Applied Deep Learning

Dr. Philippe Blaettchen
Bayes Business School (formerly Cass)

www.bayes.city.ac.uk

Learning objectives of today

Goals: Creating neural networks with Keras and TensorFlow

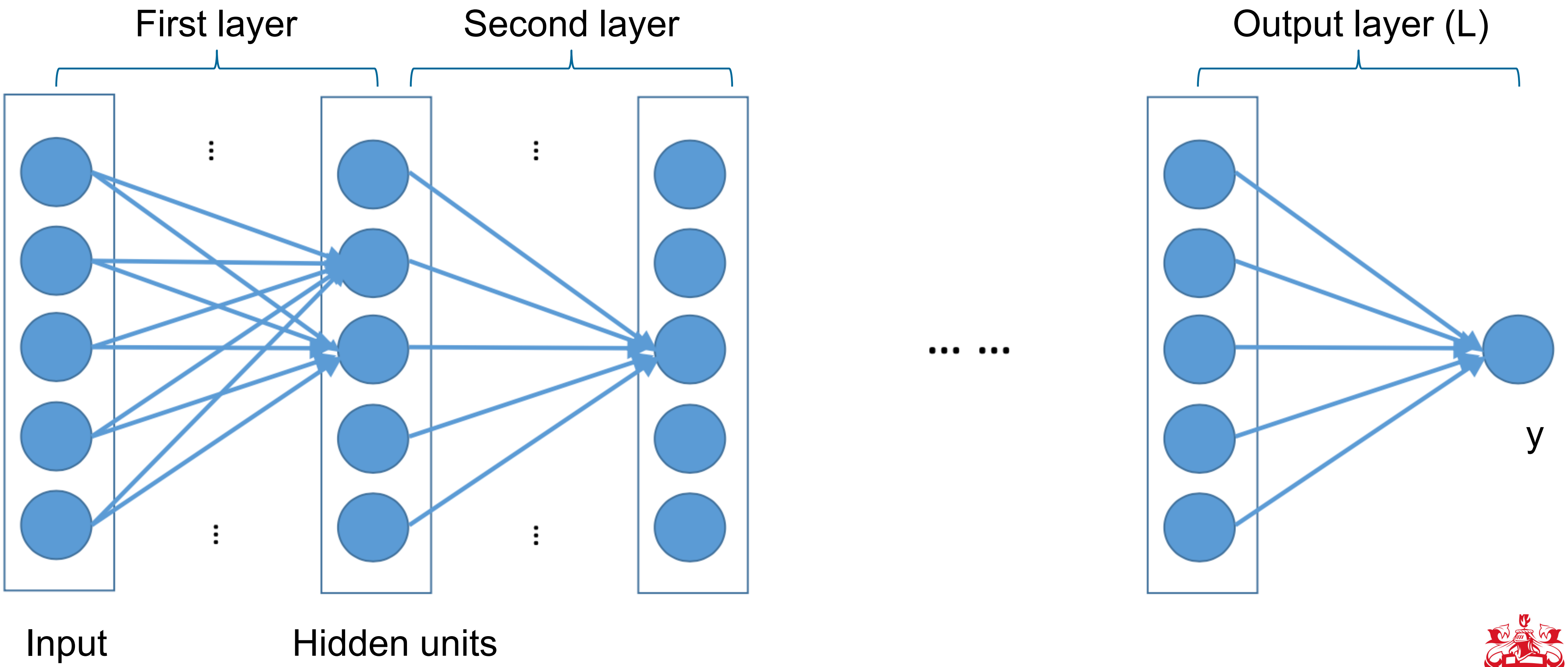
- Understanding the role that frameworks such as TensorFlow play in enabling the design, training, and use of arbitrarily complex neural networks
- Getting started on applying TensorFlow to create different types of feed-forward networks

How will we do this?

- We start with a quick recap of the functioning of neural networks
- We then introduce TensorFlow and Keras as programming frameworks for neural networks
- We create a number of neural networks, using TensorFlow's Sequential API

Recap

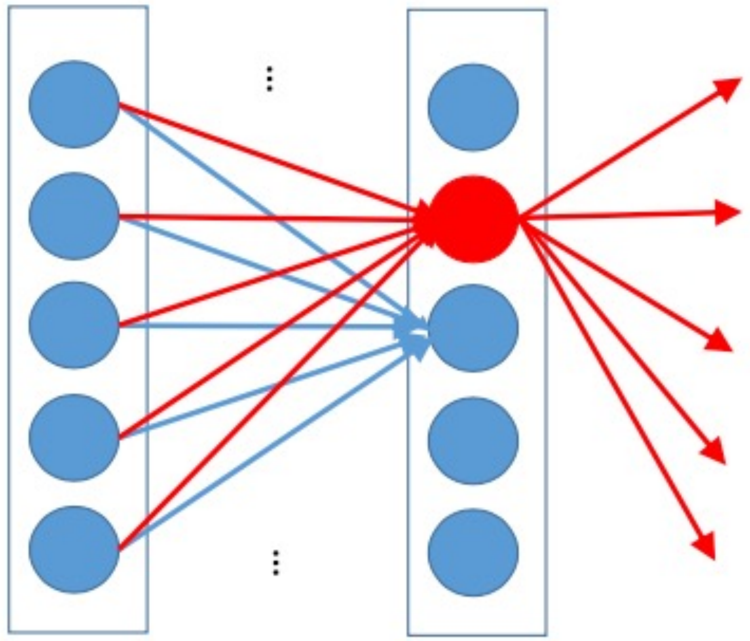
Components



Source: Liang

Hidden layers

$\text{“}\boldsymbol{x}\text{”} = \boldsymbol{a}^{[l-1]}$ $\quad z = \boldsymbol{a}^{[\boldsymbol{l}-\mathbf{1}]}\boldsymbol{w} + b$ $\quad f(\mathrm{z})$



[l-1] [l]

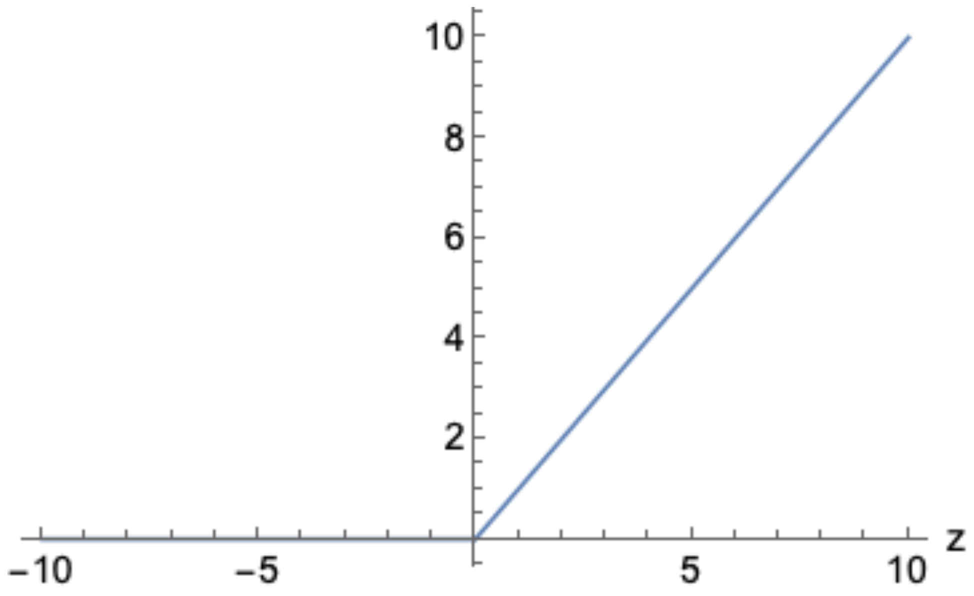
- f is what we call an “activation function”
- There are many activation functions, and new ones are invented all the time
- Many of these functions do just fine, or slightly better than existing ones

Source: Liang

BAYES BUSINESS SCHOOL
CITY, UNIVERSITY OF LONDON

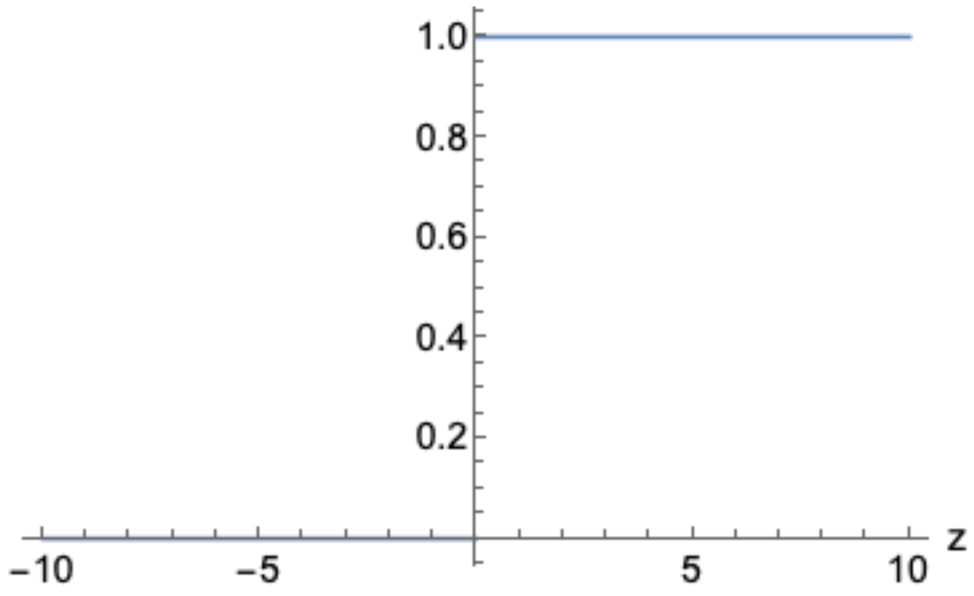
Typical activation functions: Rectified Linear Unit

Rectified Linear Unit (ReLU)



$$f(z) = \max\{0, z\}$$

"Derivative"



$$f'(z) = \begin{cases} 0, if\ z < 0 \\ 1, if\ z > 0 \end{cases}$$

BAYES BUSINESS SCHOOL
CITY, UNIVERSITY OF LONDON

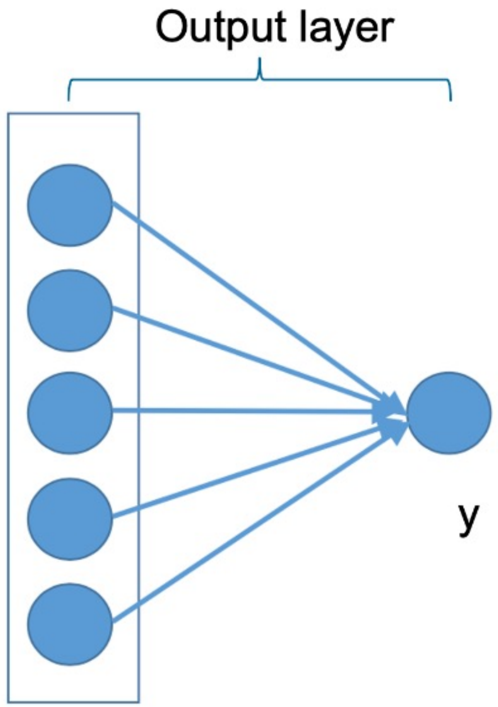
Binary classification

- Input: $\boldsymbol{a}^{[L-1](i)}$

- As usual, we make a linear transformation: $z^{[L](i)} = \boldsymbol{a}^{[L-1](i)}\boldsymbol{w}^{[L]} + b^{[L]}$

- We then use the logistic sigmoid function $\hat{y}^{(i)} = f\left(z^{[L](i)}\right) = \sigma(z^{[L](i)}) = \frac{1}{1+e^{-z^{[L](i)}}}$

- We can interpret the output as the probability of $y^{(i)} = 1$

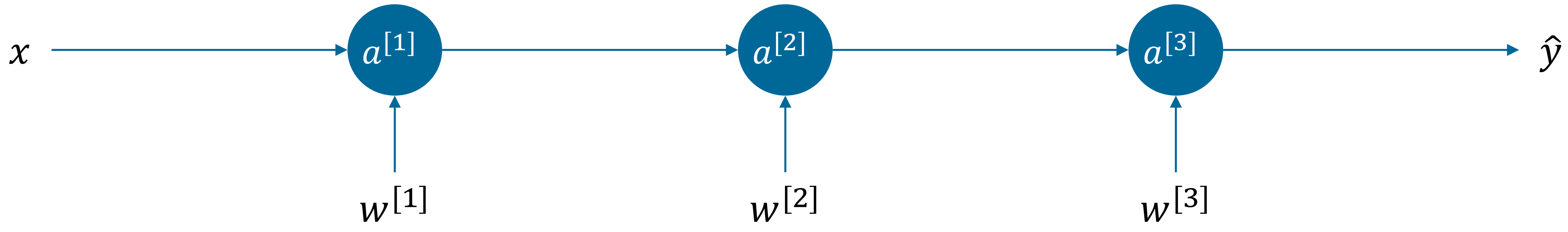


Source: Liang

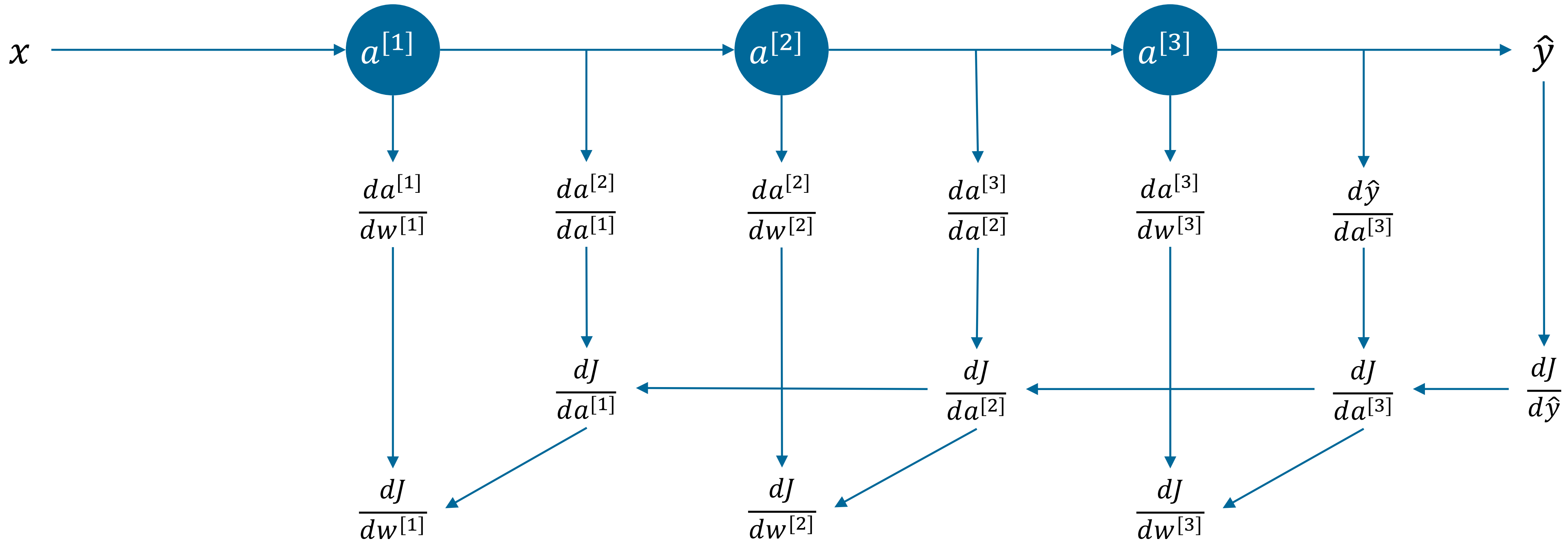
Generalizing our gradient descent algorithm

1. Decide a “learning rate” α
2. Start with some parameters $\boldsymbol{\theta}$ (initialization)
3. For a certain number of iterations
   - Compute $J(\boldsymbol{\theta})$ (forward propagation)
   - Compute $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ (back-propagation)
   - Let $\boldsymbol{\theta} := \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ (parameter update)

Step 1: Forward propagation through the computational graph



Step 2: Back-propagation through the computational graph



BAYES BUSINESS SCHOOL
CITY, UNIVERSITY OF LONDON

Deep learning programming frameworks

Why to use programming frameworks for deep learning?

- Build complex neural networks based on simple modules
- Automatically compute gradients
- Make efficient use of hardware (parallel processing, usage of GPUs and TPUs)
- Create models that are exportable to other runtimes, browsers, or even mobile devices
- Load a large range of pre-trained models

Deep learning in Python: the two main rivals

TensorFlow

- Developed by Google
- Python API based on C engine

- Largest community
- Parallelization of models and data
- Visualization (TensorBoard)
- Relatively intuitive when familiar with numpy
- Runs on most systems (e.g., TensorFlow Lite)
- Broader framework (e.g., reinforcement learning)
- Easy to use Colab-GPU

PyTorch

- Developed by Facebook
- Python API based on LUA engine (Torch)

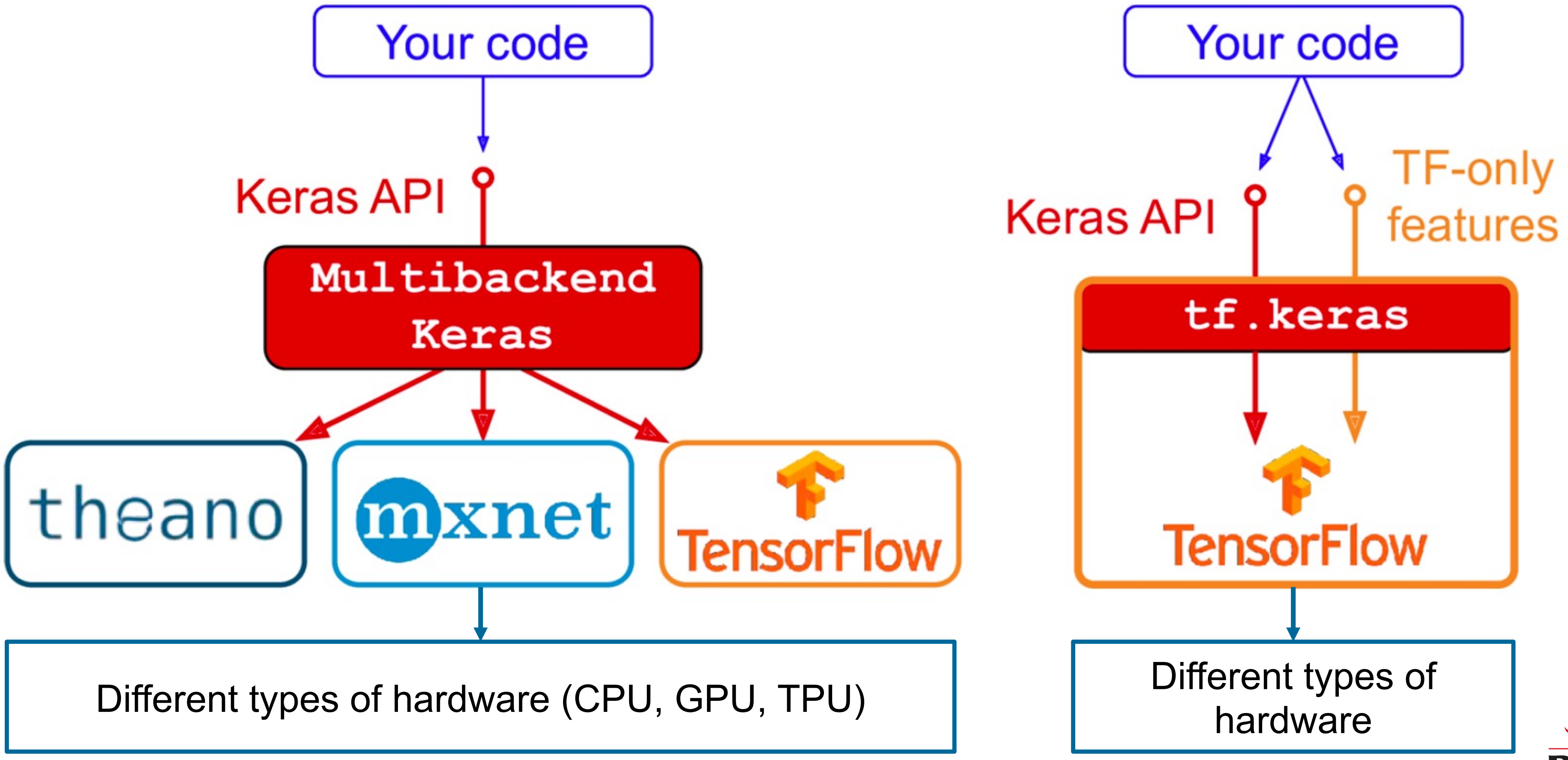
- Many pre-trained models, especially transformers
- Highly modular
- Easy to run on GPUs
- Runs on most systems (e.g., PyTorch Mobile)

Wasn't there also this thing called Keras?

- High-level deep learning API to build, train, evaluate, execute neural networks
    - Documentation: https://keras.io
- Requires computation backend:
    - TensorFlow (at this point, that's the standard)
    - Microsoft Cognitive Toolkit
    - Theano (original backend on which Keras was built)
    - Apache MXNEt
    - Core ML (Apple)
    - JavaScript & TypeScrit (for web browsers)
    - PlaidML
    - …
- TensorFlow comes with its own Keras implementation tf.keras (with some added features)
    - E.g., use Data API from TF

BAYES
BUSINESS SCHOOL
CITY, UNIVERSITY OF LONDON

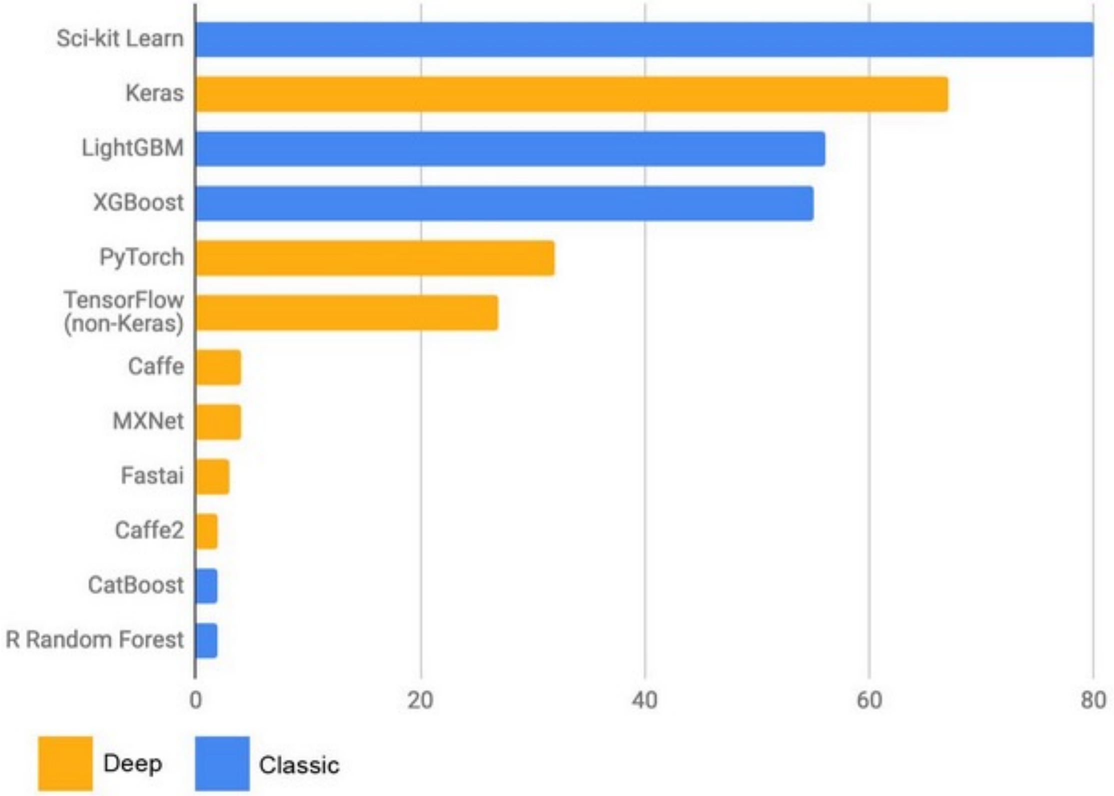
Keras in general and Keras using TensorFlow



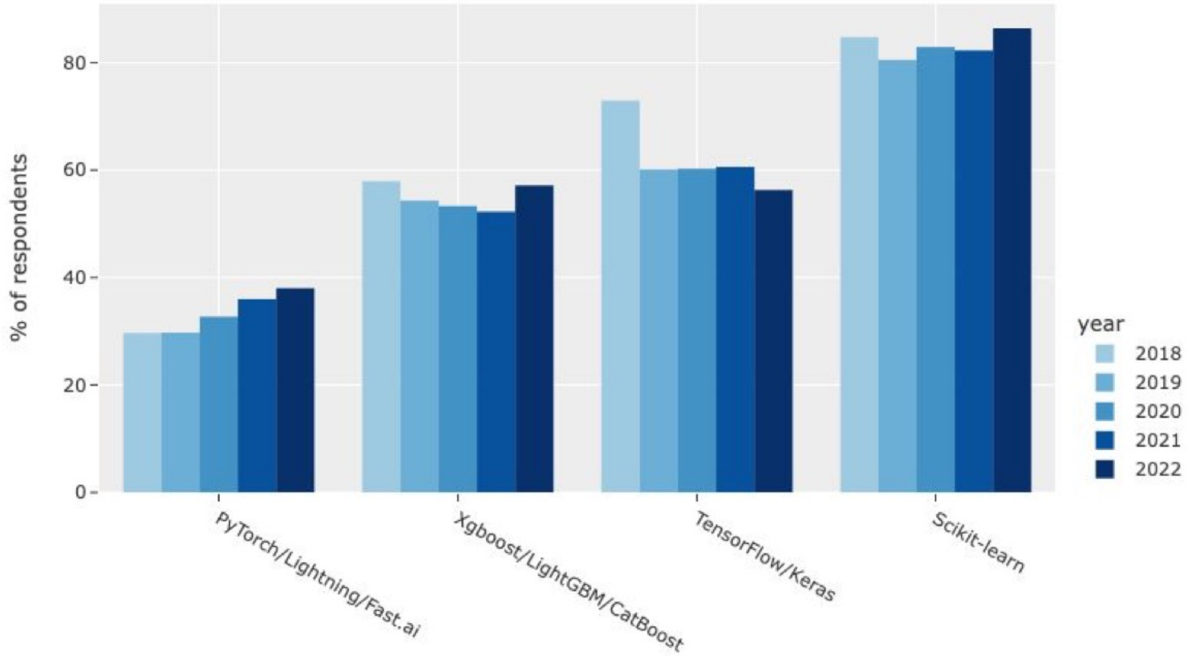
Source: Geron

Some statistics

ML tools used by top-5 teams in Kaggle competitions 2017-2019



ML tools used by professional data scientists surveyed by Kaggle



Source: Kaggle

TensorFlow in practice

Multi-class classification

- We again make a linear transformation with a matrix of weights: $\boldsymbol{z}^{[L](i)} = \boldsymbol{a}^{[L-1](i)}\boldsymbol{W}^{[L]} + \boldsymbol{b}^{[L]}$

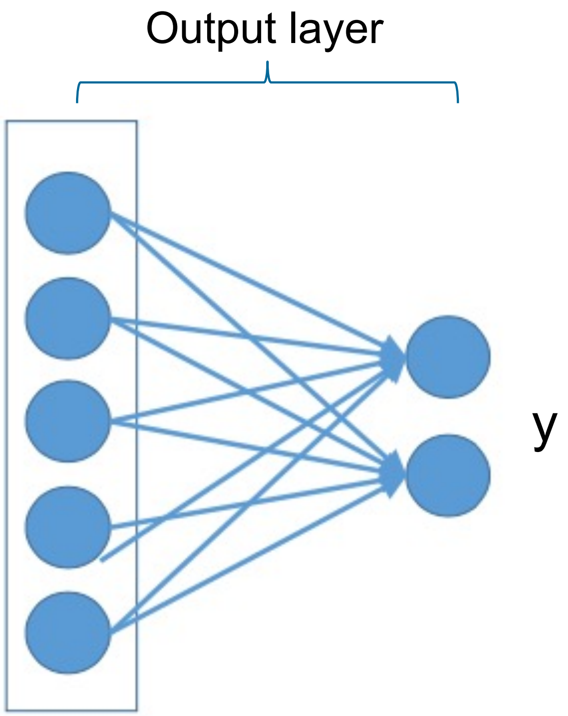
- Note that $\boldsymbol{z}^{[L](i)} = \begin{pmatrix} z_1^{[L](i)} & z_2^{[L](i)} & \cdots & z_K^{[L](i)} \end{pmatrix}$

- We then use the softmax function on each of the outputs:

$$\hat{y}_k^{(i)} = f\left(\boldsymbol{z}^{[L](i)}\right) = \frac{e^{-z_k^{[L](i)}}}{\sum_{k=1}^{K} e^{-z_k^{[L](i)}}}$$

- This implies that $\hat{y}_k^{(i)} \in (0,1)$ and $\sum_{k=1}^{K} \hat{y}_k^{(i)} = 1$

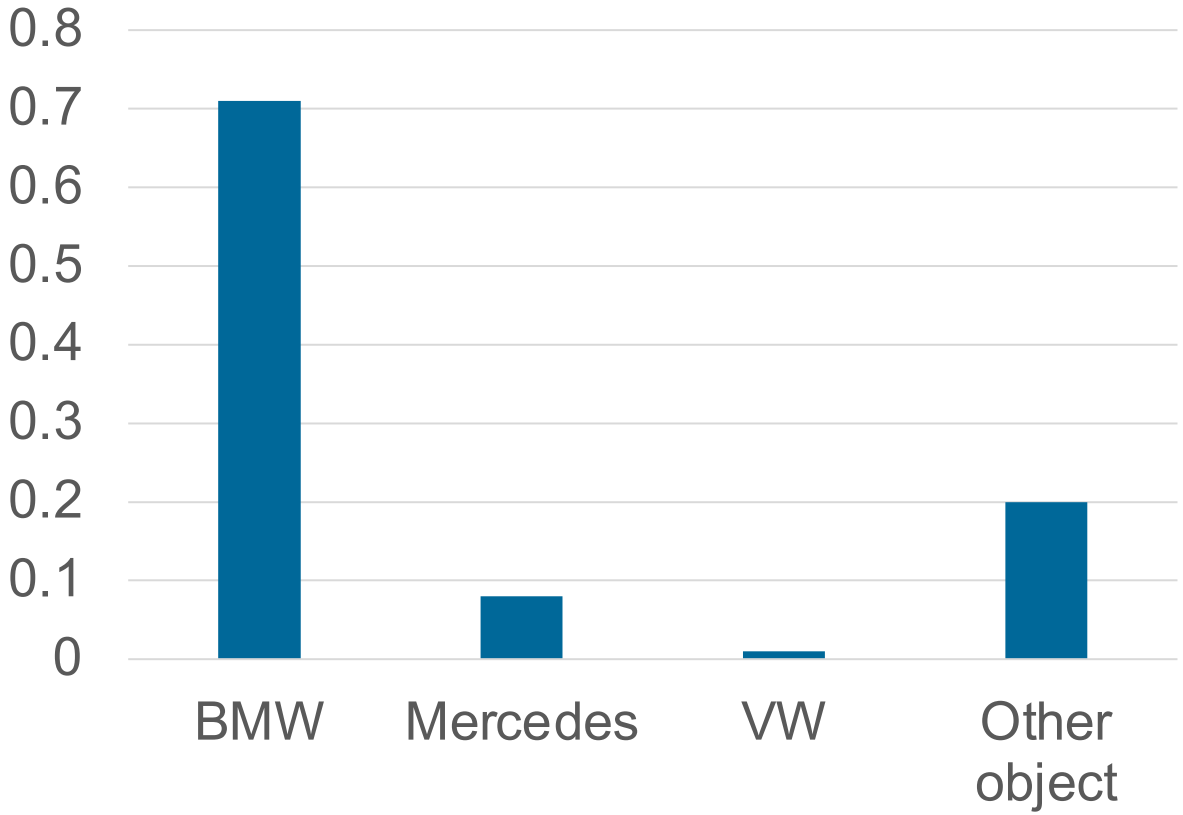
- Hence, we can interpret $\hat{y}_k^{(i)}$ as the probability that $y^{(i)} = k$ ("belongs to class k")



Source: Liang

Softmax output

- E.g., when performing object recognition, we might represent our prediction $\hat{\boldsymbol{y}}^{(i)}$ as



Softmax in practice

See you next week!

Sources

- Chollet, 2021, Deep Learning with Python (2nd edition)
- DeepLearning.AI, n.d.: deeplearning.ai
- Geron, 2019, Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow
- Goodfellow, Bengio, Courville, 2016, The Deep Learning Book: http://www.deeplearningbook.org
- Kaggle, 2019, Kaggle competitions: Why choose Keras as framework?: https://www.kaggle.com/getting-started/151903
- Kaggle, 2022, State of Data Science and Machine Learning 2022: https://www.kaggle.com/kaggle-survey-2022
- Liang, 2016, Introduction to Deep Learning: https://www.cs.princeton.edu/courses/archive/spring16/cos495/

BAYES
BUSINESS SCHOOL
CITY, UNIVERSITY OF LONDON