Machine Intelligence:: Deep Learning
Week 1

Beate Sick, Elvis Murina, Oliver Dürr

Institut für Datenanalyse und Prozessdesign
Zürcher Hochschule für Angewandte Wissenschaften

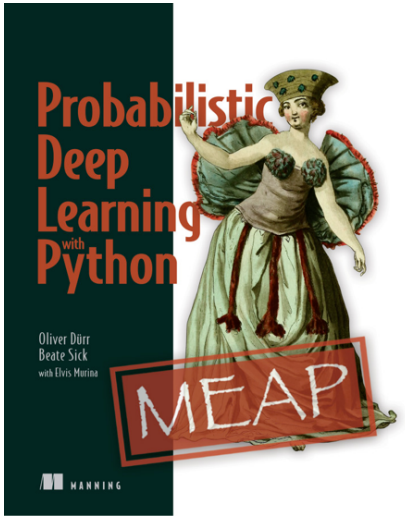
Winterthur, 18. Feb. 2020

Tensorchiefs

$F^{\mu\nu}$

- Jointly organize
  - The CAS Deep Learning Course
    - https://tensorchiefs.github.io/dl_course_2020/
  - The Deep Learning Workshops “Deep Learning Day”
    - https://tensorchiefs.github.io/dlday2018/
    - https://tensorchiefs.github.io/dlday2017/
    - https://sites.google.com/site/sdsdlday2016/

- Currently write a book about deep learning
  - Can be used in addition to the course
  - https://tensorchiefs.github.io/dl_book/



Tell us something about you

- CS Background
  - Fluent in python?
- Statistics / Math
  - Who visited CAS Data Analysis II
  - What is a distribution
  - Vector times Matrix
    - Please make sure to check https://tensorchiefs.github.io/dl_course_2020/prerequistites.html
- Any contacts with deep learning yet?

Before we start (check technical details)

- Website
  - https://tensorchiefs.github.io/dl_course_2020/

- 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)
  - As an alternative you can use the anaconda distribution
    - We need TF 2.0 and TensorFlow Probability >= 0.8

- Before we start make sure you have the latest image
  - `docker pull oduerr/dl_book_docker`

- Alternatively Anaconda installation (see website)

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 us latest by week4)

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

Organizational Issues: Times

- Dates and times: see our webpage
- Usually afternoon session
  - 13:30 – 15:00
  - 15:30 – 17:00
    - Exceptions
      - 24 March and 7 April is in the morning
- 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 we talk the better!**

Outline of the DL Module (tentative)

- Day 1: Jumpstart to DL
  - What is DL
  - Basic Building Blocks
  - Keras
- Day 2: CNN I
  - ImageData
- Day 3: CNN II and RNN
  - Tips and Tricks
  - Modern Architectures
  - 1-D Sequential Data
- Day 4: Looking at details
  - Linear Regression
  - Backpropagation
    - Resnet
  - Likelihood principle
- Day 5: Probabilistic Aspects
  - TensorFlow Probability (TFP)
  - Negative Loss Likelihood NLL
  - Count Data
- Day 6: Probabilistic models in the wild
  - Complex Distributions
  - Generative modes with normalizing flows
- Day 7: Uncertainty in DL
  - Bayesian Modeling
- Day 8: Uncertainty cont'd
  - Bayesian Neural Networks
  - Projects

Day 1-4 should get you ready for your project.

Learning Objectives for today

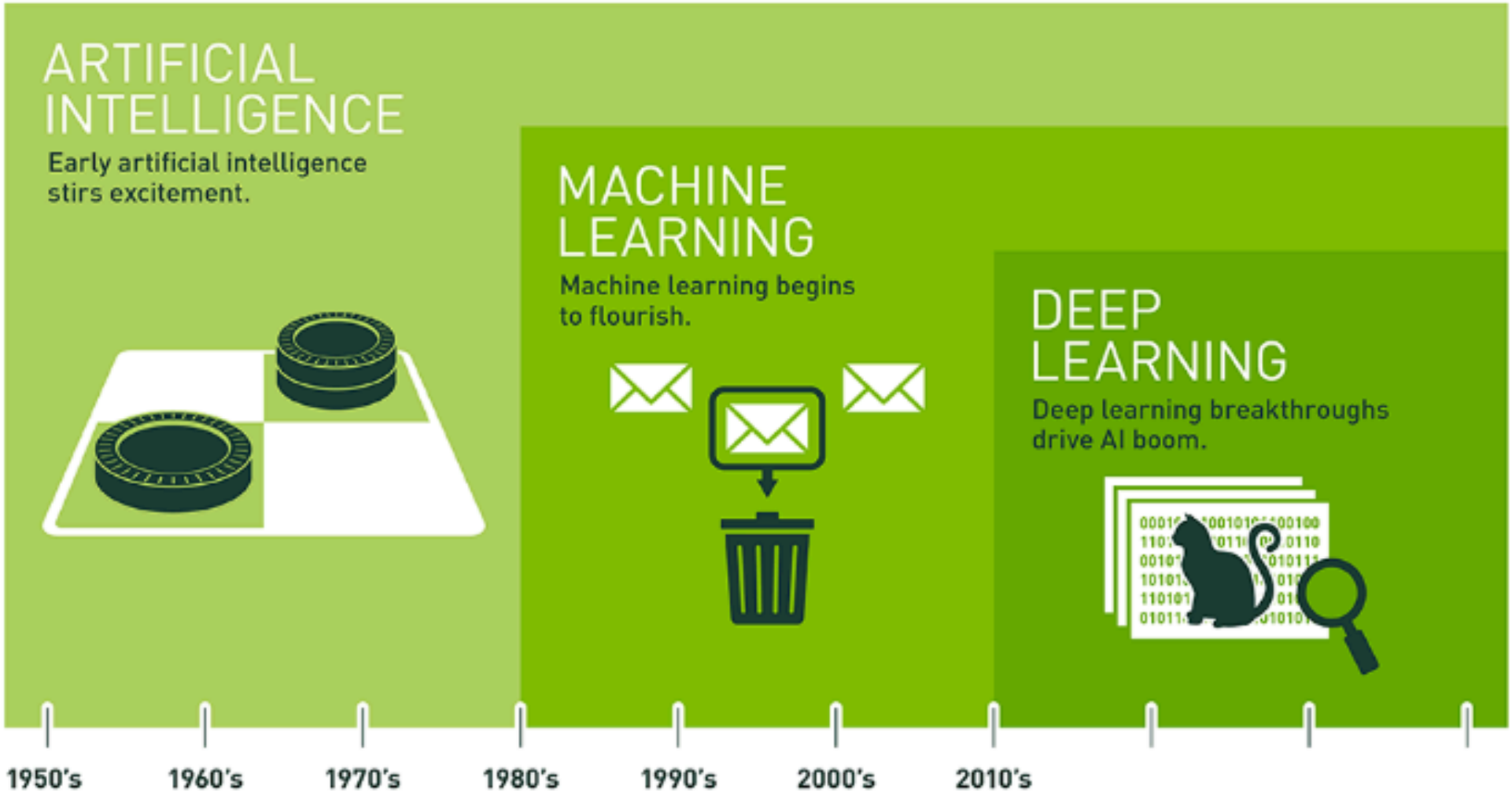
- Get a rough idea what the DL is about

- 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?

AI, Machine Learning, Deep Learning



Slide credit: https://www.datasciencecentral.com/profiles/blogs/artificial-intelligence-vs-machine-learning-vs-deep-learning

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

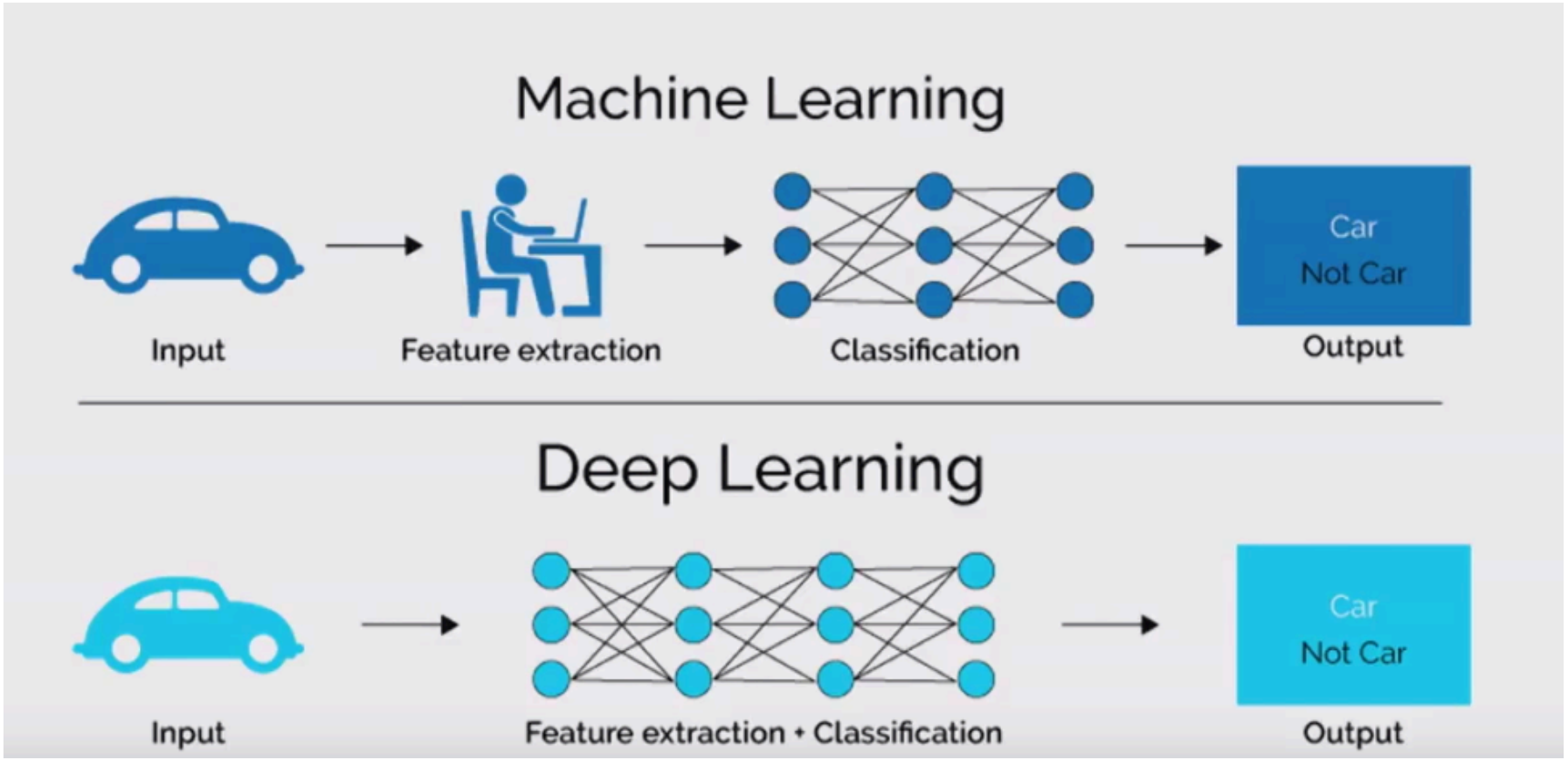
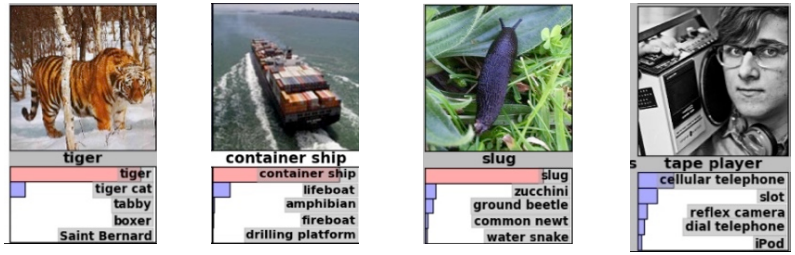


image credit: Siraj Raval

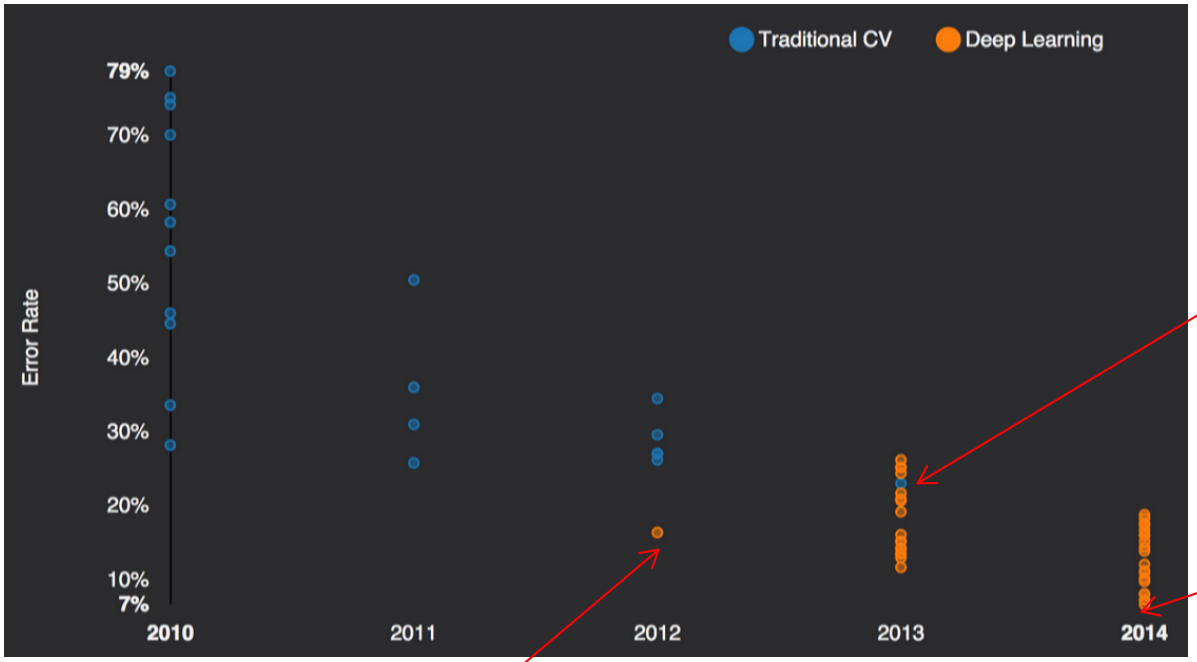
The most convincing case for DL (subjective view)

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

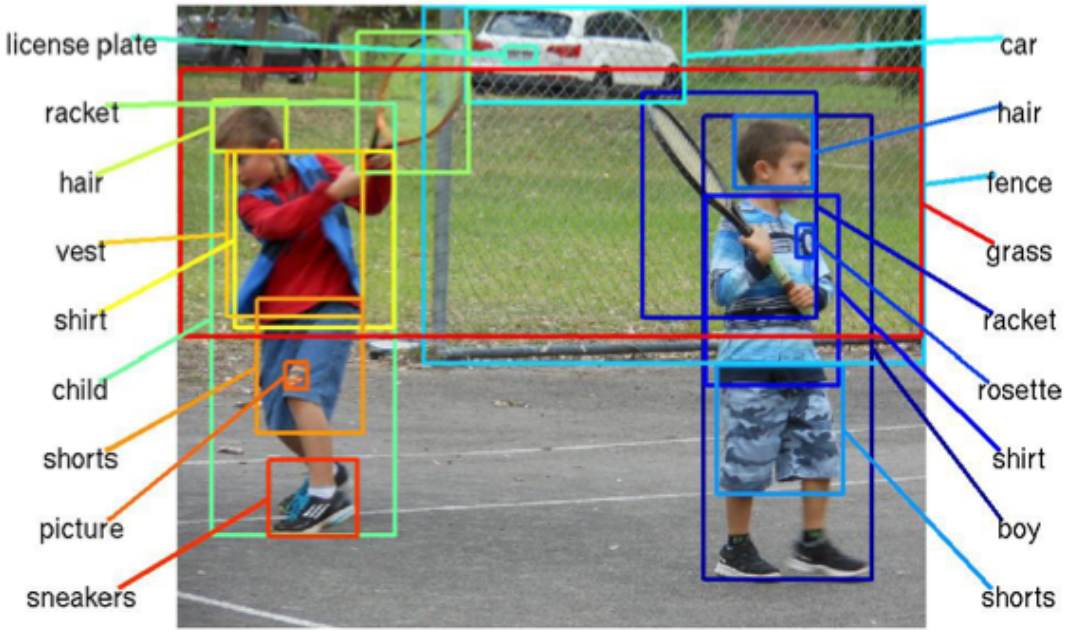
- 4.95% Microsoft (Feb 6 surpassing human performance 5.1%)
- 4.8% Google (Feb 11) -> further improved to 3.6 (Dec)?
- 4.58% Baidu (May 11 banned due too many submissions)
- 3.57% Microsoft (Resnet winner 2015)

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

The importance of seeing: the Cambrian explosion

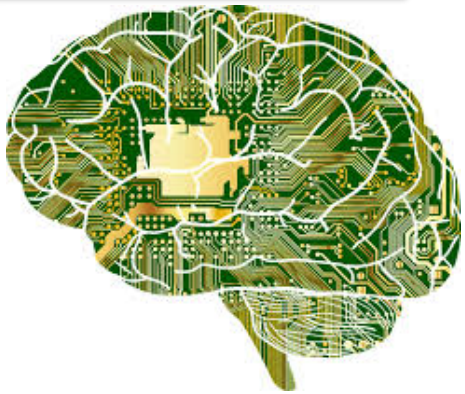
In a short period of time ~20 mio yrs (~540 mio yrs ago) most animal groups appeared.

So extraordinary fast that this is taken as an explanation intelligent design.

A simpler explanation

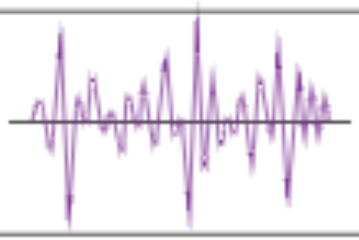
creatures learned to see

A more liable equivalence of deep learning than …



Idea: Stolen from CS231n, brain wikipedia

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 öffnet Tür zu hören, sehen und Texten. (kein Verstehen aber statistische Zusammenhänge).

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

Deep Learning vs. Machine Learning

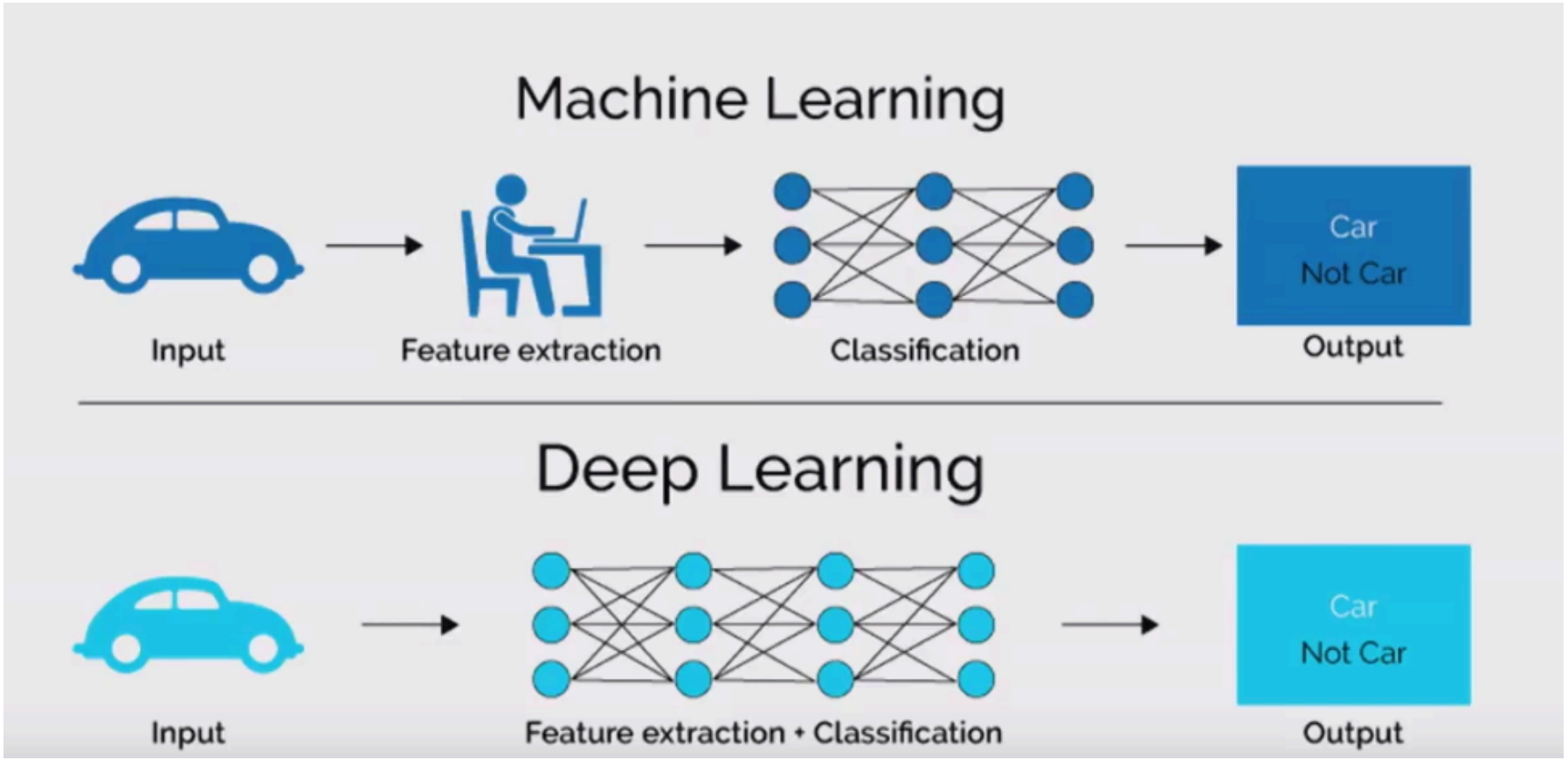
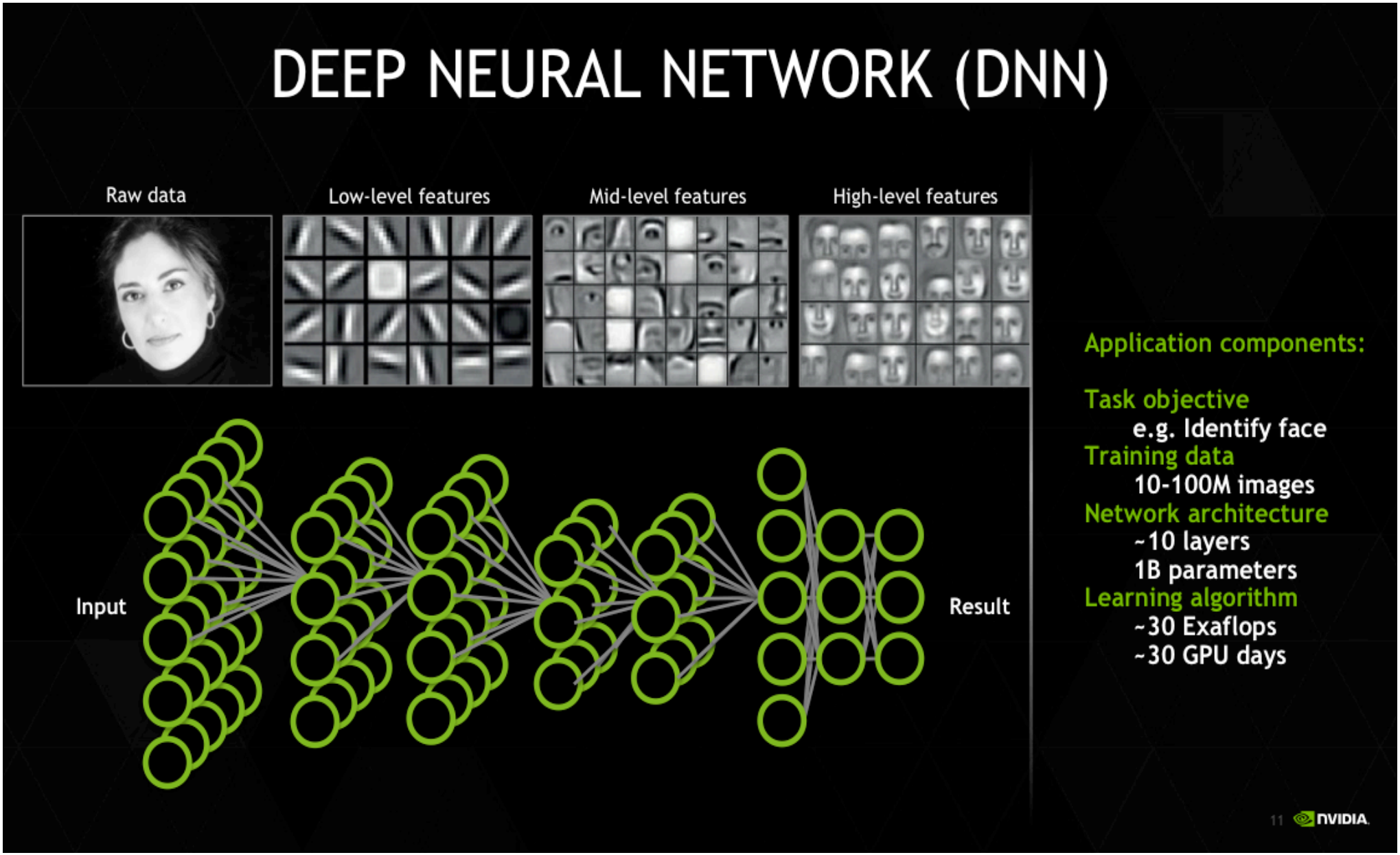


image credit: Siraj Raval

Main Idea in DL



Learn hierarchy of features

First Neural Network

The Single Cell: Biological Motivation

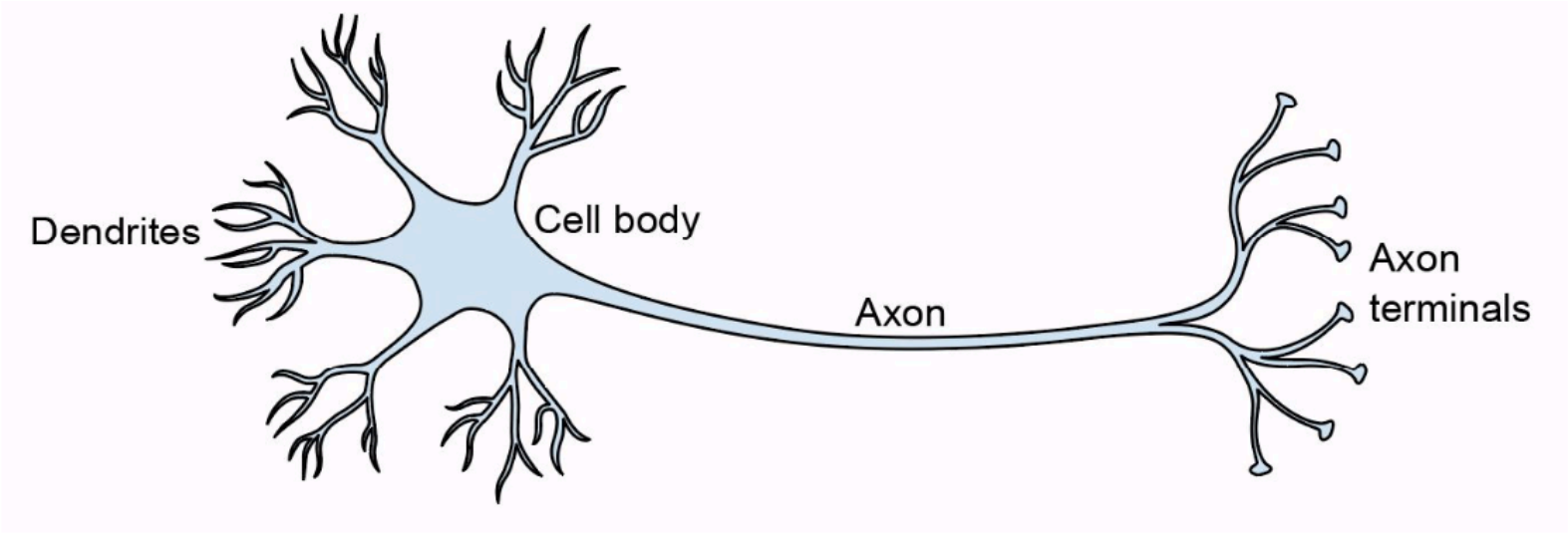
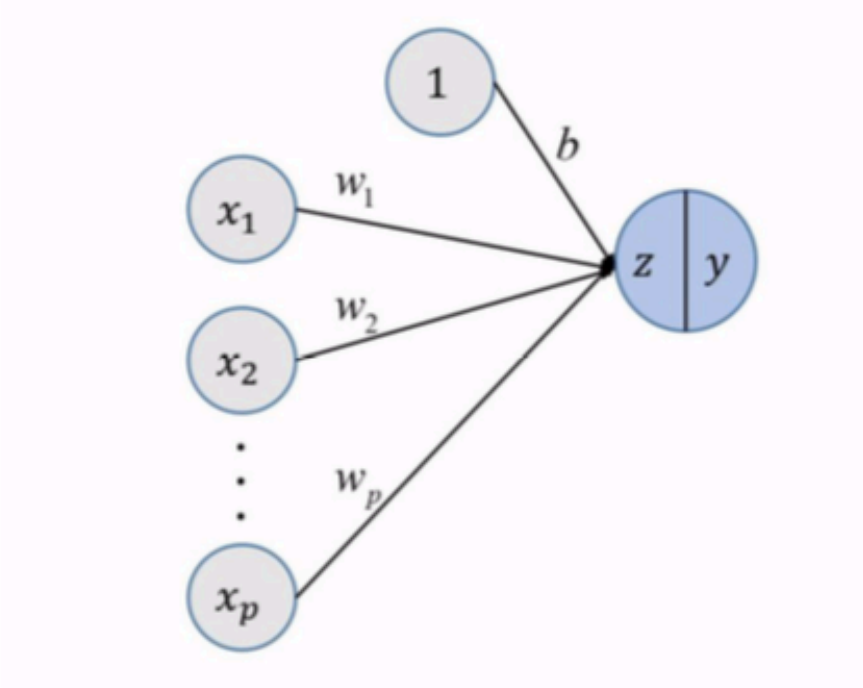


Figure 2.2 A single biological brain cell. The neuron receives the signal from other neurons via its dendrites shown on the left. If the cumulated signal exceeds a certain value, an impulse is sent via the axon to the axon terminals, which, in turn, couples to other neurons.

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)

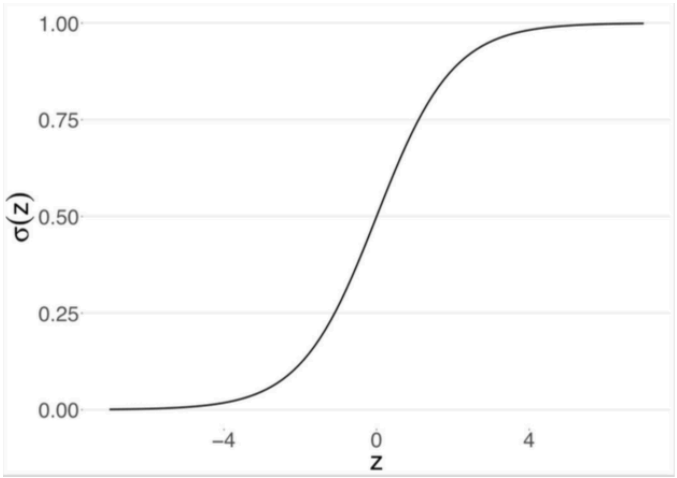


Figure 2.3 The mathematical abstraction of a brain cell (an artificial neuron). The value *z* is computed as the weighted sum of the *p* input values, *x*1 to *xp*, and a bias term *b* that shifts up or down the resulting weighted sum of the inputs. The value *y* is computed from *z* by applying an activation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

Toy Task

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

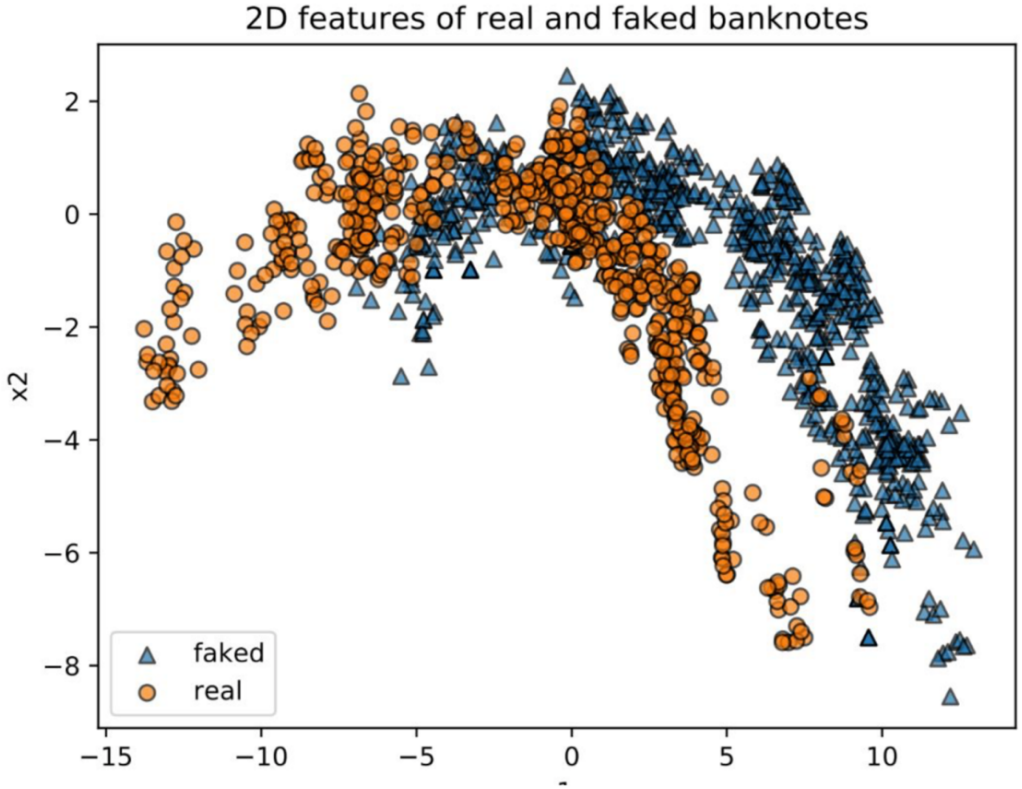
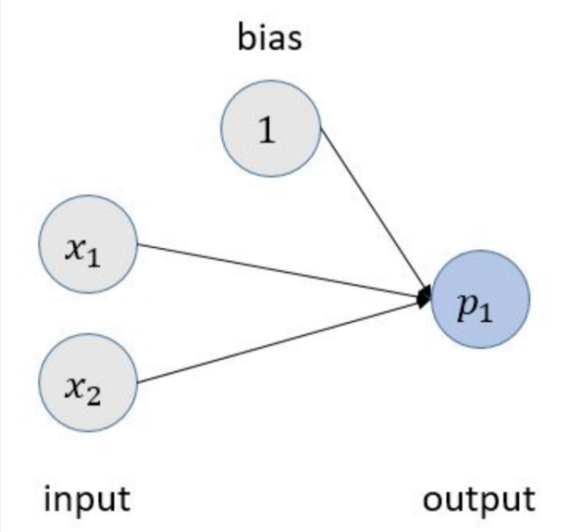
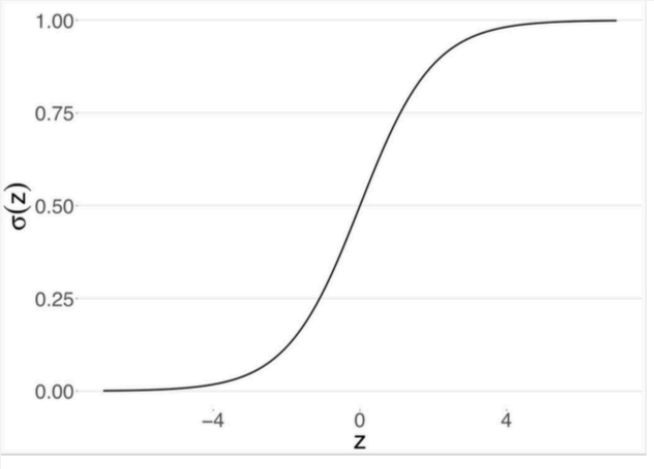


Figure 2.5 The (training) data points for the real and faked banknotes

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

Exercise: Part 1





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

TASK:

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

$w_1 = 0.3, w_2 = 0.1$, and $b = 1.0$

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

1. What is the probability that a banknote characterized by x_1=1 and $x_2 = 2.2$ is fake?
2. Have a look at the decision plot.

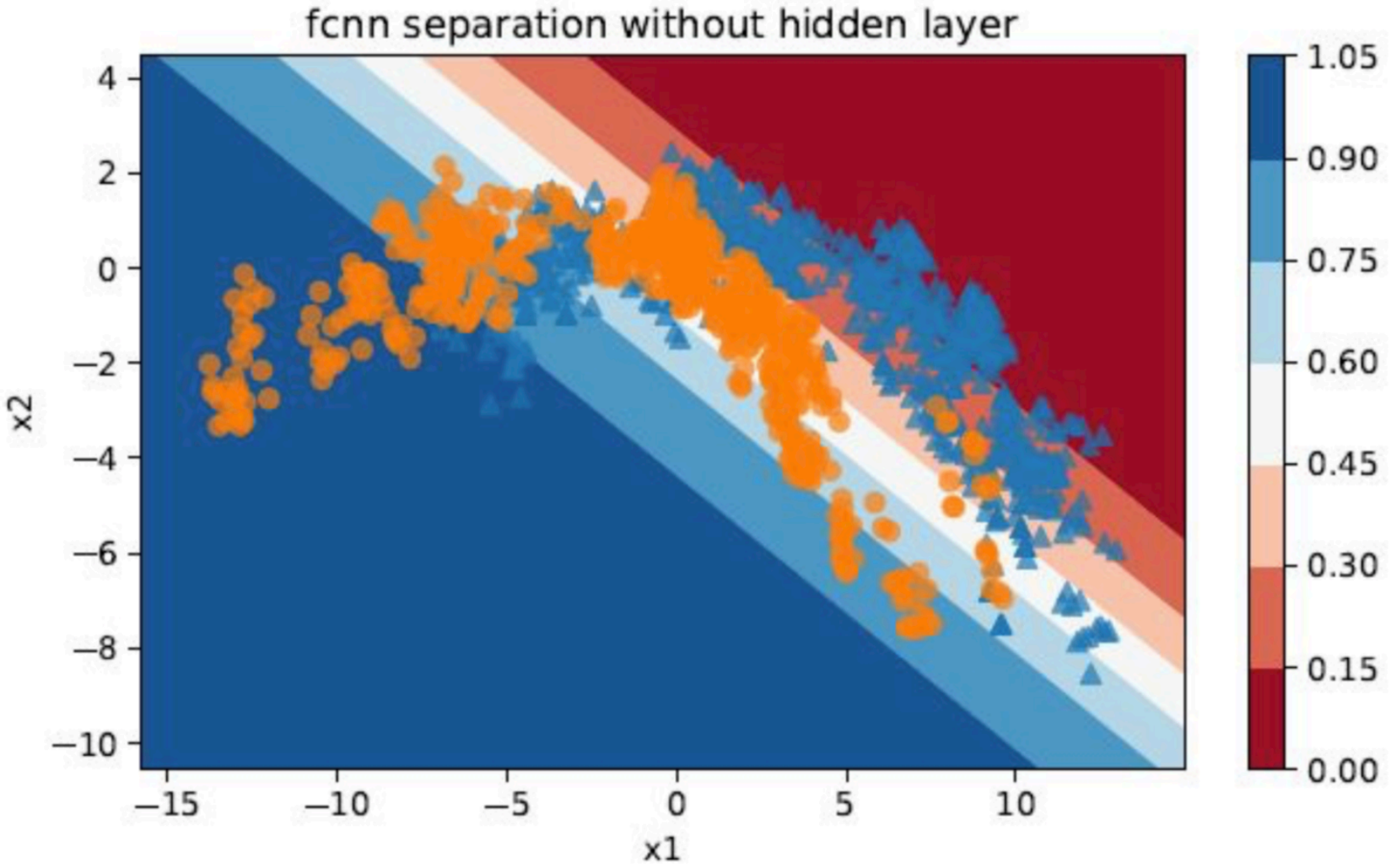
GPUs love Vectors

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

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

DL: better to have column vectors

Result

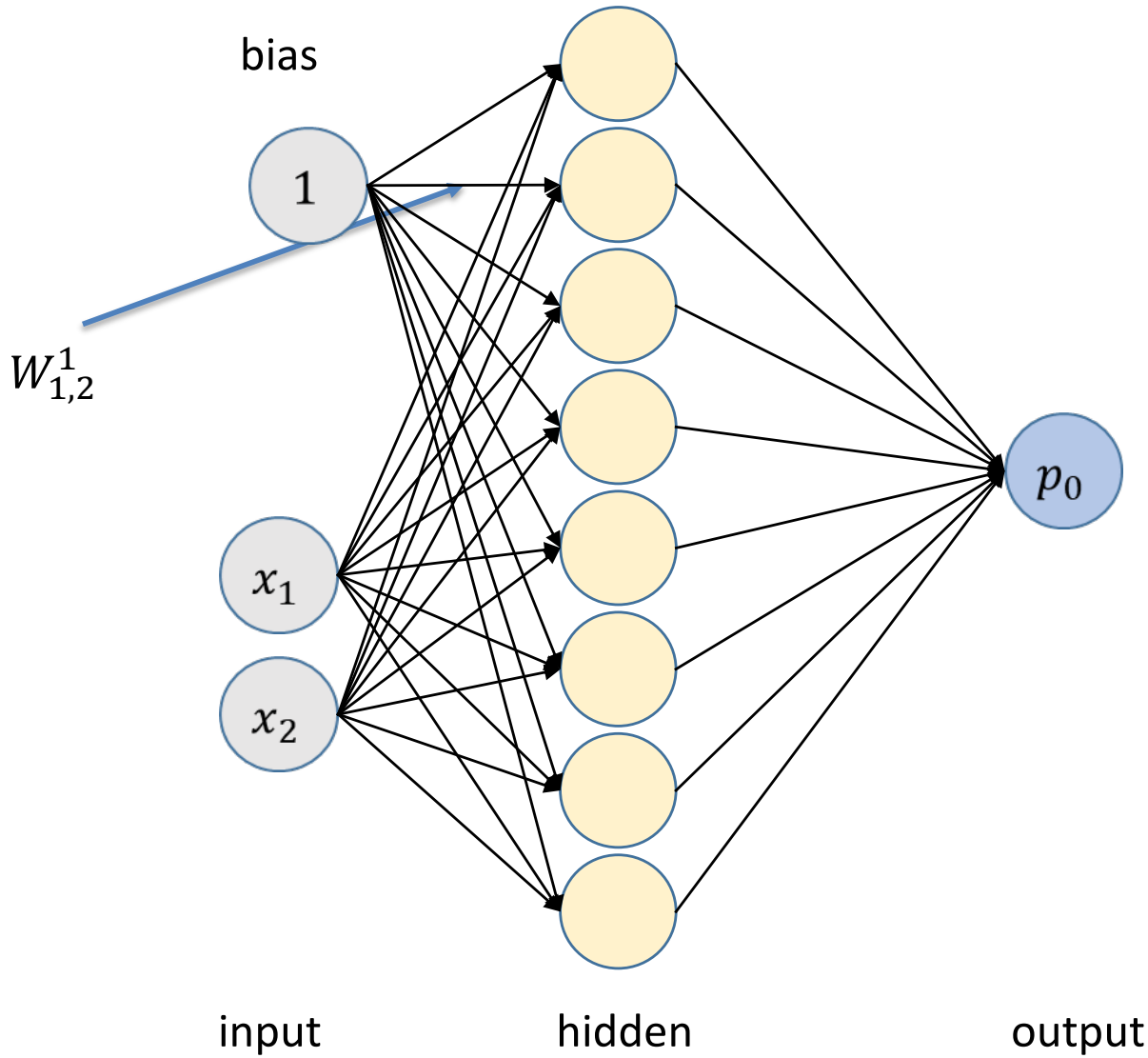


General rule: Networks without hidden layer have linear decision boundary.

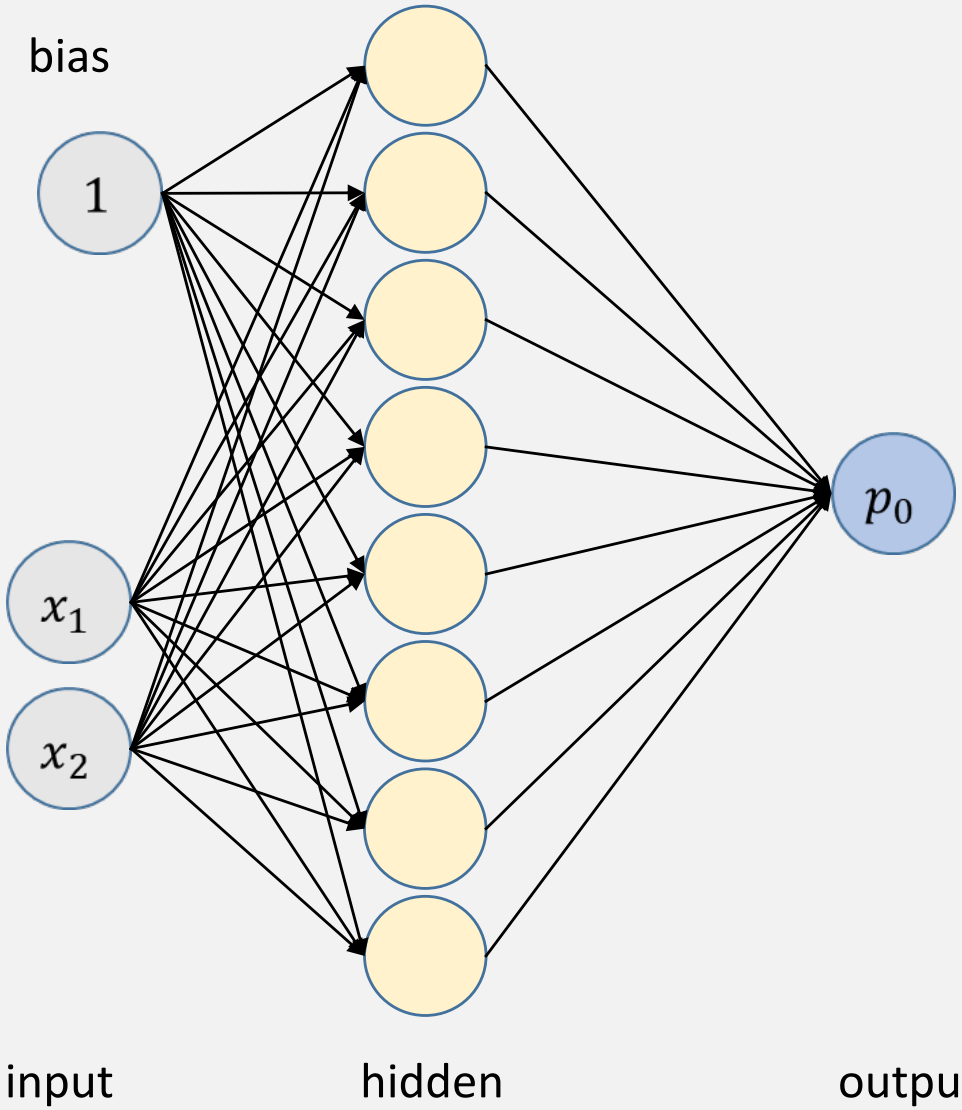

WE NEED TO GO
DEEPER
memegene

A first deep network

$W_{\text{from,to}}$

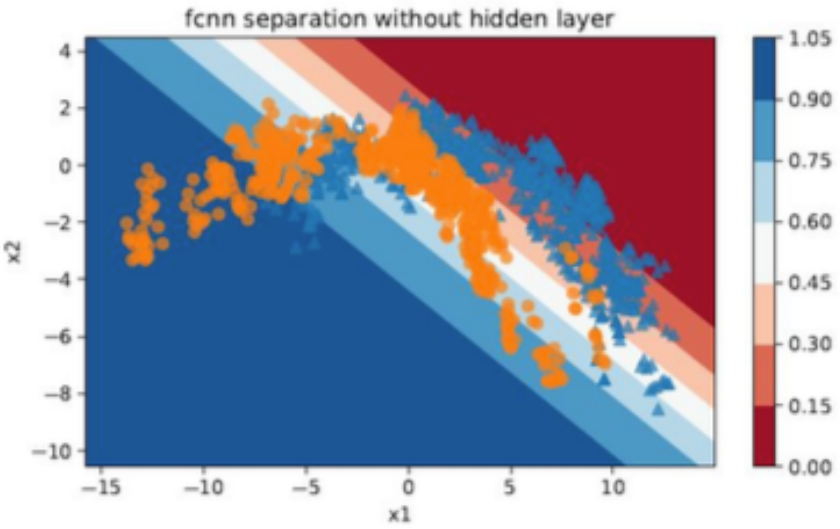


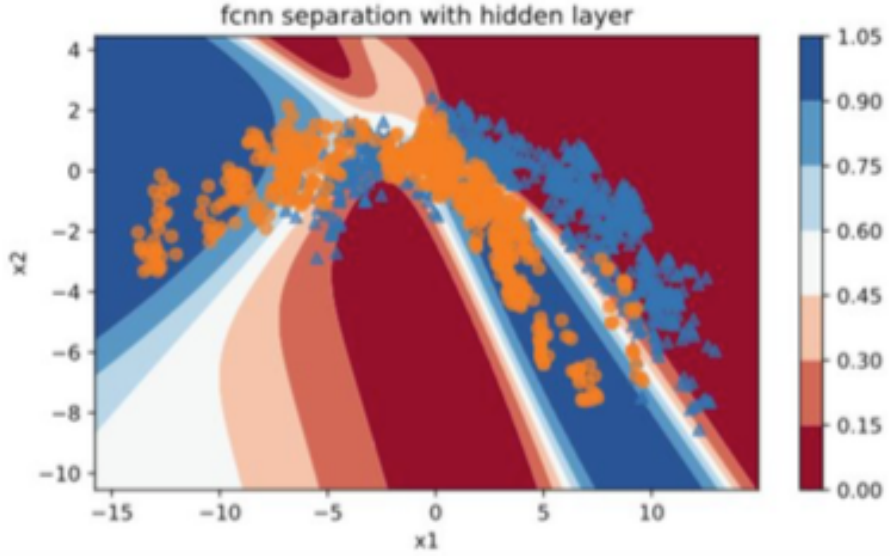
Exercise: Part 2 Including a hidden layer



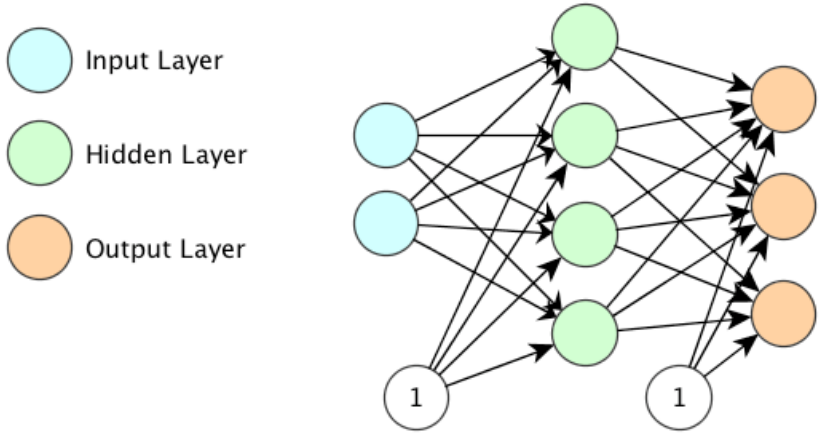
Open NB 01_simple_forward_pass.ipynb and do exercise

The benefit of hidden layers

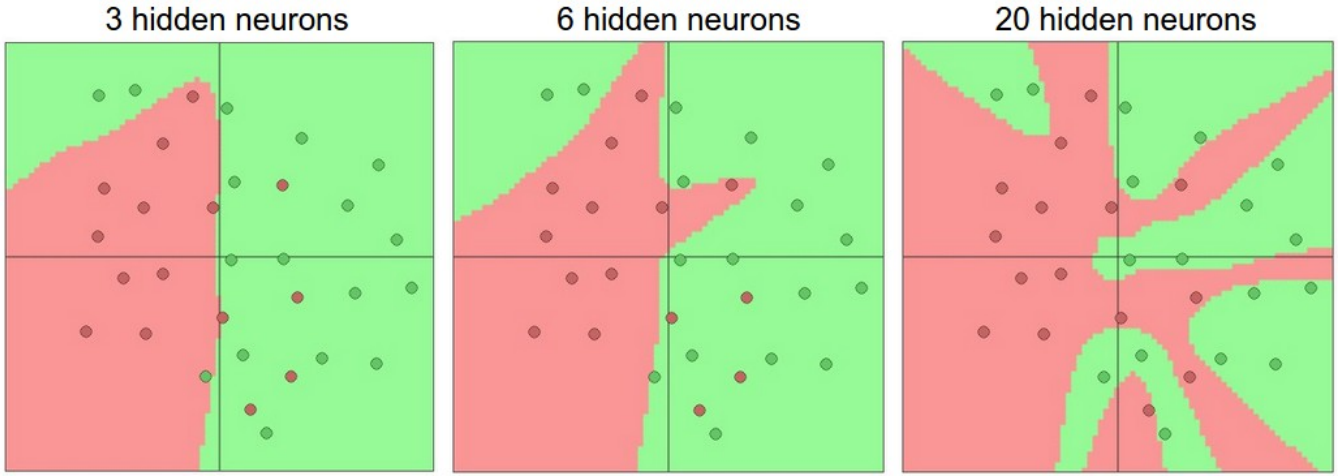




One hidden Layer

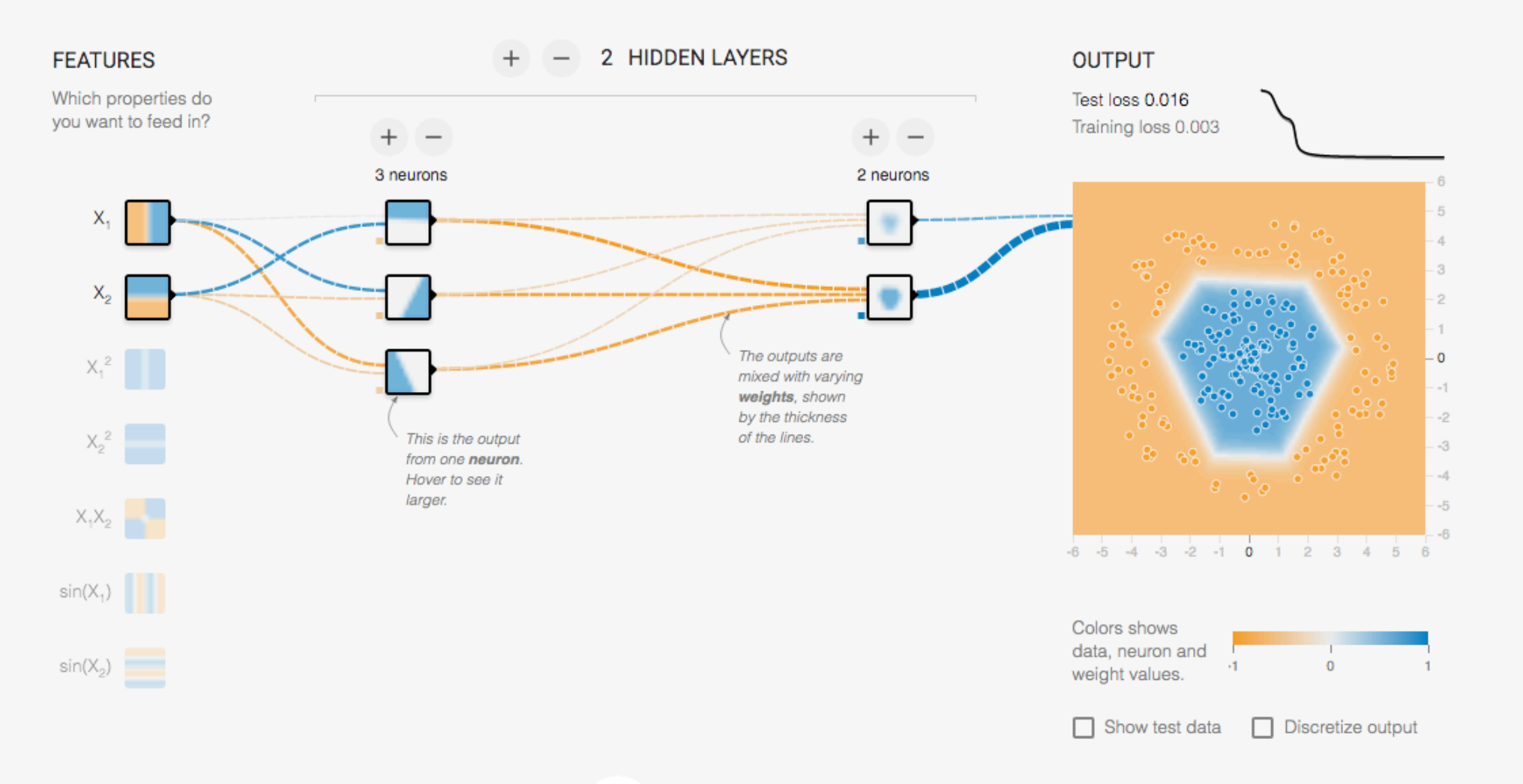


A network with one hidden layer is a universal function approximator!



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

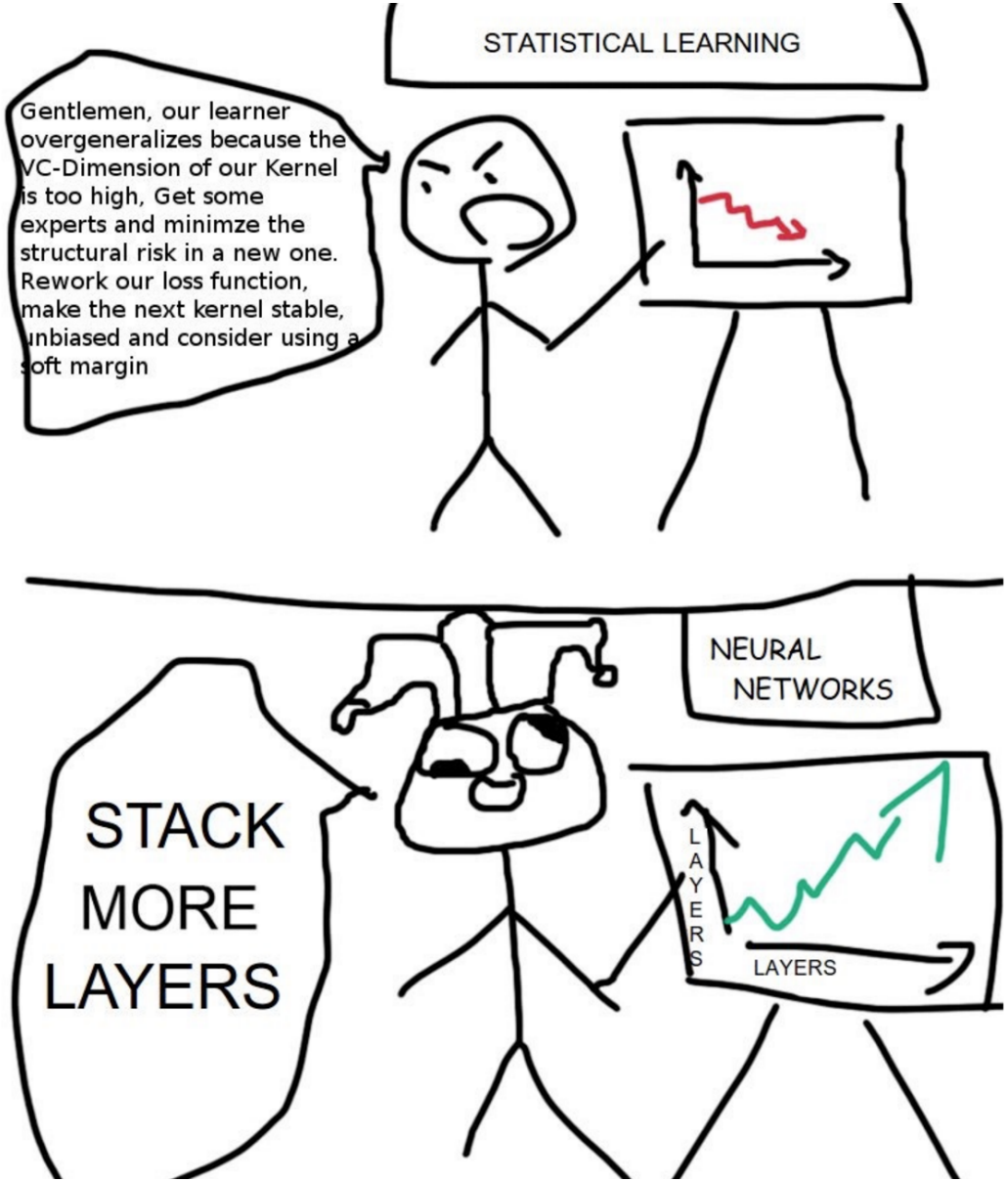
Experiment yourself (homework)



http://playground.tensorflow.org

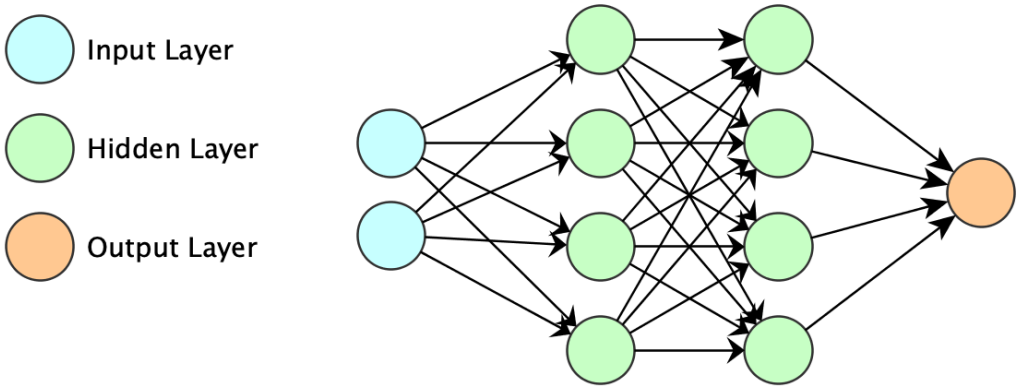
Let's you explore the effect of hidden layers

DL vs Machine Learning Meme



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

Structure of the network



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

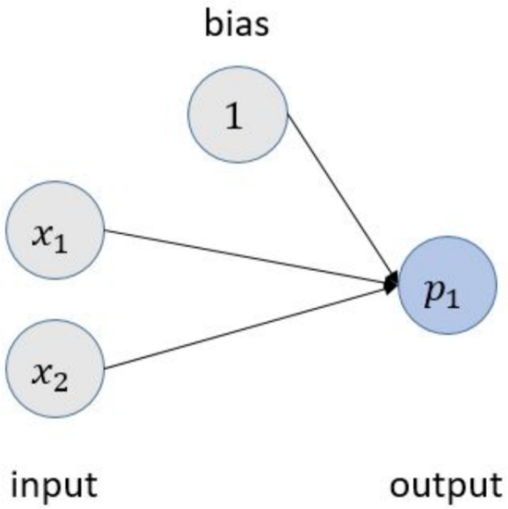
$$p = f(f(f(\mathrm{x}\,\mathrm{W}^1)\mathrm{W}^2))$$

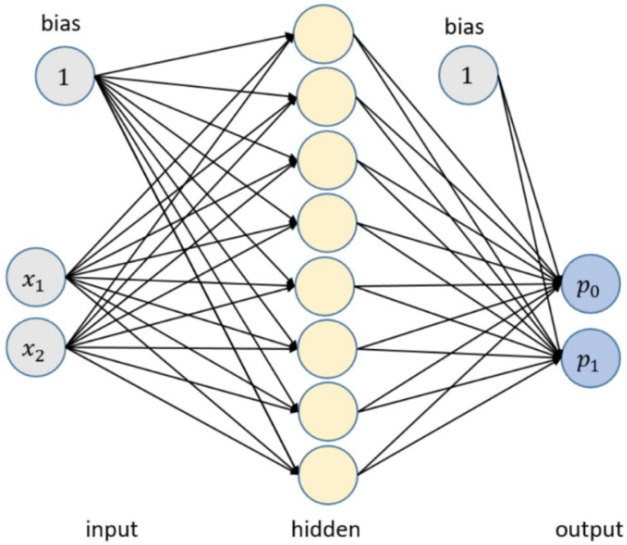
Looks a bit like onions, matryoshka (Russian Dolls) or lego bricks

Training NN

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

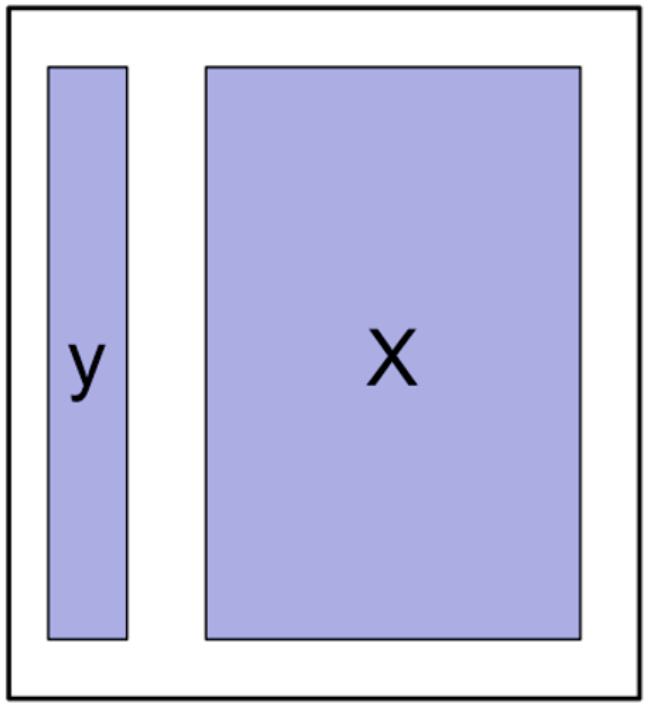




Loss Functions

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

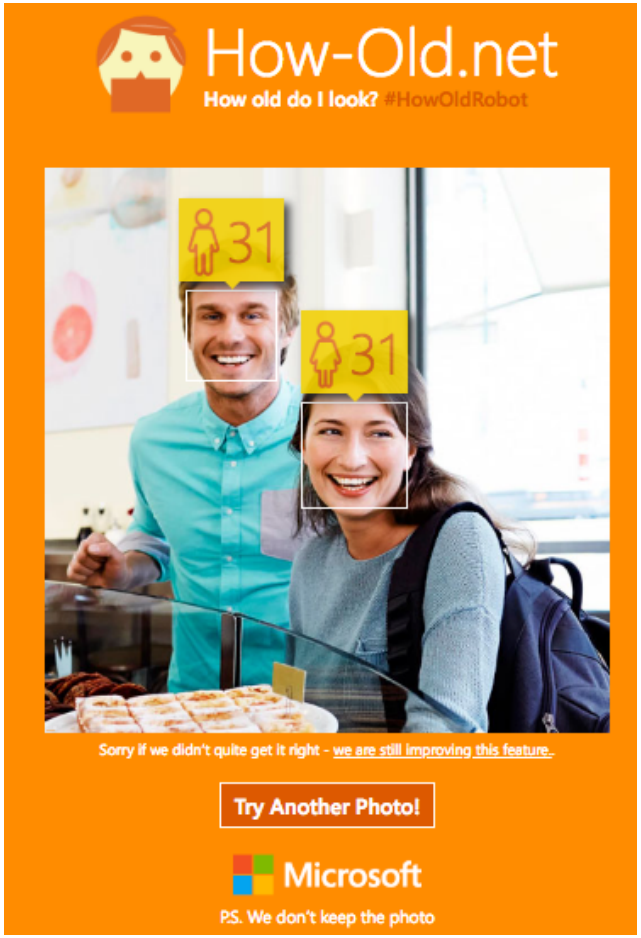
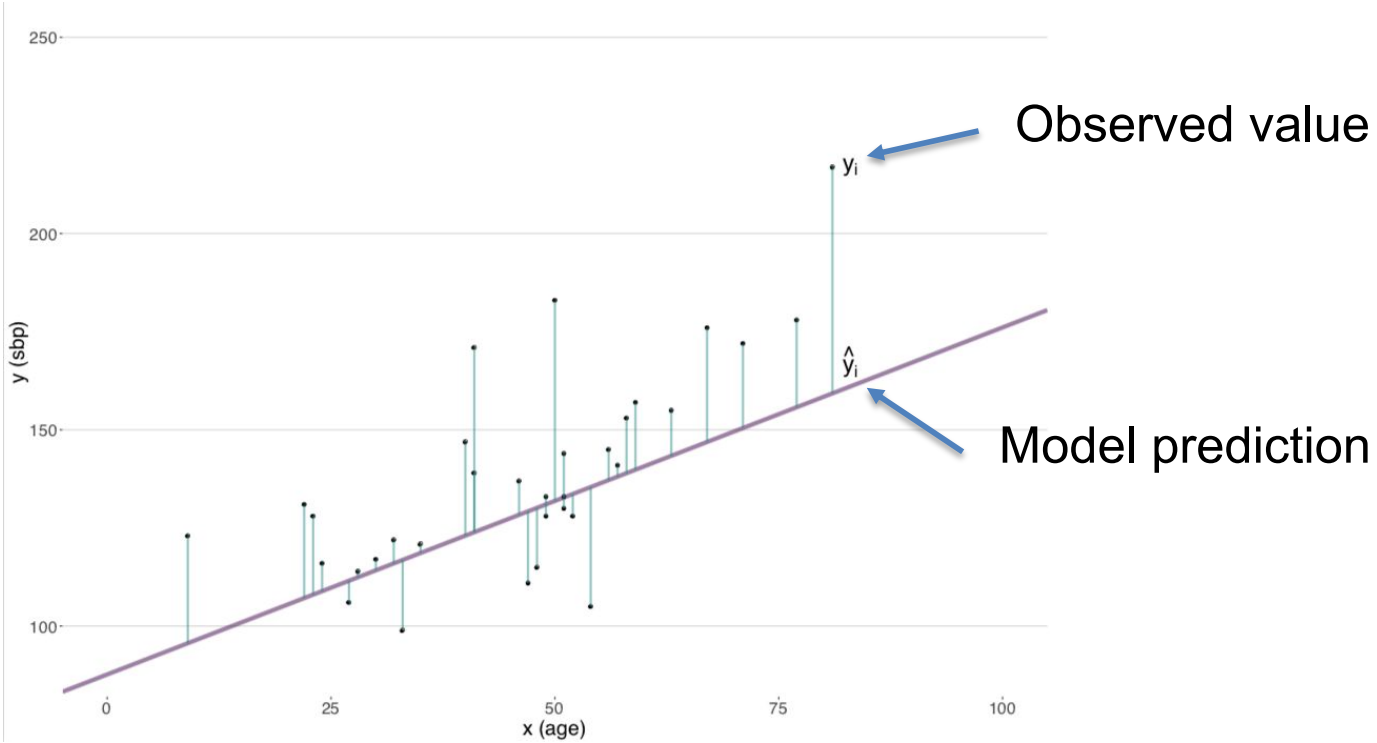


Image --> Age

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

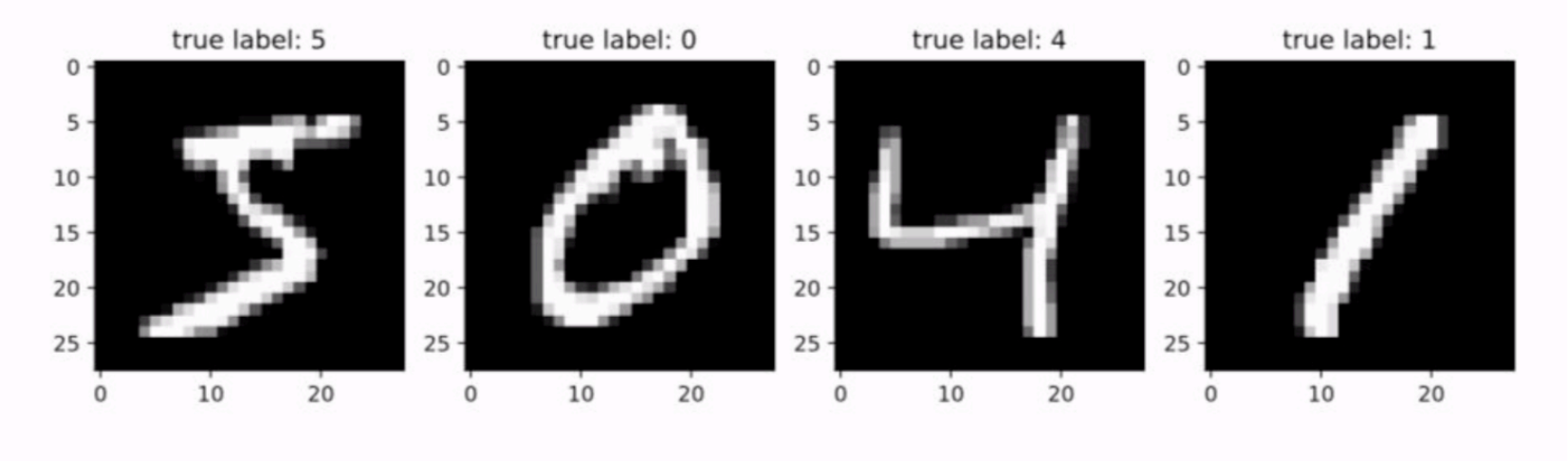
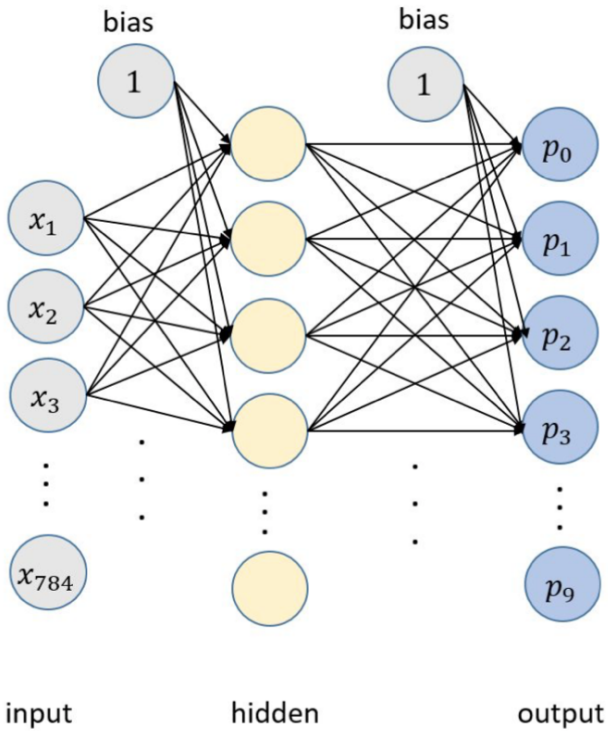


Figure 2.11 The first four digits of the MNIST data set—the standard data set used for benchmarking NN for images classification

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

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

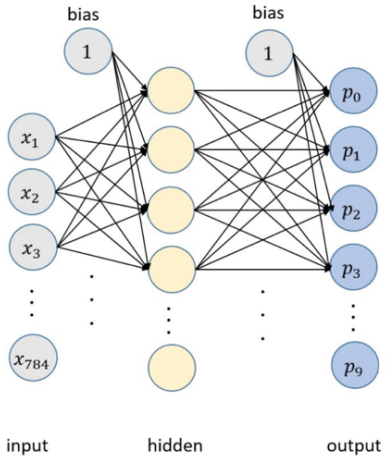
Makes outcome positive

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.

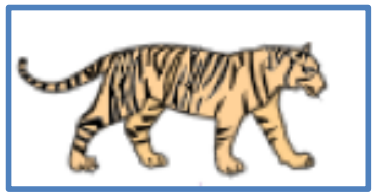
- As MSE Loss is averaged of individual losses l_i of training data $\mathrm{i} = 1, \ldots N$
- Want l_i
  - 0 for perfect match, i.e. predicts class of training example $y^{(i)}$ with probability 1
  - ∞ for worst match, i.e. predicts class $y^{(i)}$ with probability 0
  - $l_i = -\log\left(p_{model}\left(y^{(i)}\middle|x^{(i)}\right)\right)$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

Training / Gradient Descent

Prinzipielle Funktionsweise: Training Bild Klassifikation

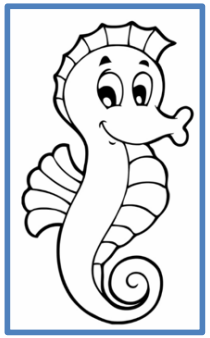
Wahre Klasse

Vorhergesagtes Label

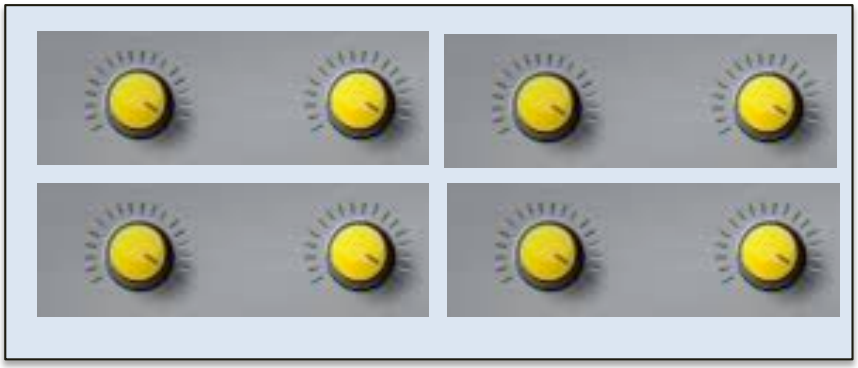


Tiger → Seehund 👎

Tiger → Tiger 👍



Seepferd → Seepferd 👍



...

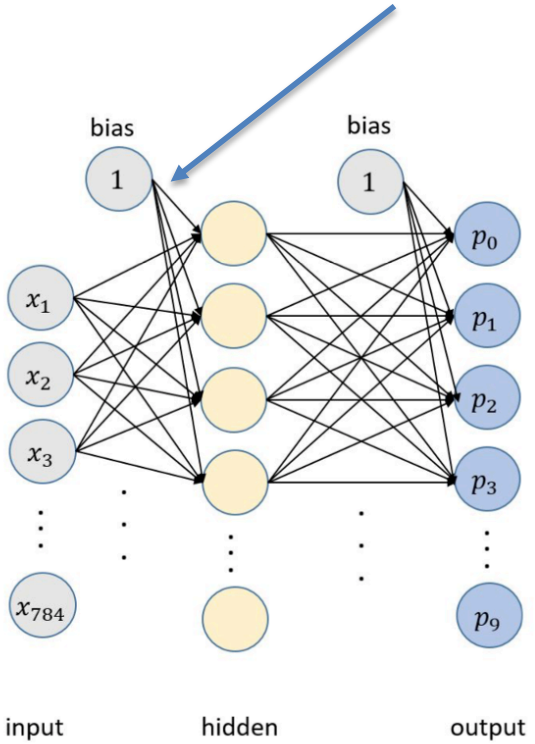
Typisch 1 Mio. Trainingsdaten

Trainingsprinzip:
Parameter werden so eingestellt, dass möglichst wenige Fehler in den Trainingsdaten gemacht werden.

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

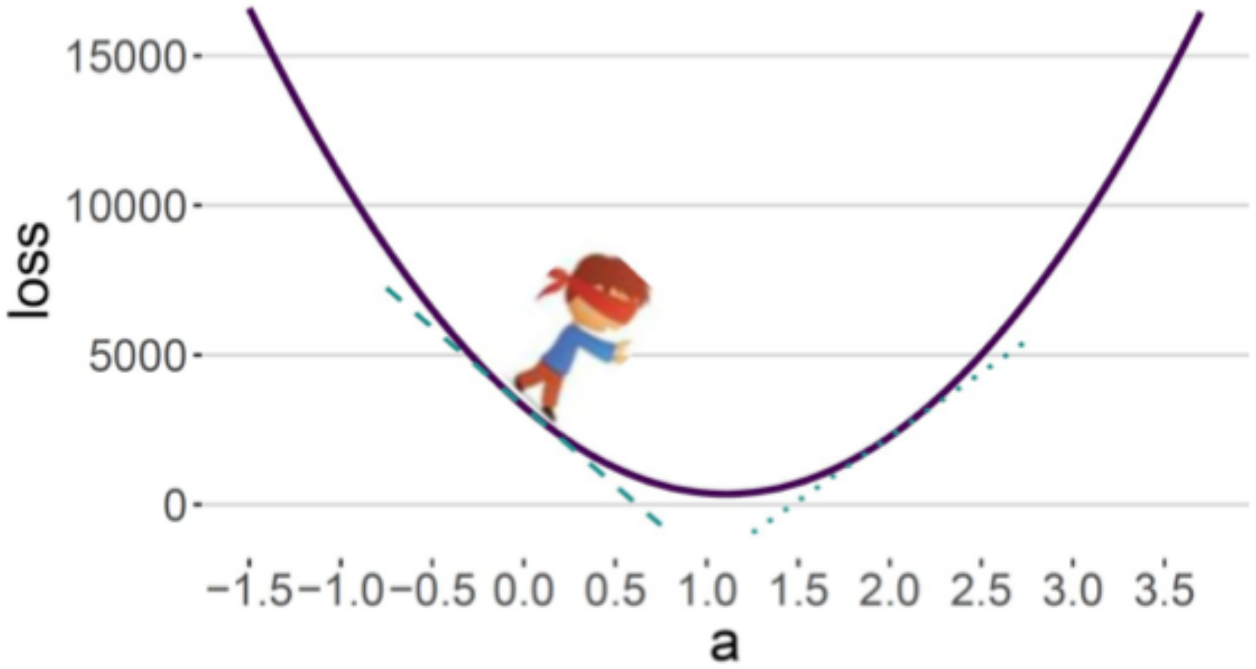
Parameters of the network are the weights.



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 (see lecture 4)

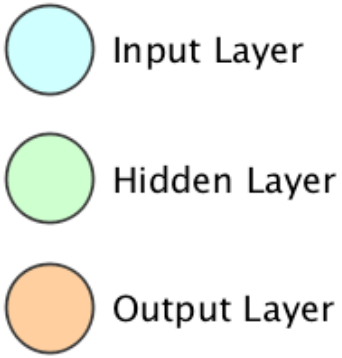
- We need to calculate the derivative of the loss function w.r.t. all weights

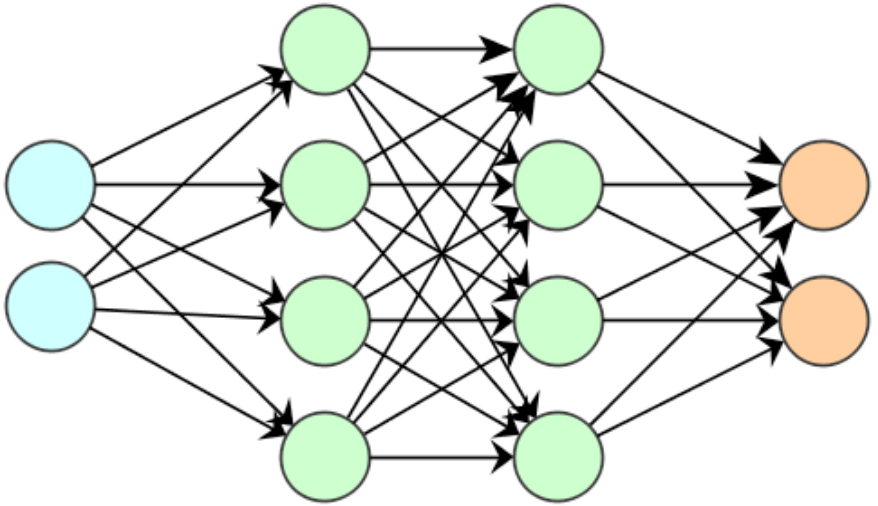
- Doing this efficiently (on graphic cards GPU) by hand is tedious

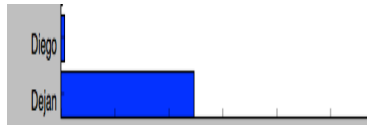
- Enter:
  - Deep Learning Frameworks

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

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...

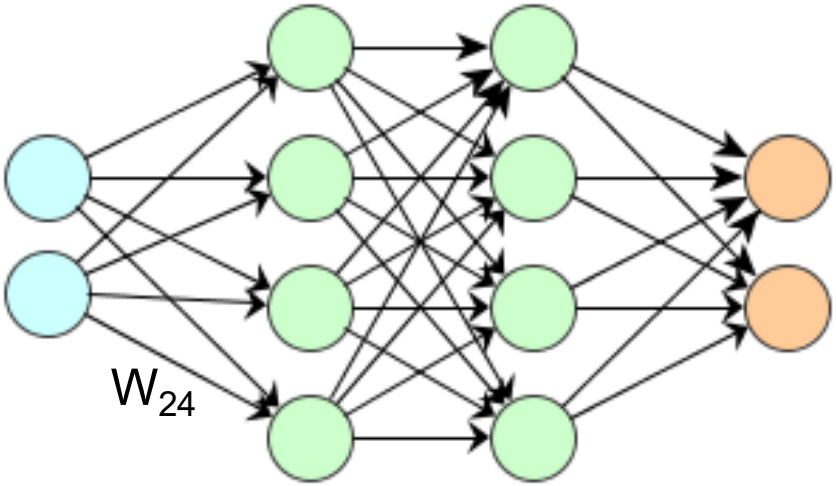
Sharpe, R. W. (1997). Differential Geometry: Cartan's Generalization of Klein's Erlangen Program. Berlin, New York: Springer-Verlag. p. 194. ISBN 978-0-387-94732-7.

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

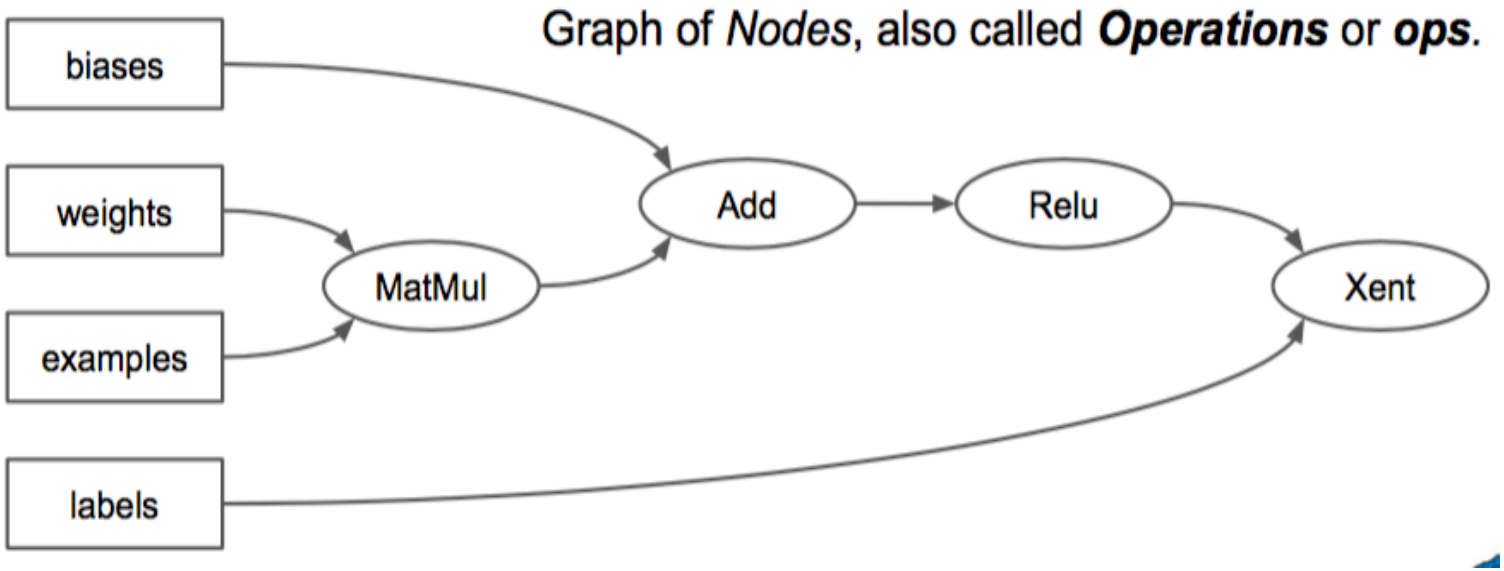


- The input can be understood as a vector
- A mini-batch of size 64 of input vectors can be understood as tensor of order 2
  - (index in batch, x_j)
- The weights going from e.g. Layer L_1 to Layer L_2 can be written as a matrix (often called W)
- A mini-batch of size 64 images with 256,256 pixels and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

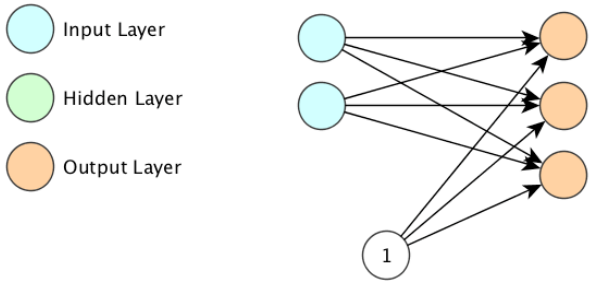
Data Flow and computational graph

Computational Graph (Nodes)

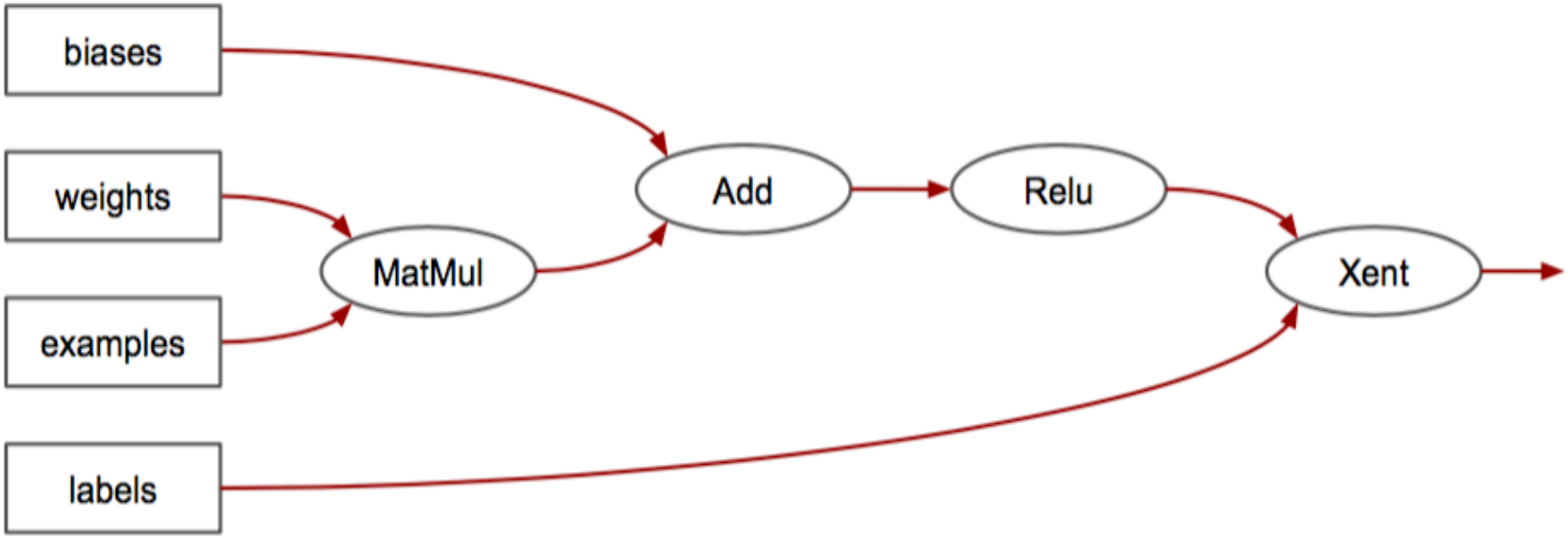
- Computation is expressed as a dataflow graph



Computational Graph (Edges)



- Edges are N-dimensional Arrays: Tensors



Technically Deep Learning is about Tensors on computational graphs.

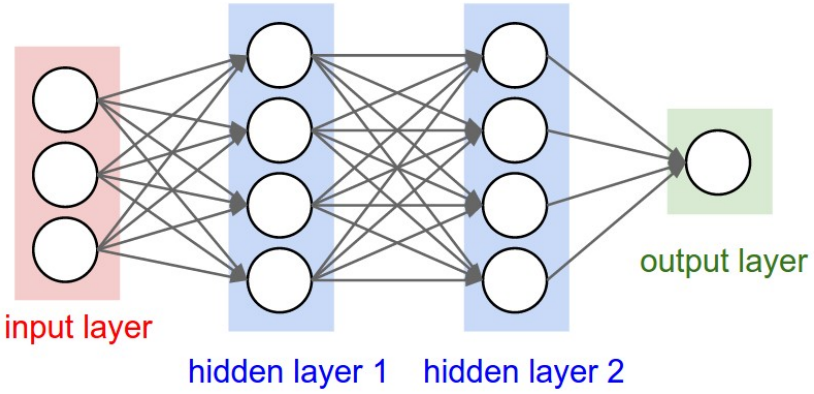
TensorFlow

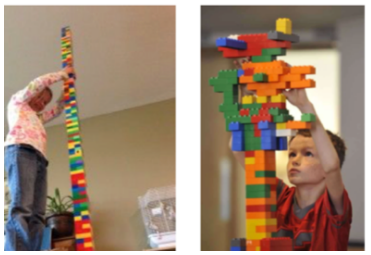
Quite perfect name and logo for a framework

Introduction to Keras

Keras as High-Level library to TensorFlow

- There is a multitude of libraries (currently too many!) which help you with training and setting up the networks.
- Main competitors on low-level pytorch and TensorFlow
- We use Keras as high-level library
- Libraries make use of the Lego like block structure of networks





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

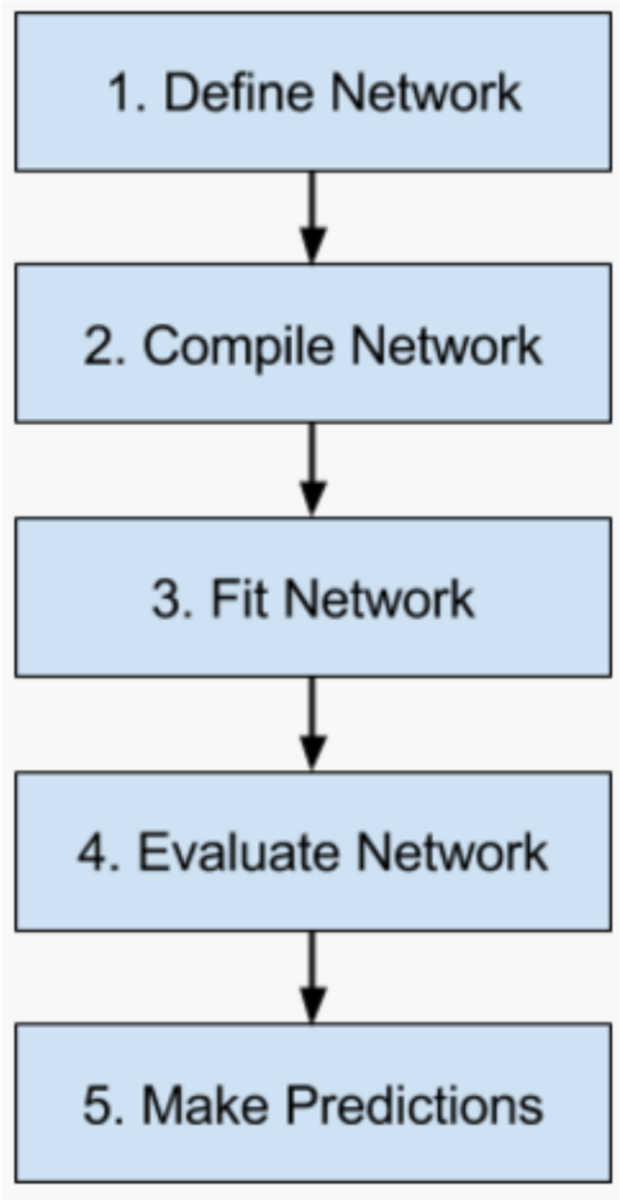
[marketing]

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

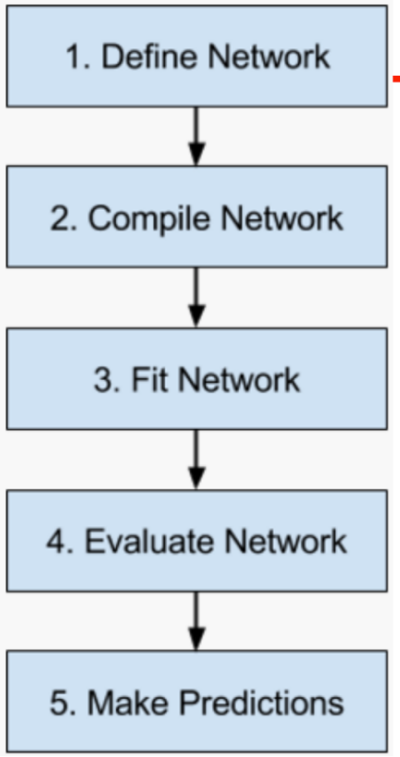
Fit network to training data

Evaluate network on test data

Use in production

A first run through

Define the network



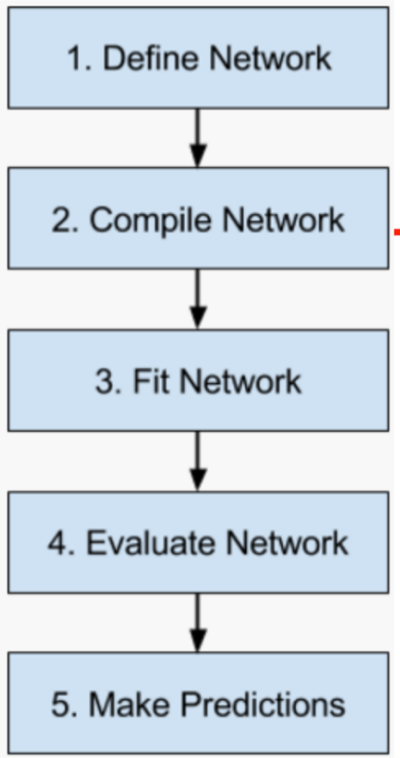
Number of neurons in (first) hidden dense layers (will be input to next layer)

```
model = Sequential()

model.add(Dense(500, batch_input_shape=(None, 784)))
model.add(keras.layers.normalization.BatchNormalization())
model.add(Dropout(0.3))
model.add(Activation('relu'))

model.add(Dense(50))
model.add(keras.layers.normalization.BatchNormalization())
model.add(Dropout(0.3))
model.add(Activation('relu'))

model.add(Dense(10, activation='softmax'))
```

Input shape needs to be defined only at the beginning.

Alternative: `input_dim=784`

Compile the network



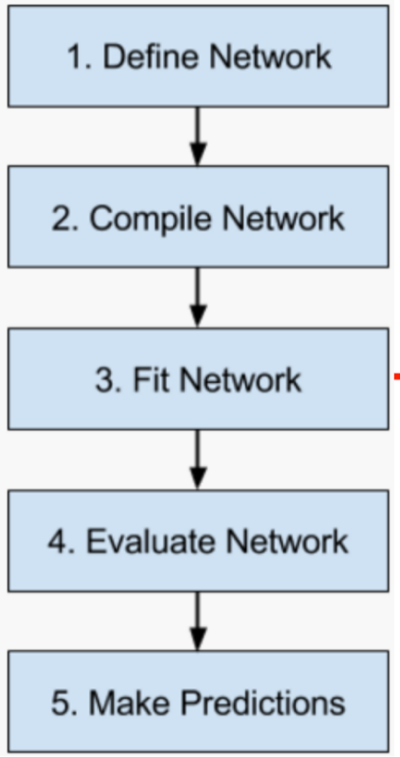
loss function that will be minimized

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

easiest optimizer is SGD
(stochastic gradient descent)

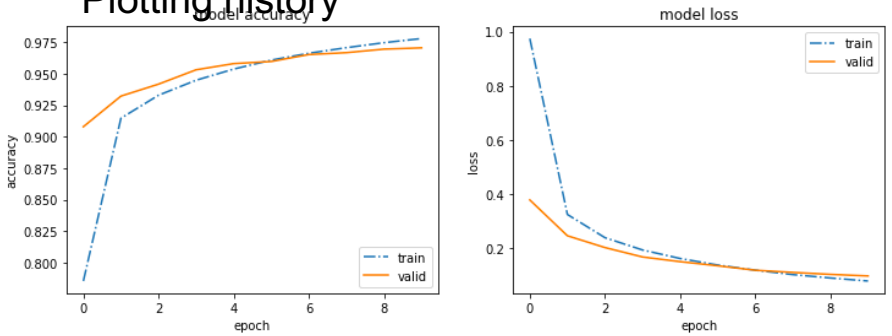
Which metrics besides 'loss' do we want to collect during training

Fit the network

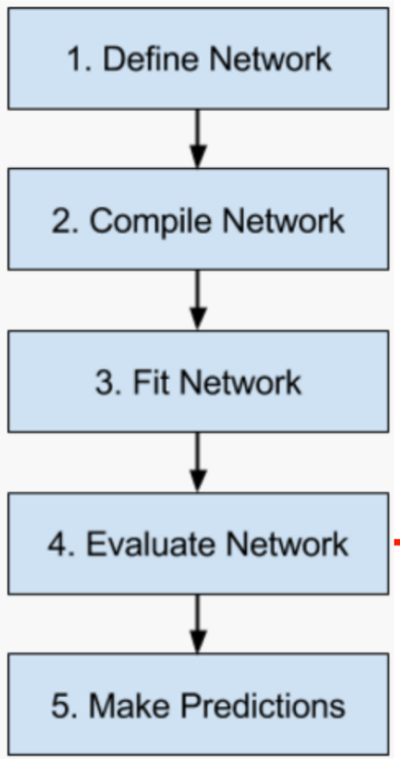



```
# define information required for tensorboard
tensorboard = keras.callbacks.TensorBoard(
    log_dir='tensorboard/mnist_small/' + name + '/',
    write_graph=True,
    histogram_freq=1)

# train the model, memorize training history
history = model.fit(X[0:2400],                    } input data (train)
          convertToOneHot(y[0:2400],10),          } output/label (train)
          epochs=30,                              } How and how often provide training data
          batch_size=128,
          callbacks=[tensorboard],
          validation_data=[X[2400:3000],
                           convertToOneHot(y[2400:3000],10)])
```

Plotting history



Evaluate the network




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

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

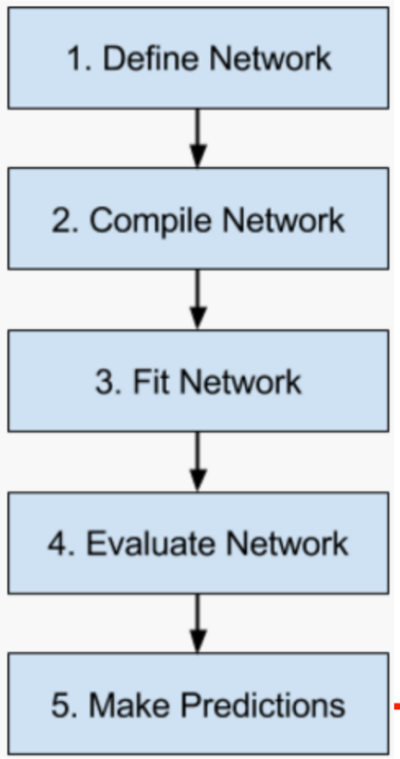
```
[2.5549037284851073, 0.1085]
```

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])
```

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]

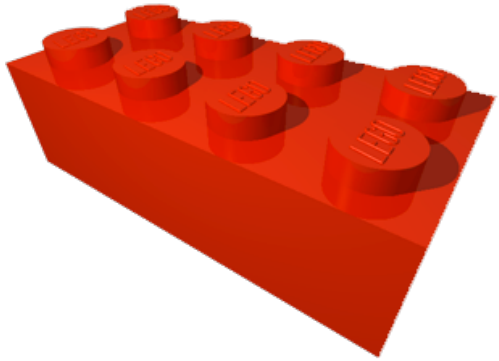
Model subclassing

```
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

- 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

Output Layer

The last layer of the network is a bit special

- Classification
  - # of nodes = # of classes
    - For binary classification sometime only probability of class 1 is reported
  - Usually output is probability for class
    - Use softmax in that case

- Regression (simple distributions)
  - In the interpretation the output of a NN is a probability
  - #nodes = 1
  - Gaussian with fixed variance (the usual regression)
  - Poisson (count data) see later

- Regression (more complicated distributions)
  - #nodes = #number of parameters for distribution

Tasks

1. Finish NB 01_simple_forward_pass.ipynb have a look at the Keras implementation
2. Do exercise NB 02
   https://github.com/tensorchiefs/dl_course_2020/blob/master/notebooks/02_fcnn_with_banknote.ipynb