Deep Neural Networks Vision

hands on session

Alexander Boettcher

Topics

- Introduction
- DNN
- CNN

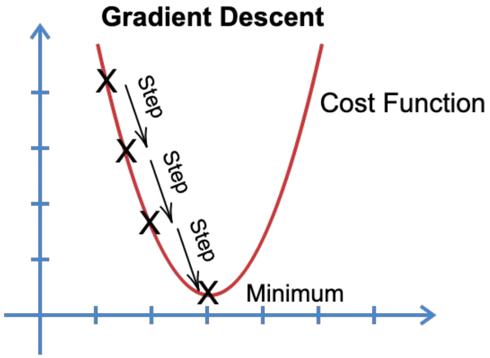
Introduction - Deep Learning preconditions

- Theory
  Already developed staring in the 1970s
- Data
  Available since digital cameras and the internet spread
- Processors
  Parallelization in GPUs is available
- Frameworks
  Popular frameworks currently Pytorch and Tensorflow

Introduction - Theory

- Mathematical numerical optimization

$$\underset{x}{\text{minimize}} \qquad f(x)$$

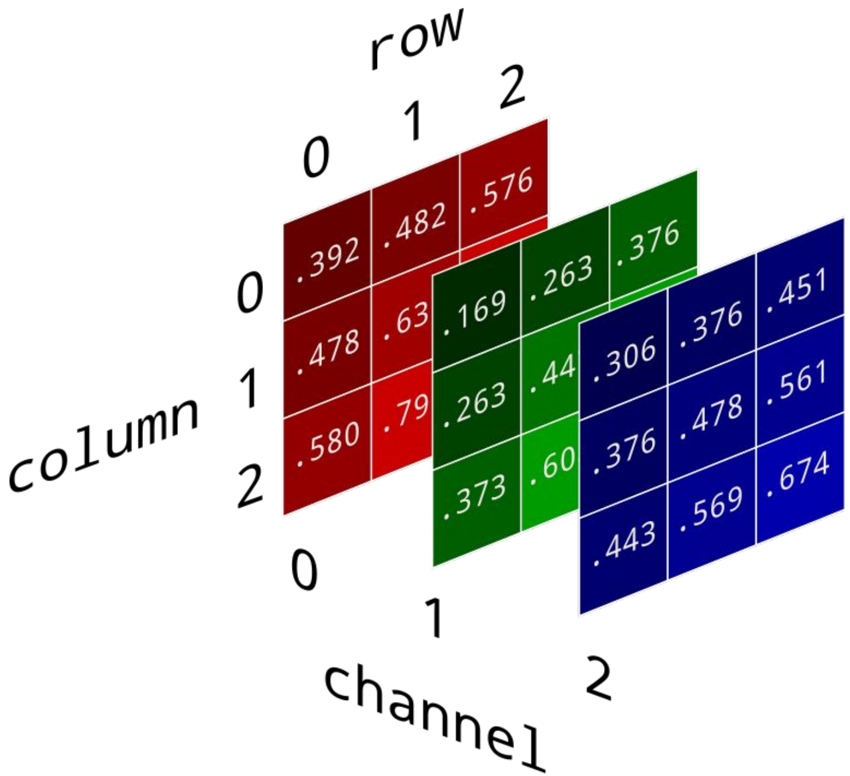


- First order optimization (e.g. gradient descent)
  - In DL it is easy to obtain derivative
    (simple basic functions and chain rule)

- Fast optimization is key
  - Aim: reach the (local) optimum using few evaluations of the function and its derivative

Introduction - Data

- Sets of images
- Stored in matrixes (or tensors)
- Common shape of a data tensor (with four axes):
    - number of images,
    - number of channels,
    - number of columns,
    - number of row



Introduction - Processors

- GPUs (Graphics processing units) are popular in Deep Learning
    - Specialized on linear algebra
        - matrix multiplication
        - Convolutions
    - Many cores for parallelization of matrix operations.
    - Most operations during optimization are matrix operations!

Introduction - Frameworks

- Important features of deep learning frameworks
  - Automatic differentiation
  - Set of popular optimizers
  - Popular (pretrained) networks

Introduction - Summary

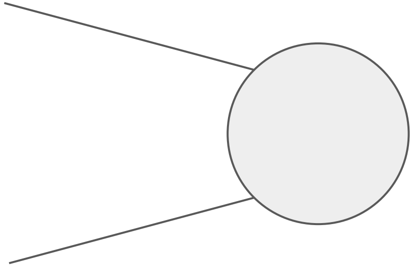
Takeaways:

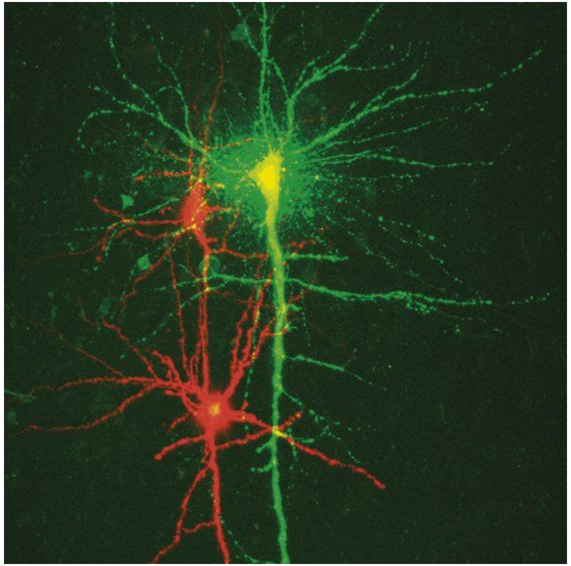
- Deep learning is based on **mathematical numerical optimization**
- The workhorse of deep learning are **optimizers** (e.g. gradient descent)
- Deep learning frameworks calculate the gradient using **automatic differentiation**
- Deep learning **libraries** provide everything needed to do all of the above at ease.

Introduction - Hands on

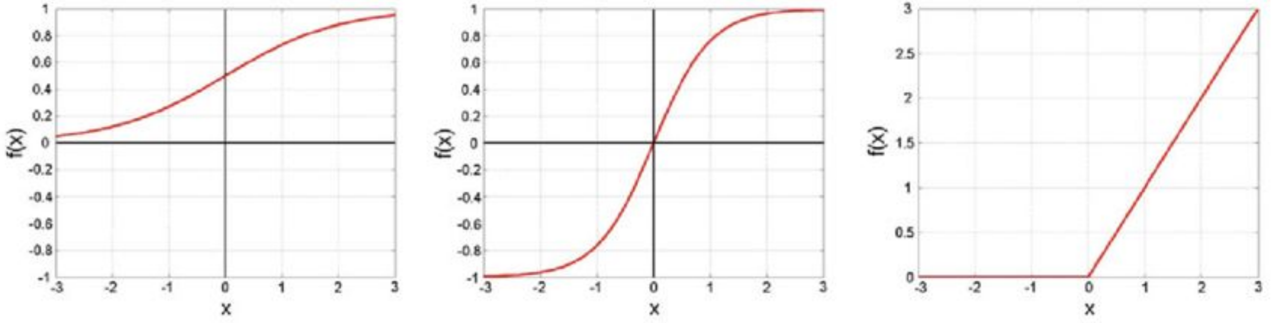
Jupyter Notebook: Session_1_introduction

DNN - Neuron



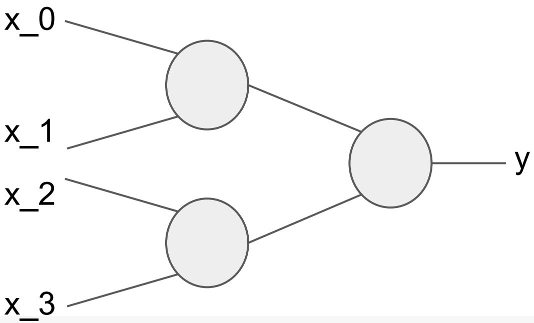


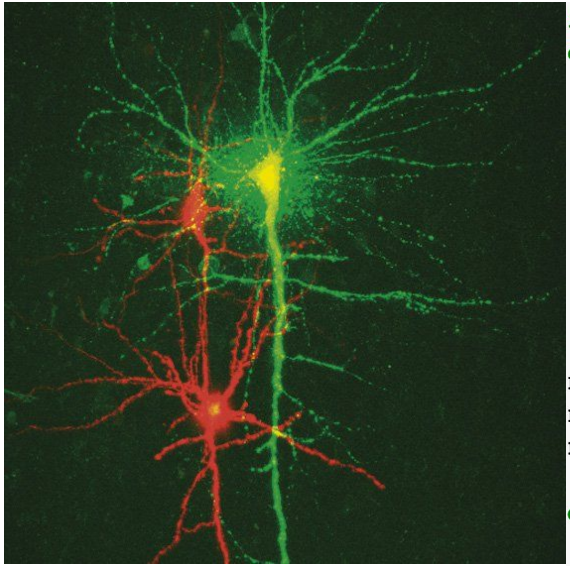
```python
import torch
class Neuron:
    def __init__(self):
        self.w_0 = torch.autograd.Variable(torch.Tensor([1]), requires_grad=True)
        self.w_1 = torch.autograd.Variable(torch.Tensor([1]), requires_grad=True)
        self.bias = torch.autograd.Variable(torch.Tensor([1]), requires_grad=True)

    def f(self, x_0, x_1):
        linear = x_0*self.w_0 + x_1*self.w_1 + bias
        non_linear = np.clip(linear, 0., np.inf)
        return non_linear
n = Neuron()
```



(a) (b) (c)

DNN - Neural Network

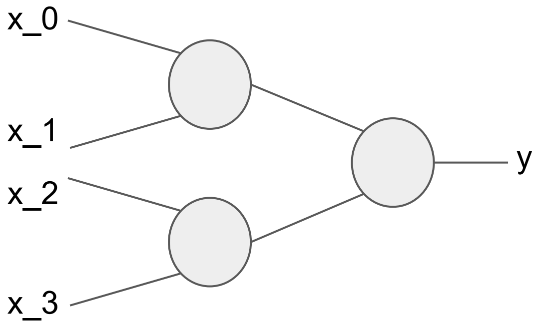




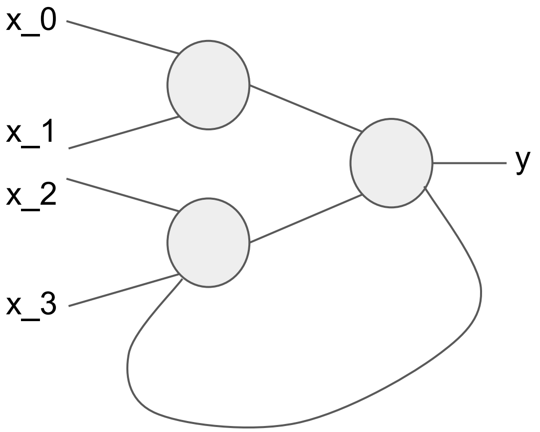
```python
import torch
class Neuron:
    def __init__(self):
        self.w_0 = torch.autograd.Variable(torch.Tensor([1]), requires_grad=True)
        self.w_1 = torch.autograd.Variable(torch.Tensor([1]), requires_grad=True)
        self.bias = torch.autograd.Variable(torch.Tensor([1]), requires_grad=True)

    def f(self, x_0, x_1):
        linear = x_0*self.w_0 + x_1*self.w_1 + bias
        non_linear = np.clip(linear, 0., np.inf)
        return non_linear
n_0 = Neuron()
n_1 = Neuron()
n_2 = Neuron()

def network(x_0, x_1, x_2, x_3):
    y = n_0(n_1(x_0, x_1), n_2(x_2, x_3))
    return y
```

DNN - Neural Network Architectures

Feed-forward networks
(no loops allowed)



Recurrent networks
(loops allowed)



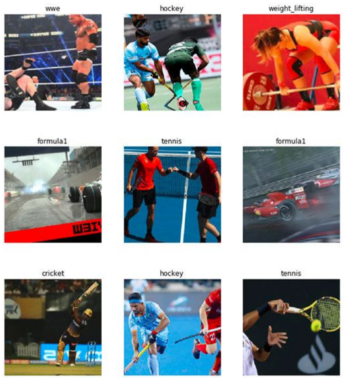
Introduction - Hands on

Jupyter Notebook: Session_2_neural_networks

Tasks

- Classification

  Out of a given set of classes y predict the most likely class



- Regression

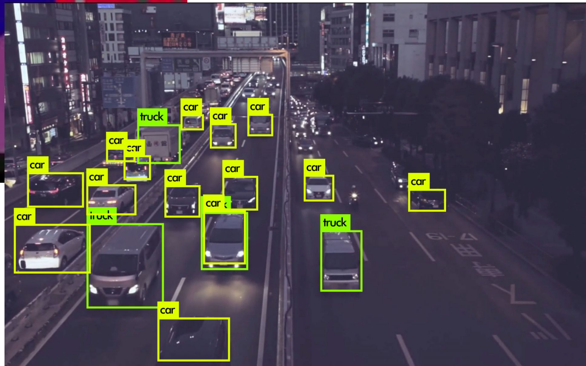
  y predicts a floating point number

- Segmentation

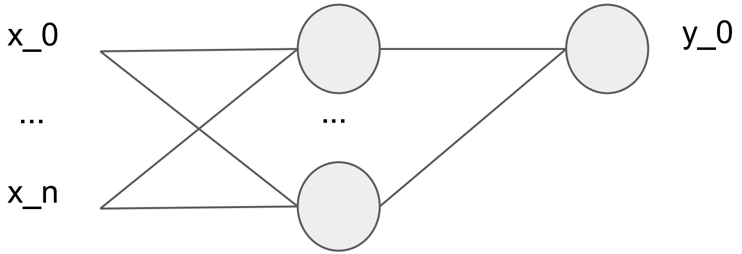
  Out of a given set of classes y predicts the most likely class for each pixel in th input image

- Detection

  Out of a given set of classes y predicts boxes for each instance of the classes in the input image



Classification - Example

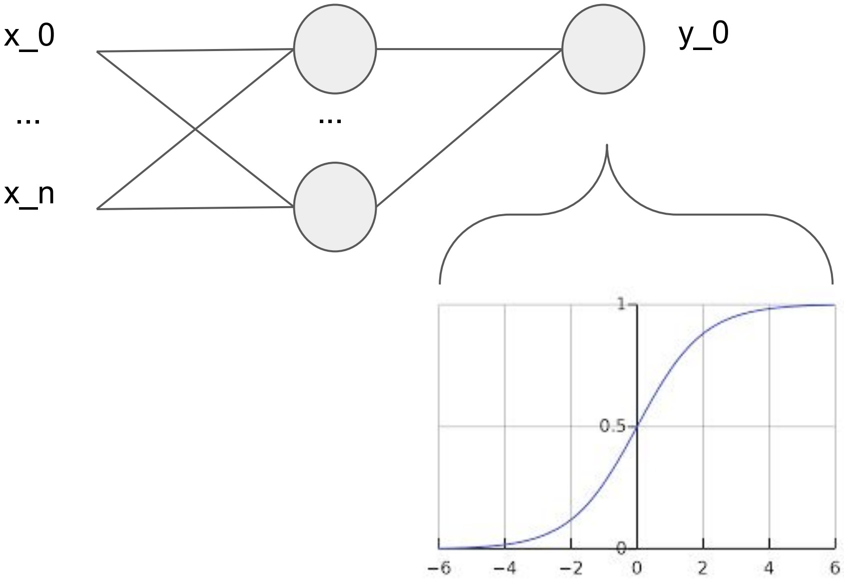


Example: classify if the input image shows a car

- no car: y_0 = 0
- car: y_0 = 1

How to make y_0 to return binary classification results?

Classification - Example

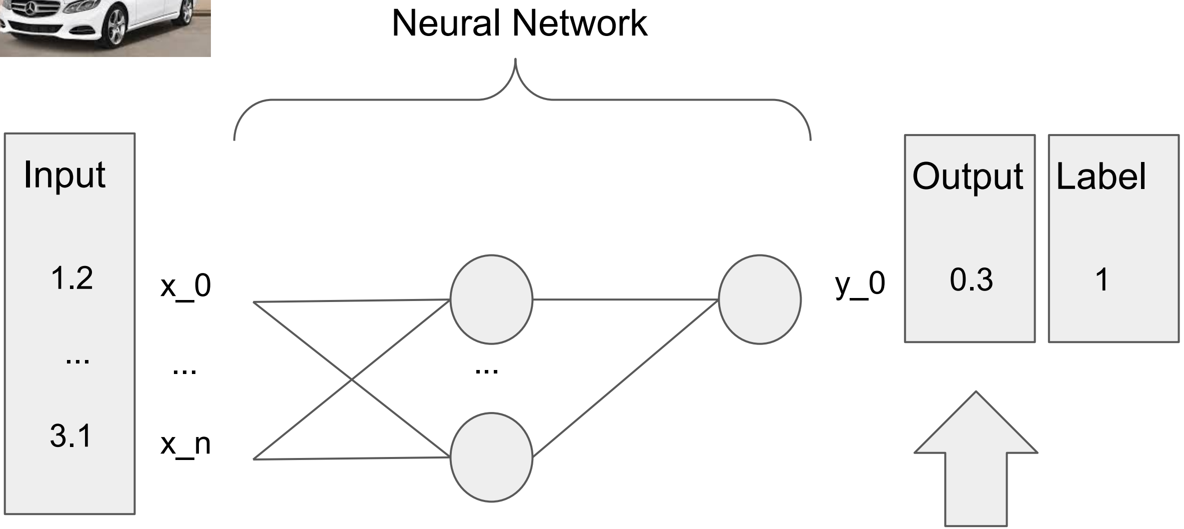


Use a sigmoidal non-linearity in the last layer

Example: classify if the input image shows a car

- no car: y_0 = 0
- car: y_0 = 1

Classification - Example

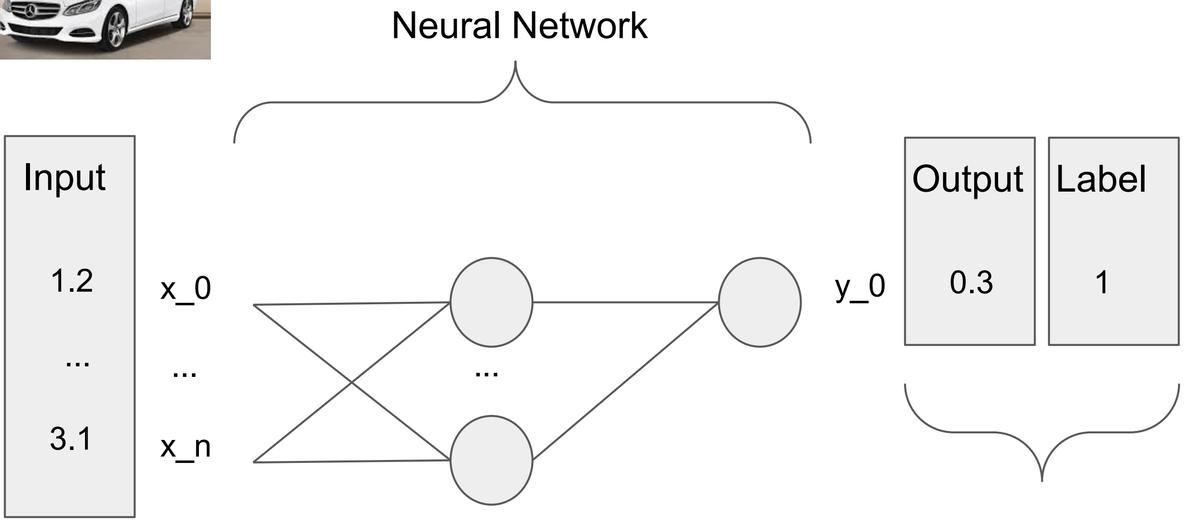


Random output due to random initialization of the network

Example: classify if the input image shows a car

- no car: y_0 = 0
- car: y_0 = 1

Classification - Example



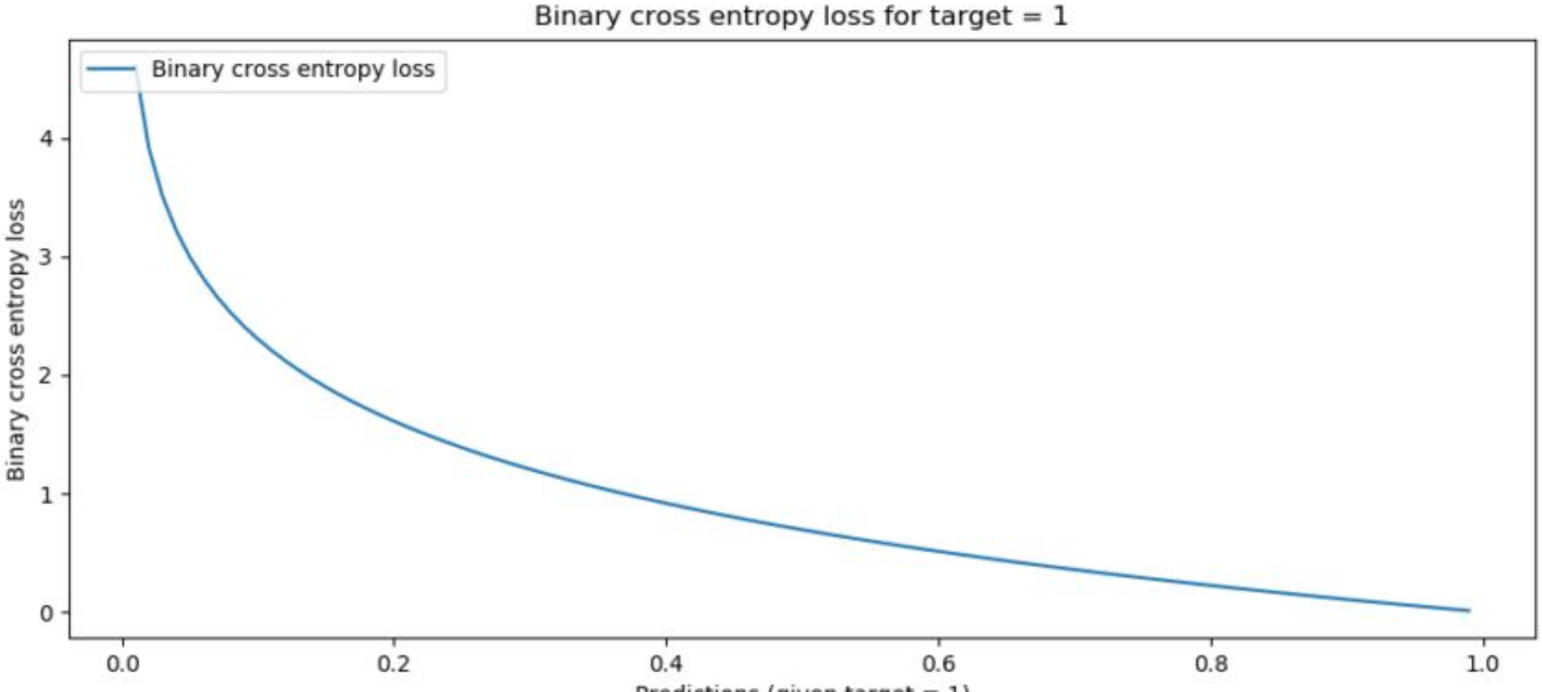
Example: classify if the input image shows a car

- no car: $y_0 = 0$
- car: $y_0 = 1$

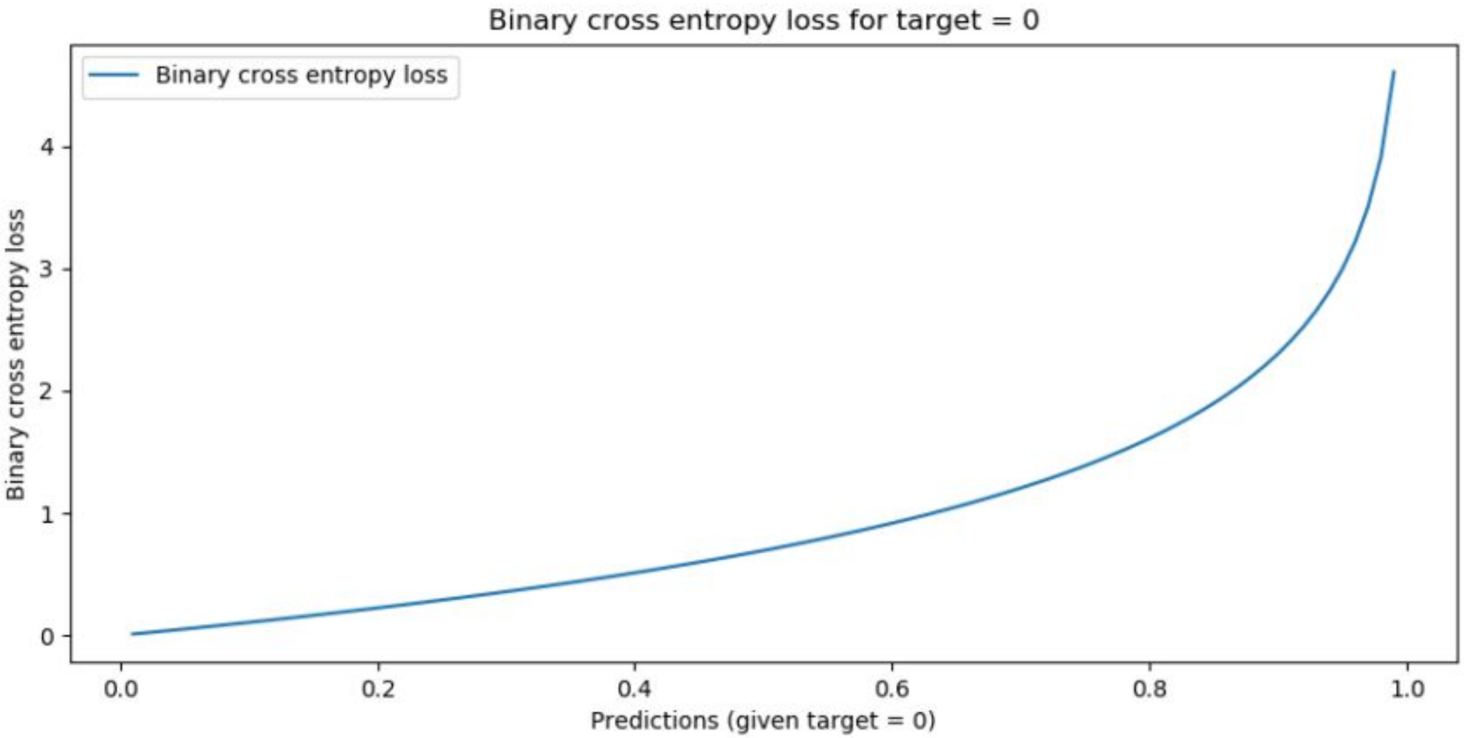
Optimize the difference (loss) between the output and the label

Classification - Cross entropy loss

loss(model(x), label=1)



Classification - cross entropy loss

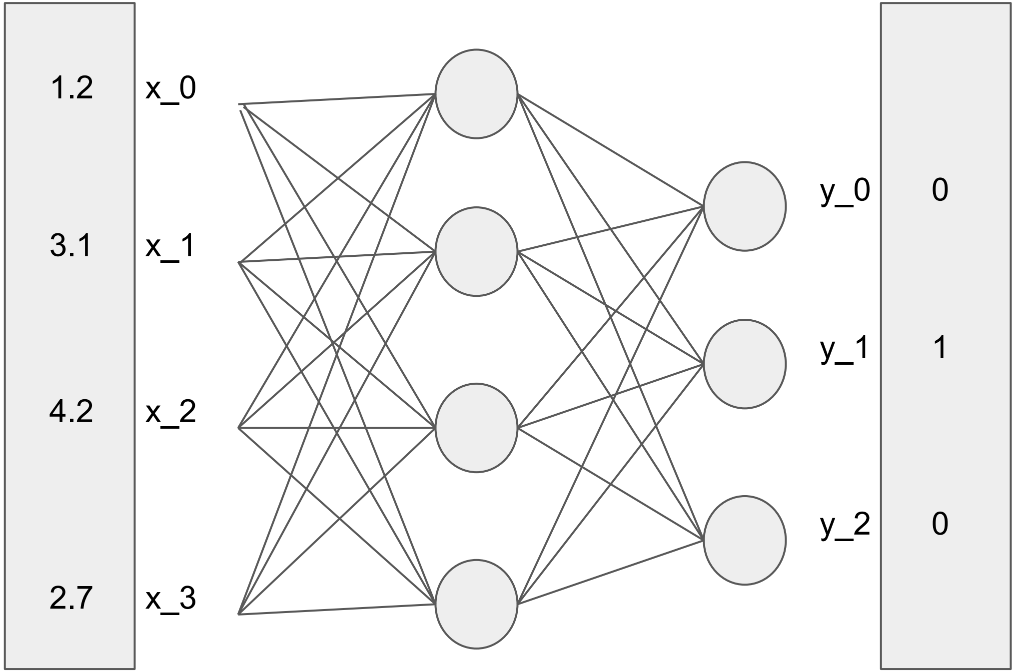


Introduction - Hands on

Jupyter Notebook: Session_3_neural_networks_optimization

Classification - Example

Iris Versicolor Iris Setosa Iris Virginica



Example network with

- 4 inputs
- 3 output classes

Classes (one hot encoding):

- Iris Setosa: [1, 0, 0]
- Iris Versicolour: [0, 1, 0]
- Iris Virginica: [0, 0, 1]

Pytorch

As the examples get more complicated, let’s use Pytorch to specify the model and load the data.

Introduction - Hands on

Jupyter Notebook: Session_4_DNNs

CNNs

Natural images are spatially invariant:

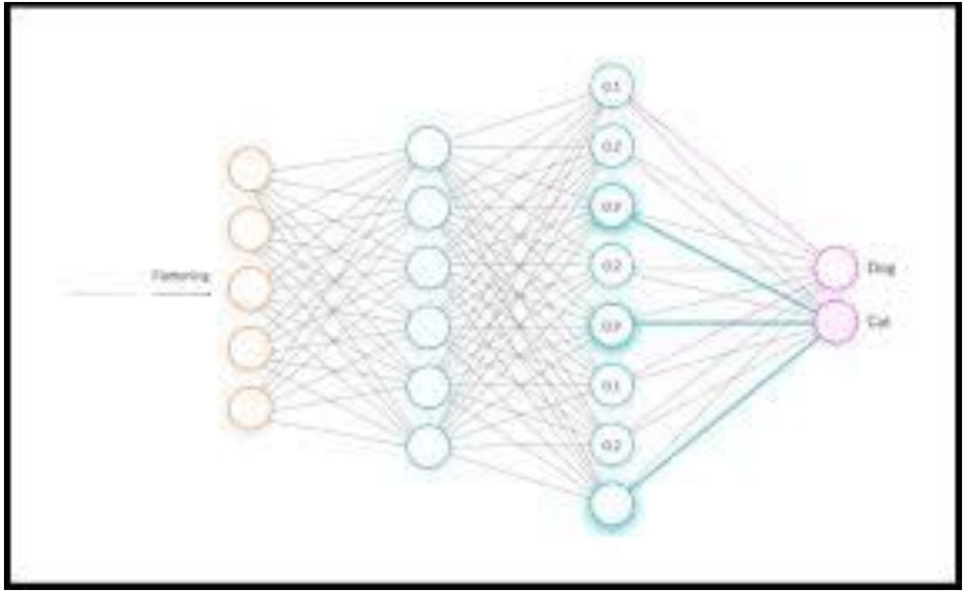
- The statistics is over all the space the same
- A neuron that detects and edge conducts the same task no matter at with spatial location

The correlation of pixel intensities decrease the further away the pixels are:

- Closeby pixels are related e.g. an edge might affect several collocated pixels
- Pixels far apart have almost no relation to each other

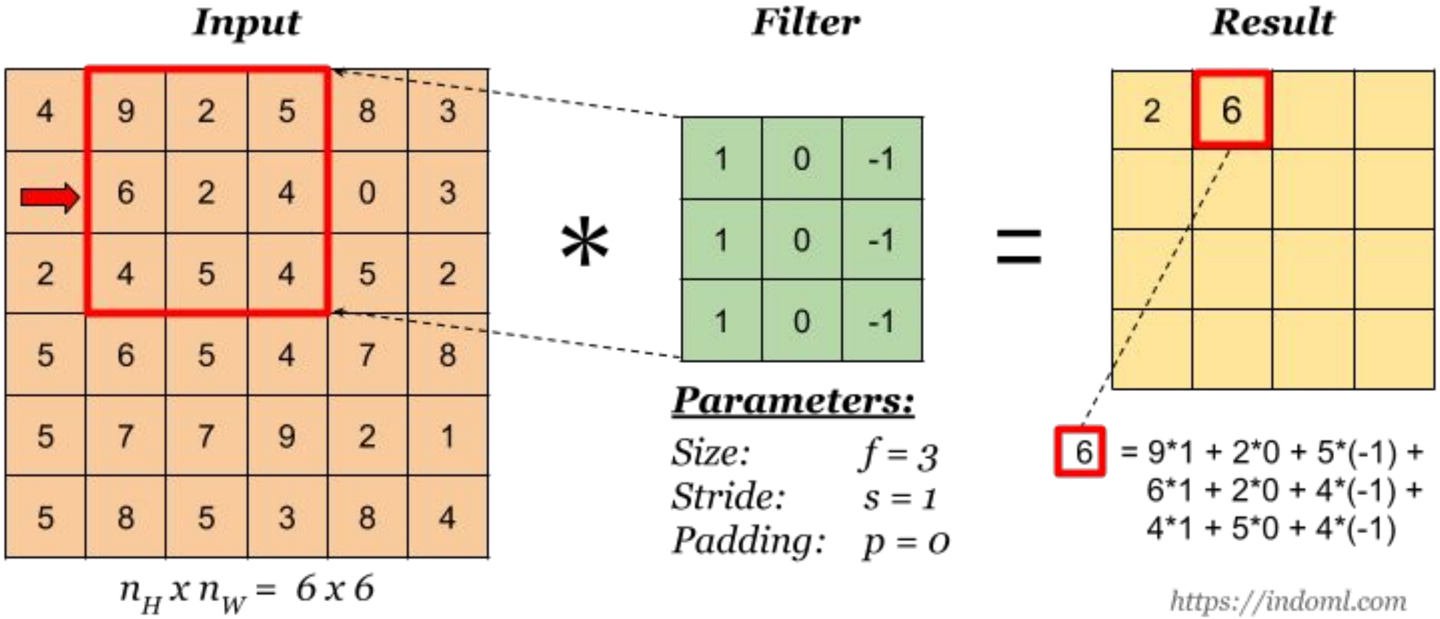
CNNs

Fully connected layers connect all neurons of the lower layer with all layers of the upper layer:



CNNs

Fully connected layers connect all neurons of the lower layer with all layers of the upper layer:



Introduction - Hands on

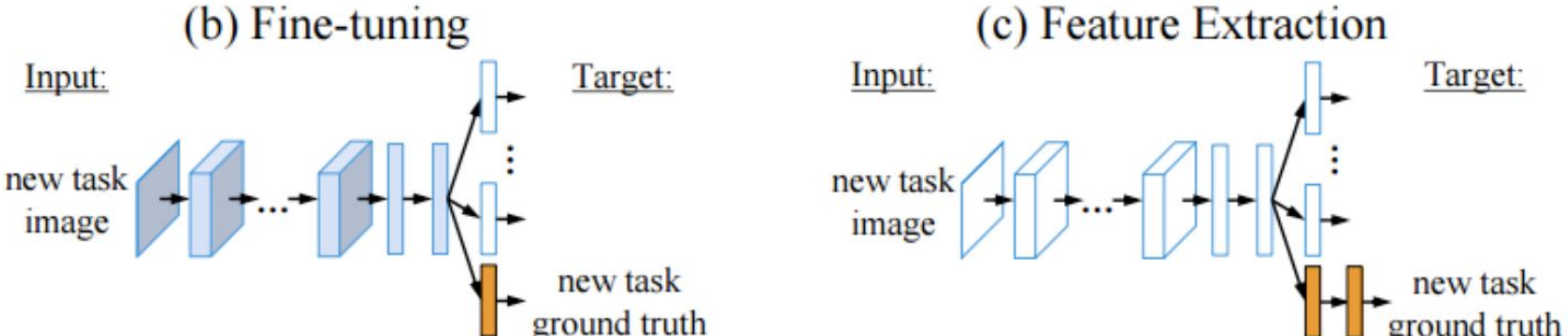
Jupyter Notebook: Session_5_CNNs

Finetuning

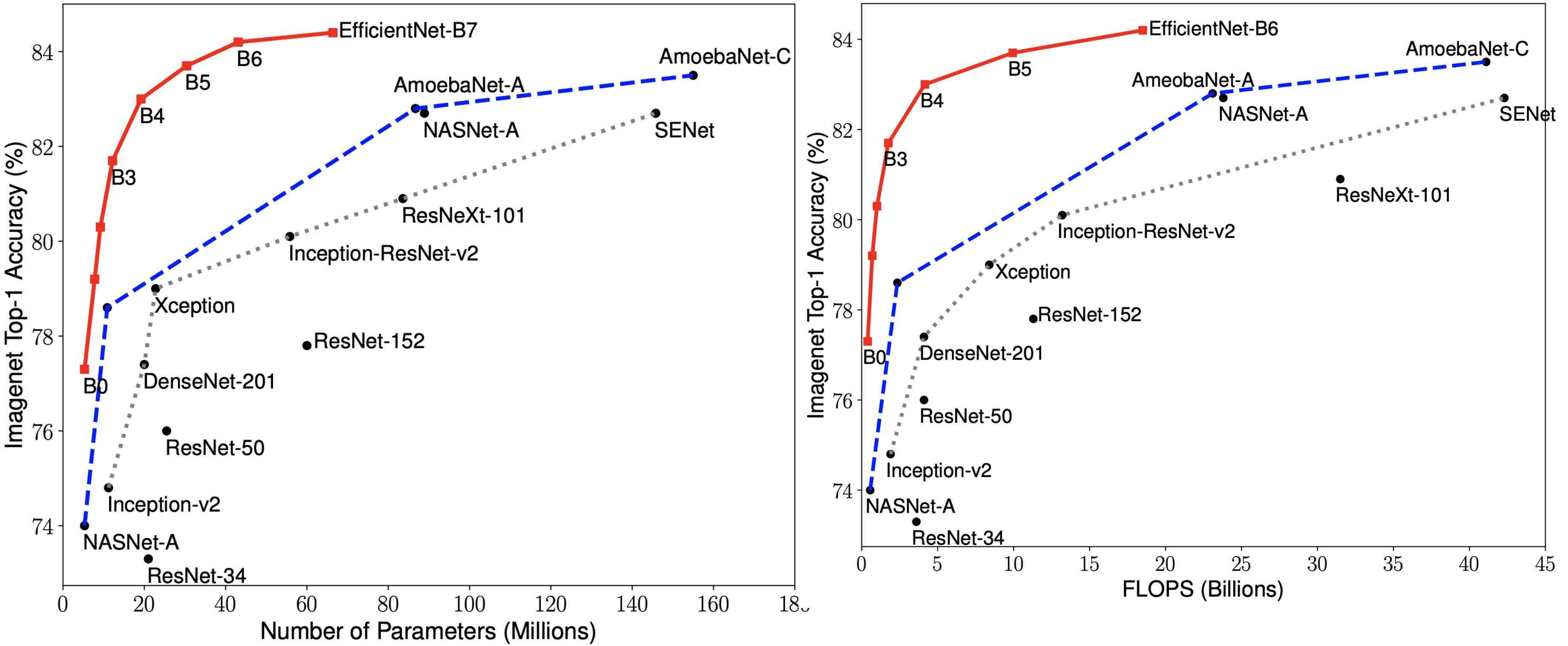
Learning big networks from scratch

- Demands large amount of data
- Takes long time
- Demands fast GPUs

It is easier to reuse a trained network and only retrain a part of it.



Finetuning



EfficientNet-B7
B6
B5
B4
B3
B0
AmoebaNet-C
AmoebaNet-A
NASNet-A
SENet
ResNeXt-101
Inception-ResNet-v2
Xception
ResNet-152
DenseNet-201
ResNet-50
Inception-v2
NASNet-A
ResNet-34
Imagenet Top-1 Accuracy (%)
Number of Parameters (Millions)
EfficientNet-B6
B5
B4
B3
B0
AmoebaNet-C
AmoebaNet-A
NASNet-A
SENet
ResNeXt-101
Inception-ResNet-v2
Xception
ResNet-152
DenseNet-201
ResNet-50
Inception-v2
NASNet-A
ResNet-34
Imagenet Top-1 Accuracy (%)
FLOPS (Billions)

Introduction - Hands on

Jupyter Notebook: Session_4_finetuning