Hochschule Konstanz
Fakultät Informatik

Oliver Dürr

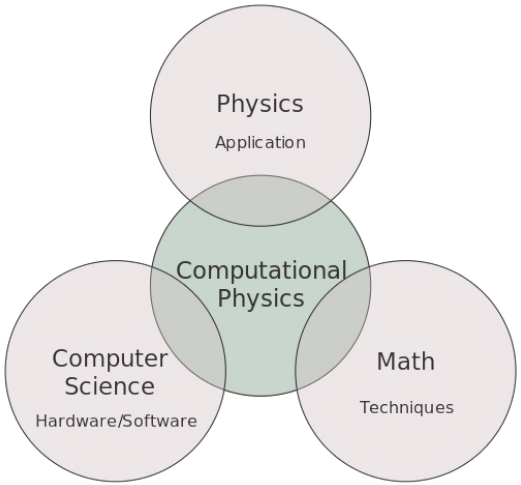
Short course on deep learning
Lesson 1

Southern Taiwan University of Science and Technology

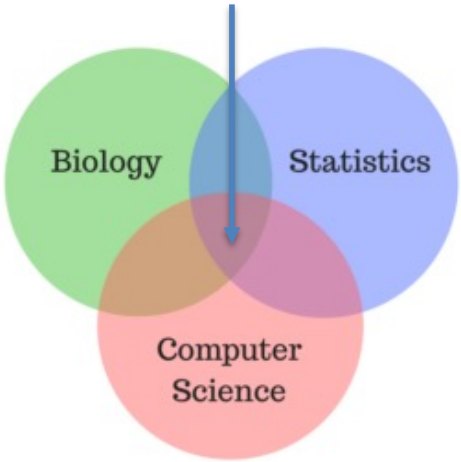
This course is a short version of dl_course_2022 I created with Beate Sick.

Oliver: From Computational Physics → Data Science

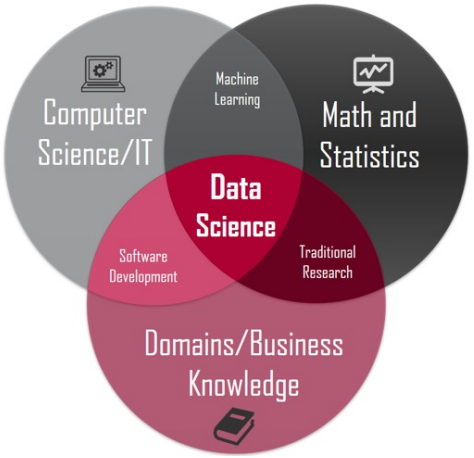
Computational Physics



Bioinformatics



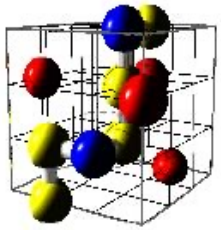
Data Science

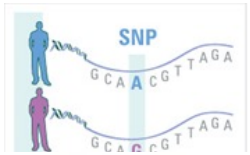


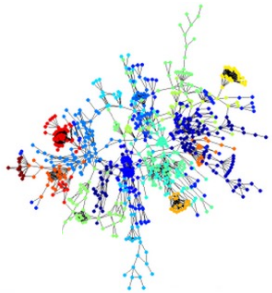
1990's
Uni-Konstanz

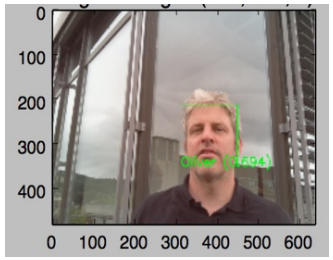
2000's
Genedata Basel

2010's
ZHAW Winterthur HTWG Konstanz









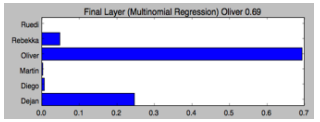


Image credits: Wikipedia, http://omgenomics.com/what-is-bioinformatics/ https://www.fox.temple.edu/institutes-and-centers/data-science/

Tell us something about you

- CS Background
  - Fluent in python?
    - numpy?
    - pandas
- Statistics / Math
  - What is a distribution?
  - Gradient
  - Gradient Descent?
  - Vector times Matrix?
    - Please make sure to check https://tensorchiefs.github.io/dl_course_2022/prerequistites.html
- Any contacts with deep learning yet?
- Motivation for DL, Data you want to work with?

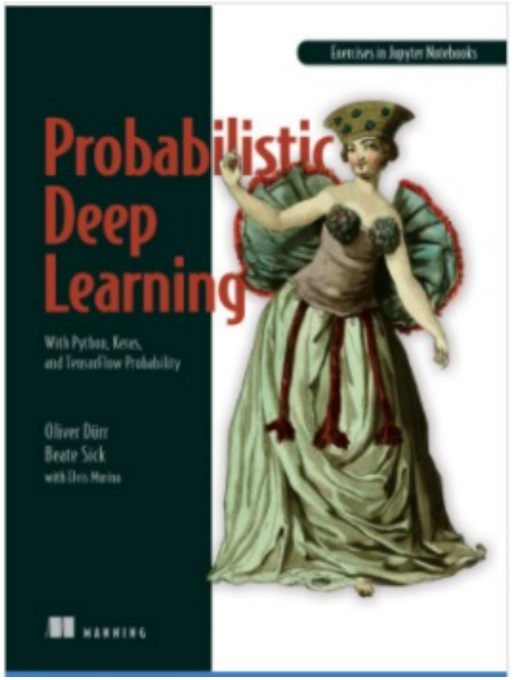
Technical details for this course

- Running the code:
  - Could (need internet)
    - **Colab Notebooks**
    - Kaggle Notebooks
    - Deepnote
  - Local Installation
    - Anaconda
    - Docker image: **`oduerr/dl_book_docker`**
      - Based on the official TF docker images with additional libraries needed for the course
      - How to use docker (see website)
      - **`docker pull oduerr/dl_book_docker`**

Material for the course

- Website and Github repository
  - The CAS Deep Learning Course
    - https://oduerr.github.io/dl_scourse_2022/

- Book
  - Many examples used in the course are from the book
  - Can be used in addition to the course
  - https://www.manning.com/books/probabilistic-deep-learning-with-python?a_aid=probabilistic_deep_learning&a_bid=78e55885
  - https://github.com/tensorchiefs/dl_book



Organizational Issues: ~~Test~~ Projects

- Projects (2-3 People)

- Presented on the last day
  - Spotlight talk (5 Minutes)
  - Poster
- Topics
  - You can / should choose a topic of your own (have to be discussed with me by Monday 11/14)

  - Possible Topics (see website)
    - Take part in a Kaggle Competition (e.g. Leaf Classification / Dogs vs. Cats)
    - Music classification
    - Polar bear detection
    - ...

- Computing: colab, laptop (or cloud computing)

Organizational Issues: Times

- Dates and times: see our webpage
- Theory and exercises will be mixed
  - Could be 50 minutes theory 30 minutes exercises
  - Could be vice versa
- **Please interrupt us if something is unclear! The less I talk the better!**

Outline of the DL Module Day 1 – Day 3 is fixed needs decission

- Block 1: Jumpstart to DL
  - What is DL
  - Basic Building Blocks
  - Keras
- Block 2: CNN I
  - ImageData
- Block 3: CNN II and RNN(optinal)
  - Tips and Tricks
  - Modern Architectures
  - 1-D Sequential Data
- Block 4
  - Linear Regression
  - Backpropagation
    - Resnet
  - Likelihood principle

- L *4*: Probabilistic Aspects
  - TensorFlow Probability (TFP)
  - Negative Loss Likelihood NLL
  - Count Data

The content of L 3 and L 4 depends a bit on your need

Day 1-3 should get you ready for your project.

Learning Objectives for Block 1

- Get a rough idea what the DL is about

- Get a first idea on patterns in NN / DL
  - Computational Graph
  - Flow of tensors
    - Matrix and Tensor operations on a computational graph
  - Backpropagation
    - To fit the weights of a network efficiently

- Framework
  - Introduction to Keras

Introduction to Deep Learning --what's the hype about?

Machine Perception

- Computers have been quite bad in things which are easy for humans (images, text, sound)
- A Kaggle contest 2012
- In the following we explain why

Kaggle dog vs cat competition

Deep Blue beat Kasparov at chess in 1997.
Watson beat the brightest trivia minds at Jeopardy in 2011.
Can you tell Fido from Mittens in 2013?

Deep Learning vs. Machine Learning

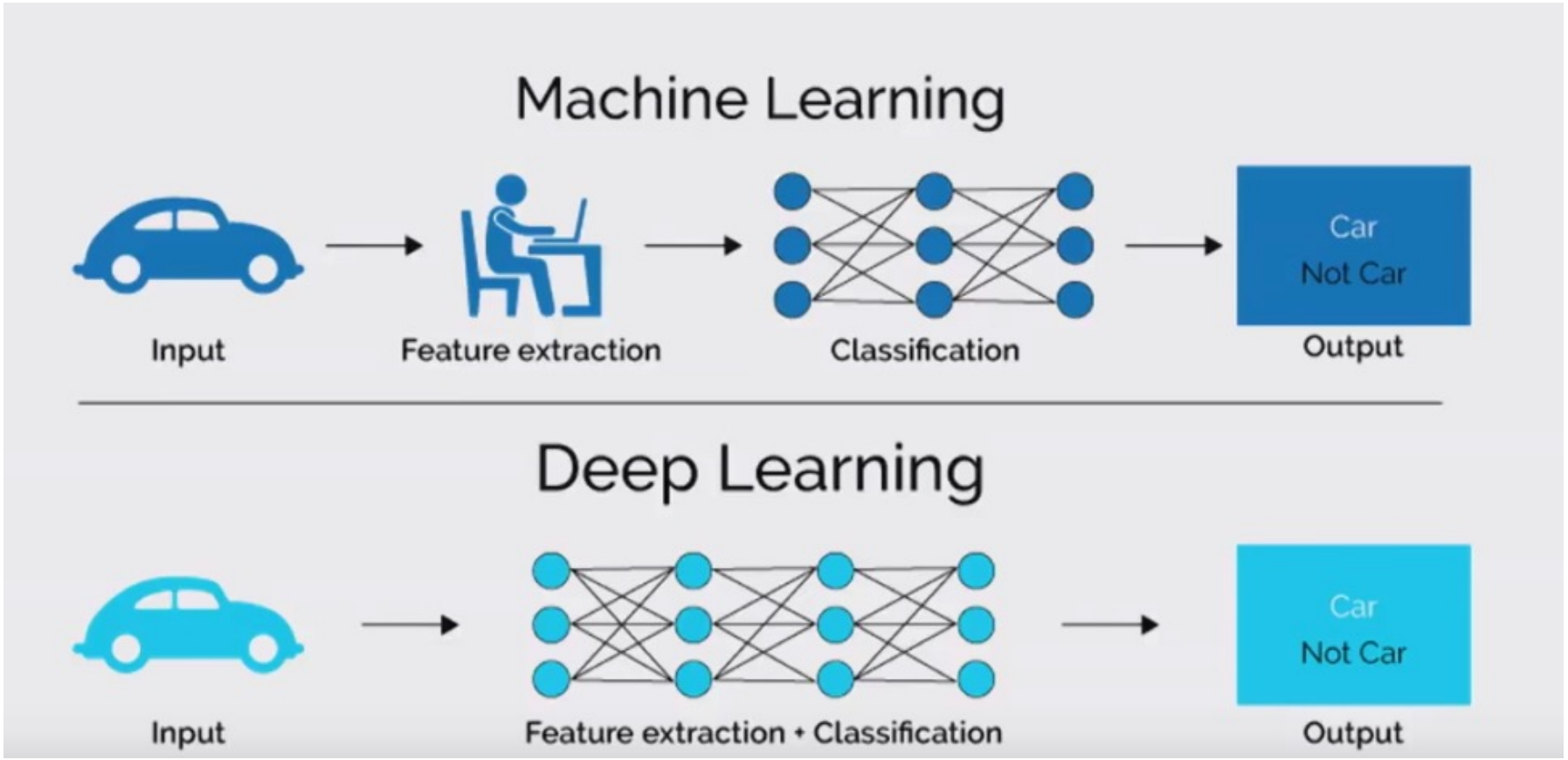
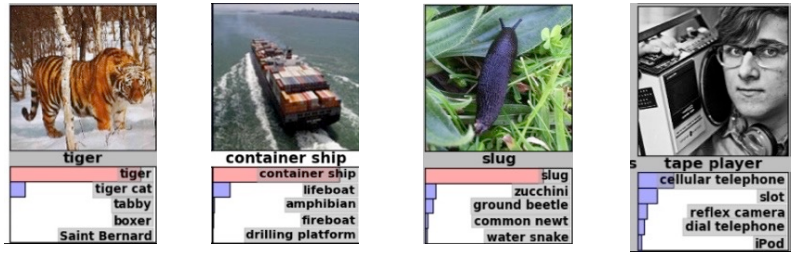


image credit: Siraj Raval

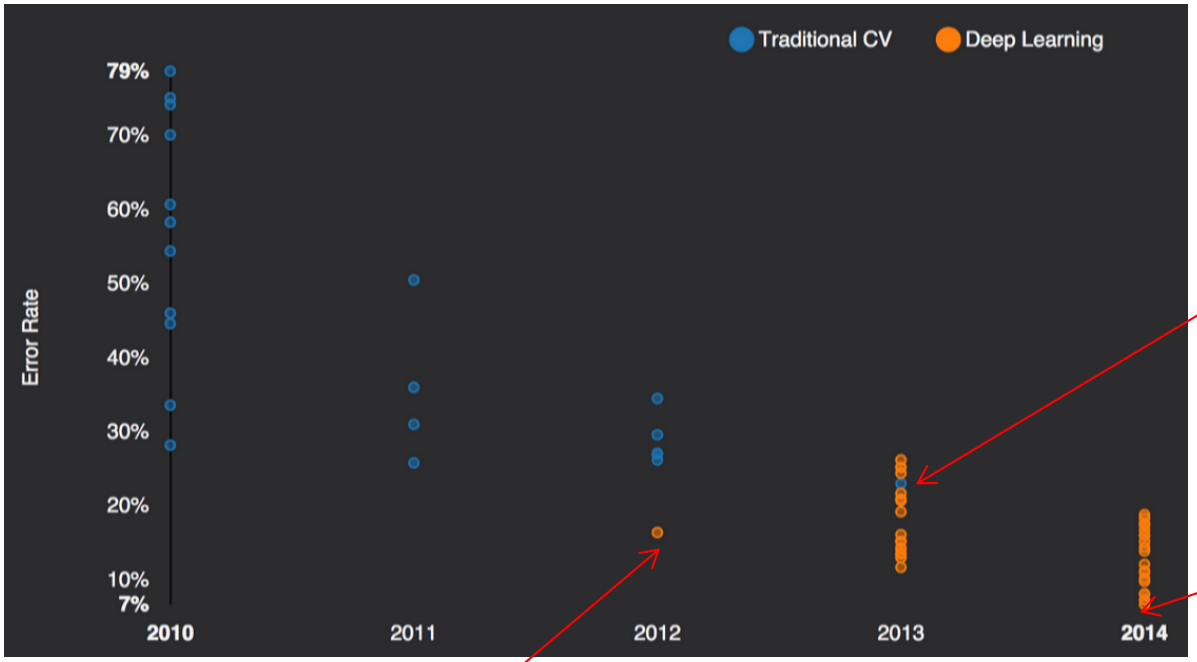
The most convincing case for DL (subjective view)

Why DL: Imagenet 2012, 2013, 2014, 2015

1000 classes
1 Mio samples



...



Human: 5% misclassification

Only one non-CNN approach in 2013

GoogLeNet 6.7%

A. Krizhevsky
first CNN in 2012
Und es hat zoom gemacht

2015: It gets tougher

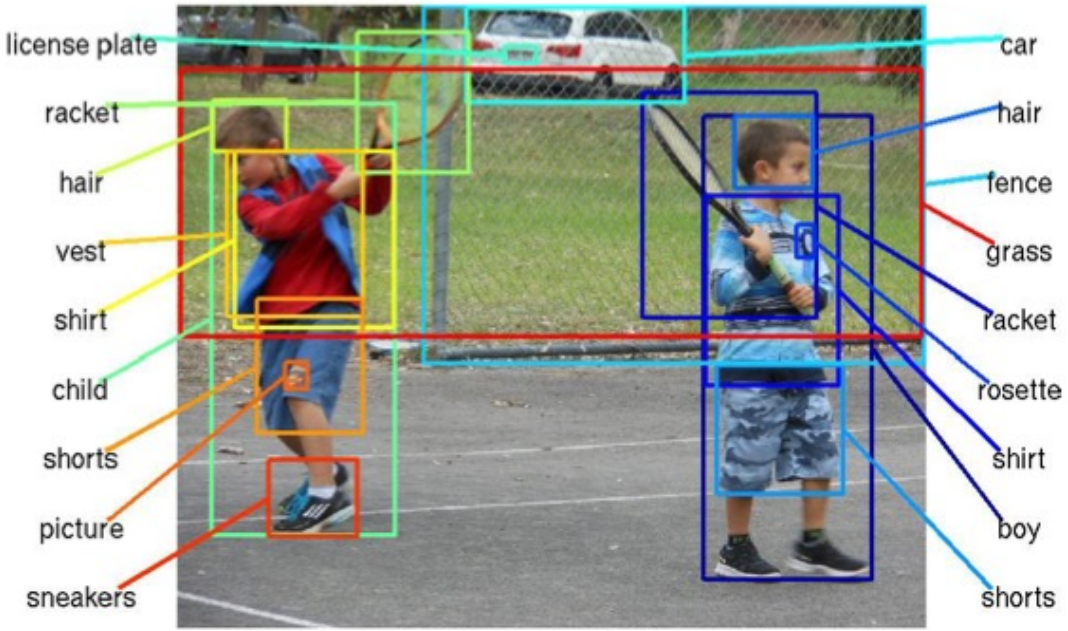
- 4.95% Microsoft (Feb 6 surpassing human performance 5.1%)
- 4.8% Google (Feb 11) -> further improved to 3.6 (Dec)?
- 4.58% Baidu (May 11 banned due too many submissions)
- 3.57% Microsoft (Resnet winner 2015) → task solved!

Figure: https://medium.com/global-silicon-valley/machine-learning-yesterday-today-tomorrow-3d3023c7b519

The computer vision success story

- With DL it took approx. 3 years to solve object detection and other computer vision task

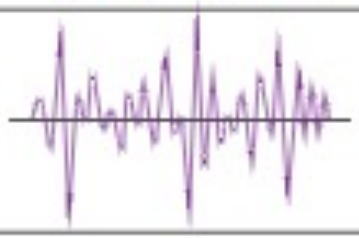
Deep Blue beat Kasparov at chess in 1997.
Watson beat the brightest trivia minds at Jeopardy in 2011.
Can you tell Fido from Mittens in 2013?



"man in black shirt is playing guitar."

Images form cs229n

Use cases of deep learning

Input x to DL model	Output y of DL model	Application
Images 	Label "Tiger"	Image classification
Audio 	Sequence / Text "see you tomorrow"	Voice recognition
ASCII-Sequences "Hallo, wie gehts?"	Unicode-Sequences "你好，你好吗?"	Translation
ASCII-Sequence This movie was rather good	Label (Sentiment) positive	Sentiment analysis

Deep Learning opens the door to perception. (But no undestanding just statistitical correlations).

Source: Oliver Dürr, Beate Sick, Elvis Murina in Deep Learning in Depth, Manning

Deep learning

Artificial Intelligence?

All the impressive achievements of deep learning amount to just curve fitting

Juda Pearl, 2018

Pearl's ladder of causality

Current DL

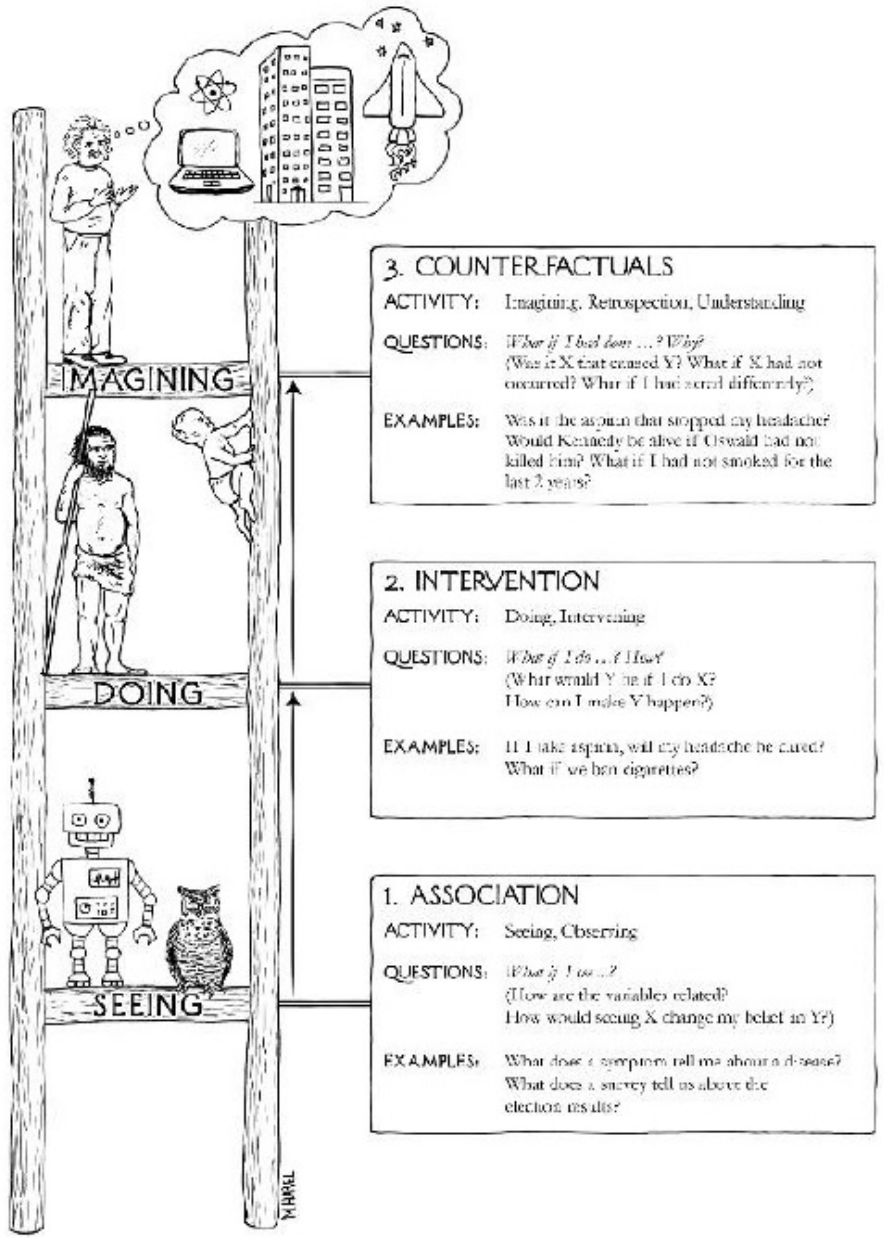
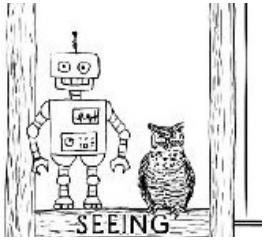


FIGURE 1.2. The Ladder of Causation, with representative organisms at each level. Most animals, as well as present-day learning machines, are on the first

Image credits: "The Book of Why"

On the first rung of the ladder
DL is currently as good as a ensemble of pigeons ;-)



https://www.youtube.com/watch?v=NsV6S8EsC0E

PLOS ONE

OPEN ACCESS

Citation: Levenson RM, Krupinski EA, Navarro VM, Wasserman EA (2015) Pigeons (*Columba livia*) as Trainable Observers of Pathology and Radiology Breast Cancer Images. PLoS ONE 10(11): e0141357. doi:10.1371/journal.pone.0141357

Editor: Jonathan A Coles, Glasgow University, UNITED KINGDOM

Received: August 25, 2015

Accepted: October 7, 2015

Published: November 18, 2015

RESEARCH ARTICLE

Pigeons (*Columba livia*) as Trainable Observers of Pathology and Radiology Breast Cancer Images

Richard M. Levenson[1]*, Elizabeth A. Krupinski[3], Victor M. Navarro[2], Edward A. Wasserman[2]*

1 Department of Pathology and Laboratory Medicine, University of California Davis Medical Center, Sacramento, California, United States of America, **2** Department of Psychological and Brain Sciences, The University of Iowa, Iowa City, Iowa, United States of America, **3** Department of Radiology & Imaging Sciences, College of Medicine, Emory University, Atlanta, Georgia, United States of America

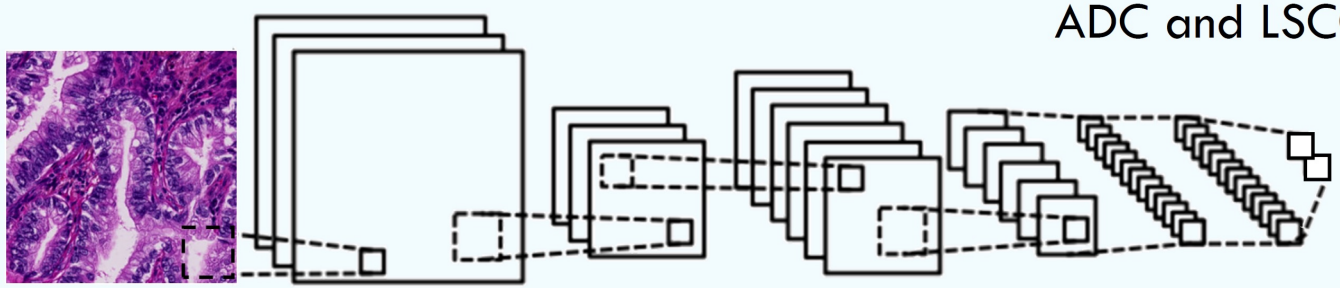
* levenson@ucdavis.edu (RML); ed-wasserman@uiowa.edu (EAW)

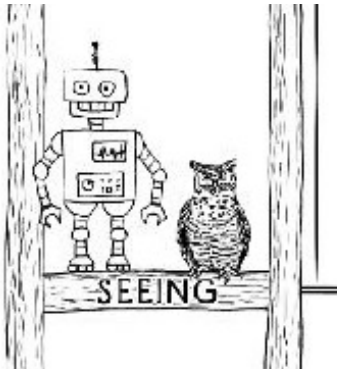
Abstract

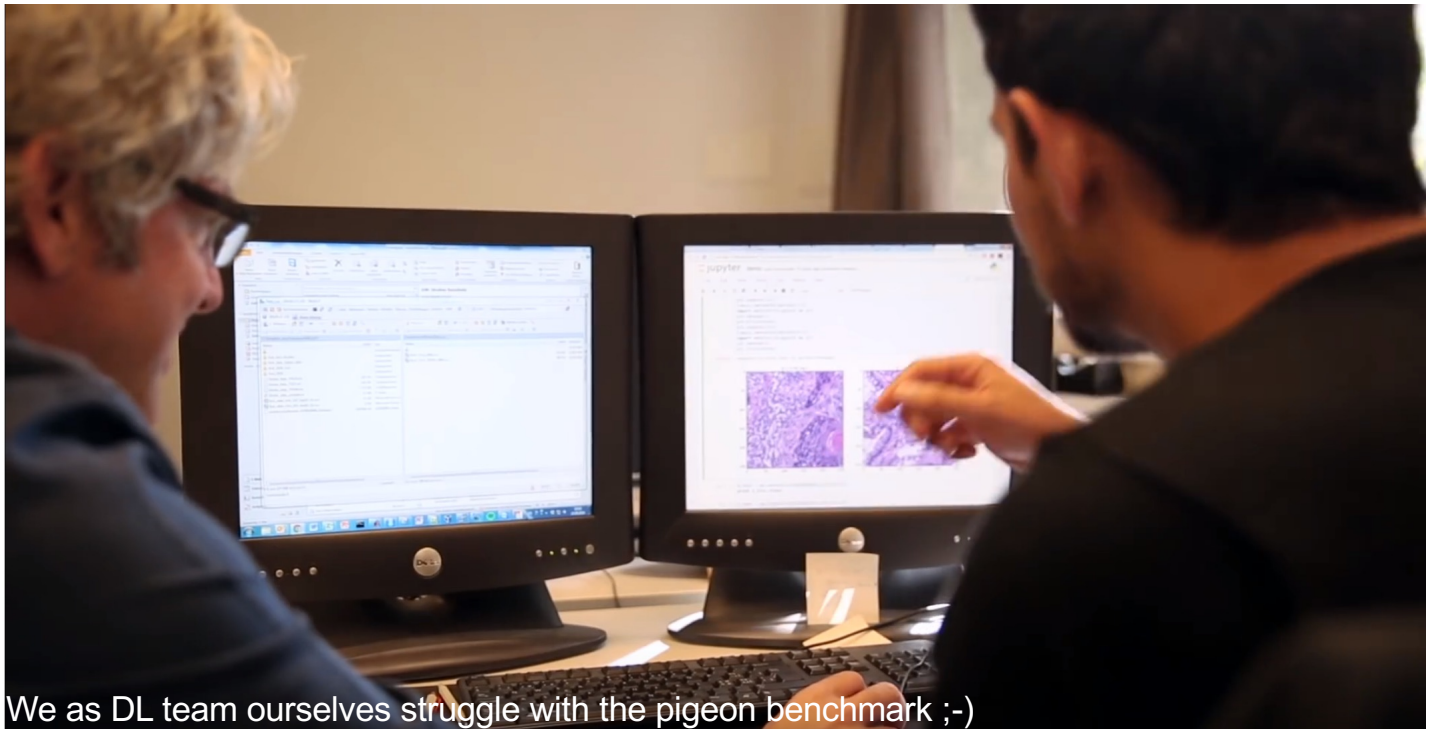
Pathologists and radiologists spend years acquiring and refining their medically essential visual skills, so it is of considerable interest to understand how this process actually unfolds and what image features and properties are critical for accurate diagnostic performance. Key insights into human behavioral tasks can often be obtained by using appropriate animal models. We report here that pigeons (*Columba livia*)—which share many visual system properties with humans—can serve as promising surrogate observers of medical images, a capability not previously documented. The birds proved to have a remarkable ability to distinguish benign from malignant human breast histopathology after training with differential food reinforcement; even more importantly, the pigeons were able to generalize what they had learned when confronted with novel image sets. The birds' histological accuracy, like that of humans, was modestly affected by the presence or absence of color as well as by degrees of image compression, but these impacts could be ameliorated with further training. Turning to radiology, the birds proved to be similarly capable of detecting cancer-relevant microcalcifications on mammogram images. However, when given a different (and for humans quite difficult) task—namely, classification of suspicious mammographic densities (masses)—the pigeons proved to be capable only of image memorization and were unable

On the first rung of the ladder
DL is currently as good as an ensemble of pigeons

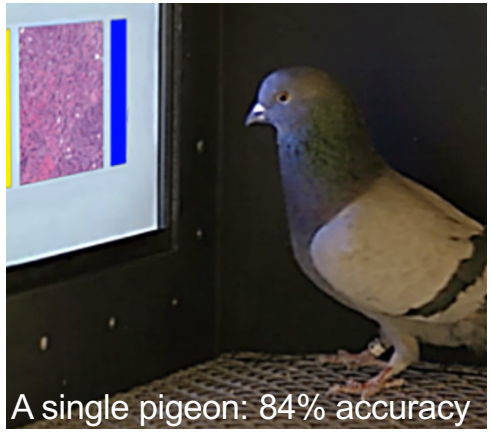
Our DL model achieves ~90% accuracy on image level







We as DL team ourselves struggle with the pigeon benchmark ;-)



A single pigeon: 84% accuracy

First Neural Network

The Single Cell: Biological Motivation

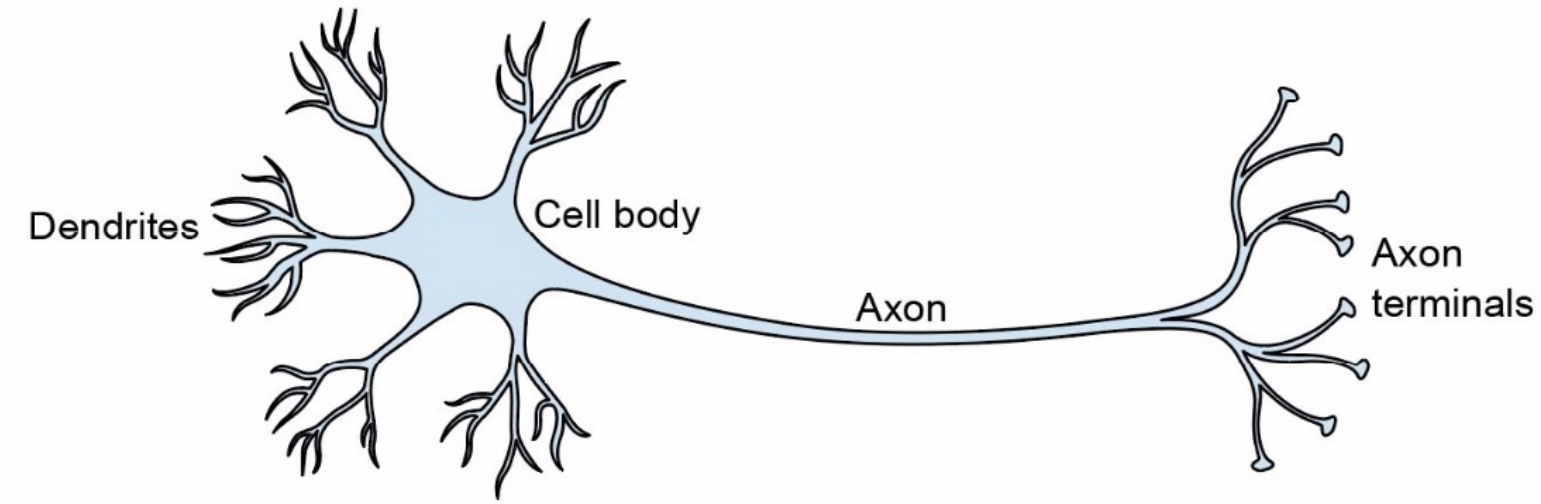


Figure 2.2 A single biological brain cell. The neuron receives the signal from other neurons via its dendrites shown on the left. If the cumulated signal exceeds a certain value, an impulse is sent via the axon to the axon terminals, which, in turn, couples to other neurons.

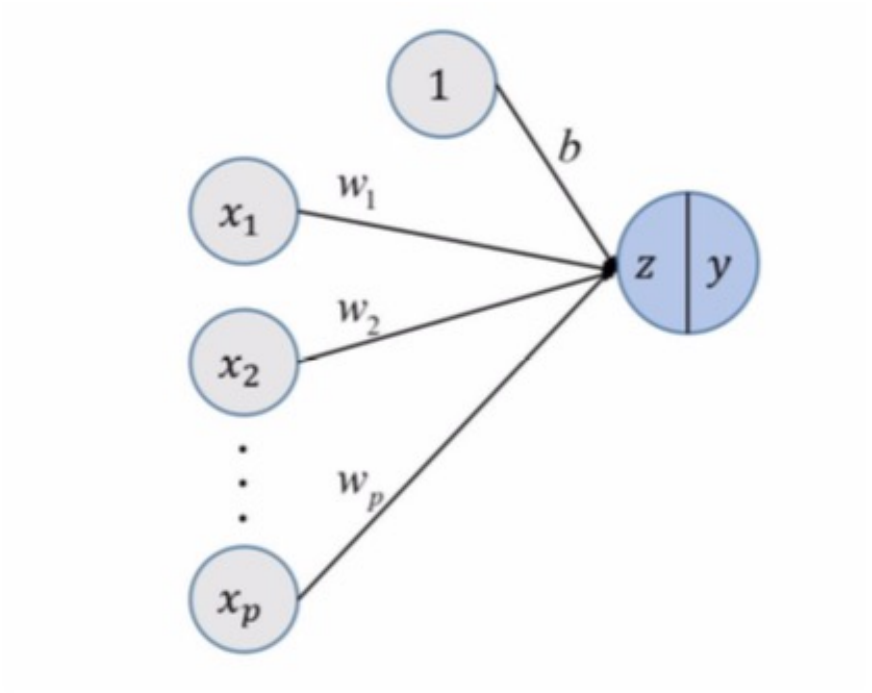
Neural networks are **loosely** inspired by how the brain works

The Single Cell: Mathematical Abstraction

$$z = b + x_1 \cdot w_1 + x_2 \cdot w_2 + \cdots x_p \cdot w_p$$

$$z = b + \sum x_i \cdot w_i = b + \boldsymbol{x} \cdot \boldsymbol{w}$$

Activation (many possibilities)



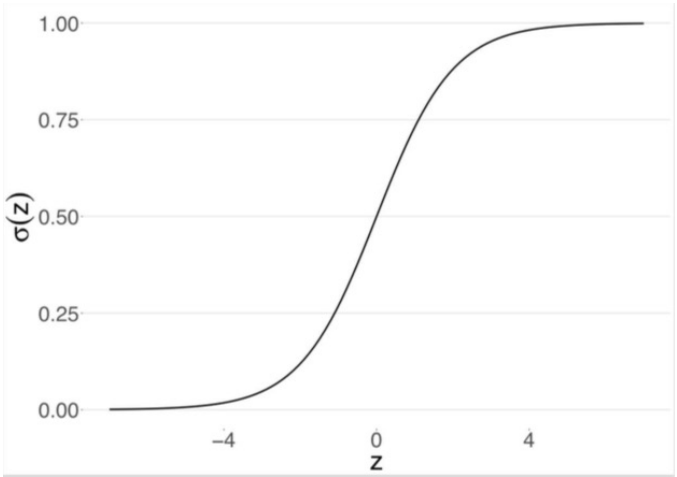


Figure 2.3 The mathematical abstraction of a brain cell (an artificial neuron). The value *z* is computed as the weighted sum of the *p* input values, *x*1 to *xp*, and a bias term *b* that shifts up or down the resulting weighted sum of the inputs. The value *y* is computed from *z* by applying an activation function.

```
# definition of the sigmoid function
def sigmoid(z):
    return (1 / (1 + np.exp(-z)))
```

Toy Task

- Task tell fake from real banknotes
- Banknotes described by two features

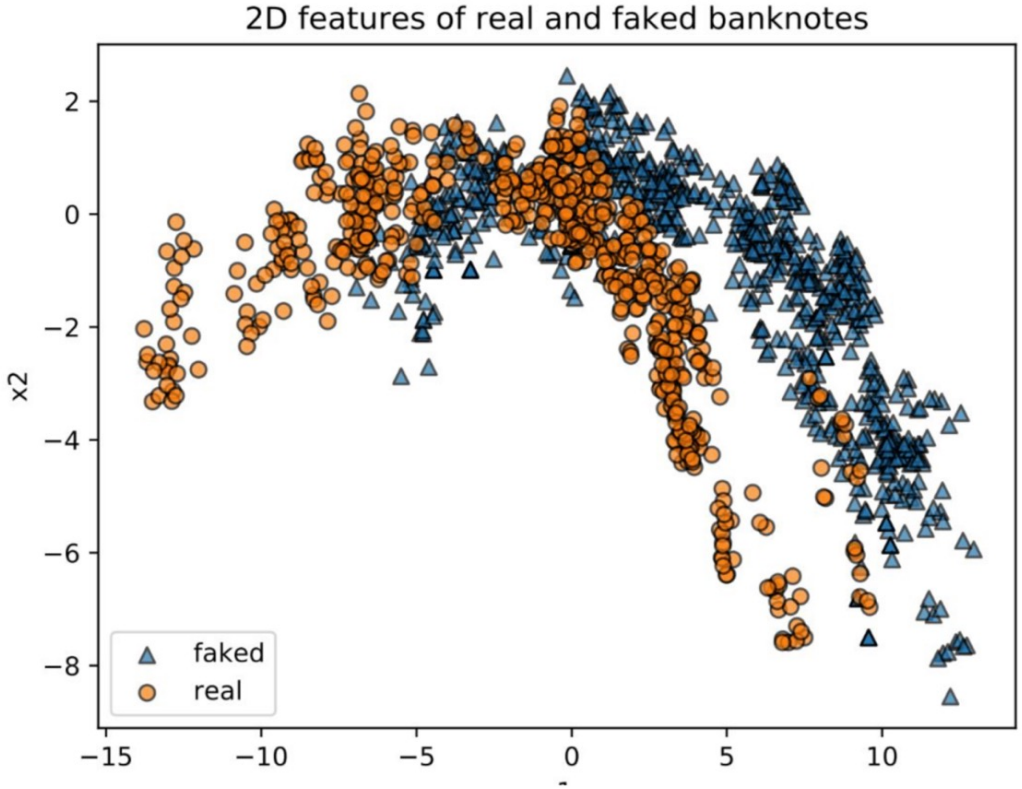
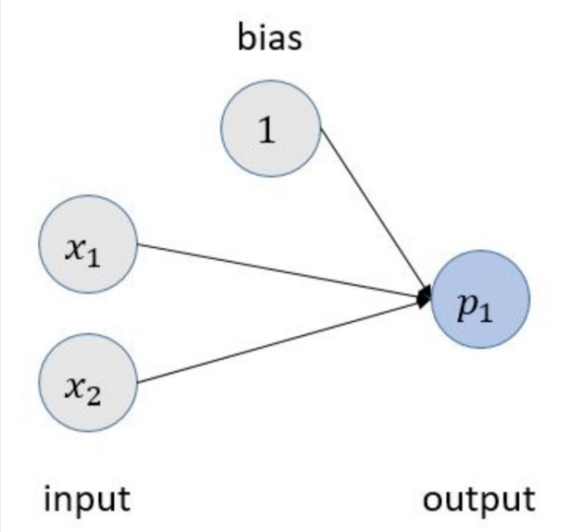
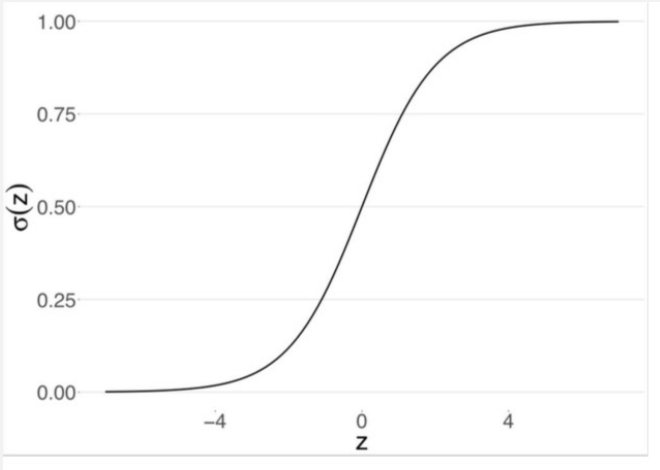


Figure 2.5 The (training) data points for the real and faked banknotes

https://github.com/tensorchiefs/dl_book/blob/master/chapter_02/nb_ch02_01.ipynb

Exercise: Part 1 [Pen and Paper]





Model: The above network models the **probability** p_1 that a given banknote is false.

TASK (with pen and paper)

The weights (determined by a training procedure later) are given by

$$w_1 = 0.3, w_2 = 0.1, \text{ and } b = 1.0$$

The probability can be calculated from z using the function sigmoid(z)

What is the probability that a banknote, that is characterized by x_1=1 and $x_2 = 2.2$, is a faked banknote?

GPUs love Vectors

$$F^{\mu\nu}$$

In Math:

$$p_1 = \text{sigmoid}\left(\begin{pmatrix} x_1 & x_2 \end{pmatrix} \cdot \begin{pmatrix} w_1 \\ w_2 \end{pmatrix} + b \right)$$

DL: better to have column vectors

In code:

```
## function to return the probability output after the matrix multiplication
def predict_no_hidden(X):
    return sigmoid(np.matmul(X,W)+b)
```

Bis hier hin erste Stunde 50 Minuten

Recap: Matrix Multiplication aka dot-product of matrices

We can only multiply matrices if their dimensions are compatible.

A × **B** = **C**

$(m \times \mathbf{n}) \times (\mathbf{n} \times p) = (m \times p)$

$$\mathbf{A}_{3x3} \times \mathbf{B}_{3x2} = \mathbf{C}_{3x2}$$

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} x \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \\ c_{31} & c_{32} \end{bmatrix}$$

$$c_{11} = a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31}$$
$$c_{12} = a_{11}b_{12} + a_{12}b_{22} + a_{13}b_{32}$$
$$c_{21} = a_{21}b_{11} + a_{22}b_{21} + a_{23}b_{31}$$
$$c_{22} = a_{21}b_{12} + a_{22}b_{22} + a_{23}b_{32}$$
$$c_{31} = a_{31}b_{11} + a_{32}b_{21} + a_{33}b_{31}$$
$$c_{32} = a_{31}b_{12} + a_{32}b_{22} + a_{33}b_{32}$$

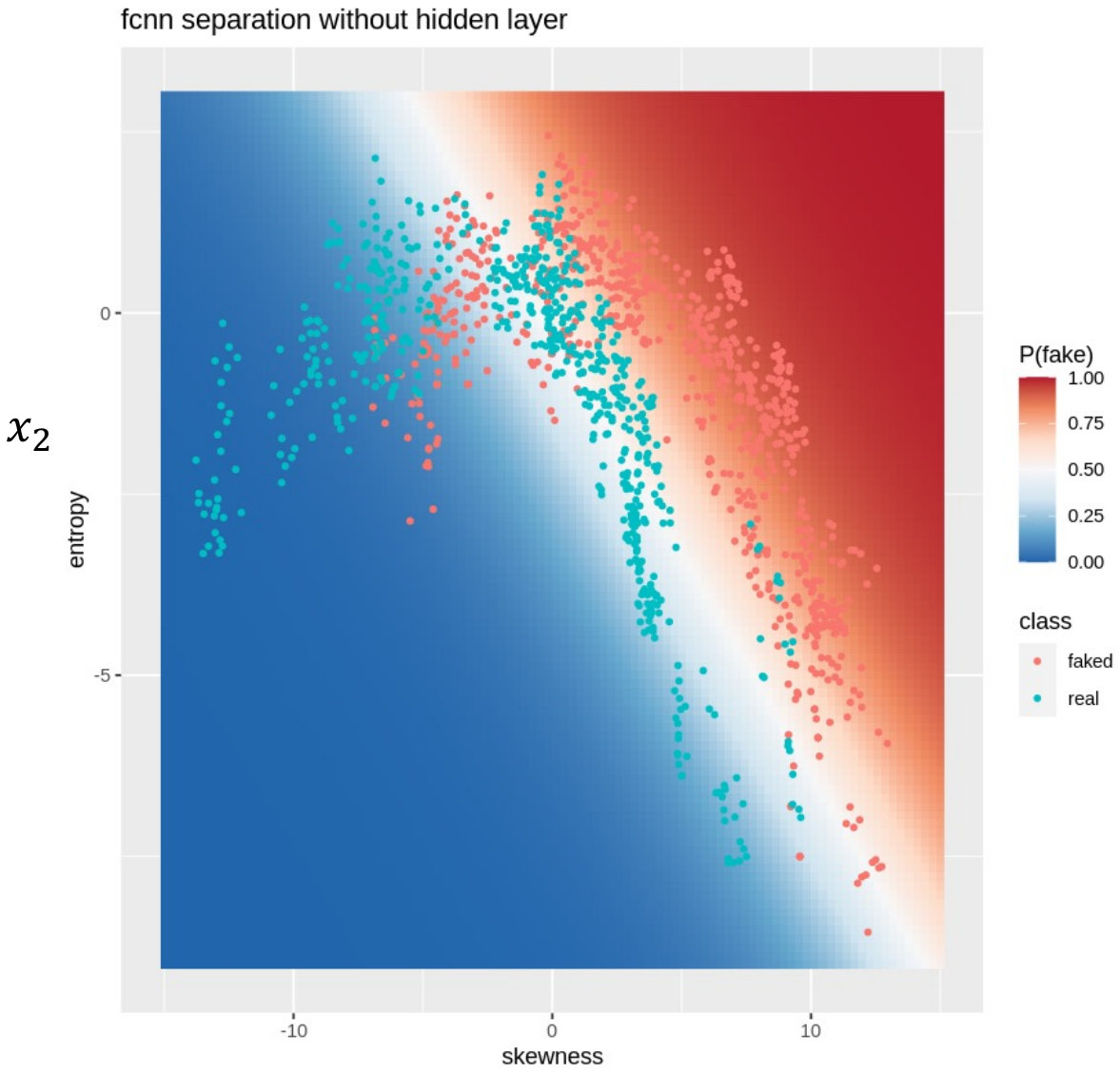
Example:

$$\mathbf{A}_{2x2} = \begin{pmatrix} 2 & 1 \\ 0 & 3 \end{pmatrix} \quad \mathbf{B}_{2x3} = \begin{pmatrix} 3 & 1 & 7 \\ 8 & 2 & 4 \end{pmatrix} \quad \mathbf{C}_{2x3} = \mathbf{A}_{2x2} \cdot \mathbf{B}_{2x3} = \begin{pmatrix} 11 & 4 & 18 \\ 24 & 6 & 12 \end{pmatrix}$$

Result (see later in the notebook)

General rule: Networks without hidden layer have linear decision boundary.

Result*

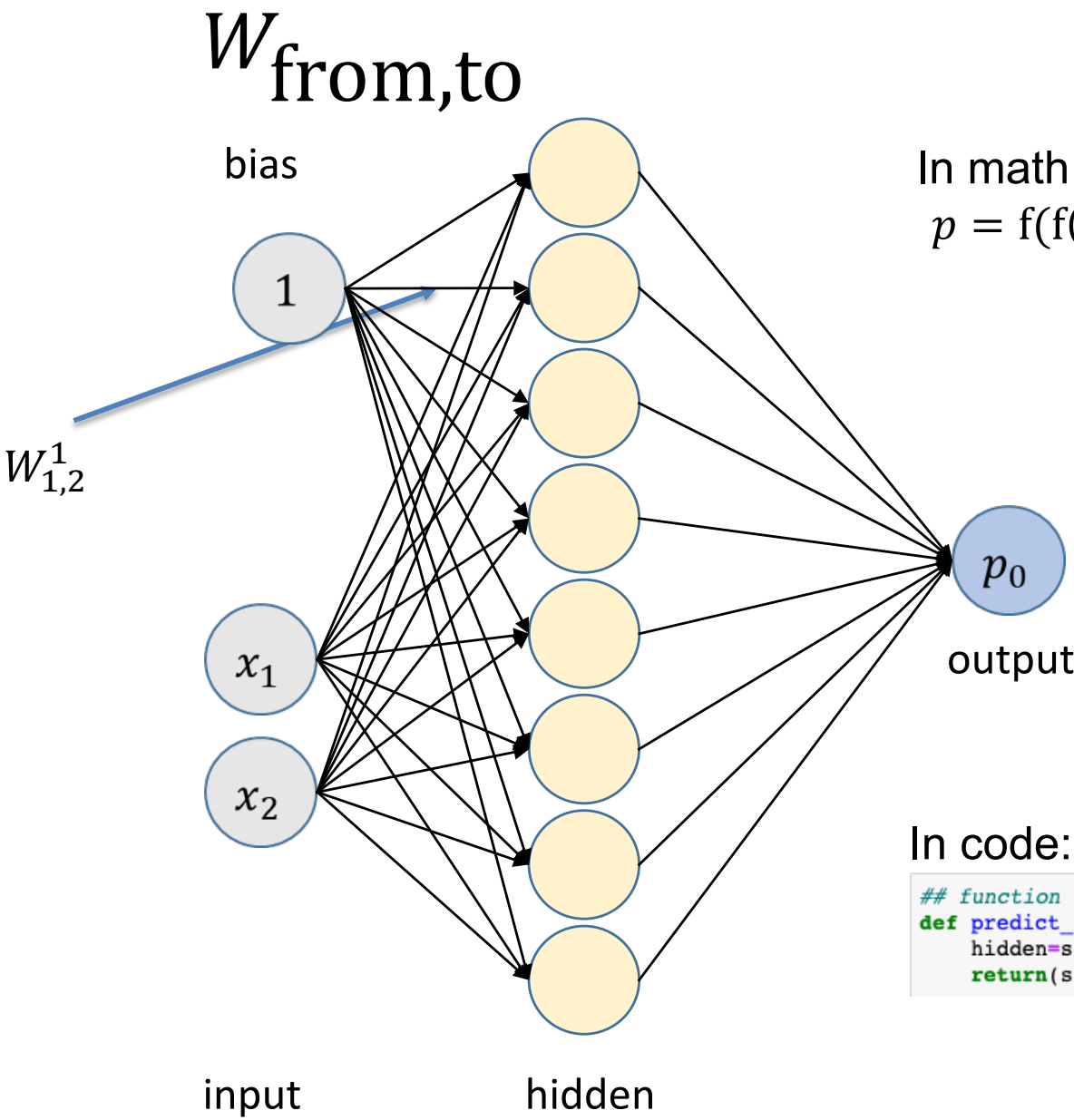


General rule: Networks without hidden layer have linear decision boundary.

*Details of training later


WE NEED TO GO
DEEPER
memegene

A first deep network



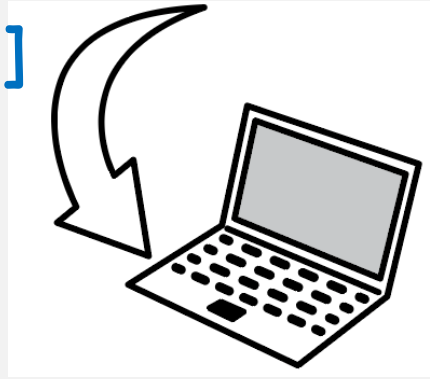
In math (f = sigmoid)

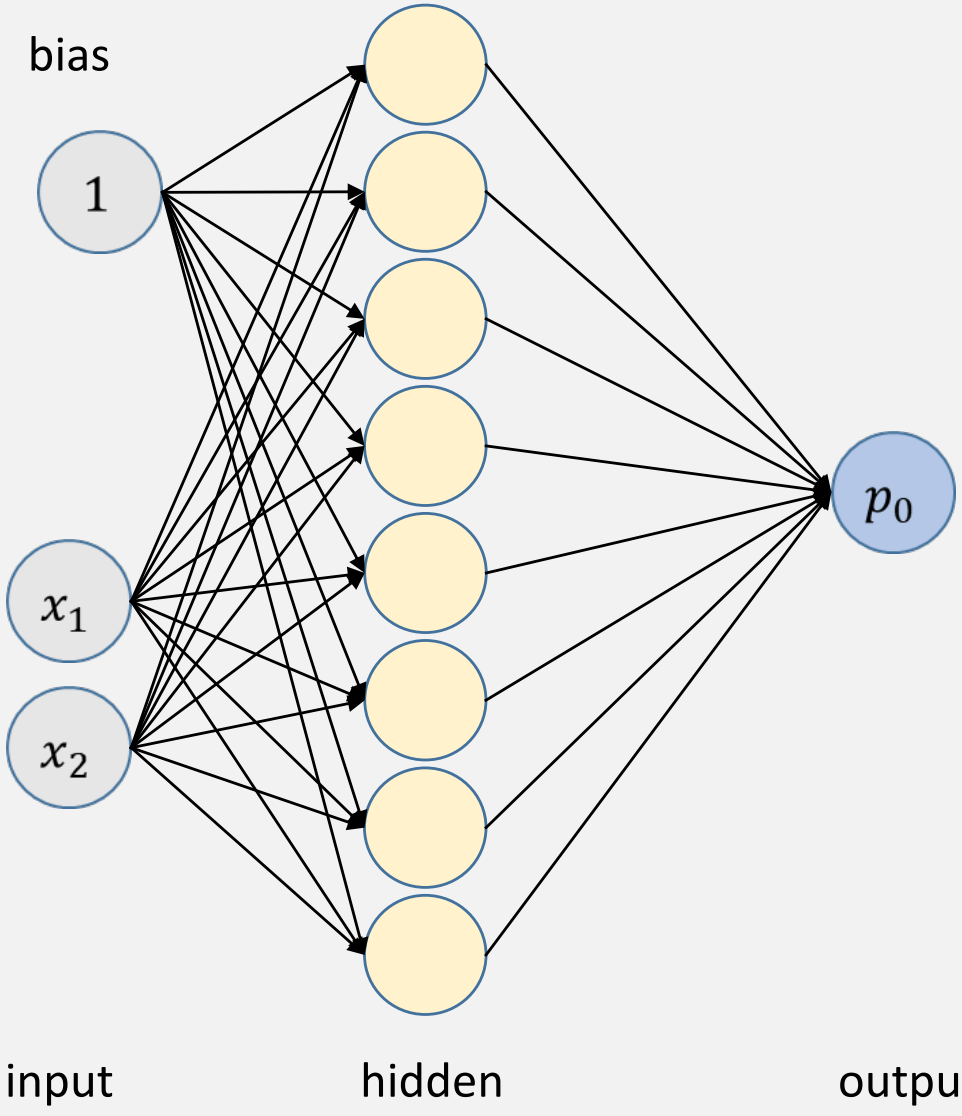
$$p = \mathrm{f}(\mathrm{f}(X \cdot W_1 + b_1) \cdot W_2 + b_2)$$

In code:

```
## function to return the probability output after the hidden layer
def predict_hidden(X):
    hidden=sigmoid(np.matmul(X,W1)+b1)
    return(sigmoid(np.matmul(hidden,W2)+b2))
```

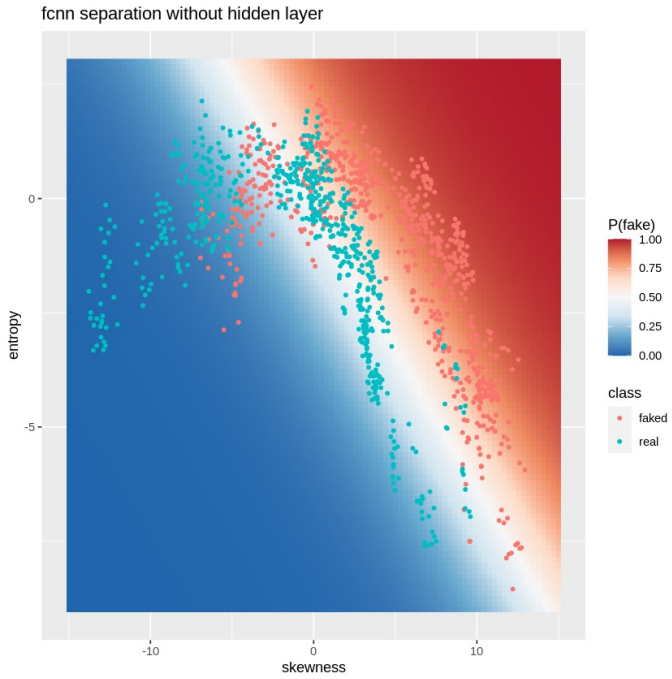
Exercise: Part 2 Including a hidden layer [30 min]

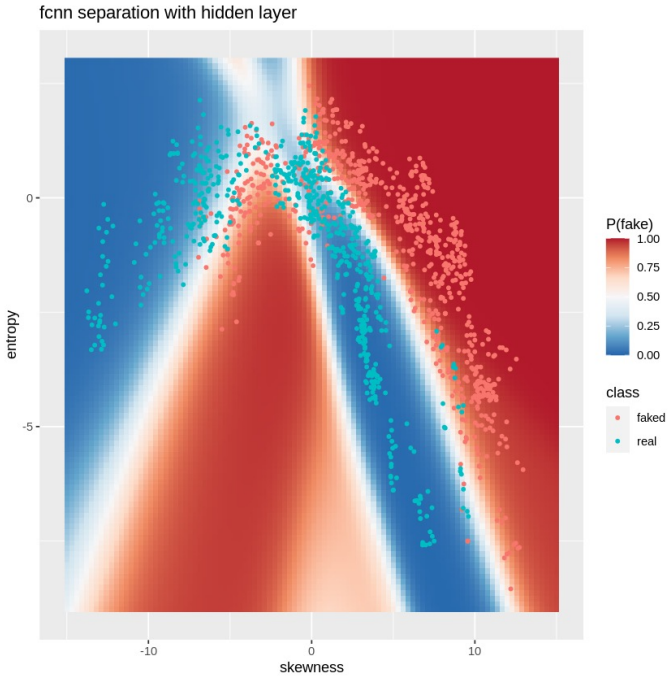


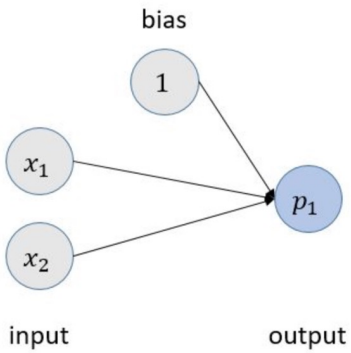


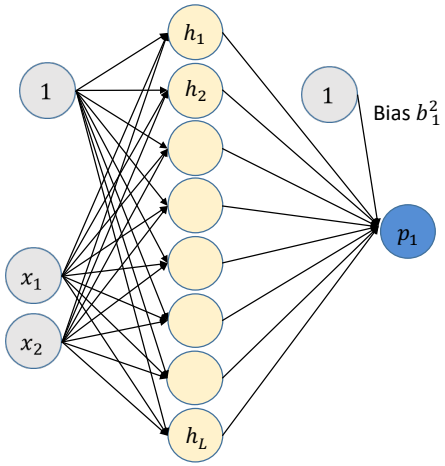
Open NB 01_simple_forward_pass.ipynb and do exercise stop before Keras

The benefit of hidden layers (Trained See next Lecture)

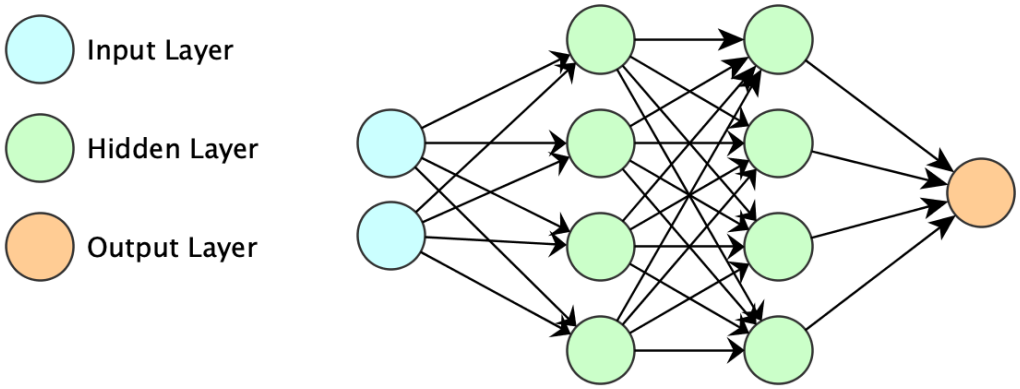








Structure of the network (with two hidden)



In code:

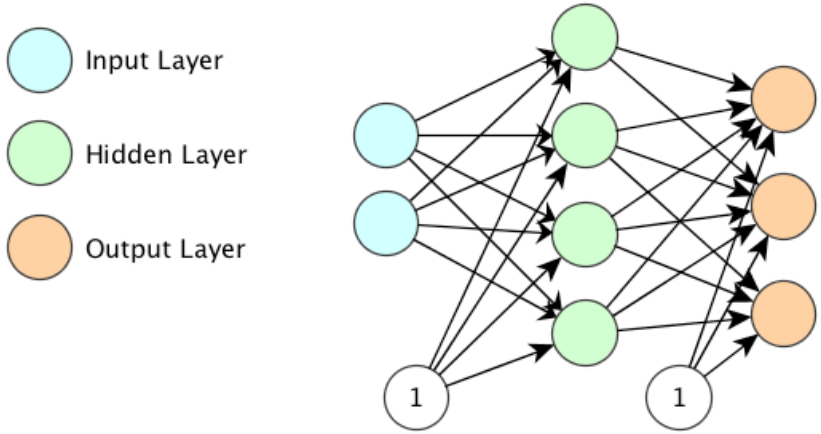
```
## Solution 2 hidden layers
def predict_hidden_2(X):
    hidden_1=sigmoid(np.matmul(X,W1)+b1)
    hidden_2=sigmoid(np.matmul(hidden_1,W2)+b2)
    return(sigmoid(np.matmul(hidden_2,W3)+b3))
```

In math (f = sigmoid) and b1=b2=b3=0

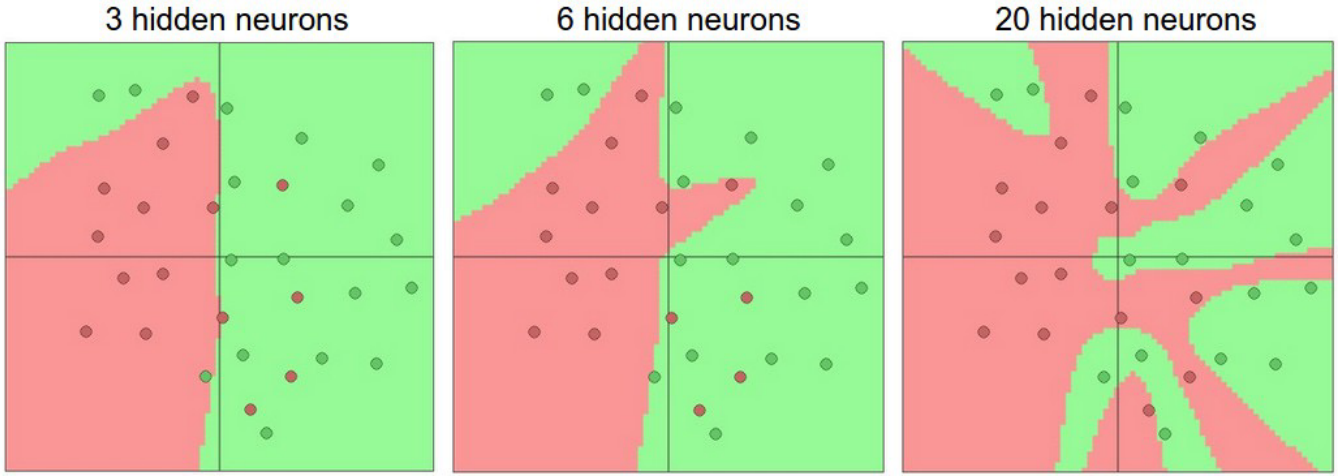
$$p = f(f(f(\mathrm{x}\,\mathrm{W}^1)\mathrm{W}^2))$$

Looks a bit like onions, matryoshka (Russian Dolls) or lego bricks

One hidden Layer

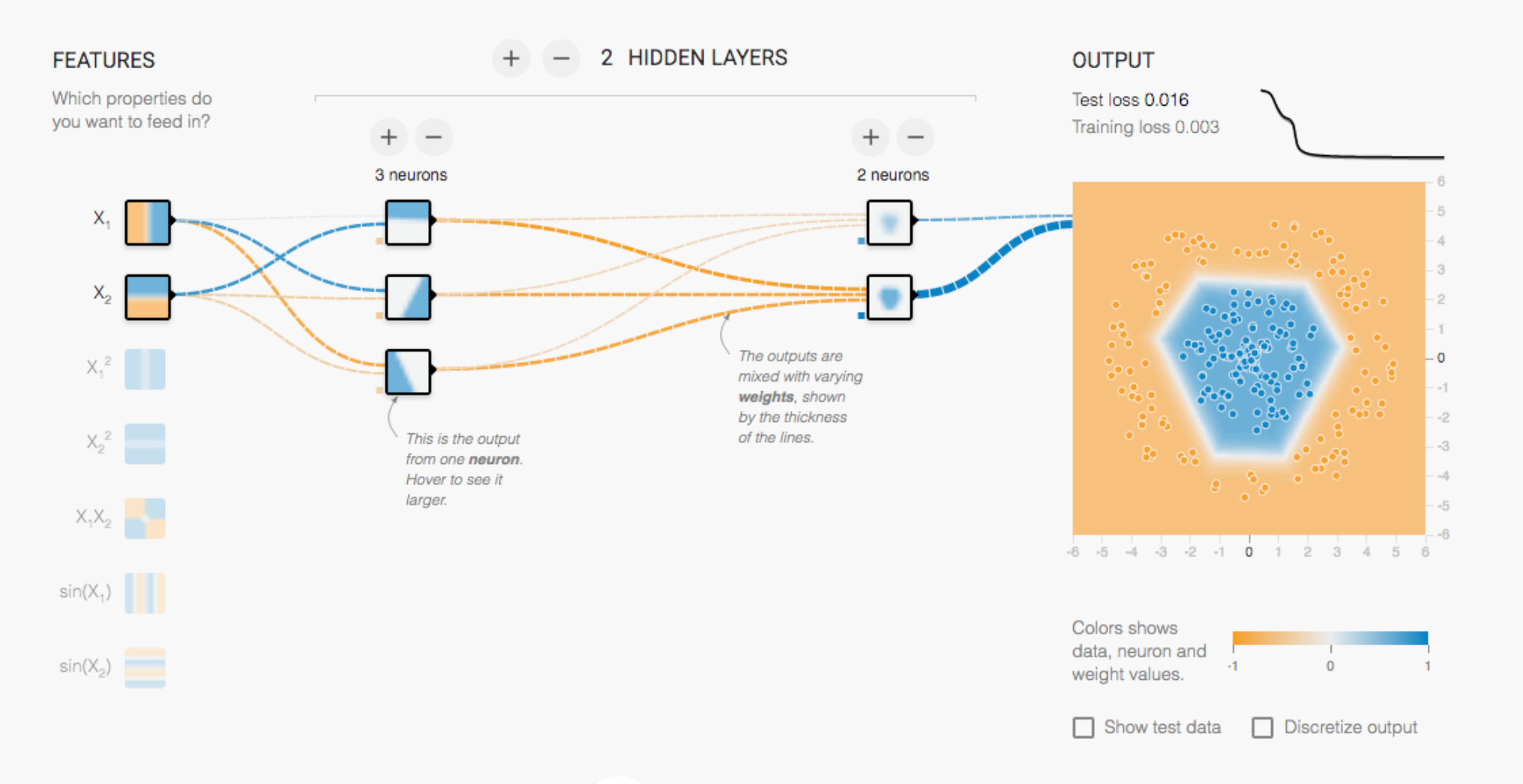


A network with one hidden layer is a universal function approximator!



http://cs231n.github.io/neural-networks-1/

Experiment yourself (homework)



http://playground.tensorflow.org

Let's you explore the effect of hidden layers

To go deep non-linear activation functions are needed

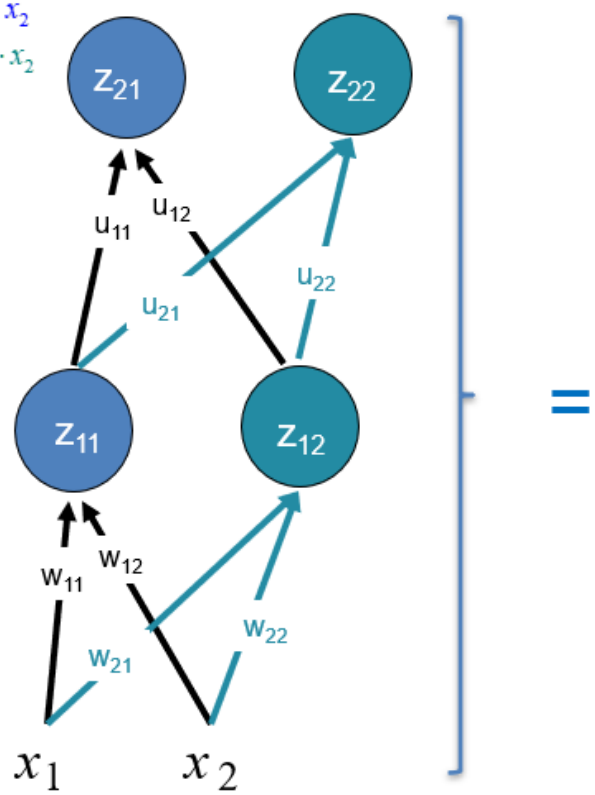
2 linear layers can be replaced by 1 linear layer -> can't go deep with linear layers!

$$z = (x \cdot W) \cdot U = x \cdot (W \cdot U) = x \cdot V$$

$$z_{21} = z_{11} \cdot u_{11} + z_{12} \cdot u_{12} = (w_{11} \cdot x_1 + w_{12} \cdot x_2) \cdot u_{11} + (w_{21} \cdot x_1 + w_{22} \cdot x_2) \cdot u_{12}$$
$$= x_1 \cdot (w_{11} \cdot u_{11} + w_{21} \cdot u_{12}) + x_2 \cdot (w_{12} \cdot u_{11} + w_{22} \cdot u_{12})$$

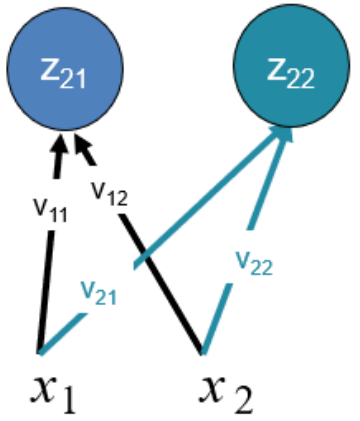
$z_{11} = w_{11} \cdot x_1 + w_{12} \cdot x_2$

$z_{12} = w_{21} \cdot x_1 + w_{22} \cdot x_2$



=

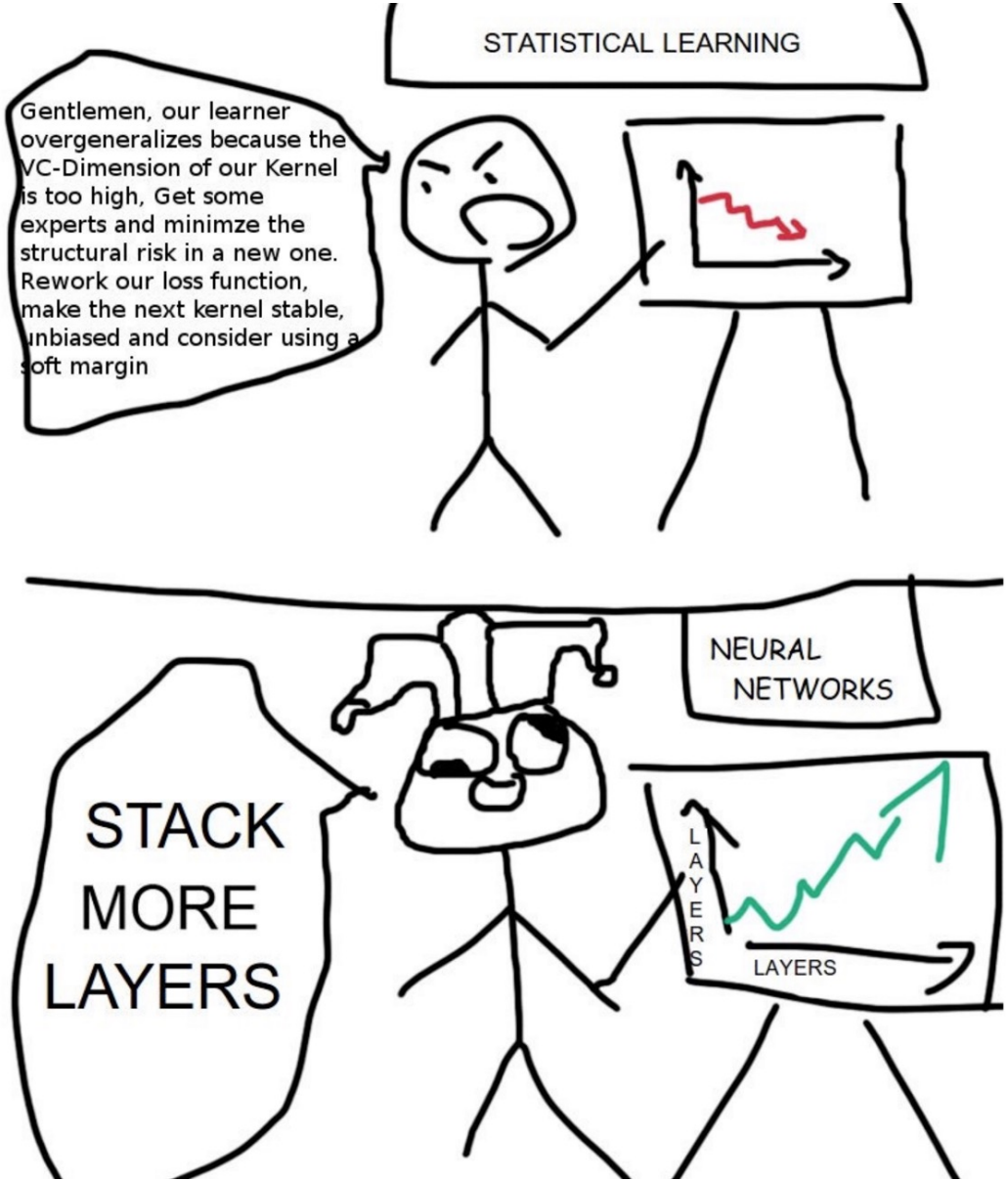
$z_{21} = v_{11} \cdot x_1 + v_{12} \cdot x_2$



$$v_{11} = w_{11} \cdot u_{11} + w_{21} \cdot u_{12}$$
$$v_{12} = w_{12} \cdot u_{11} + w_{22} \cdot u_{12}$$
$$v_{21} = w_{11} \cdot u_{21} + w_{21} \cdot u_{22}$$
$$v_{22} = w_{12} \cdot u_{21} + w_{22} \cdot u_{22}$$

Remark: biases are ignored here, but do not change fact

DL vs Machine Learning Meme

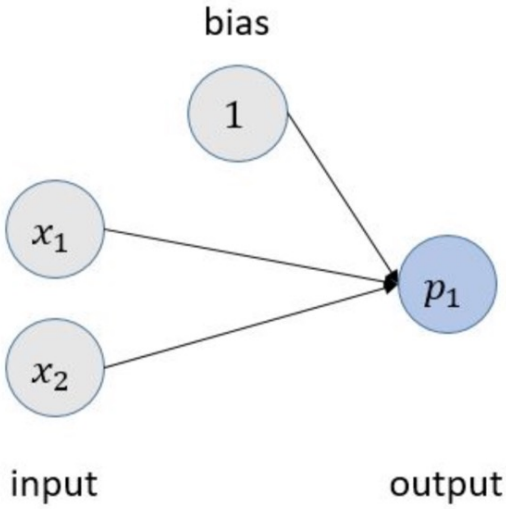


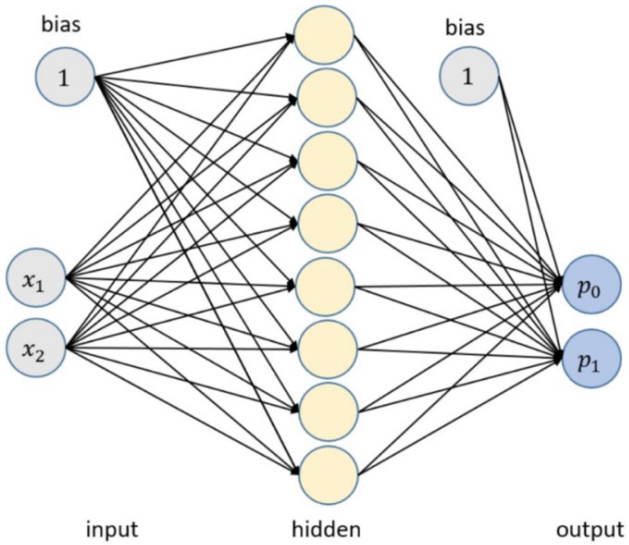
https://www.reddit.com/r/ProgrammerHumor/comments/8c1i45/stack_more_layers/

Training NN
Second part (starting with hour 3)

How to determine the weights

- The output of a NN is defined by the weights and biases
- These weights and biases are tuned so that the NN bests fits the training data
- The goodness of fit to the training data is quantified by a loss

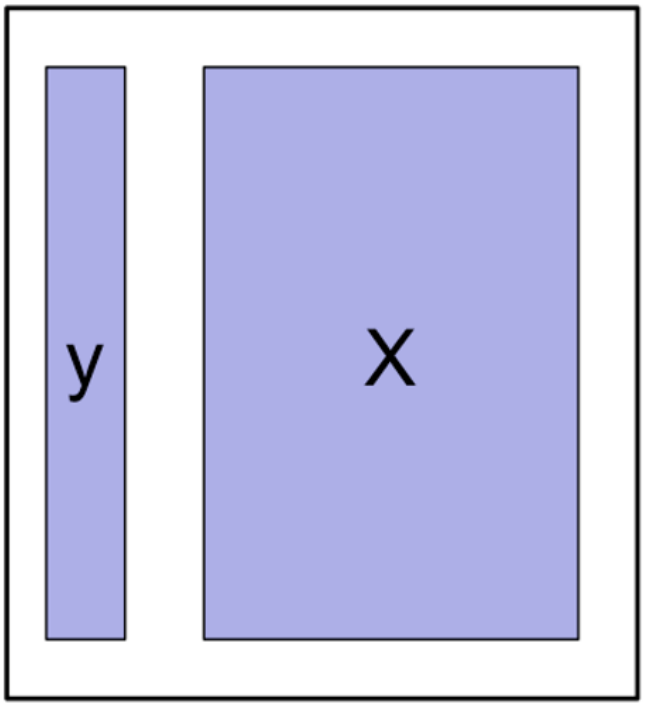




Loss Functions

Tasks in DL

- The loss function depends on the task
- 2 Main tasks in DL predict y given x
  - Regression
    - Predict a number*
  - Classification
    - Predict a class*



Supervised Learning

*Later we refine this notion

Example Regression: Estimate Age From Picture

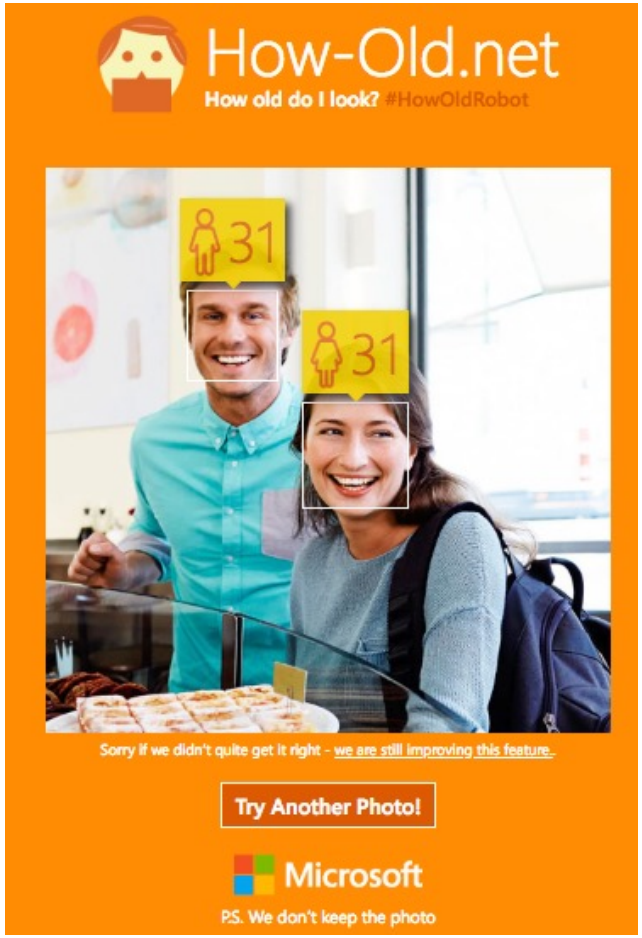
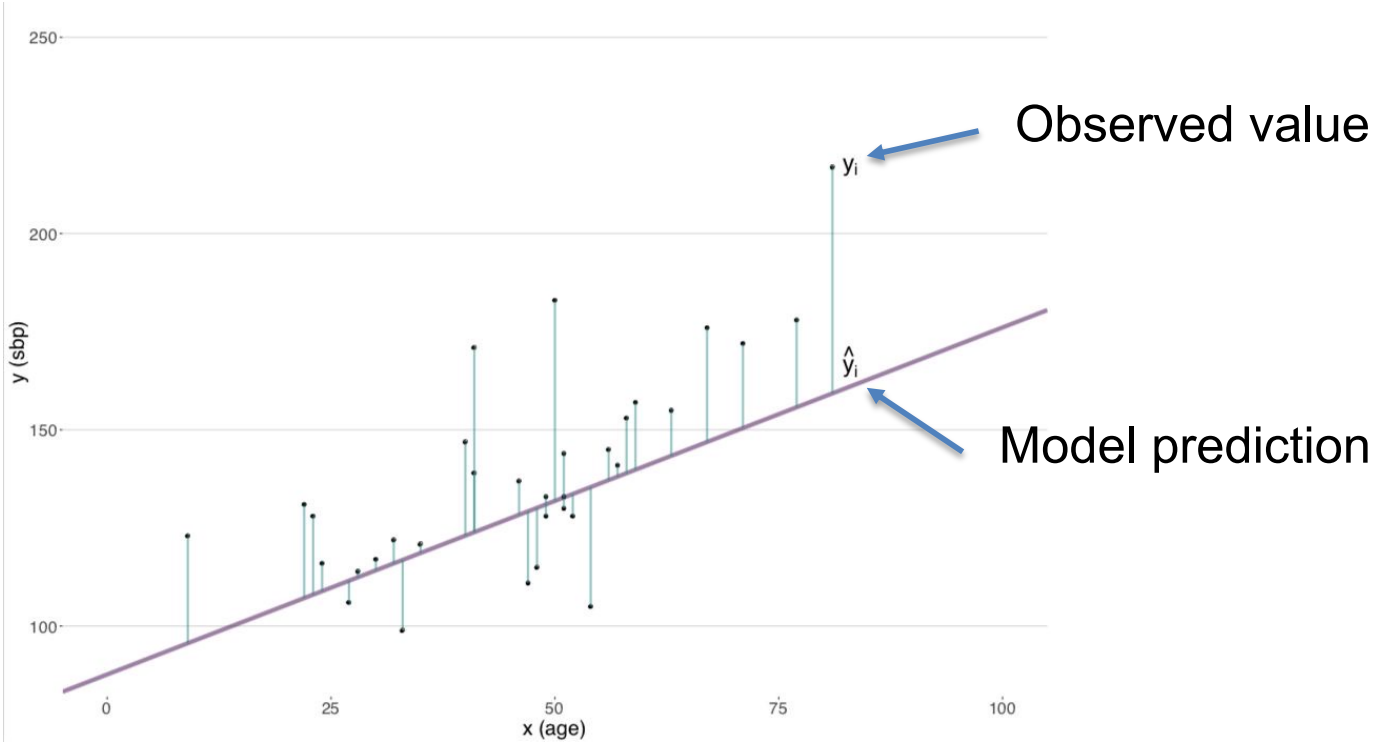


Image --> Age

https://www.how-old.net/

Loss Functions for Regression Problems

- Regression = Predict a number
- Example (Linear Regression)
  - Blood pressure of 31 women vs. age (training data)



Loss: Mean Squared Error (MSE) $\frac{1}{N}\sum_i (y_i - \widehat{y_i})^2$ also for regression problems (not only linear regression)

Classification

- Predict class
- Usually in DL the model predicts a probability for a class
- Example:
  - Banknote from exercise
  - Typical example Number from hand-written digit

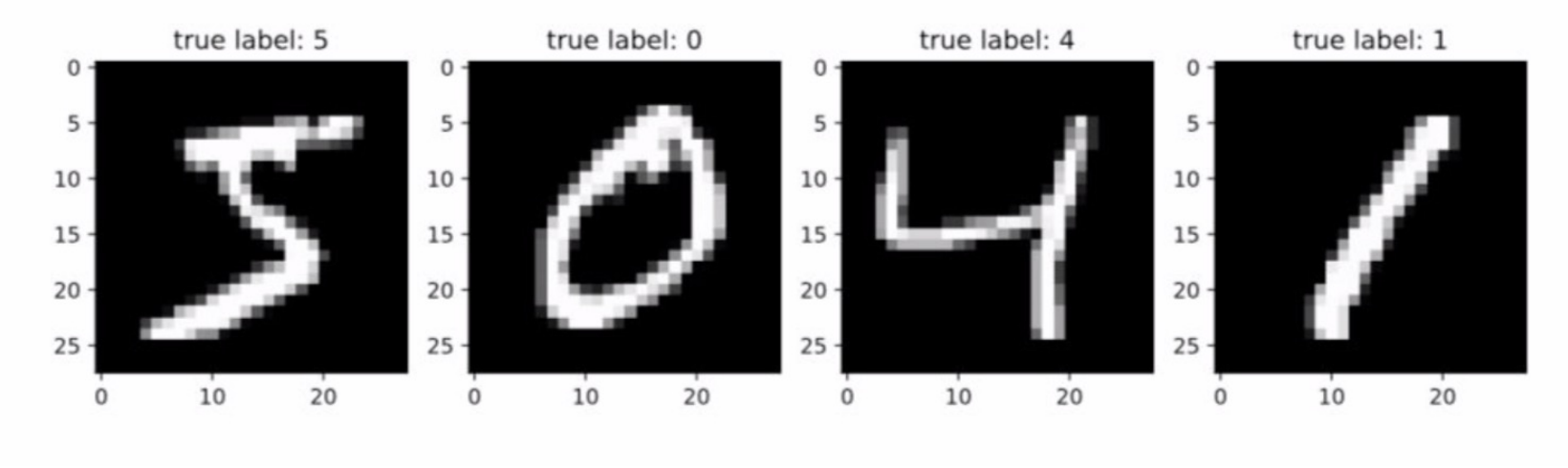
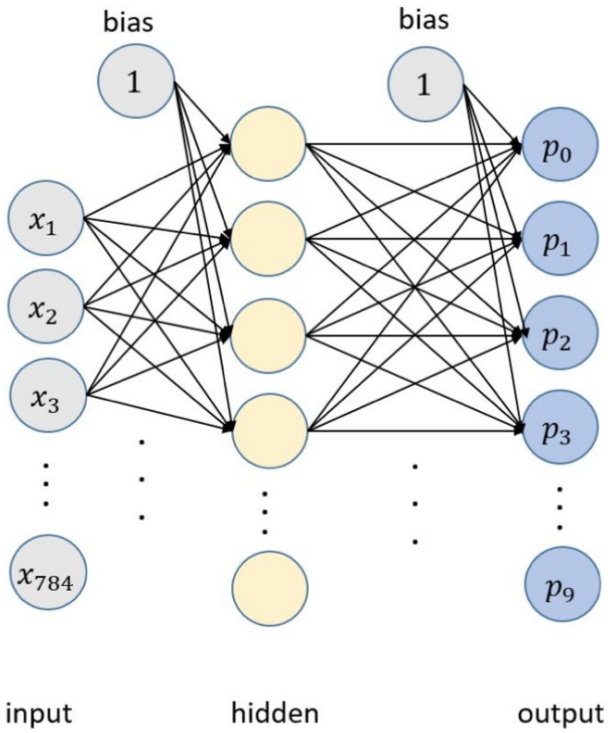


Figure 2.11 The first four digits of the MNIST data set—the standard data set used for benchmarking NN for images classification

Classification: Softmax Activation



$p_o, p_1 \ldots p_9$ are probabilities for the classes 0 to 9.

Activation of last layer z_i incomming

$$p_i = \frac{e^{z_i}}{\sum_{j=0}^{9} e^{z_j}}$$

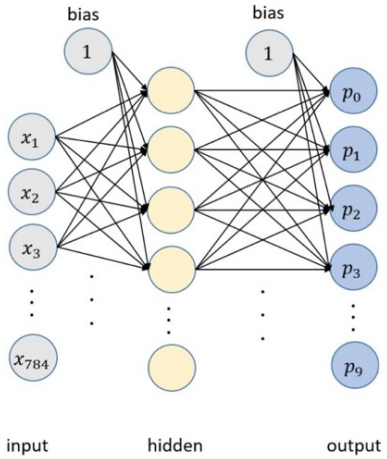
Makes outcome positive

Ensures that pi's sum up to one

This activation is called softmax

Figure 2.12: A fully connected NN with 2 hidden layers. For the MNIST example, the input layer has 784 values for the 28 x 28 pixels and the output layer out of 10 nodes for the 10 classes.

Loss for classification ('categorical cross-entropy')



$p_o, p_1 \ldots p_9$ are probabilities for the classes 0 to 9.

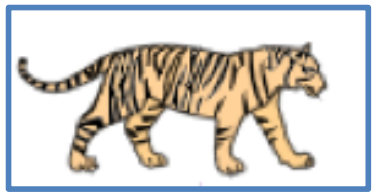
- As MSE Loss is averaged of individual losses l_i of training data $\mathrm{i} = 1, \ldots N$
- Want l_i
  - 0 for perfect match, i.e. predicts class of training example $y^{(i)}$ with probability 1
  - ∞ for worst match, i.e. predicts class $y^{(i)}$ with probability 0
  - $l_i = -\log\left(p_{model}\left(y^{(i)}\middle|x^{(i)}\right)\right)$
  - loss = $\frac{1}{N}\sum l_i$

Training / Gradient Descent

Idee of training: Image classification

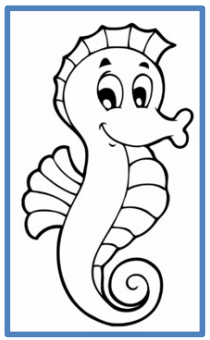
True Label

Vorhergesagtes Label

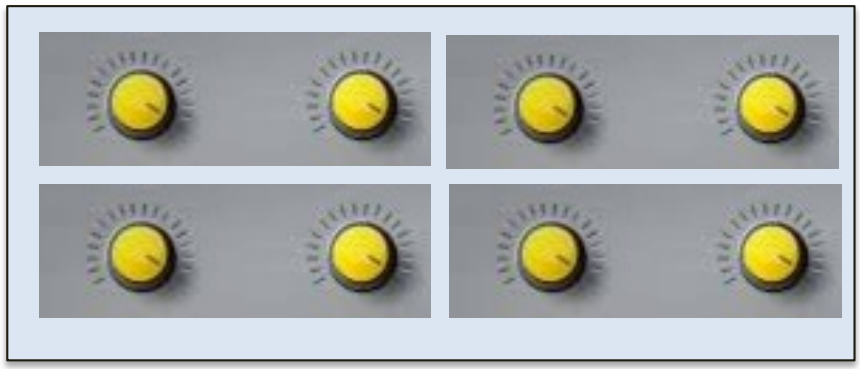


Tiger

Tiger



Seah orse



→ Dog 👎

→ Tiger 👍

→ Seahorse 👍

...

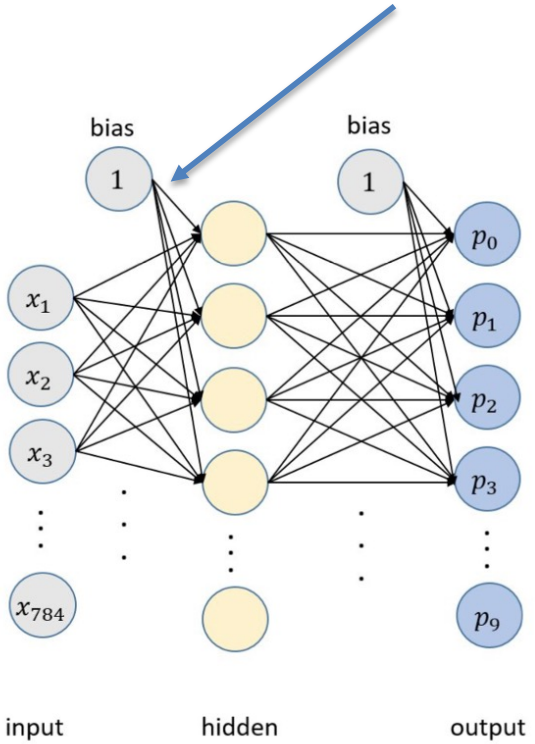
Typical 1 Mio. Trainsdata

Pincipal:
Parameters are set so that as few errors as possible are made in the training data.

Optimization in DL

- DL many parameters
  - Optimization by gradient descent
- Algorithm
  - Take a batch of training examples
  - Calculate the loss of that batch
  - Tune the parameters so that loss gets minimized

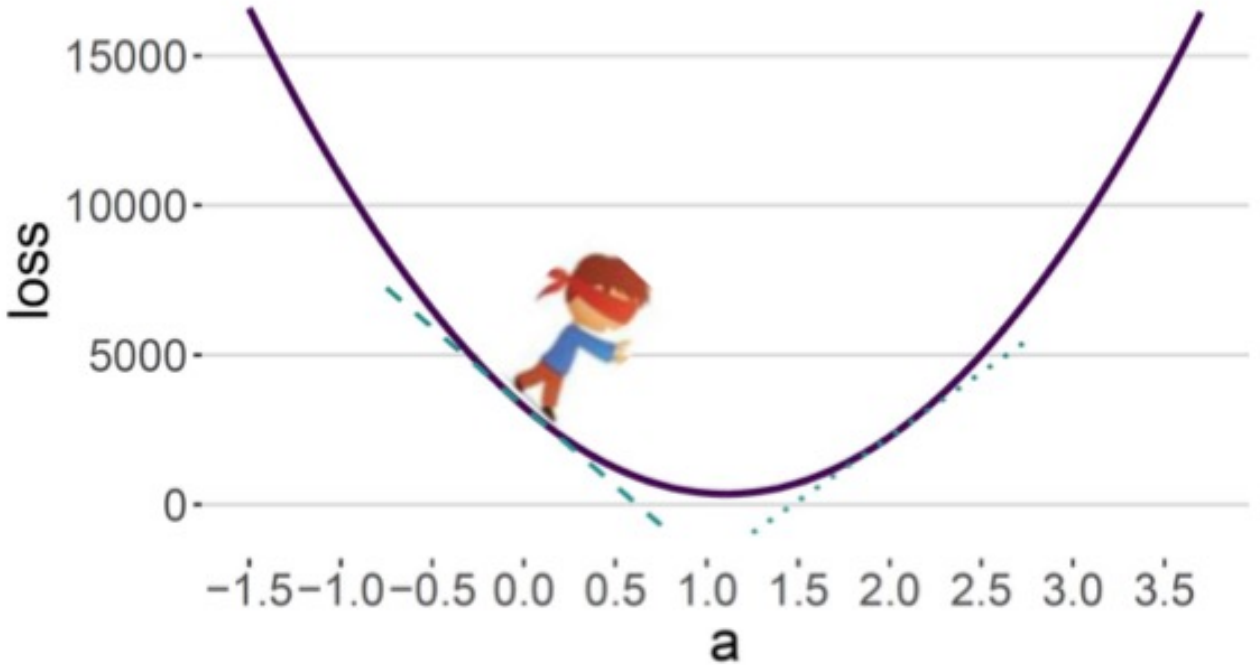
Parameters of the network are the weights.



Modern Networks have Billions (10^9) of weights. Record 2020 1.5E9 https://openai.com/blog/better-language-models/

Idea of gradient descent

- Shown loss function for a single parameter *a*



- Take a large step if slope is steep (you are away from minimum)
- Slope of loss function is given by gradient
- Iterative update of the parameters
  - $a_{t+1} = a_t - \eta \; grad\_a(loss)$

Backpropagation

- There is an efficient way to update all parameters of the network
- This is called Backpropagation

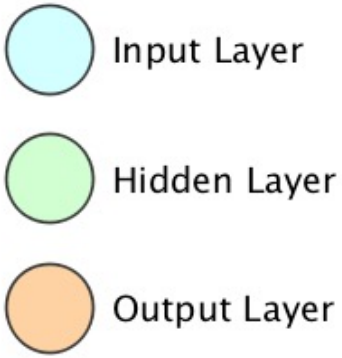
- We need to calculate the derivative of the loss function w.r.t. all weights

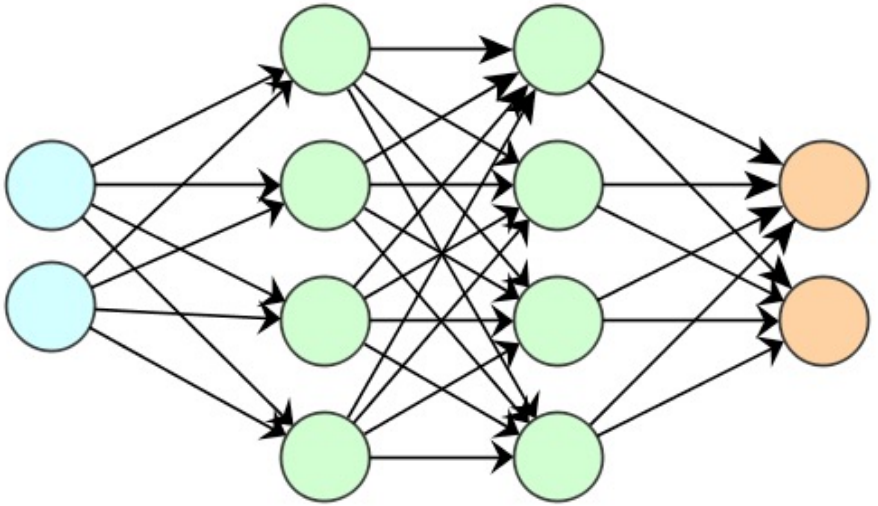
- Doing this efficiently (on graphic cards GPU) by hand is tedious

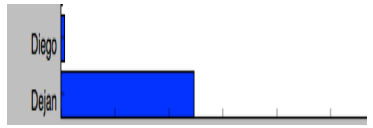
- Enter:
  - Deep Learning Frameworks

Deep Learning Frameworks

Recap: The first network







- The input: e.g. intensity values of pixels of an image
  - (Almost) no pre-processing
- Information is processed layer by layer from building blocks
- Output: probability that image belongs to certain class
- Arrows are weights (these need to be learned / training)

Deep Learning Frameworks (common)

- Computation needs to be done on GPU or specialized hardware (compute performance)
- On GPU: almost exclusively on NVIDIA using the cuda library, cudnn
- Data Structure are multidimensional arrays (*tensors*) which are manipulated
- Learning require to calculate derivatives of the network w.r.t parameters.

In this course: TensorFlow with Keras

Low Level Deep Learning Libraries for Tensor Manipulations

- Torch / pytorch
  - Facebook, quite flexible, lua (Jan 2017 also in python, **pytorch**)
  - Dynamic computational graph
- TensorFlow
  - Open sourced by Google Dec 2015
  - TF 1.x static computational graphs
  - Since version 2.0 also dynamic computational graphs in eager mode
- JAX
  - New kid on the block
  - Numpy on stereoids (GPU / TPU replacement)
- Chainer
  - Flexible, build graph on the fly
- MXNet
  - Can be used in many languages, build graph on the fly?
- Caffe
  - Inflexible, good of CV
  - Calculate gradients by hand
- Theano
  - Around since 2008,
  - Active development abandoned
  - Slow compiling of graph (due to optimization)

We will work (mainly) with high level libraries Keras ontop of TensorFlow

TensorFlow

What is TensorFlow

- It's API about ***tensors***, which ***flow*** in a ***computational graph***

https://www.tensorflow.org/

- What is a computational graph? But first..
- What are **tensors**?

What is a tensor?

In this course we only need the simple and easy accessible definition of Ricci:

Definition. A tensor of type (*p*, *q*) is an assignment of a multidimensional array

$$T^{i_1\dots i_p}_{j_1\dots j_q}[\mathbf{f}]$$

to each basis $\mathbf{f} = (\mathbf{e}_1,\dots,\mathbf{e}_n)$ of a fixed *n*-dimensional vector space such that, if we apply the change of basis

$$\mathbf{f} \mapsto \mathbf{f}\cdot R = \left(\mathbf{e}_i R^i_1, \dots, \mathbf{e}_i R^i_n\right)$$

then the multidimensional array obeys the transformation law

$$T^{i'_1\dots i'_p}_{j'_1\dots j'_q}[\mathbf{f}\cdot R] = (R^{-1})^{i'_1}_{i_1}\cdots(R^{-1})^{i'_p}_{i_p}\, T^{i_1,\dots,i_p}_{j_1,\dots,j_q}[\mathbf{f}]\, R^{j_1}_{j'_1}\cdots R^{j_q}_{j'_q}.$$

Just kidding...

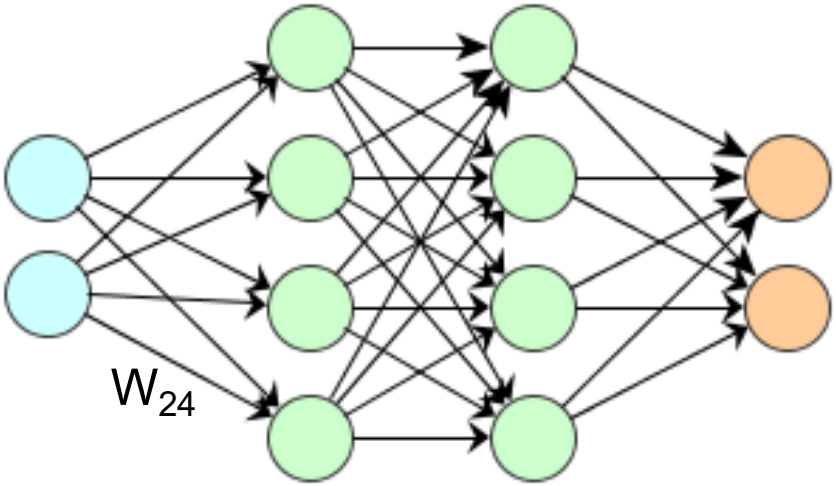
Sharpe, R. W. (1997). Differential Geometry: Cartan's Generalization of Klein's Erlangen Program. Berlin, New York: Springer-Verlag. p. 194. ISBN 978-0-387-94732-7.

What is a tensor?

For TensorFlow: A tensor is an array with several indices (like in numpy).
Order (a.k.a rank) are number of indices and shape is the range.

```
In [1]: import numpy as np

In [2]: T1 = np.asarray([1,2,3]) #Tensor of order 1 aka Vector
        T1

Out[2]: array([1, 2, 3])

In [3]: T2 = np.asarray([[1,2,3],[4,5,6]]) #Tensor of order 2 aka Matrix
        T2

Out[3]: array([[1, 2, 3],
               [4, 5, 6]])

In [4]: T3 = np.zeros((10,2,3)) #Tensor of order 3 (Volume like objects)

In [6]: print(T1.shape)
        print(T2.shape)
        print(T3.shape)

        (3,)
        (2, 3)
        (10, 2, 3)
```

Typical Tensors in Deep Learning

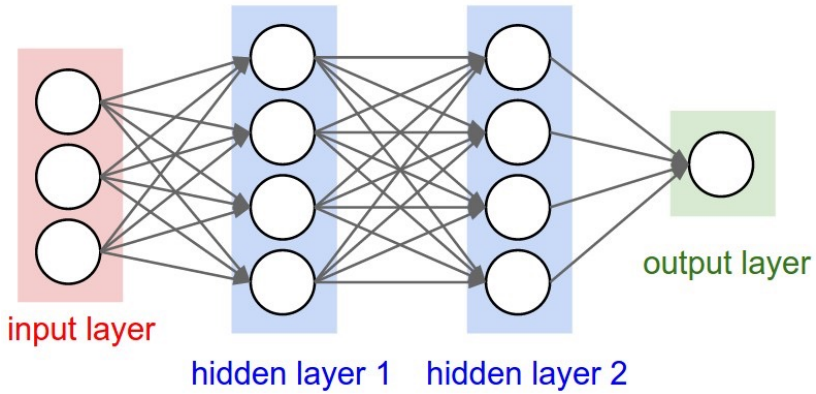


- The input can be understood as a vector
- A mini-batch of size 64 of input vectors can be understood as tensor of order 2
  - (index in batch, x_j)
- The weights going from e.g. Layer L_1 to Layer L_2 can be written as a matrix (often called W)
- A mini-batch of size 64 images with 256,256 pixels and 3 color-channels can be understood as a tensor of order 4.

Introduction to Keras

Keras as High-Level library to TensorFlow

- We use Keras as high-level library
- Libraries make use of the Lego like block structure of networks



High Level Libraries

- Keras
  - Keras is now part of TF core
  - https://keras.io/

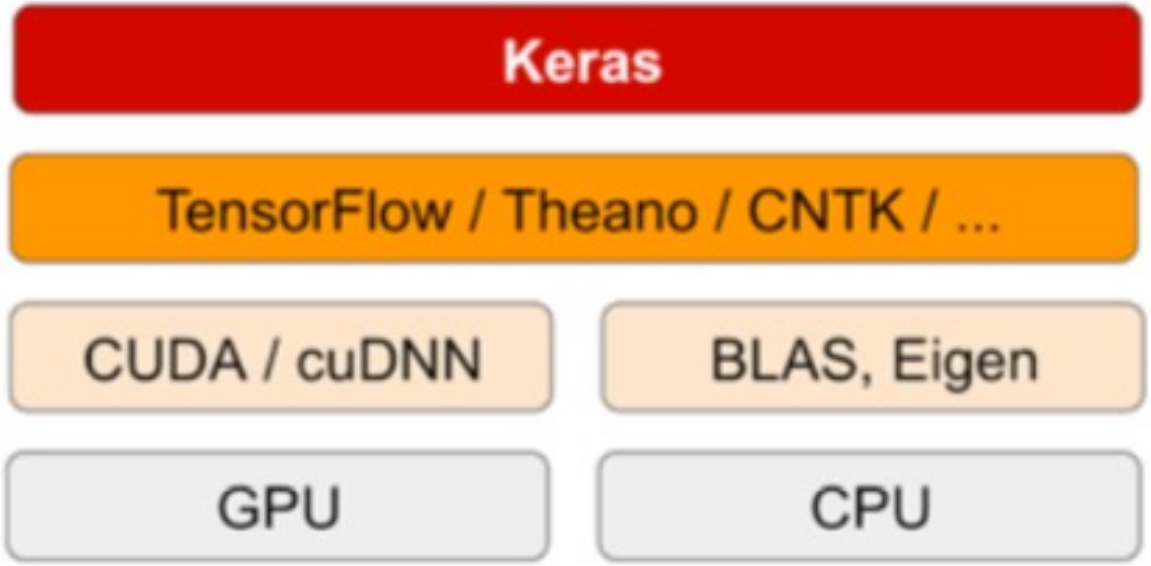


Figure: From Deep Learning with Python, Francois Chollett

The Keras user experience [marketing]

The Keras user experience

Keras is an API designed for human beings, not machines. Keras follows best practices for reducing cognitive load: it offers consistent & simple APIs, it minimizes the number of user actions required for common use cases, and it provides clear and actionable feedback upon user error.

This makes Keras easy to learn and easy to use. As a Keras user, you are more productive, allowing you to try more ideas than your competition, faster -- which in turn helps you win machine learning competitions.

This ease of use does not come at the cost of reduced flexibility: because Keras integrates with lower-level deep learning languages (in particular TensorFlow), it enables you to implement anything you could have built in the base language. In particular, as tf.keras, the Keras API integrates seamlessly with your TensorFlow workflows.

https://web.stanford.edu/class/cs20si/lectures/march9guestlecture.pdf

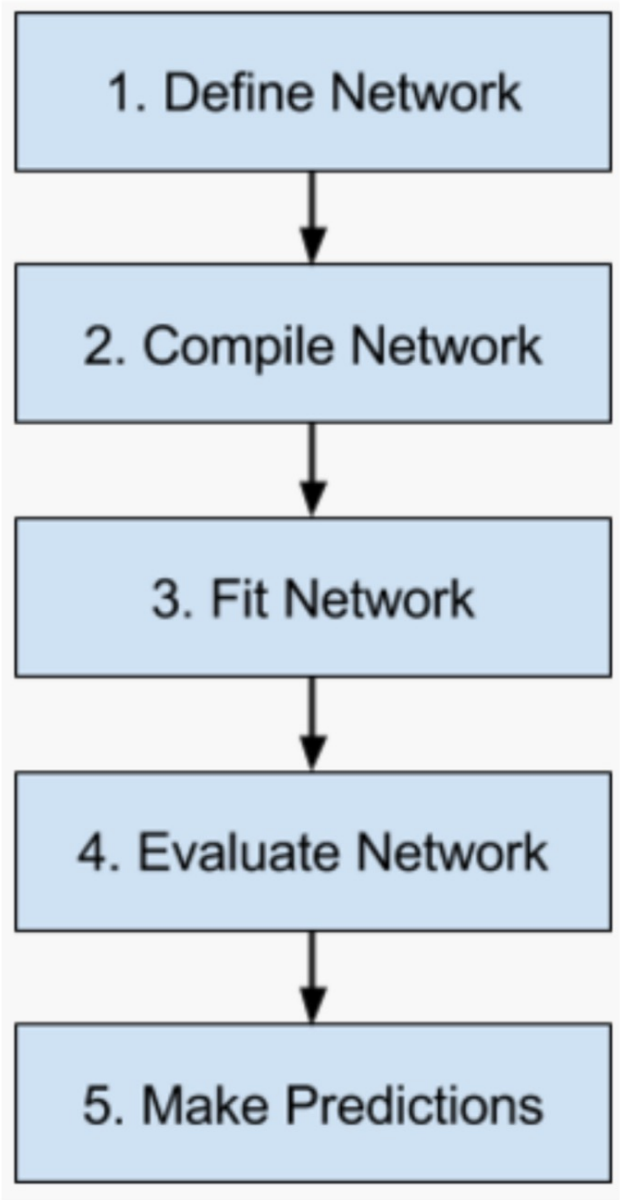
[marketing]

Keras is multi-backend, multi-platform

- Develop in Python, R
  - On Unix, Windows, OSX
- Run the same code with...
  - TensorFlow
  - CNTK
  - Theano
  - MXNet
  - PlaidML
  - ??
- CPU, NVIDIA GPU, AMD GPU, TPU...

https://web.stanford.edu/class/cs20si/lectures/march9guestlecture.pdf

Keras Workflow



Define the network (layerwise)

Add loss and optimization method

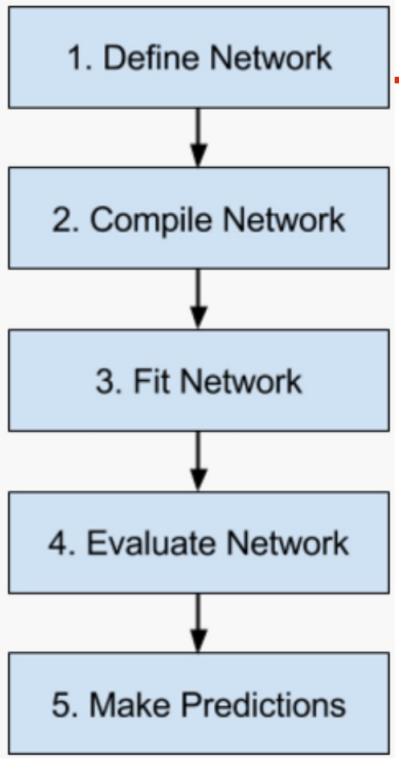
Fit network to training data

Evaluate network on test data

Use in production

A first run through

Define the network

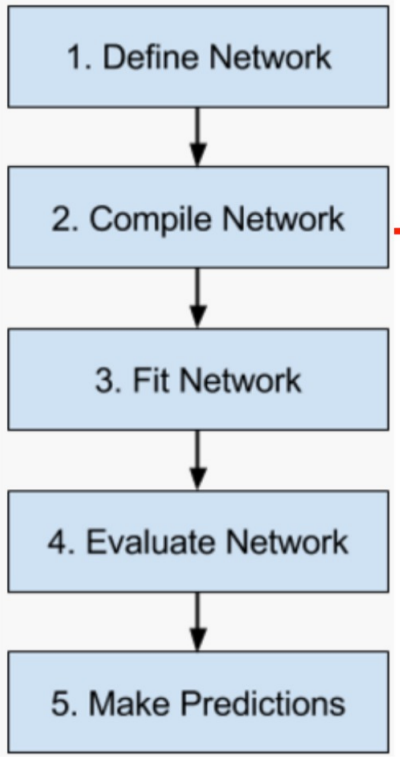



```
# define fcNN with 2 hidden layers
model = Sequential()

model.add(Dense(100, batch_input_shape=(None, 784)))
model.add(Activation('sigmoid'))
model.add(Dense(50))
model.add(Activation('sigmoid'))
model.add(Dense(10))
model.add(Activation('softmax'))
```

Input shape needs to be defined only at the beginning.

Alternative: `input_dim=784`

Compile the network



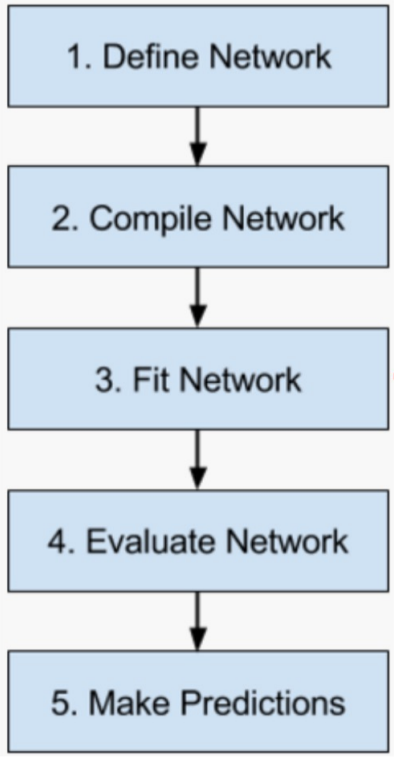
loss function that will be minimized

```
model.compile(loss='categorical_crossentropy',
              optimizer='adadelta',
              metrics=['accuracy'])
```

easiest optimizer is SGD
(stochastic gradient descent)

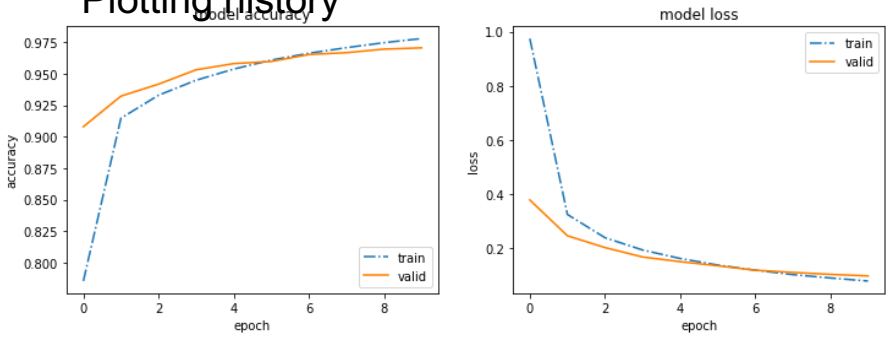
Which metrics besides 'loss' do we want to collect during training

Fit the network

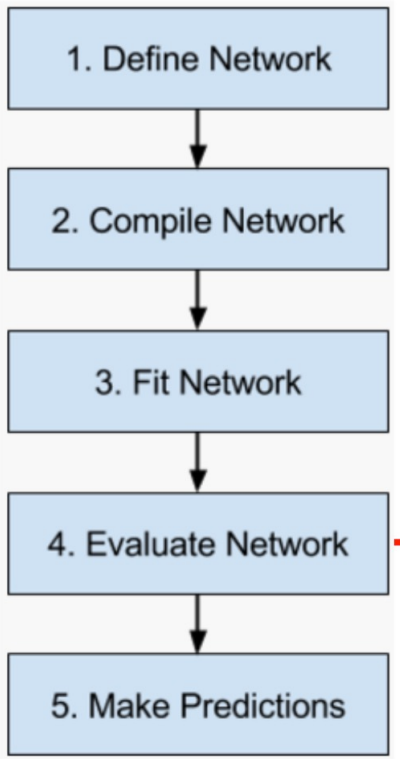



```
# Training of the network
history = model.fit(X, Y,
          epochs=400,
          batch_size=128,
          verbose=0)
```

Plotting history



Evaluate the network




```
model.evaluate(X[0:2000], convertToOneHot(y[0:2000],10))
```

```
2000/2000 [==============================] - 0s 215us/step
```

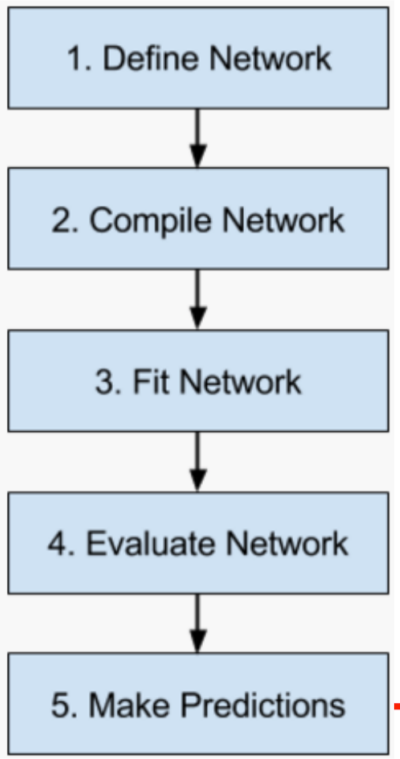
```
[2.5549037284851073, 0.1085]
```

loss

metrics: accuracy

```
model.compile(loss='categorical_crossentropy',
              optimizer='adadelta',
              metrics=['accuracy'])
```

Make Predictions




```
model.predict_classes(X[0:10])
```

```
array([0, 9, 9, 1, 6, 9, 8, 9, 0, 0])
```

Demo Time Example in Keras [only if time permits]

Demo Time: https://github.com/tensorchiefs/dl_book/blob/master/chapter_02/nb_ch02_02a.ipynb

Colab https://colab.research.google.com/github/tensorchiefs/dl_book/blob/master/chapter_02/nb_ch02_02a.ipynb

Building NN (with keras)

- Lego Blocks (Layers)
- Way of stacking them together API Style

Building a network (API Styles)

Three API styles

- The Sequential Model
  - Dead simple
  - Only for single-input, single-output, sequential layer stacks
  - Good for 70+% of use cases
- The functional API
  - Like playing with Lego bricks
  - Multi-input, multi-output, arbitrary static graph topologies
  - Good for 95% of use cases
- Model subclassing
  - Maximum flexibility
  - Larger potential error surface

Sequential API

The Sequential API

```
import keras
from keras import layers

model = keras.Sequential()
model.add(layers.Dense(20, activation='relu', input_shape=(10,)))
model.add(layers.Dense(20, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))

model.fit(x, y, epochs=10, batch_size=32)
```

Functional (do you spot the error?)

The functional API

```
import keras
from keras import layers

inputs = keras.Input(shape=(10,))
x = layers.Dense(20, activation='relu')(x)
x = layers.Dense(20, activation='relu')(x)
outputs = layers.Dense(10, activation='softmax')(x)

model = keras.Model(inputs, outputs)
model.fit(x, y, epochs=10, batch_size=32)
```

Subclassing (do you spot the error?) [for completness]

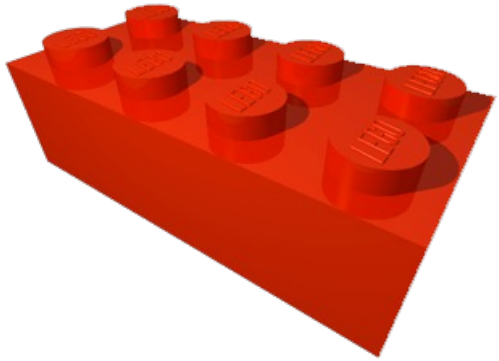
Model subclassing

```
import keras
from keras import layers

class MyModel(keras.Model):

    def __init__(self):
        super(MyModel, self).__init__()
        self.dense1 = layers.Dense(20, activation='relu')
        self.dense2 = layers.Dense(20, activation='relu')
        self.dense3 = layers.Dense(10, activation='softmax')

    def call(self, inputs):
        x = self.dense1(x)
        x = self.dense2(x)
        return self.dense3(x)

model = MyModel()
model.fit(x, y, epochs=10, batch_size=32)
```

layers

Dense fully connected

```
keras.layers.core.Dense(
        output_dim,
        init='glorot_uniform',
        activation='linear',
        weights=None,
        W_regularizer=None,
        b_regularizer=None,
        activity_regularizer=None,
        W_constraint=None,
        b_constraint=None,
        input_dim=None)
```



https://keras.io/layers/

More layers (to come)

- Dropout
  - keras.layers.Dropout
- Convolutional (see lecture on CNN)
  - keras.layers.Conv2D
  - keras.layers.Conv1D
- Pooling (see lecture on CNN)
  - keras.layers.MaxPooling2D
- Recurrent (See Lecture on RNN)
  - keras.layers.SimpleRNNCell
  - keras.layers.GRU
  - keras.layers.LSTM

- Roll your own:
  - Implement keras.layers.Layer class
  - https://keras.io/layers/writing-your-own-keras-layers/

Activation

```
keras.layers.Activation(activation)
```

Applies an activation function to an output.

Arguments

e.g. `'relu'`, `'softmax'`, `'tanh'`,...

- **activation**: name of activation function to use (see: activations), or alternatively, a Theano or TensorFlow operation.

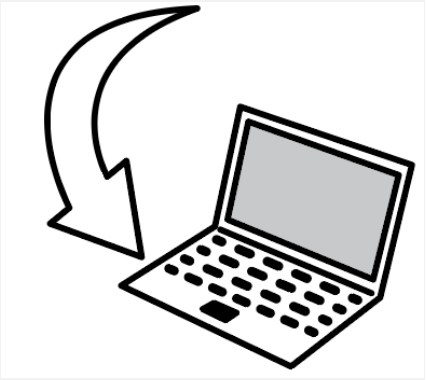
```
from keras.layers import Activation, Dense

model.add(Dense(64))
model.add(Activation('tanh'))
```

This is equivalent to:

```
model.add(Dense(64, activation='tanh'))
```

Note: Activations are also layers

Tasks



1. Finish NB 01_simple_forward_pass.ipynb have a look at the Keras implementation
2. Do exercise NB 02