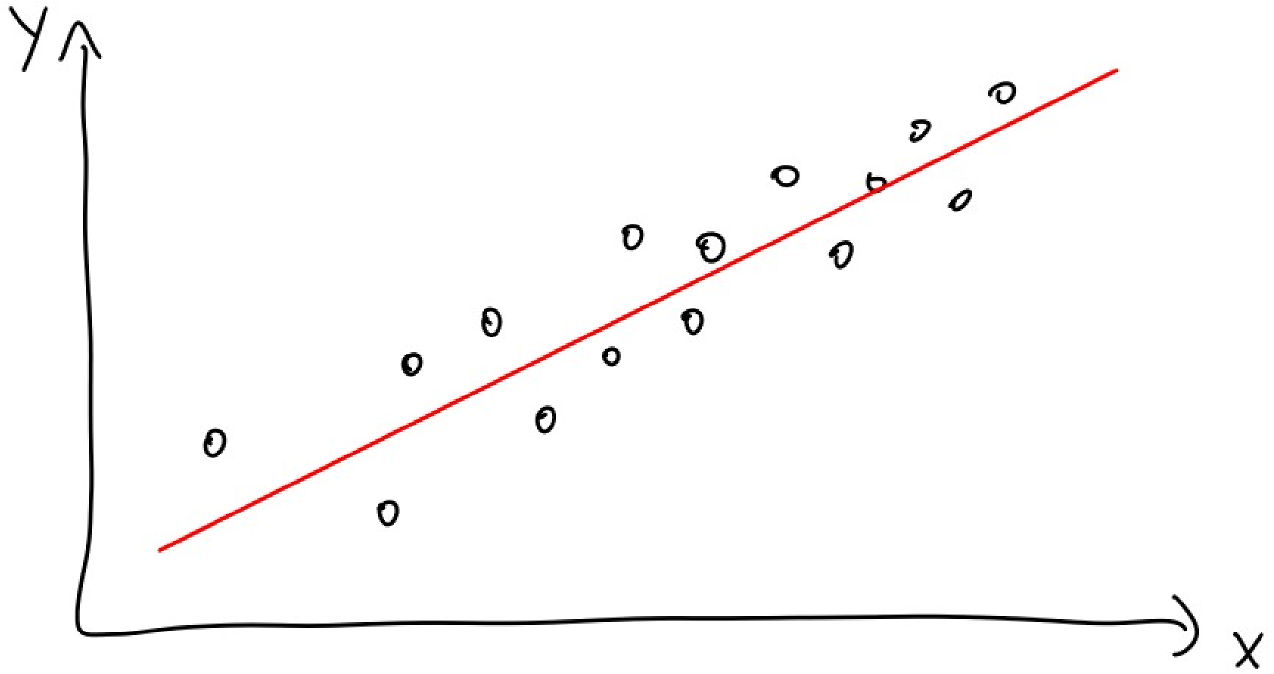
This scratches only the surface!

Introduction

- Examples of simple function approximators:
  - Linear function: $f(x) = ax + b$
  - Quadratic function: $f(x) = ax^2 + bx + c$
- All of them:
  - Parameterised by a set of parameters
  - Can be used to fit given data, $\{(x, y)_i\}_i$

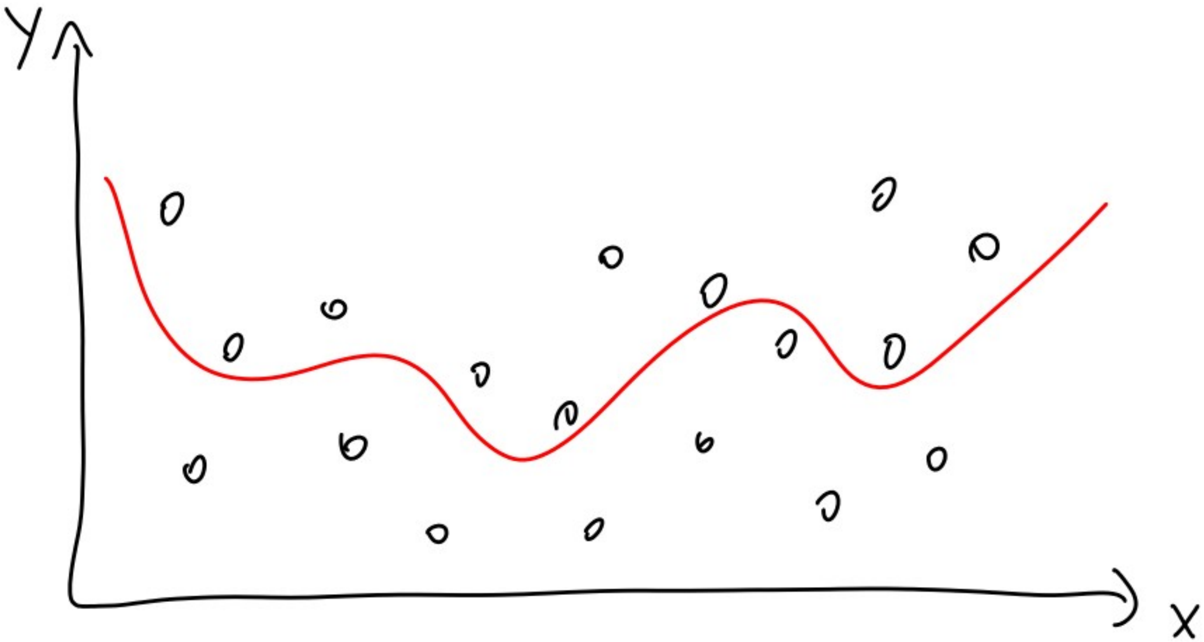
Introduction

Example of fitting a linear function:



Introduction

Example of fitting a non-linear function:



Introduction

[Live demo – Fitting function]

Introduction

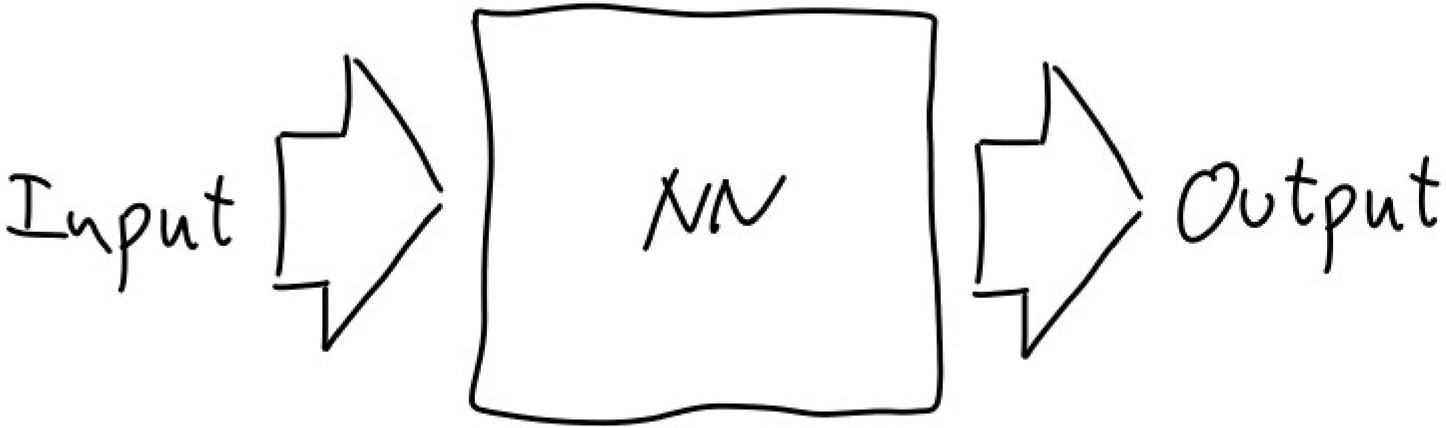
- Steps for fitting a function
  - Data to fit
  - A function to fit – called **model** for here
  - An optimisation procedure to perform the fitting
  - A cost function that is minimised
- Question for this course: How to choose model?
- Solution: Use NNs and DL!

NN structure

- A NN is a function approximator
  - … a complicated function approximator
  - … a parameterised function approximator
  - … a non-linear function approximator
- In fact, so complicated that considered a black box for the most part

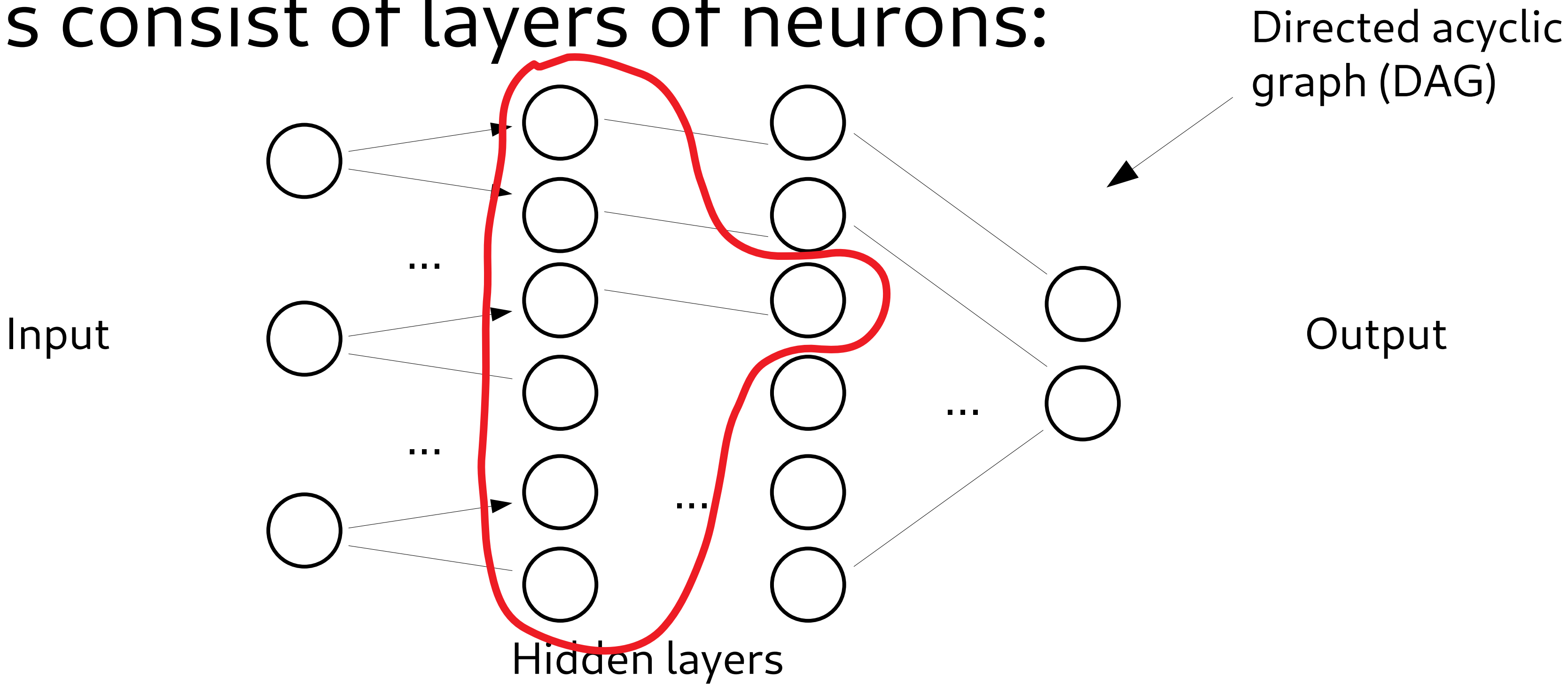
NN structure

Black box character of NNs:



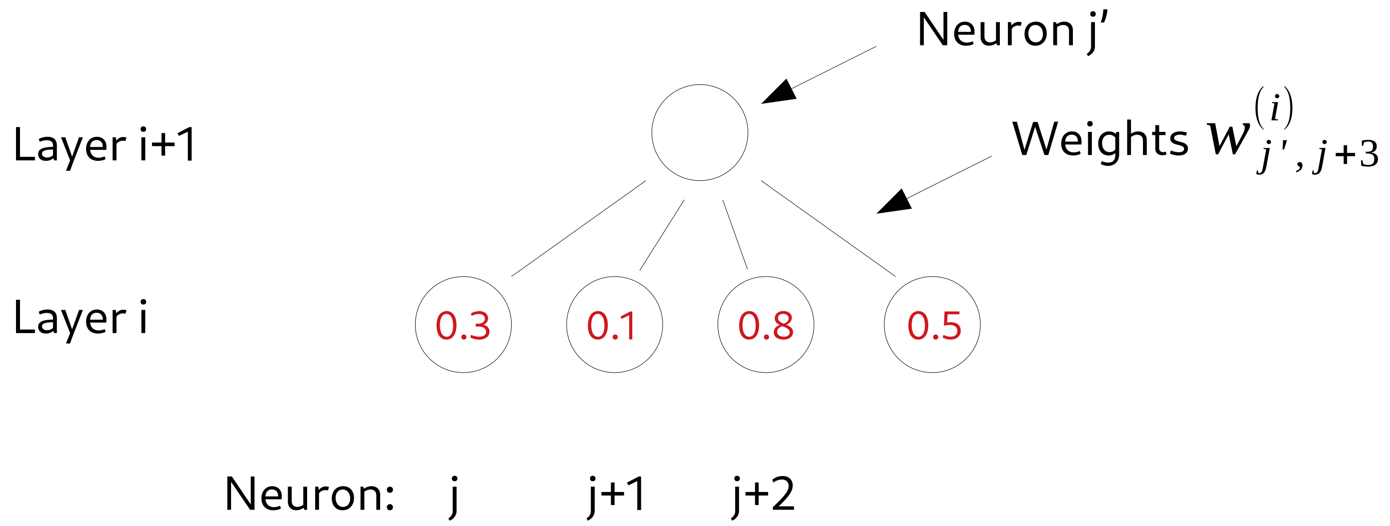
NN structure

- NN structure inspired by neurons in brain
- NNs consist of layers of neurons:



NN structure

Focus on one single layer:



NN structure

(Forward) evaluation of NN:

1) Linear combination of previous layer
2) Activation function applied to step 1)

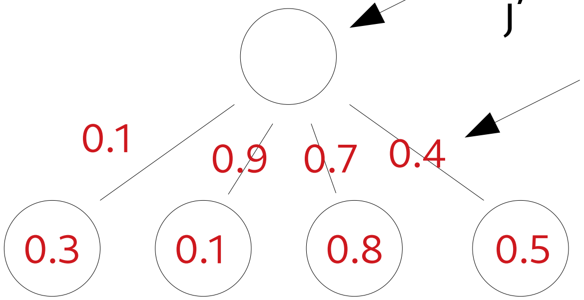
NN structure

Focus on one single layer:

Neuron j'

Weights $w^{(i)}_{j',j+3}$

Layer i+1



Layer i

z = 0.1 x 0.3 + 0.9 x 0.1
+ 0.7 x 0.8 + 0.4 x 0.5 = 0.88

Neuron: j j+1 j+2 j+3

Weighted sum

Linear combination: $z^{(i+1)}_{j'} = w^{(i)}_{j',j} y^{(i)}_{j} + w^{(i)}_{j',j+1} y^{(i)}_{j+1} + w^{(i)}_{j',j+2} y^{(i)}_{j+2} + w^{(i)}_{j',j+3} y^{(i)}_{j+3}$

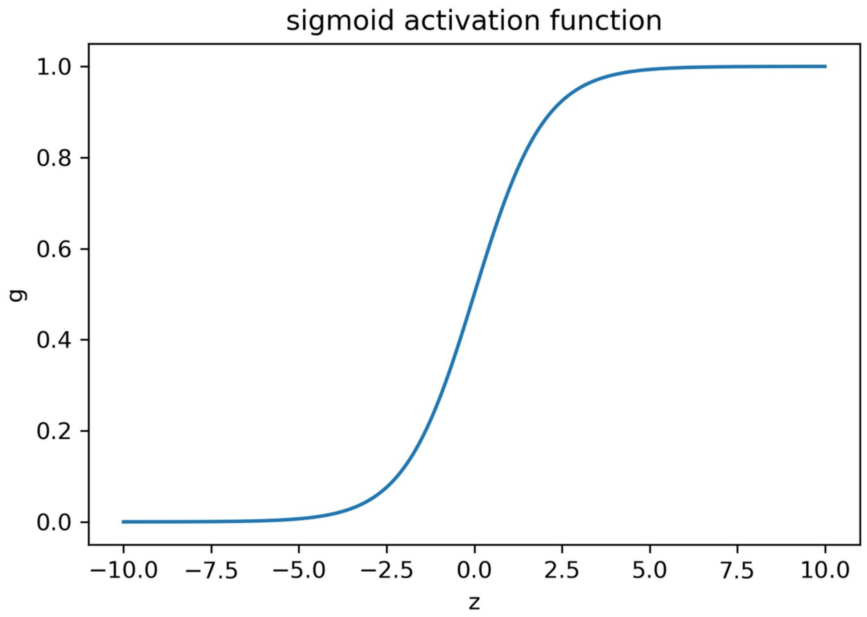
Non-linear function ("activation"): $y^{(i+1)}_{j'} = g\left(y^{(i+1)}_{j'}\right)$

Note: Almost all linear operations can be cast into linear algebra operations => very fast!

NN structure

Activation functions:

- Anything close to a step function
- Fast to compute
- Easy to differentiate



NN structure

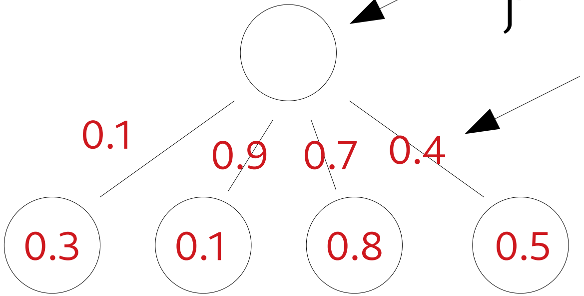
Focus on one single layer:

Neuron j'

Weights $w^{(i)}_{j',j+3}$

Layer i+1

Layer i



z = 0.1 x 0.3 + 0.9 x 0.1
+ 0.7 x 0.8 + 0.4 x 0.5 = 0.88

Neuron: j j+1 j+2 j+3

Linear combination: $z^{(i+1)}_{j'} = w^{(i)}_{j',j}\, y^{(i)}_{j} + w^{(i)}_{j',j+1}\, y^{(i)}_{j+1} + w^{(i)}_{j',j+2}\, y^{(i)}_{j+2} + w^{(i)}_{j',j+3}\, y^{(i)}_{j+3}$

Non-linear function (“activation”): $y^{(i+1)}_{j'} = g\left(y^{(i+1)}_{j'}\right)$

y = sigma(0.88) = 0.71

NN structure

remember: linear vs quadratic

Intermediate summary:

- NNs are built from layers — Model
- Weights connect adjacent layers — Parameters
- Information flows from input to output

Remaining:

- Cost function:

also called "loss"

True output value of sample *i*

NN prediction for output value of sample *i*

$$L = \frac{1}{2} \sum_{i=1}^{N} \left(y_i - y_{NN}(x_i; \theta) \right)^2$$

Sum over training samples

Values of all weight matrices

NN structure

Summary:

- We know how to evaluate a given feed forward fully connected NN

Now:

- Fit NN to regression task

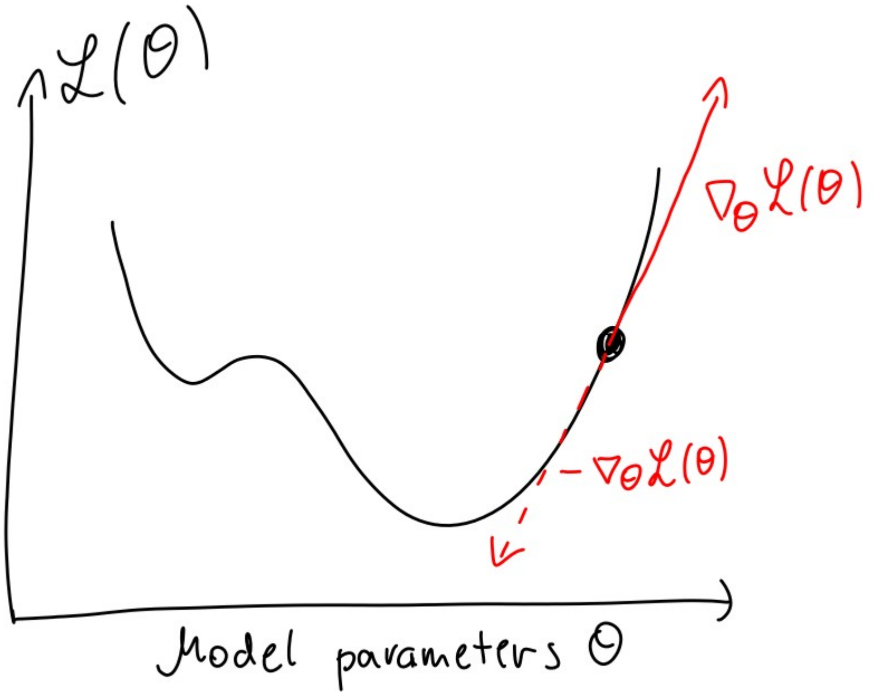
Called “training” in analogy to human learning

Classical optimisation task

Stochastic Gradient Decent (SGD)

- Optimisation procedure to minimise cost function L
- Typical first-order optimisation routine: gradient descent
  - Follow negative gradient
  - Update: $\theta' = \theta - \epsilon \nabla_\theta L(\theta)$

Step size or “learning rate”



SGD

- Key problem: $\theta' = \theta - \epsilon \boxed{\nabla_\theta L(\theta)}$
- Why?

Veeeeeeery large! ~O(1e5)

$$L = \frac{1}{2} \sum_{i=1}^{\boxed{N}} \left(y_i - y_{NN}(x_i; \theta) \right)^2$$

- Solution:

batch size

  - Use m << N randomly chosen samples

$$L = \frac{1}{2} \sum_{i=1}^{m} \left(y_i - y_{NN}(x_i; \theta) \right)^2$$

SGD

Then:

$$\nabla_\theta L = \frac{1}{2} \sum_{i=1}^{m} \nabla_\theta \left(y_i - y_{NN}(x_i; \theta) \right)^2$$

$$= -\sum_{i=1}^{m} \left(y_i - y_{NN}(x_i; \theta) \right) \boxed{\nabla_\theta y_{NN}(x_i; \theta)}$$

The neural network structure is known.

Hence: It is possible to compute it!

SGD

Then:

$$\nabla_\theta L = \frac{1}{2}\sum_{i=1}^{m} \nabla_\theta \left(y_i - y_{NN}(x_i;\theta)\right)^2$$

$$= -\sum_{i=1}^{m} \left(y_i - y_{NN}(x_i;\theta)\right) \boxed{\nabla_\theta y_{NN}(x_i;\theta)}$$

The neural network structure is known.

Hence: It is possible to compute it!

Stunning side note: SGD helps with avoiding to get stuck at saddle points or (weak) local minima.

"Backprop" to the rescue!

Backprop

- “Backprop” = back propagation
- Very rough concept:
  - Forward pass: Feed data into the network
  - Backward pass: Propagate errors backwards to adapt weights
- Remember for now:
  - Backprop makes computing the derivatives easy
  - It is **much simpler** than computing the gradient naively

Frameworks

- Nobody uses plain Python implementations
- Nobody who does NN and DL in 2023 at least
- Solution: e.g. Tensorflow and Keras!

Deals with tensors and differentiation

Deals with NNs

Frameworks: Keras

- High-level Python library
- Useful for:
  - Assembly of NN
  - Training of NN
  - Evaluation of NN (training vs validation vs test data)
  - And some others

Frameworks: Keras

- NNs, as presented before, are modeled as layers
- The user has to provide data
- Data:
  - Training data
  - Validation data
  - Test data

Used for training

Used for design optimisations of NN using **unseen** data

Test of how well the NN performs on **unseen** data

Frameworks: Keras

[Live demo – Learn 2D function]

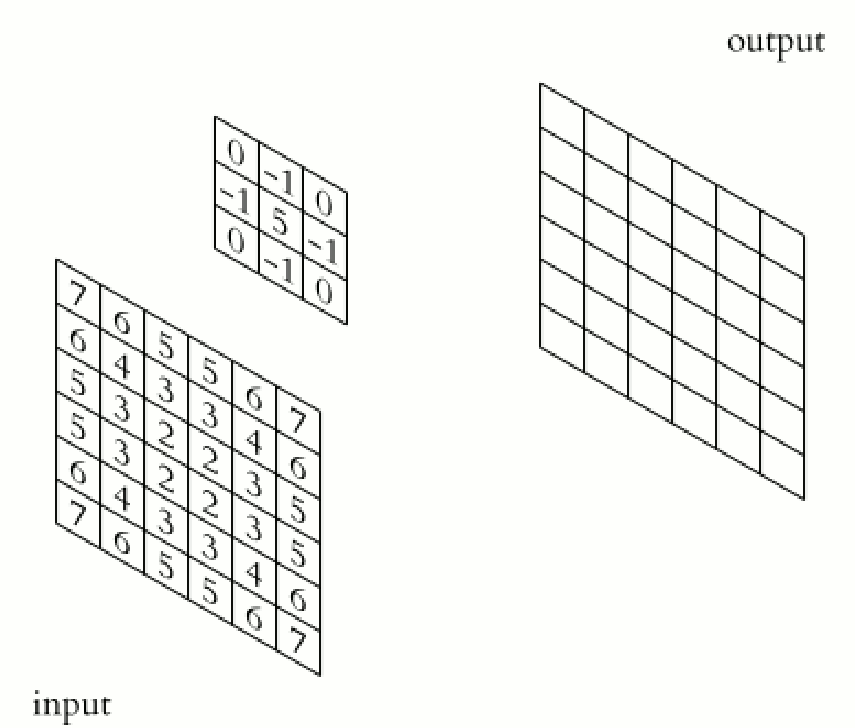
Deep learning (DL)

- DL = Use NNs that have many layers
- Previously, we discussed “feed forward” layers (called “Dense” in Keras)
- But: There are many more types of layers
  - For images (ConvLayers)
  - For time series (LSTMs)

In fact: You are only limited by your imagination when designing layers (if they perform well is, of course, a different question)

Convolutional NNs (CNNs)

- A layer type that learns convolution kernels
- Typically used for images
- Advantage:
  - Less weights than Dense
  - Good for translationally invariant features

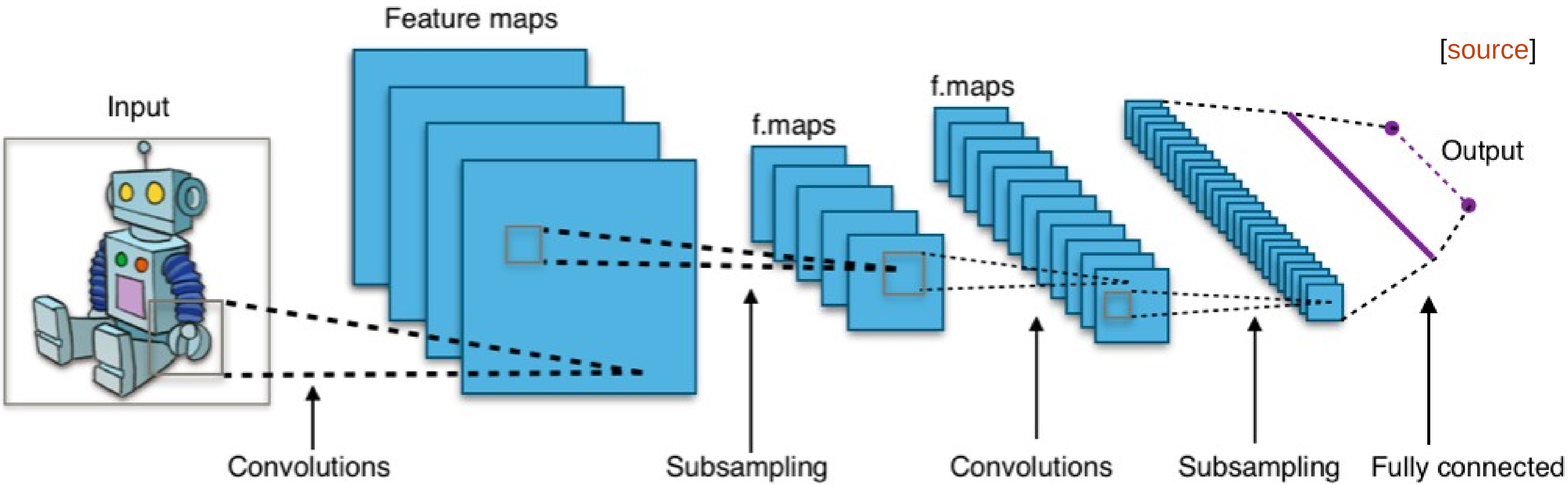


[source]

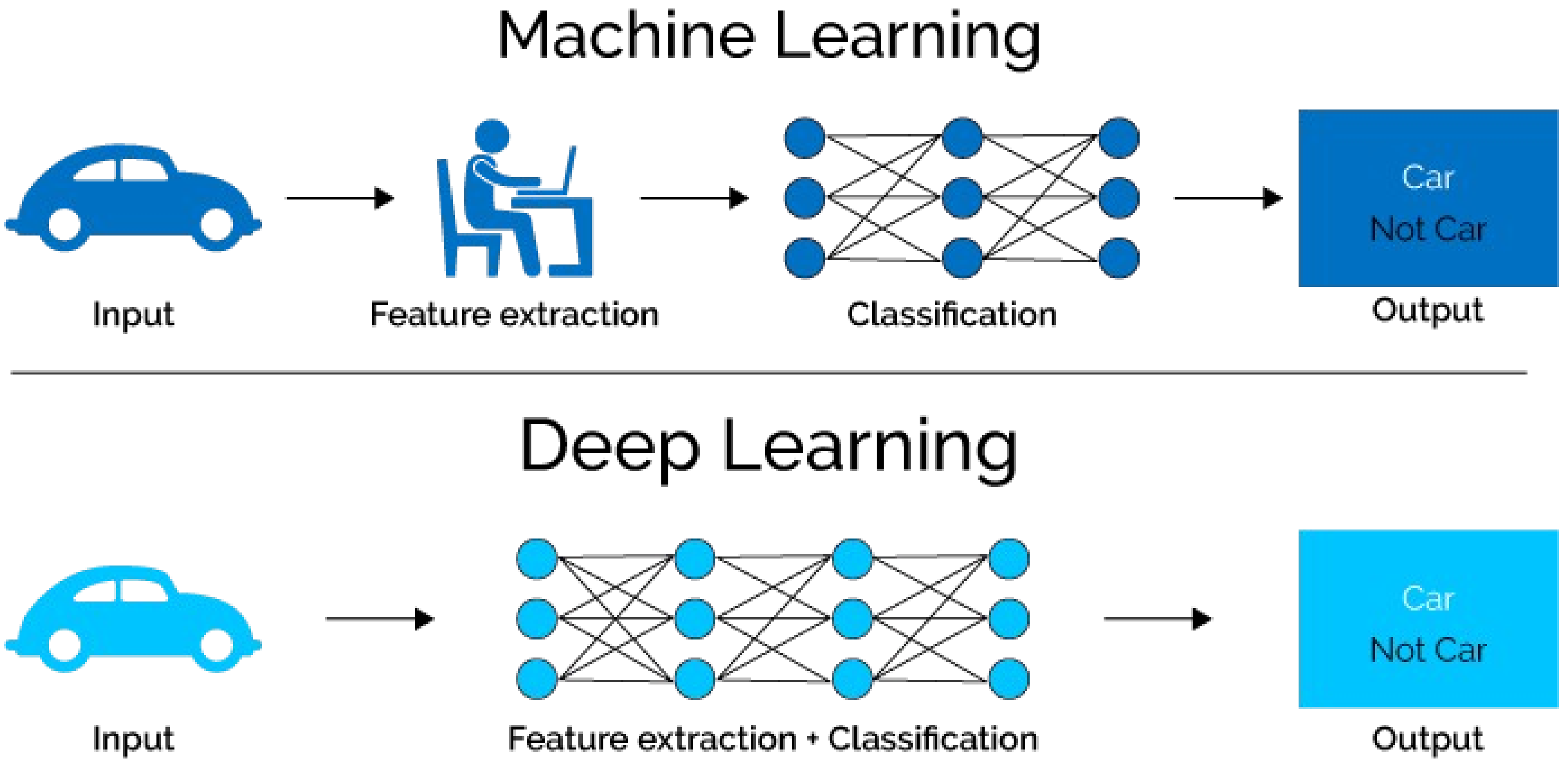
Another nice visualisation!

CNNs

[source]


Feature maps
Input
f.maps
f.maps
Output
Convolutions
Subsampling
Convolutions
Subsampling
Fully connected

DL vs ML

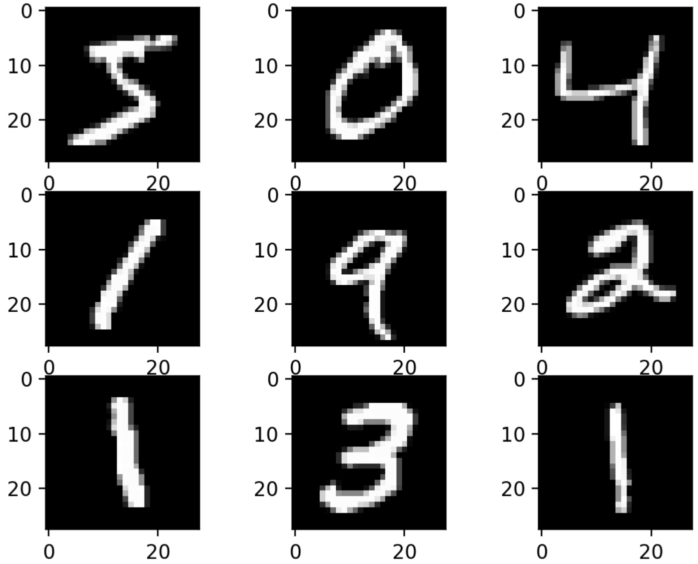


→ **DL learns the features by itself!**

MNIST dataset

- A dataset to test machine learning model performances
- Widely used in DL to benchmark how well a NN performs
- Content: Images of handwritten digits

- Exampes:



Advanced example

- We train a CNN on MNIST data
- We use different layer types
- We use other best practises
- We use a larger model!
  … and see some drawbacks of NNs

They are computationally expensive to train

Frameworks: Keras

[Live demo – MNIST with CNN]

Colab

- A platform provided by Google
- I presented the examples using **Jupyter Notebooks**
  - "Python in the cloud"
- Colab uses Jupyter Notebooks that run on Google infrastructure
- Advantage: They provide free GPU access whi

http://colab.research.google.com

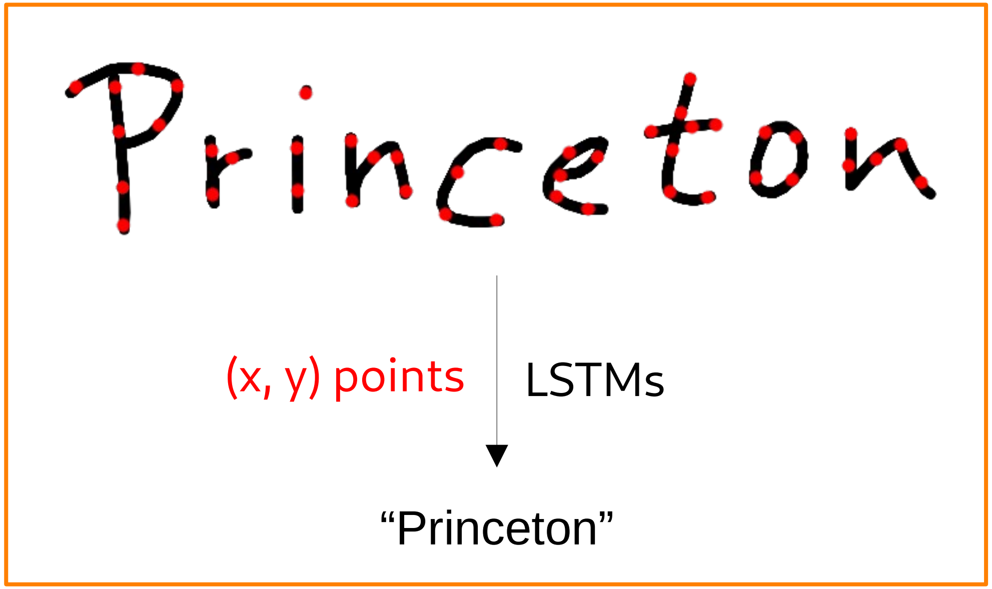
Outlook

- There is much more to DL
- Examples:
  - LSTM layers
  - Autoencoder
  - GANs
  - VAEs
  - Information theory (to understand more details)

Re LSTMs: my weekend project

Small Sunday-only hobby project of mine:

- Given pen dynamics, predict written text
- Uses LSTMs
- Impact: O(10,000) users of open source application



Project link & demo video:
https://github.com/PellelNitram/xournalpp_htr

Sources

- Backprop:
  - https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/
- CNNs: https://de.wikipedia.org/wiki/Convolutional_Neural_Network
- CNN & MNIST example:
  - https://keras.io/examples/mnist_cnn/
  - Colab: https://colab.research.google.com

Get in contact

E-Mail: martin.lellep@gmail.com

Tech Blog:
http://lellep.xyz/blog

LinkedIn:
https://www.linkedin.com/in/martin-lellep-858600152/

Questions

?

Feel free to ask questions!

Next session

Neural networks
& deep learning