Deep Learning in 60 Minutes

DI Alexander Dür MSc

Content

- Why Deep Learning?
- Definition
- From linear models to Deep Learning
- Ingredients of modern Deep Learning
- How to get started

Why you should care about Deep Learning

- **State-of-the-Art performance for many problems:**
  - Image Classification
  - Translation
  - Speech Synthesis
- **Versatile Tools:**
  - improve scaling of other statistical models (linear models, probabilistic graphical models, etc.)

Goal of this Talk

After this talk you should

- understand the **motivation** behind Deep Learning
- know enough **theory to understand simple Deep Learning models**
- be able to **build** simple Deep Learning **models in Keras**

What is Deep Learning?

- class of machine learning methods that
  - use **layers of non-linear computations**
  - are optimized via **Stochastic Gradient Descent**

→ sounds complicated, why not use linear models?

Why not use linear models?

- Simple example – price houses:
  - 120 m^2 living space
  - 200 m^2 garden
  - 1 garage

Linear model for housing prices:

- Coefficients (cost per unit):
  - coef_living_space = 300
  - coef_garden = 50
  - coef_garage = 20,000
  - bias = 10,000
- Our house (120 m² living space, 200 m² garden, 1 garage):
  - $(120 \quad 200 \quad 1) * \begin{pmatrix} \mathbf{300} \\ \mathbf{50} \\ \mathbf{20,000} \end{pmatrix} + \mathbf{10,000} = \mathbf{€\,76,000}$

How should we choose the coefficients?

- minimize **quadratic loss** of training data T:
  - $\min \sum_{i \in T} (p_i - \widehat{p_i})^2$
- **formula** gives us **coefficients** directly:
  - $\hat{\beta} = (X^T X)^{-1} X^T y$

Limits of our model

→Output is always a **single unconstrained** real number

Solution:

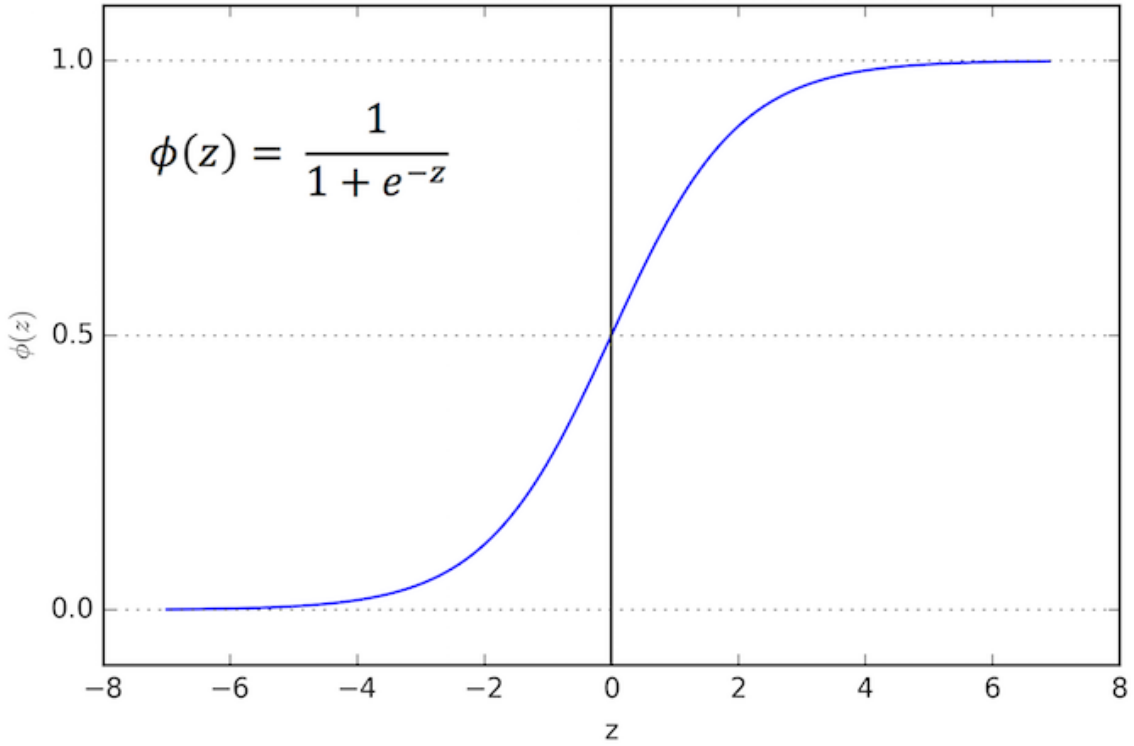
→ Generalized Linear Models

Generalized Linear Model for default prediction

- Simple example – credit scoring:
  - 30 years old
  - € 10,000 credit amount
  - makes € 2,000 per month

→ output should be **yes or no**, not any real number

Solution



- Link Functions (= Activation Functions):
  - transform real number into output we want: $p \in [\mathbf{0}; \mathbf{1}]$
- Logistic Link Function (Perceptron):
  - $logit(p) = -\log\left(\frac{1}{p} - 1\right), logit^{-1}(y) = \frac{1}{1+\exp(-y)}$

How to make GLMs work

- Adapt **link functions** to desired output
  - **extensions for all distributions** with density functions
- **Multiple GLMs** to get **multiple outputs**
  - e.g. mean and variance

How should we choose the coefficients?

- minimize **log likelihood**:

  = maximize probability of training data according to chosen distribution

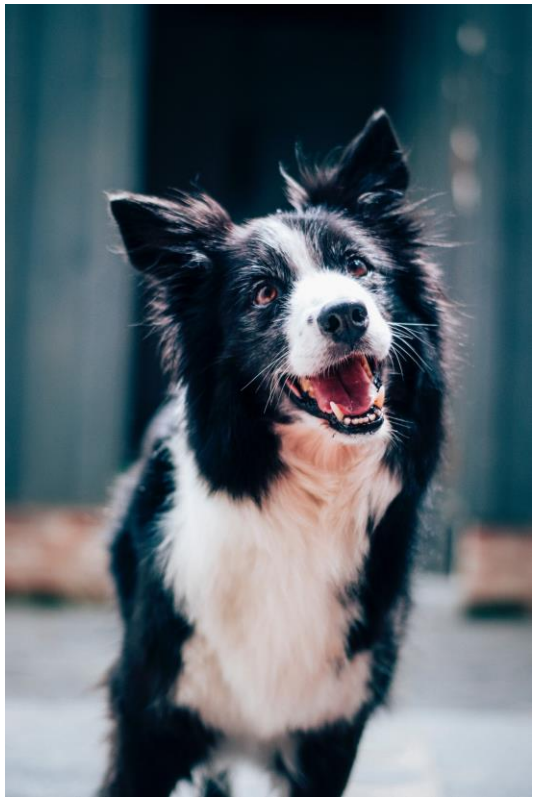
- **optimize numerically**

→link function forces us to use log likelihood

→makes model **less transparent** and **harder to use**

Limits of GLMs

- THE problem:
  - generate **identical outputs** when **inputs** have **nothing in common**
- **Root** of all limitations:
  - output is based on **weighted sum of inputs**

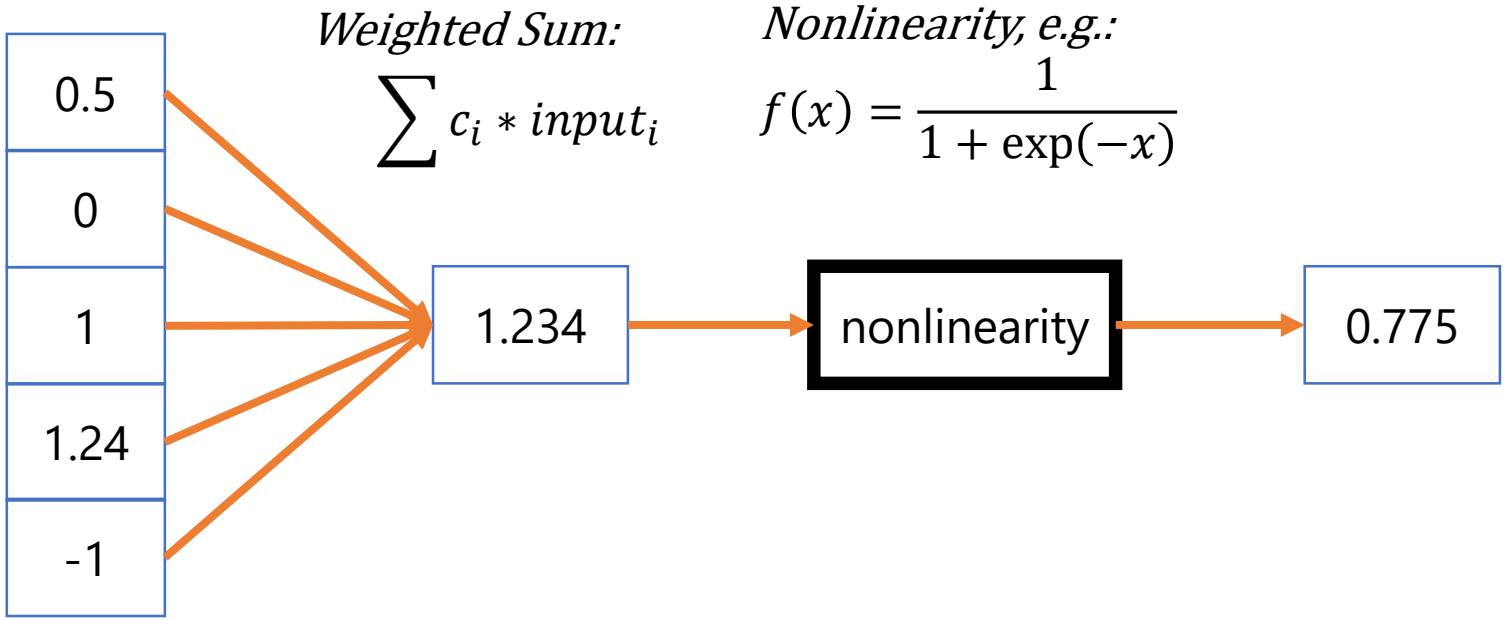


Limits of GLMs

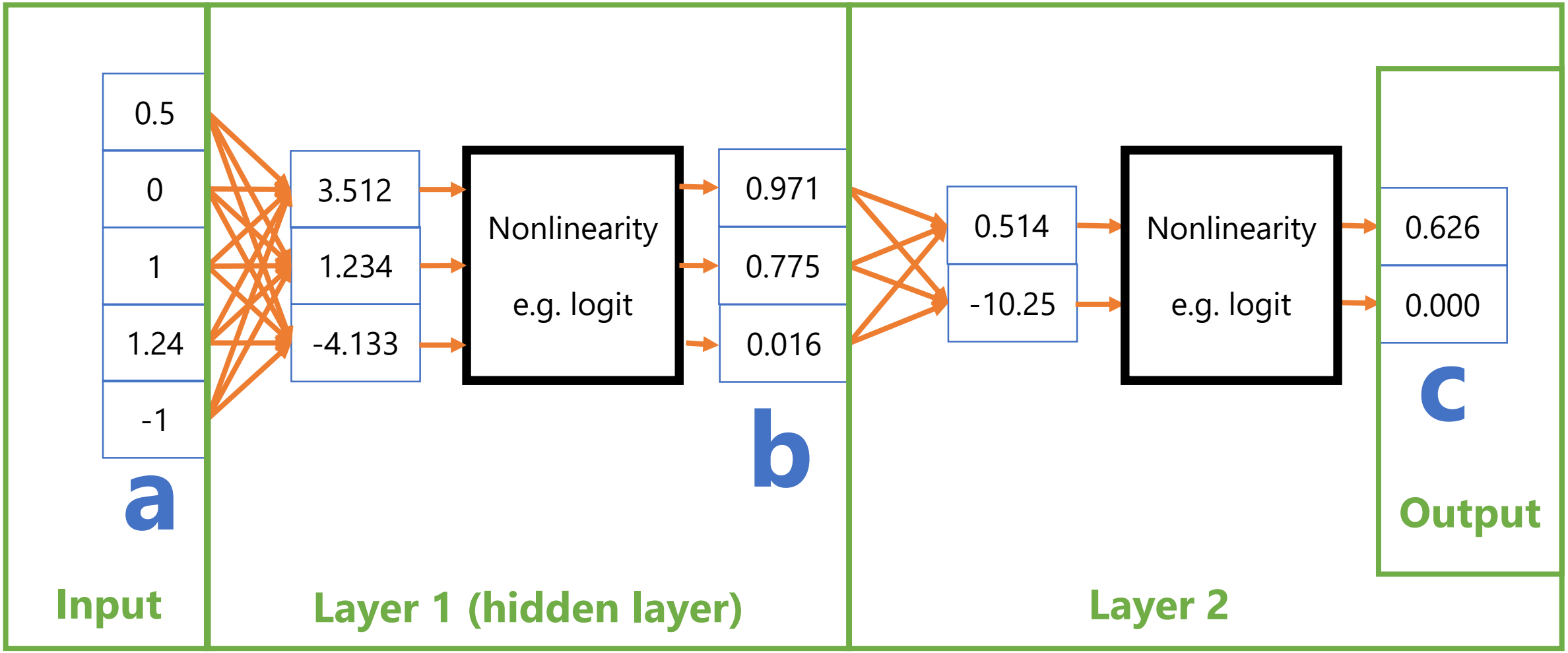
- Most popular example:
  - XOR Function
    - XOR(0 , 0) → 0
    - XOR(1 , 1) → 0
    - XOR(0 , 1) → 1
    - XOR(1 , 0) → 1

Improving GLMs

- Multilayer Perceptron:
  - basic idea: stack layers of linear models with activation functions
- Single **Perceptron = GLM** with logit link function:



Multilayer Perceptron



a

$b = logit(a^T W_b)$

$c = logit(b^T W_c)$

Feed-Forward Layers

- All layers do the same thing:

$$y = f(x^T W)$$

$$\begin{bmatrix} y_1 & y_2 \end{bmatrix} = logit\left(\begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix} * \begin{pmatrix} c_{1,1} & c_{1,2} \\ c_{2,1} & c_{2,2} \\ c_{3,1} & c_{3,2} \end{pmatrix} \right)$$

Multilayer Perceptron

- What would you use it for?
  - **Classification**
    - e.g. fraud detection, churn rate prediction
- Why?
  - **automatic feature detection**
  - in theory MLP is a universal approximator

How should we choose the coefficients?

- **minimize loss function** (= performance metric)

  e.g. squared difference between prediction and actual value

- optimize with **SGD**

→multiple layers make the model **more flexible**

→**“normal statistics” not applicable to** the **model**

Stochastic Gradient Descent

- Step 1:
  - **calculate loss** for a batch of training samples
    - compare prediction and truth
- Step 2:
  - **change** all **parameters** a bit such that **loss reduces** for batch
- Step 3:
  - back to Step 1

Example: XOR

Ingredients of a Neural Network

- Input
- Architecture
  - Connectivity
  - Activation Functions
- Loss
- Optimizer

Input

- (independent) Numbers
  - input consists of independent numbers
    - e.g. age, credit amount and monthly income
- Tensors
  - used for data that does not fit naturally into a vector
    - e.g. images, audio
- Embeddings
  - used for categorical data
    - e.g. country, musical genre

Tensors

- many datasets **don't** naturally **fit into vector**
  - e.g. **images** consists for example of **128 x 128 px** with 3 channels each
- in computer science you would use a **multidimensional array**
  - e.g. new Double()[128][128][3]
- Mathematicians names:
  - 0-dimensional array: scalar
  - 1-dimensional array: vector
  - 2-dimensional array: matrix
  - umbrella term: tensor

Scalar: 42

Vector: $\begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$

Matrix: $\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$

Embeddings

- How do linear models handle categorical data?
  - **One-hot encoding**:
    - Adventure movie → [1, 0, 0]
    - Action movie → [0, 1, 0]
    - Romantic movie → [0, 0 , 1]
- Problem:
  - assumes that all genres are **completely different** from each other
  - even larger problem as number of categories increases

Embeddings

- Solution: Embeddings
  - idea: represent categorical variables through a **set of shared factors**
  - advantages:
    - **fewer parameters** through shared factors
    - **interpretability**
  - e.g. movies:
    - not all movies are equally different
    - factors could describe the amount of action / comedy / romance that occurs in the movie

Example: Movie Lens Dataset

User:

$$\begin{pmatrix} 0.5 \\ 2 \\ 1 \end{pmatrix}$$

$$0.5$$

Movie: $(0 \quad 1 \quad 2)$ -1

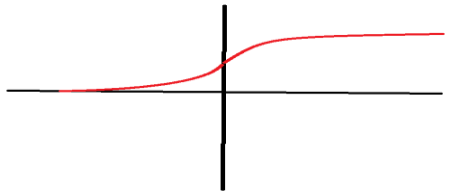
$$(0 \quad 1 \quad 2) * \begin{pmatrix} 0.5 \\ 2 \\ 1 \end{pmatrix} - 1 + 0.5 = 3.5$$

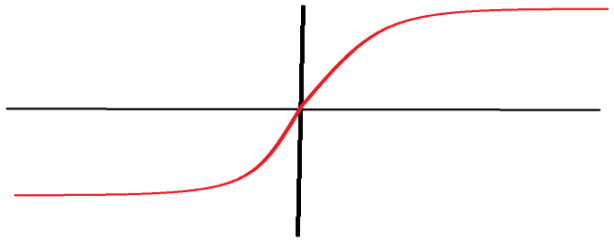
Architecture

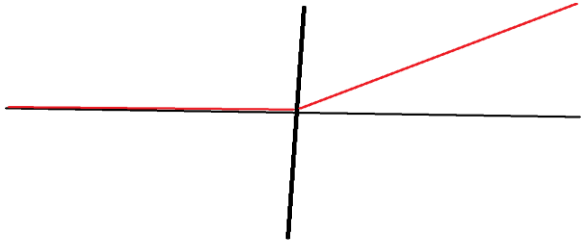
- Activation Functions
  - **non-linear functions** make models powerful
- Connectivity
  - e.g. layer might see output of **all previous** layers

Architecture: Activation Functions

- logit (=sigmoid)
  - squash values between 0 and 1
- tanh
  - squash values between -1 and 1
- relu
  - truncate values between 0 and positive infinity
- softmax
  - squash multiple values between 0 and 1, such that they sum to 1
  - like logit + normalization







Architecture: Connectivity

- Image Processing
  - Convolutions
  - Pooling

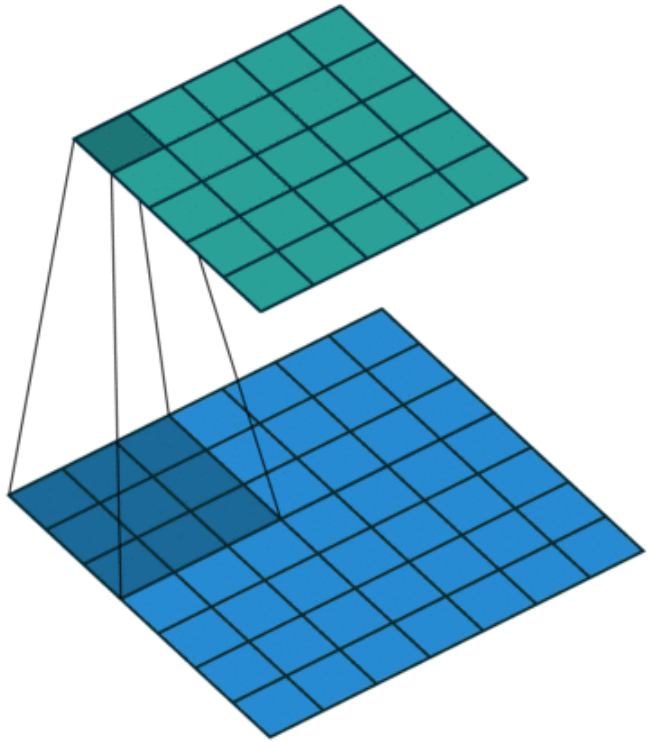
Image Processing Layers

- Aim at capturing two important properties of images:
  - **locality:** object are only on some part of the picture, the rest is background
  - **translation invariance:** a cat is a cat in the top left corner, as well as the bottom right corner



Convolutional Layers

- Idea:
  - many small neural networks reused for many parts of the image

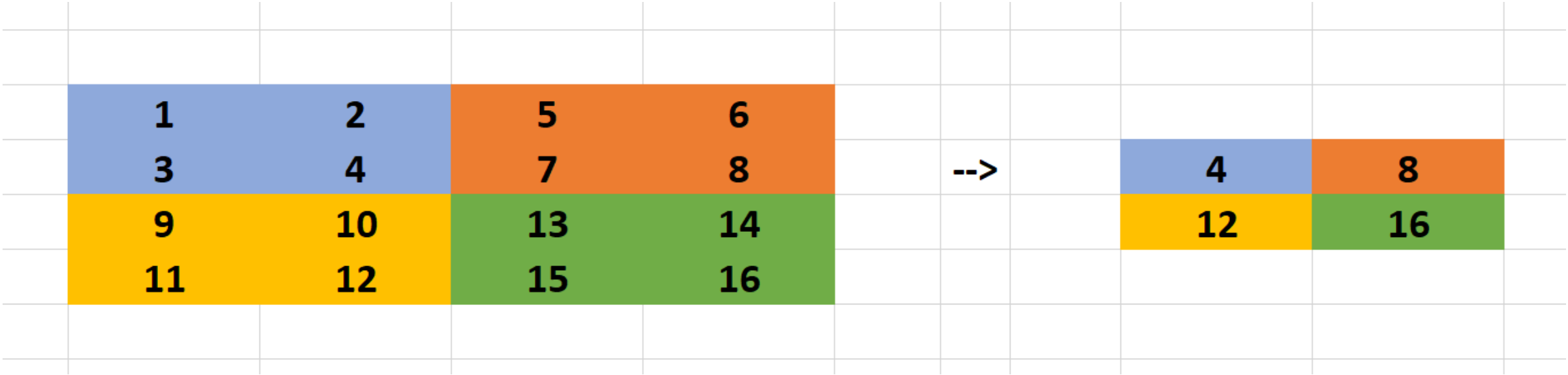


Convolutional Layers

- Excel-Sheet

Pooling Layers

- Idea:
  - combine multiple values



Example: MNIST

Losses

- **function** that describes **how good** the **output** of the network **matches** the **optimal output**
  - e.g. $loss = (predicted - actual)^2$

Getting started with Deep Learning

1. Learn about basic concepts:
   - Embeddings, convolutions, pooling
2. Choose a framework
3. Do tutorials

Frameworks

- Keras

  ++ **very simple**, **very well designed API** around TensorFlow

  \+ **lots of tutorials**

  \- **limited** in what you can do

Frameworks

- PyTorch
  + **simple, well designed API** (easy to learn)
  + run on define (numpy on the GPU)
  + supports highly dynamic networks
  - in Beta

Frameworks

- TensorFlow
  - ++ **lots of resources** (tutorials, models)
  - - **complex, badly designed API** (hard to learn)
  - - does not support highly dynamic networks (without lots of C++ code)
  - + **production ready**
  - + scales across multiple machines

Best Tutorial Resources

- fast.ai
- frameworks' official documentation
- Google

Thank you for your attention!

Questions?