Introduction to Deep Learning with TensorFlow 2

embarc Software Consulting GmbH

https://www.oose.de/seminar/deep-learning/

Slides: https://bit.ly/deep-oose

Ansicht Speichern als PDF

wifi - Name: oose-public, pw: VomWissenzumKoennen98

Rechner oose, oose

https://github.com/embarced/notebooks

embarc Software Consulting GmbH

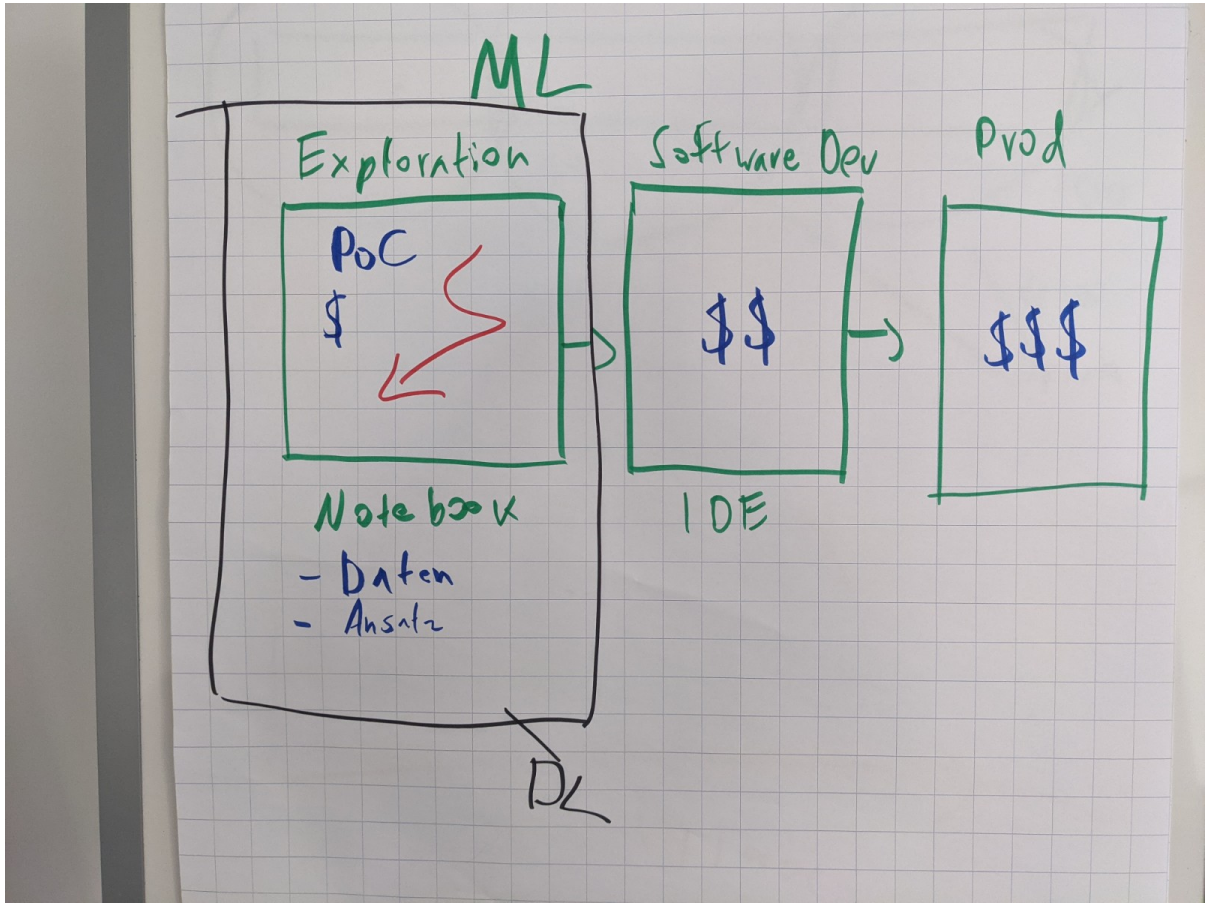
Diese Bücher braucht man

- Intro: https://www.manning.com/books/deep-learning-with-python-second-edition?gclid=Cj0KCQjwpreJBhDvARIsAF1_BU3kz9n8JeUN7JpmUhkPrah-mIdLBB9yORujbSzPMvojKXTVtvZgrXsaAjruEALw_wcB
- Advanced: https://www.oreilly.com/library/view/hands-on-machine-learning/9781492032632/

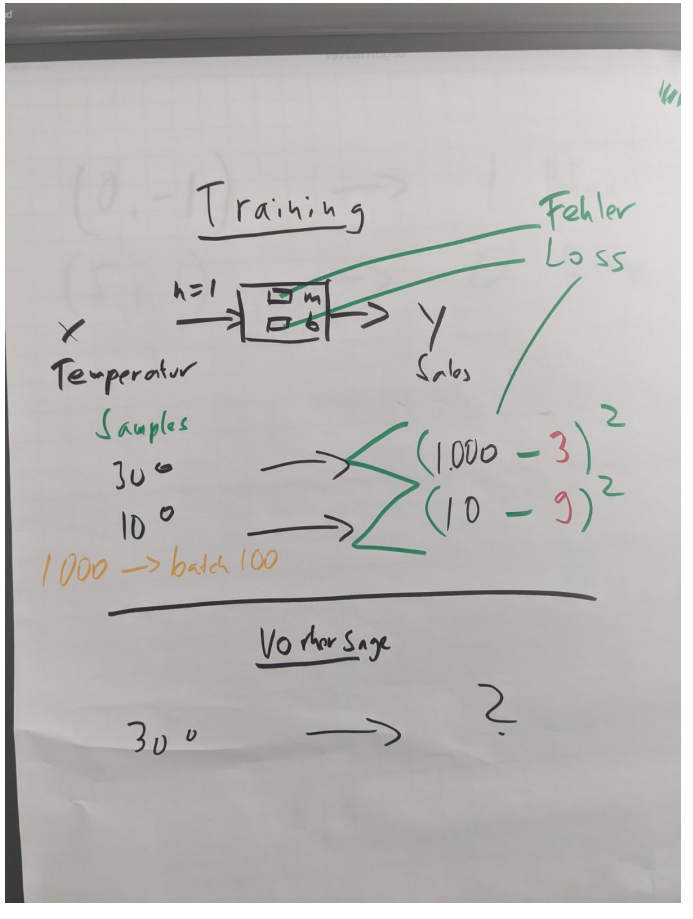
Welche Konferenz kann man besuchen?

- https://odsc.com/
- https://mlconference.ai/berlin/
- https://www.m3-konferenz.de/

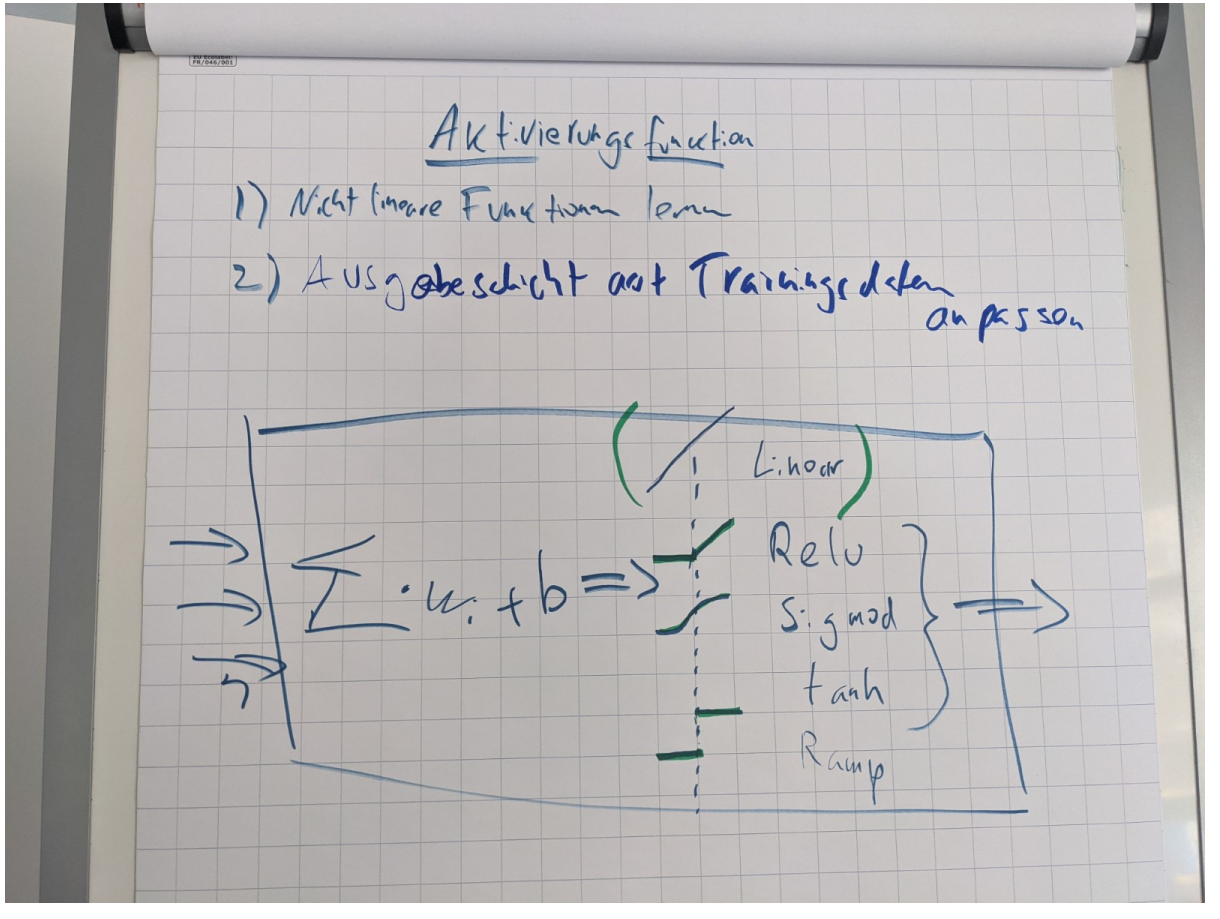
Phases of ML Projects



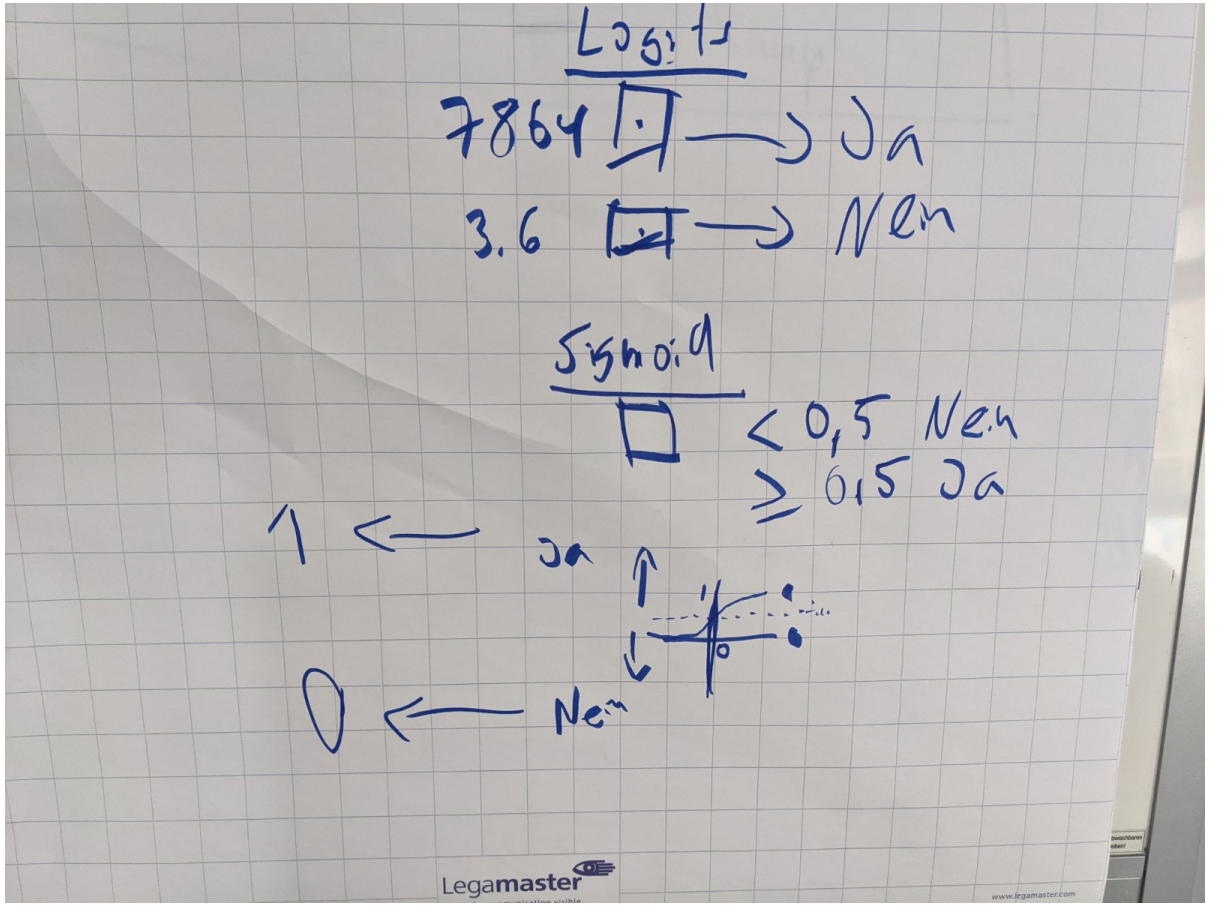
Training vs Prediction



Why Activation Functions?



Modelling Binary Decisions



Schedule

- Day 1: Introduction
    1. TensorFlow
    2. Basics of Supervised Learning
    3. Deep Learning best practices
- Day 2: Applications
    1. Tabular Data
    2. Image Recognition
    3. Sequences
- Day 3: Advanced
    1. Unsupervised Deep Learning
    2. Deep Reinforcement Learning

All Notebooks All Playgrounds

Objective: What is Deep Learning

- Basic ideas
- Why deep learning?

Why Deep Learning?

Software 1.0

Written in code (C++, …)
Requires domain expertise
1. Decompose the problem
2. Design algorithms
3. Compose into a system

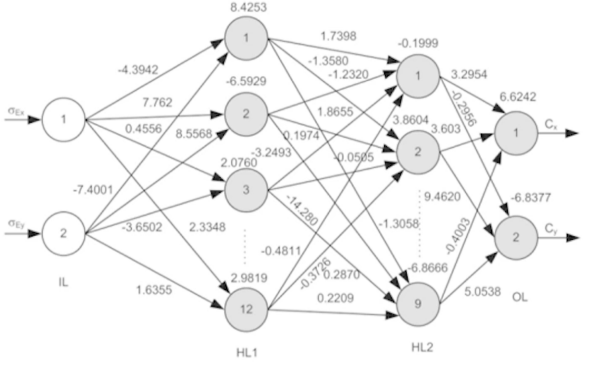
Measure performance

Andrej Karpathy - TRAIN AI 2018 - Building the Software 2.0 Stack

https://vimeo.com/272696002

Why Deep Learning?

Software 2.0



"Fill in the blanks programming"

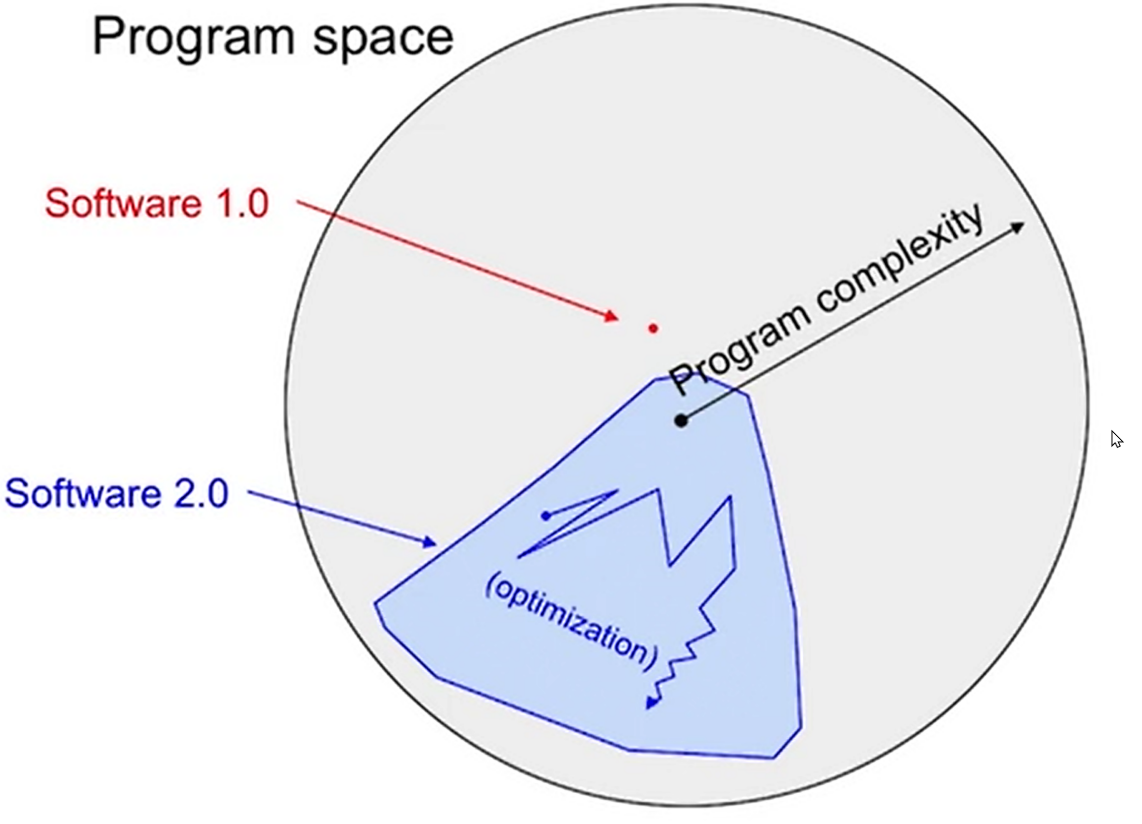
Requires much less domain expertise

1. Design a "code skeleton" ← (automate)
2. Measure performance

Andrej Karpathy - TRAIN AI 2018 - Building the Software 2.0 Stack

https://vimeo.com/272696002

Why Deep Learning?



Andrej Karpathy - TRAIN AI 2018 - Building the Software 2.0 Stack

https://vimeo.com/272696002

Approach for this course: Deep Learning fundamentally is software engineering

The ability to understand mathematical processes is widespread, but the ability to parse mathematical notation isn't. Insisting on teaching deep learning (which is fundamentally software engineering, not mathematics) with equations is like insisting on teaching Hegel in German.

Deep learning isn't a science, but rather an ever-changing set of empirically-derived engineering best practices, woven together by over-claiming, unreliable narratives.

https://twitter.com/fchollet/status/1369793906922024965
https://twitter.com/fchollet/status/1333173017678028802

Deep Learning requires special skills

- you need to understand
  - concepts, methods, processes and limitations of ML
  - many different levels of abstration from very low level code to high level concepts
- TensorFlow is not just a library whose API you can learn

Quickstart

https://colab.research.google.com/github/embarced/notebooks/blob/master/deep/intro_quickstart.ipynb?hl=en

1. Make sure you are ready to work with Colab
    - open the notebook in Chrome
    - make it run using the "Run All" command from the "Runtime" menu
    - you need to allow execution and must either have a Google login or are willing to create one
    - make a copy of the notebook to your Google Drive
2. Shared exploration of the notebook
    - Notebooks
    - Colab
    - Python
    - Using a Machine Learning Model

Schedule

- Day 1: Introduction
    1. TensorFlow
    2. Basics of Supervised Learning
    3. Deep Learning best practices
- Day 2: Applications
    1. Tabular Data
    2. Image Recognition
    3. Sequences
- Day 3: Advanced
    1. Unsupervised Deep Learning
    2. Deep Reinforcement Learning

All Notebooks All Playgrounds

Schedule

- Day 1: Introduction
    1. ***TensorFlow***
    2. Basics of Supervised Learning
    3. Deep Learning best practices
- Day 2: Applications
    1. Tabular Data
    2. Image Recognition
    3. Sequences
- Day 3: Advanced
    1. Unsupervised Deep Learning
    2. Deep Reinforcement Learning

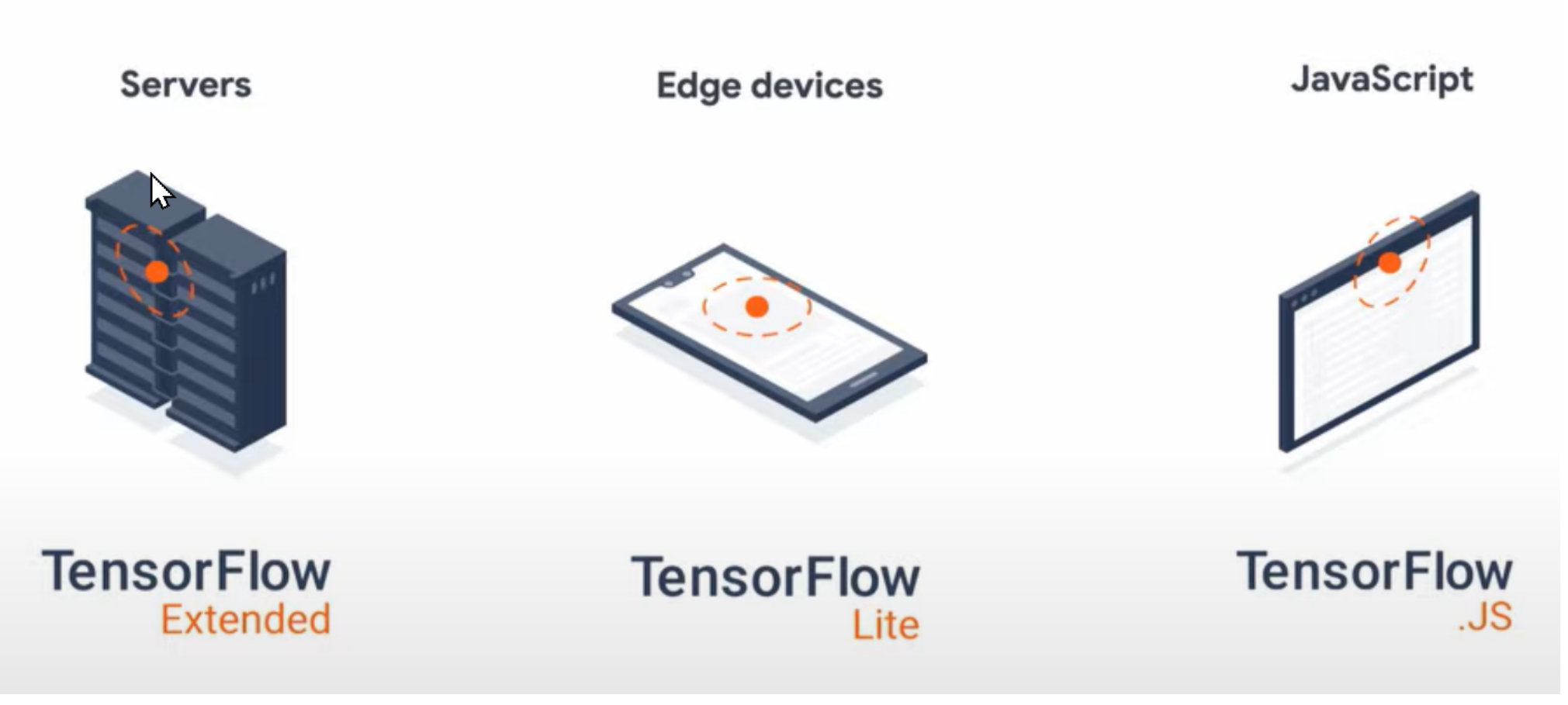
What is TensorFlow

- Platform
- Ecosystem
- Foundation for differentiable programming

Numpy++

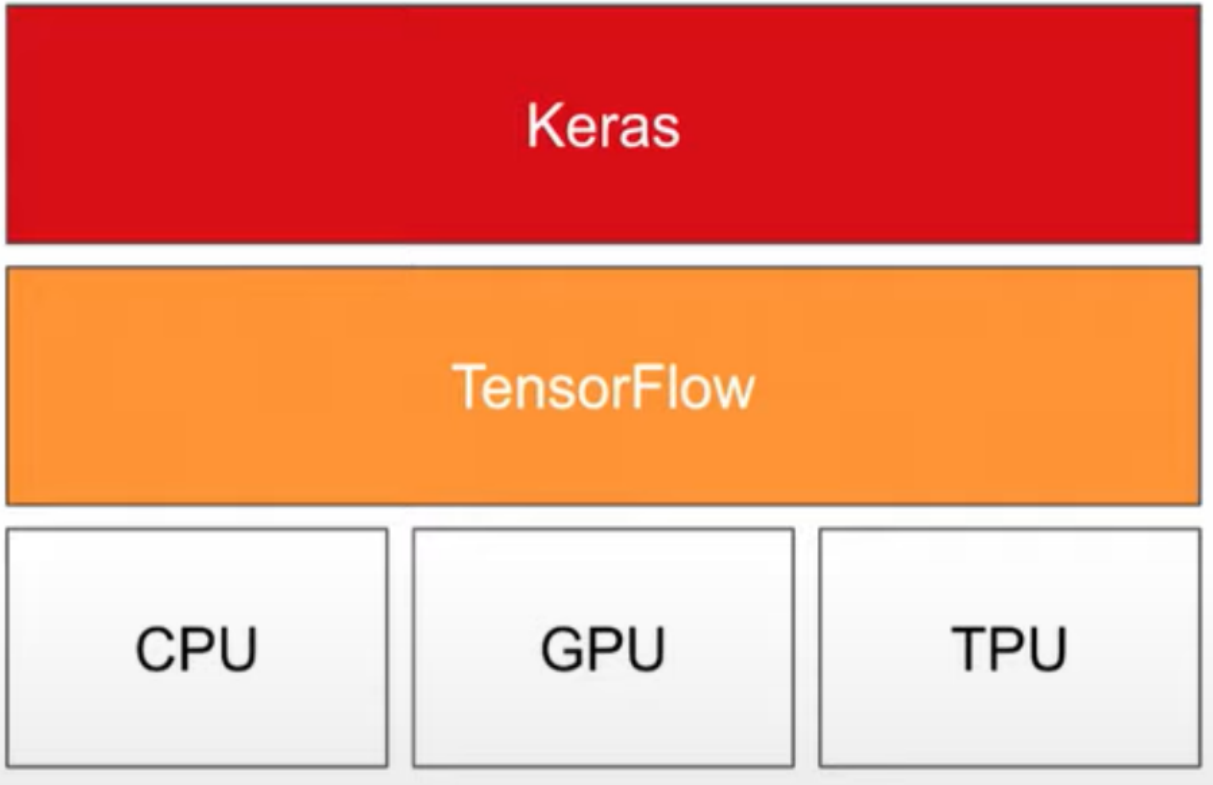
- Gradients
- GPU, TPU
- Tensor = Multi Dimensional Array

Strong Deployment Story



- https://www.tensorflow.org/tfx/guide/serving
- https://www.tensorflow.org/lite/api_docs
- https://www.tensorflow.org/js

Keras vs TensorFlow



Deep learning development: layers, models, optimizers, losses, metrics...

Tensor manipulation infrastructure: tensors, variables, automatic differentiation, distribution...

Hardware: execution

Francóis Chollet - Keras: The Next Five Years, Scaled ML

TensorFlow Low-Level Training Loop

```
optimizer = tf.keras.optimizers.SGD(learning_rate=1e-3)
loss_fn = tf.keras.losses.SparseCategoricalCrossentropy()

train_dataset = tf.data.Dataset.from_tensor_slices((x_train, y_train))
train_dataset = train_dataset.shuffle(buffer_size=1024).batch(batch_size)

val_dataset = tf.data.Dataset.from_tensor_slices((x_val, y_val))
val_dataset = val_dataset.batch(batch_size)

for epoch in range(epochs):
    for step, (x_batch_train, y_batch_train) in enumerate(train_dataset):
        with tf.GradientTape() as tape:
            y_batch_pred = model(x_batch_train, training=True)
            loss_value = loss_fn(y_batch_train, y_batch_pred)
        grads = tape.gradient(loss_value, model.trainable_weights)
        optimizer.apply_gradients(zip(grads, model.trainable_weights))
        train_acc_metric.update_state(y_batch_train, y_batch_pred)

    train_acc = train_acc_metric.result()
    train_acc_metric.reset_states()

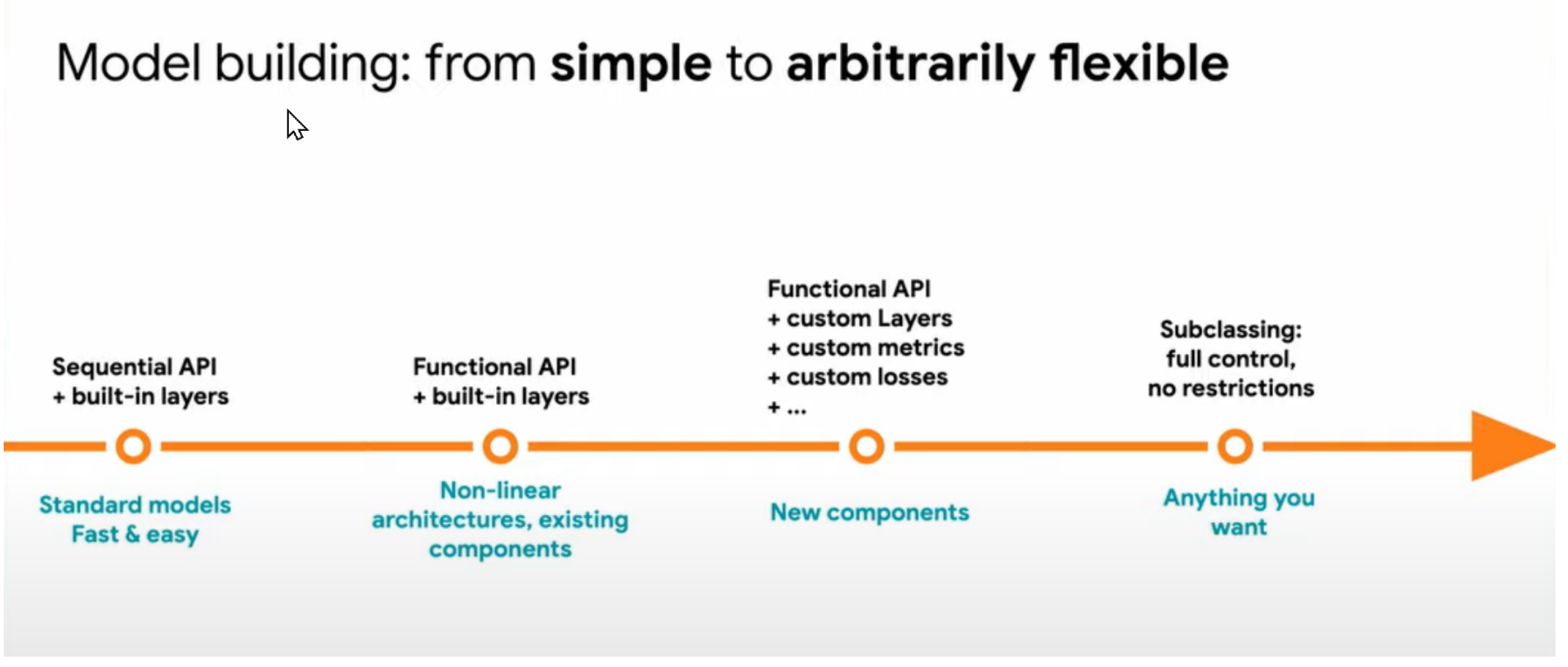
    for x_batch_val, y_batch_val in val_dataset:
        y_batch_val_pred = model(x_batch_val, training=False)
        val_acc_metric.update_state(y_batch_val, y_batch_val_pred)

    val_acc = val_acc_metric.result()
    val_acc_metric.reset_states()
```

https://www.tensorflow.org/guide/basic_training_loops

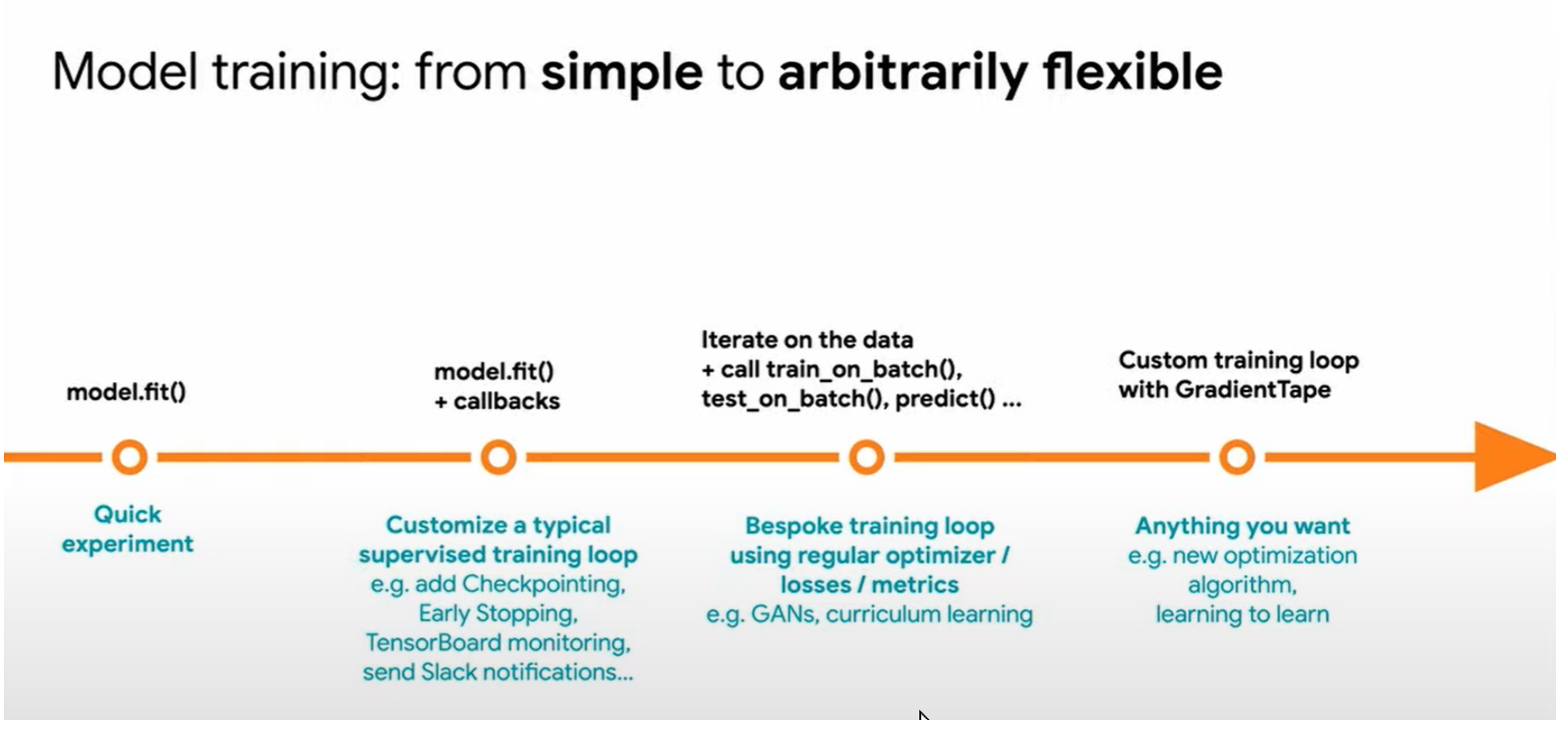
https://www.tensorflow.org/guide/keras/writing_a_training_loop_from_scratch

Keras: Progressive Disclosure of Complexity



Francóis Chollet - Keras: The Next Five Years, Scaled ML

Keras can go from Scikit-Learn API style all the way to Numpy API style



Francóis Chollet - Keras: The Next Five Years, Scaled ML

Keras High-Level Training "Loop"

```
optimizer = tf.keras.optimizers.SGD(learning_rate=1e-3)
loss_fn = tf.keras.losses.SparseCategoricalCrossentropy()

model.compile(
    optimizer=optimizer,
    loss=loss_fn,
    metrics=['accuracy']
)
model.fit(x_train, y_train, epochs=epochs, batch_size=batch_size, validation_data=(x_val, y
```

https://www.tensorflow.org/guide/basic_training_loops

https://www.tensorflow.org/api_docs/python/tf/keras/Model

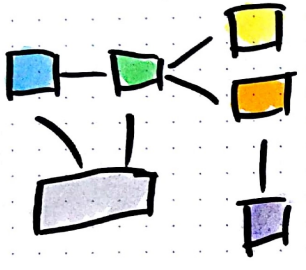
Schedule

- Day 1: Introduction
    1. TensorFlow
    2. ***Basics of Supervised Learning***
    3. Deep Learning best practices
- Day 2: Applications
    1. Tabular Data
    2. Image Recognition
    3. Sequences
- Day 3: Advanced
    1. Unsupervised Deep Learning
    2. Deep Reinforcement Learning

Supervised Learning

recover an unknown function from possibly noisy measurements

Classic Development



Domain Knowledge

⇒ analyse

Business Analyst

Requirements

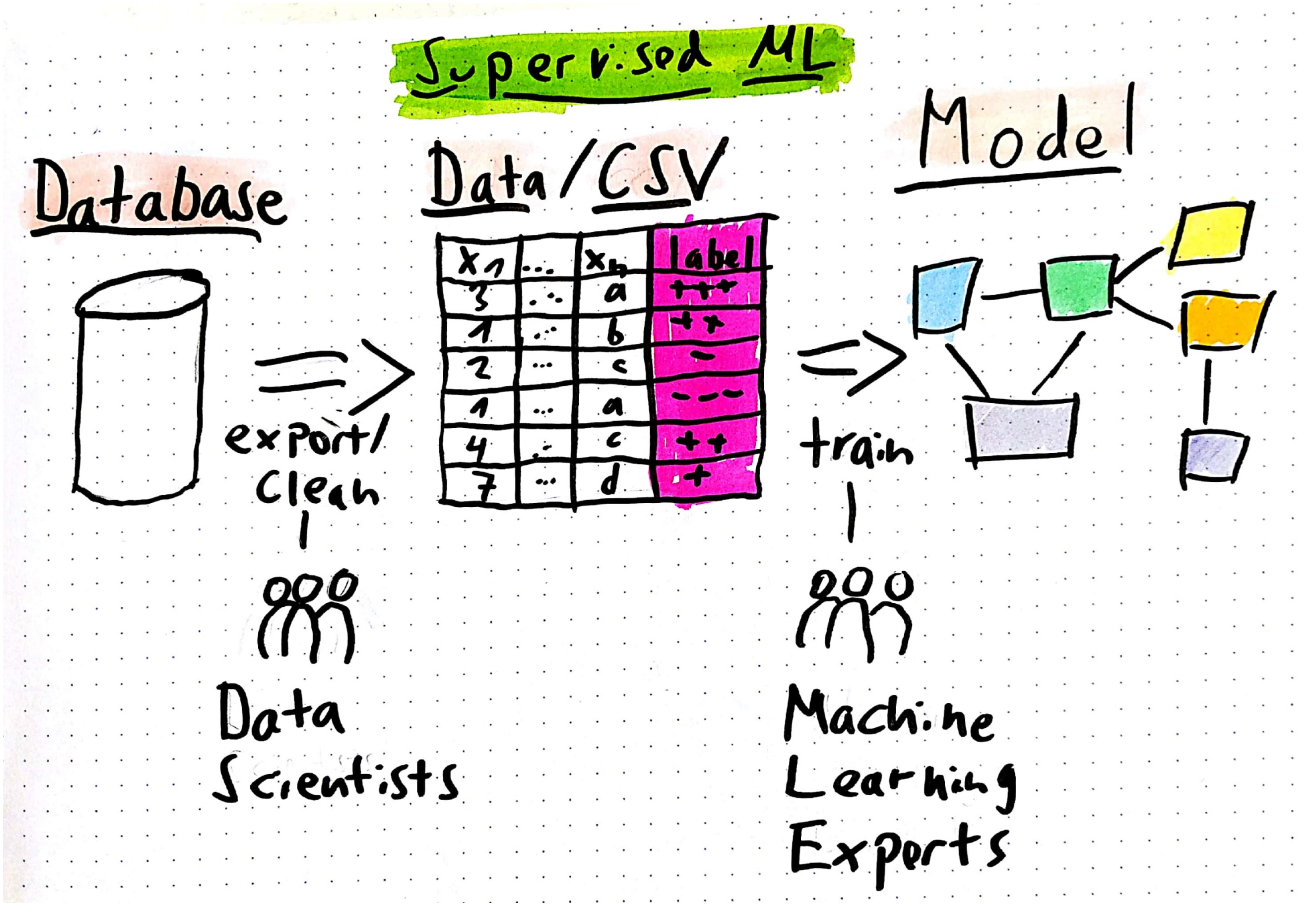
1. ~~~
2. ~~~
3. ~~~

⇒ develop

Software Developer

Programm

```
if x1 > 10:
    return ++
if x2 == 'OFFICE':
    return --
```


Supervised ML
Database
Data/CSV
Model
label
export/
clean
train
Data
Scientists
Machine
Learning
Experts

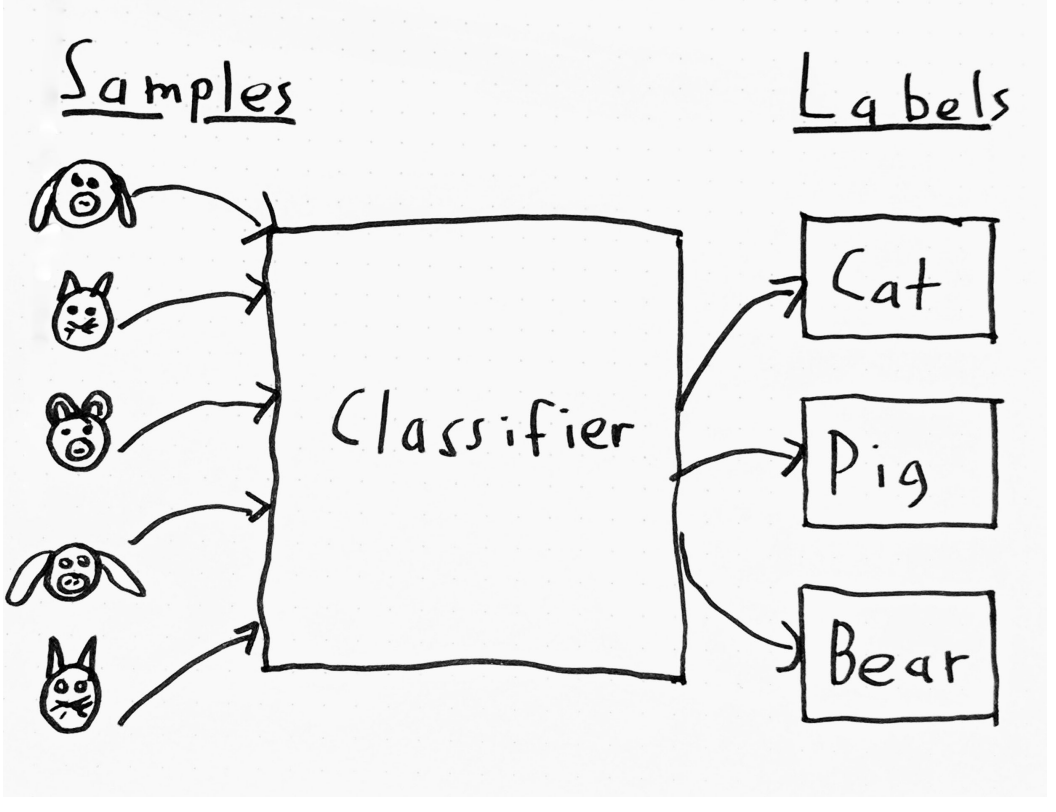
Introduction to Supervised Regression using Colab Notebooks

https://colab.research.google.com/github/embarced/notebooks/blob/master/deep/intro-regression.ipynb?hl=en

make a copy of the notebook to your Google Drive

Classification vs Regression

Regressions predict a quantity, and classifications predict a label



1. Regression: Fitting a curve (often a line) through data points
2. Classification: What category can be derived from data

Introduction to Supervised Classification using Colab Notebooks

https://colab.research.google.com/github/embarced/notebooks/blob/master/deep/intro-classification.ipynb?hl=en

make a copy of the notebook to your Google Drive

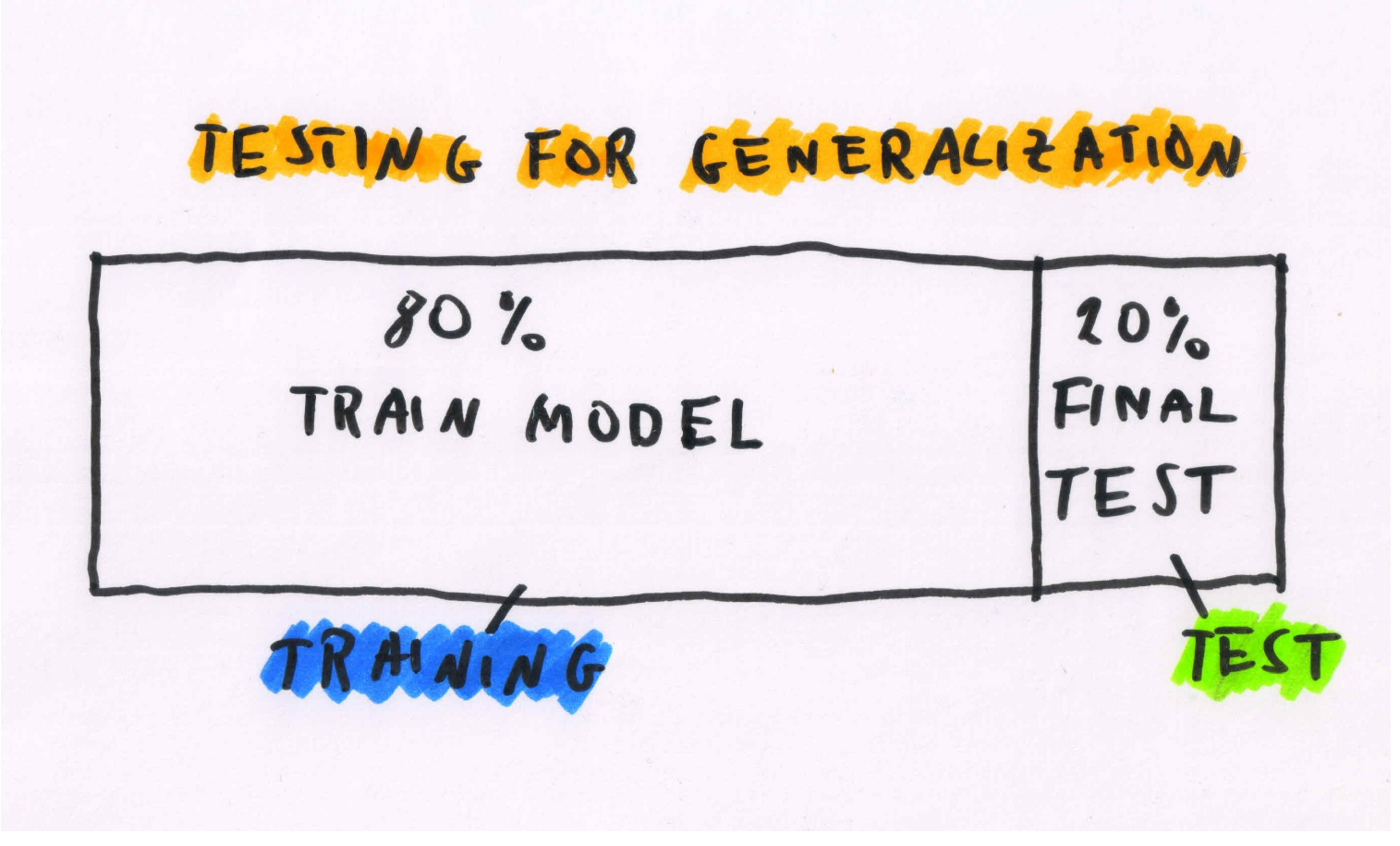
Machine Learning is all about Generalization

1. Learn from known data
2. ***Make predictions on unknown data***

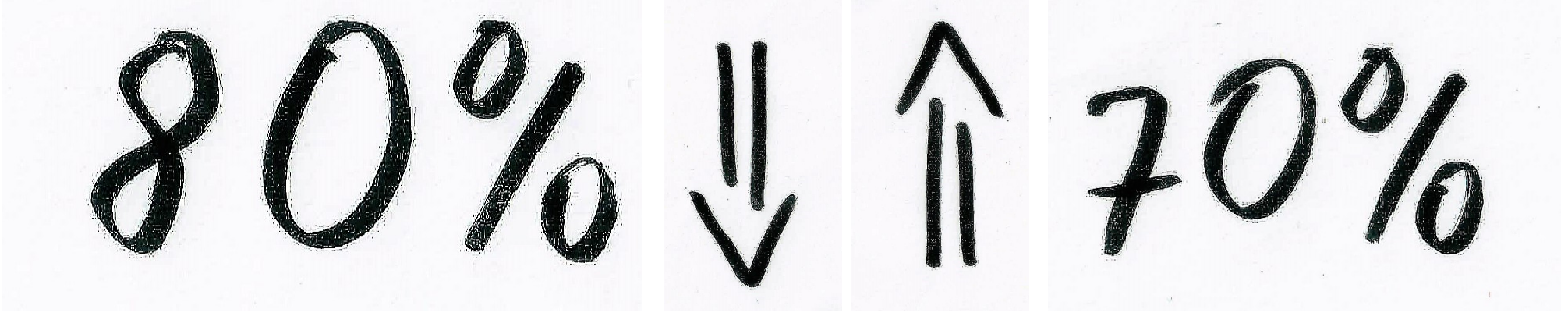
So Supervised Machine Learning is all about the data you have not seen, yet

How to make sure your classification works well on unseen data?

The trick: Split known data



The Issue: Overfitting



Training Score ***Test Score***

Training and test scores clearly divert

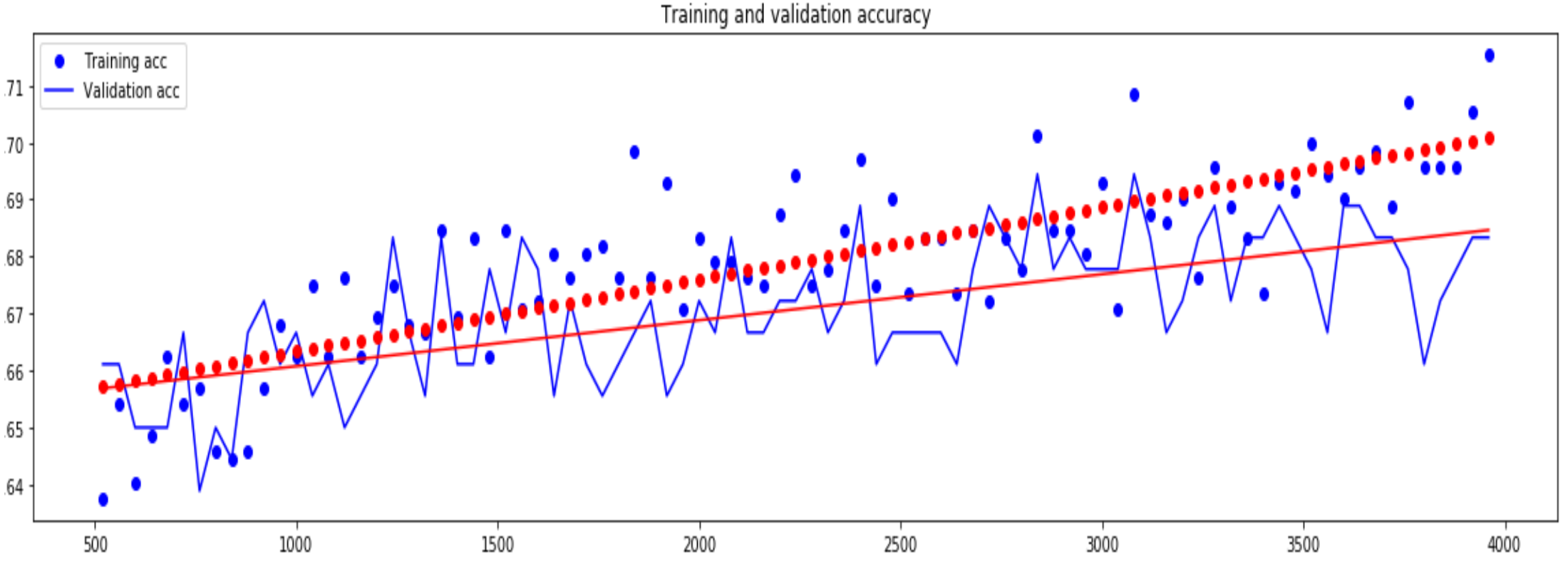
Regularization

Process to counter overfitting

- When there are more variables than data points, the problem may not have a unique solution
- There may be multiple (perhaps infinitely many) solutions that fit the data equally well
- The existence of more variables than data points, the existence of multiple solutions, and overfitting often coincide

https://stats.stackexchange.com/questions/223486/modelling-with-more-variables-than-data-points/223517#223517

First measure: Train for fewer epochs

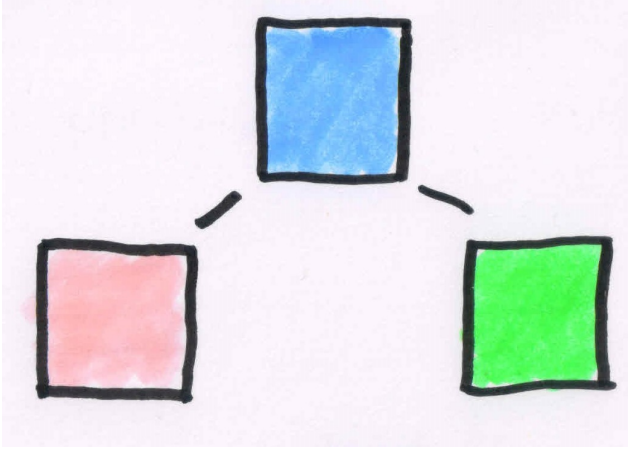


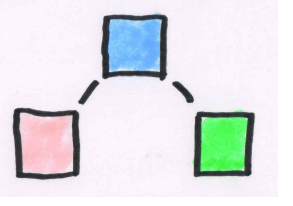
Watch where training and validation accuracy diverge and stop training there

Early stopping possible:

https://keras.io/callbacks/#earlystopping
https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/EarlyStopping

Second measure: Reduce capacity of model





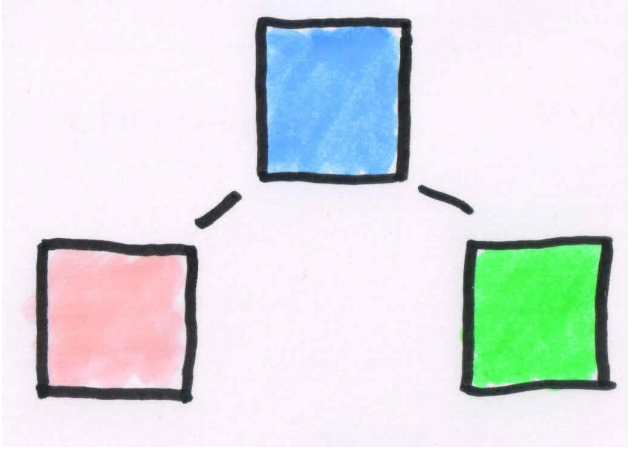
Original model

Smaller model
less hidden layers, less neurons per layer

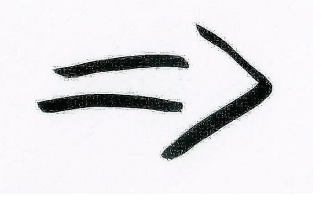
Intuition: Give model less capacity to simply memorize data

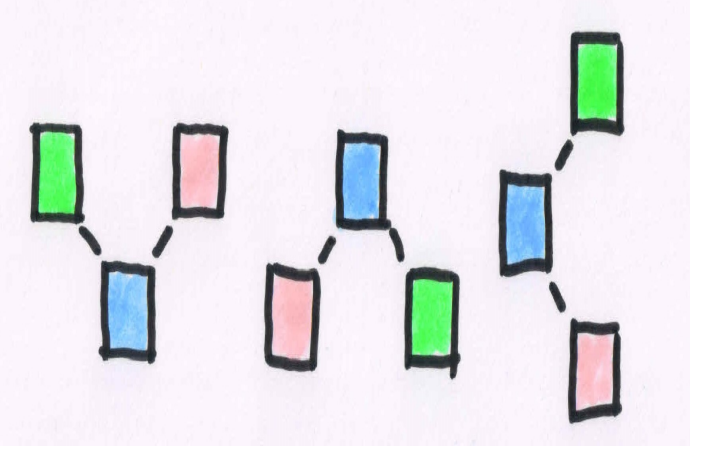
Third measure: Use Dropout

Dropouts only train a certain percentage of neurons per batch



Original model





Ensemble of small models (each one overfits on its specific batch)

Intuition: Combination of models makes result more robust

Fourth measure: Batch Normalization

- basically shifting by mean and rescaling by standard deviation
- during training: normalizes its output using the mean and standard deviation of the current batch of inputs
- during evaluation and prediction: normalizes its output using a moving average of the mean and standard deviation of the batches it has seen during training.

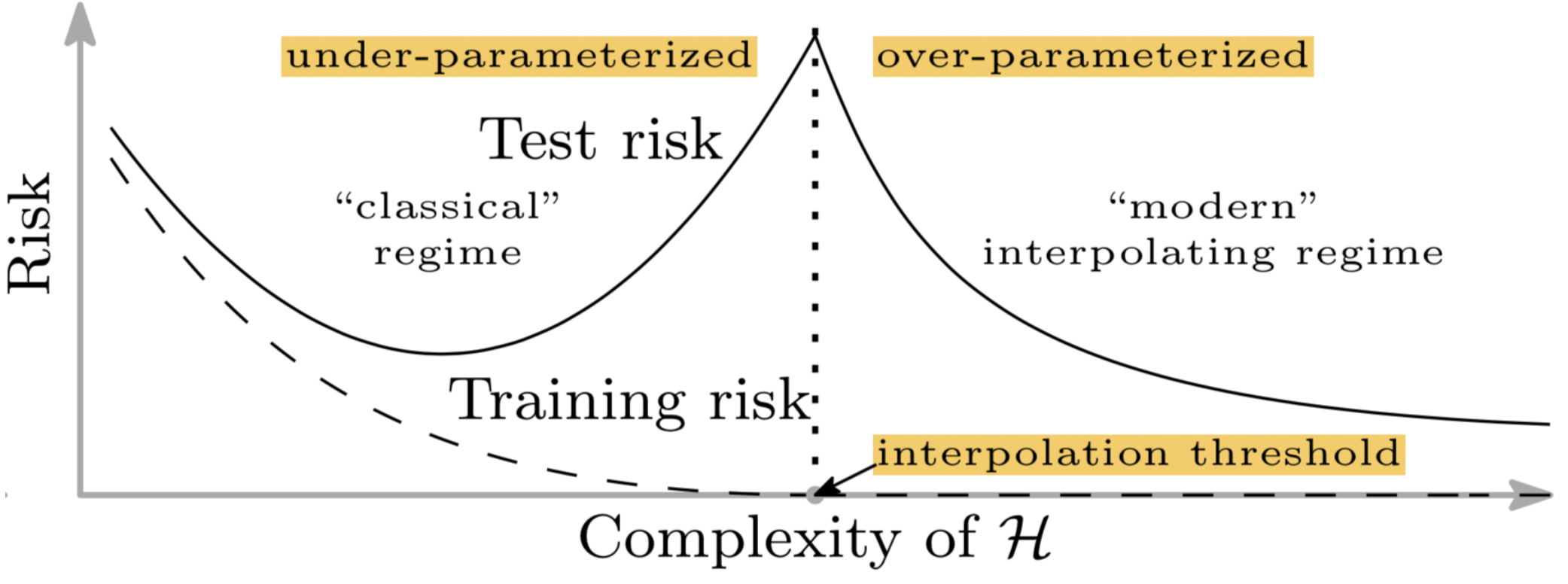
Training BatchNorm and Only BatchNorm: https://openreview.net/forum?id=vYeQQ29Tbvx
https://keras.io/api/layers/normalization_layers/batch_normalization/
BatchNorm smoothes the loss landscape: https://www.youtube.com/watch?v=ZOabsYbmBRM

Fifth approach: L1/L2 weight Regularization

- make model less complex by forcing low values for weights (less complexity, more regular)
- adds penalty term to loss function
- L1 (Lasso Regression): penalty is proportional to the absolute value of the weights coefficients
  - helps drive the weights of irrelevant or barely relevant features to exactly 0
- L2 (Ridge Regression): penalty is proportional to the square of value of the weights coefficients
  - heavily penalizes especially large coefficients

https://colab.research.google.com/github/tensorflow/docs/blob/master/site/en/tutorials/keras/overfit_and_underfit.ipynb#scrollTo=4rHoVWcswFLa
https://towardsdatascience.com/l1-and-l2-regularization-methods-ce25e7fc831c

New School of Generalization



https://arxiv.org/abs/1812.11118
https://twitter.com/IanOsband/status/1164900840106274817

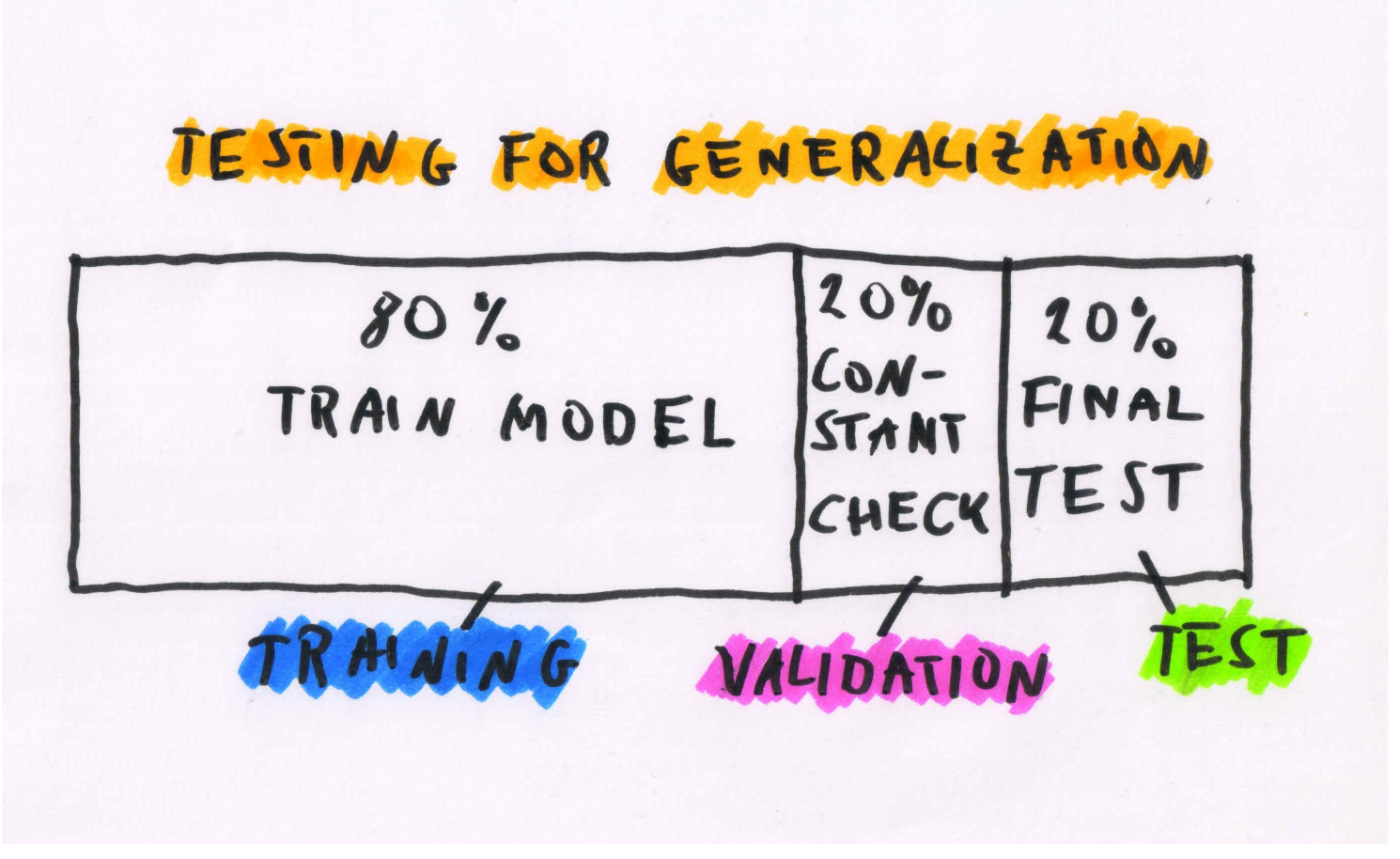
Final measure: Get more training data

if you can

if not

- try augmenting existing data
- use transfer learning (rarely works)
- carefully curate your data

Validation data set as another level of evaluation



Introduction to Regularization using Colab Notebooks

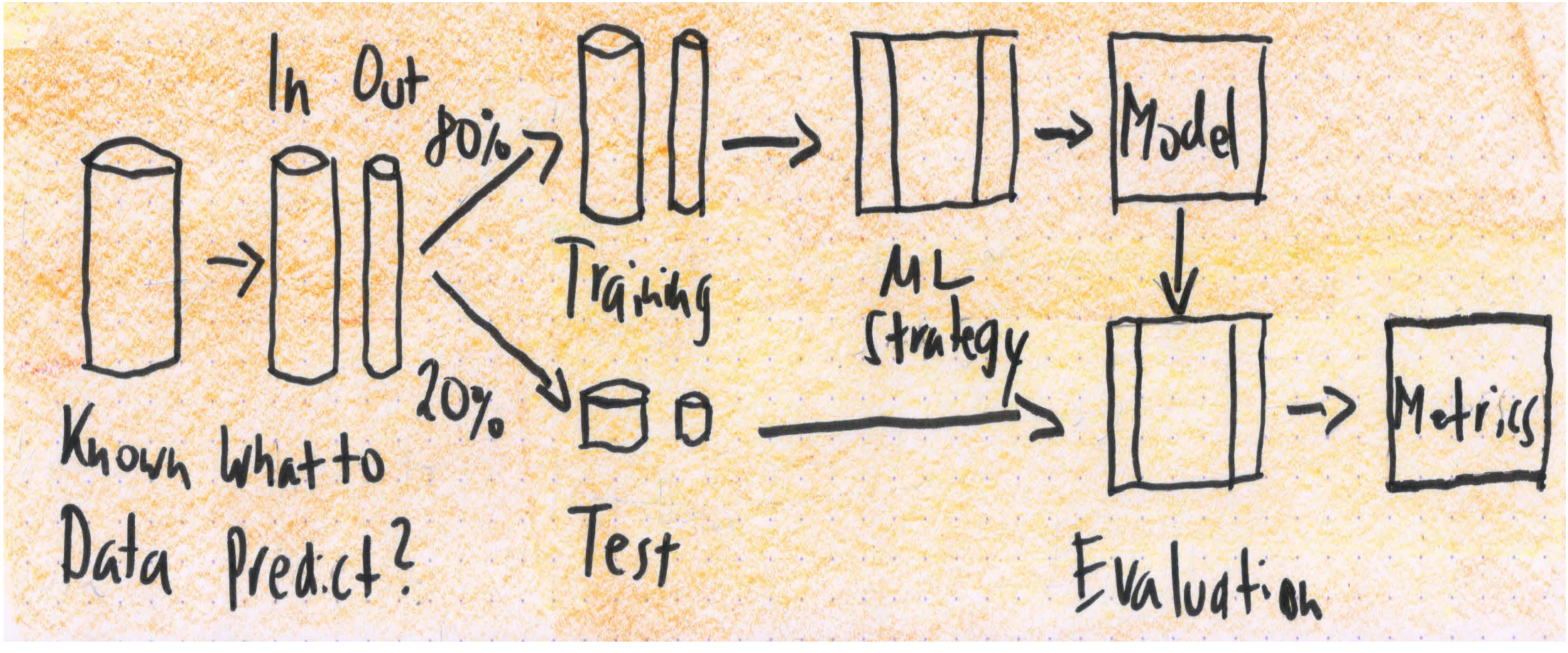
Hands-On: Choose one approach and try it out for yourself

https://colab.research.google.com/github/embarced/notebooks/blob/master/deep/intro-regularization.ipynb?hl=en

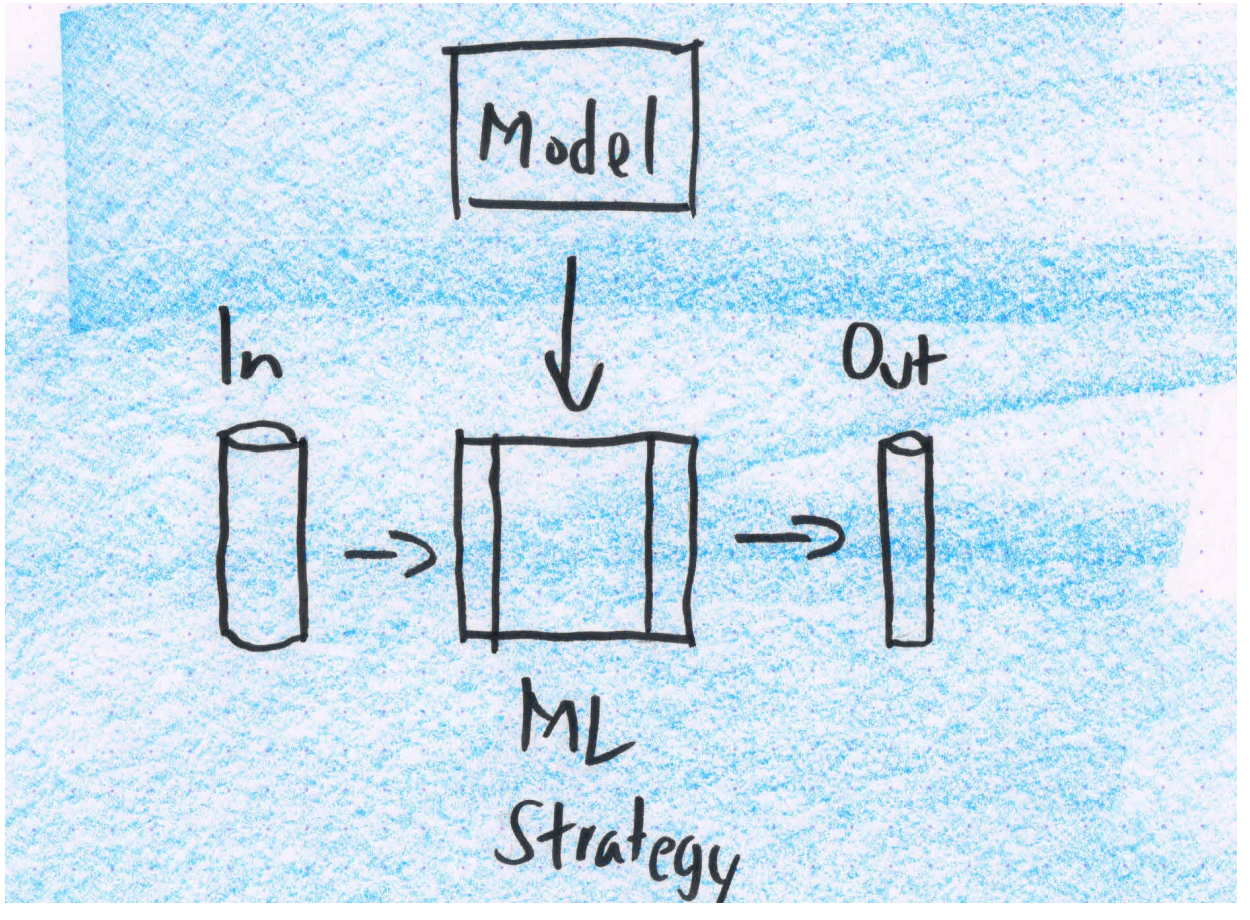
make a copy of the notebook to your Google Drive

Summary Supervised Machine Learning

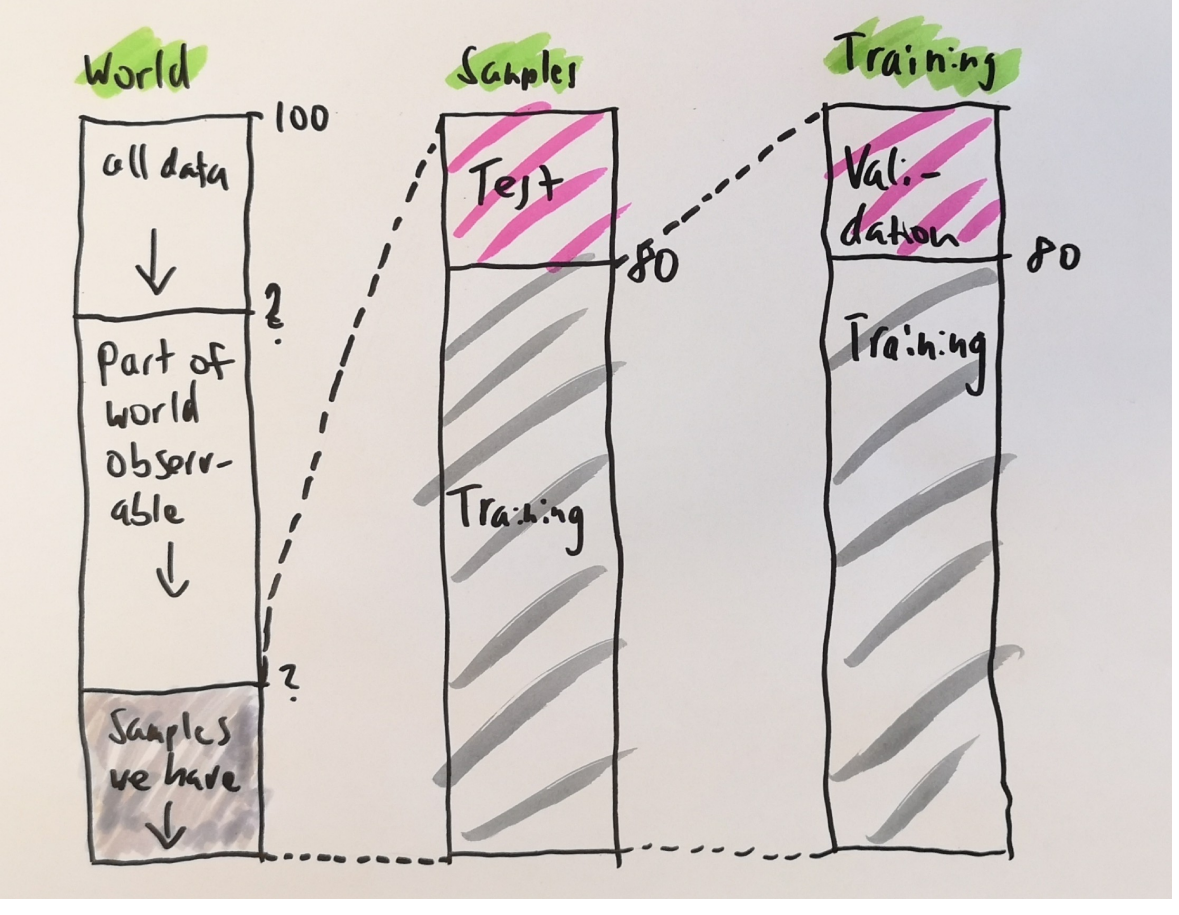
Overview: Supervised Learning Process Flow



Prediction



Be Careful: Good Test Score is no Guarantee



When to apply Supervised Learning

classify categories or predict values

- you have matching pairs of input and output and want the model to generate output for unknown input
- the solution to the problem at hand is unknown or hard to specify
- solving the problem can tolerate some error or uncertainty
- there is a clear, simple input and output
- there are patterns in your input that can be used to predict the output

what we see applied most of the time

Schedule

- Day 1: Introduction
    1. TensorFlow
    2. Basics of Supervised Learning
    3. ***Deep Learning best practices***
- Day 2: Applications
    1. Tabular Data
    2. Image Recognition
    3. Sequences
- Day 3: Advanced
    1. Unsupervised Deep Learning
    2. Deep Reinforcement Learning

What is a neural network "doing"?

Multiplication of a vector with a matrix

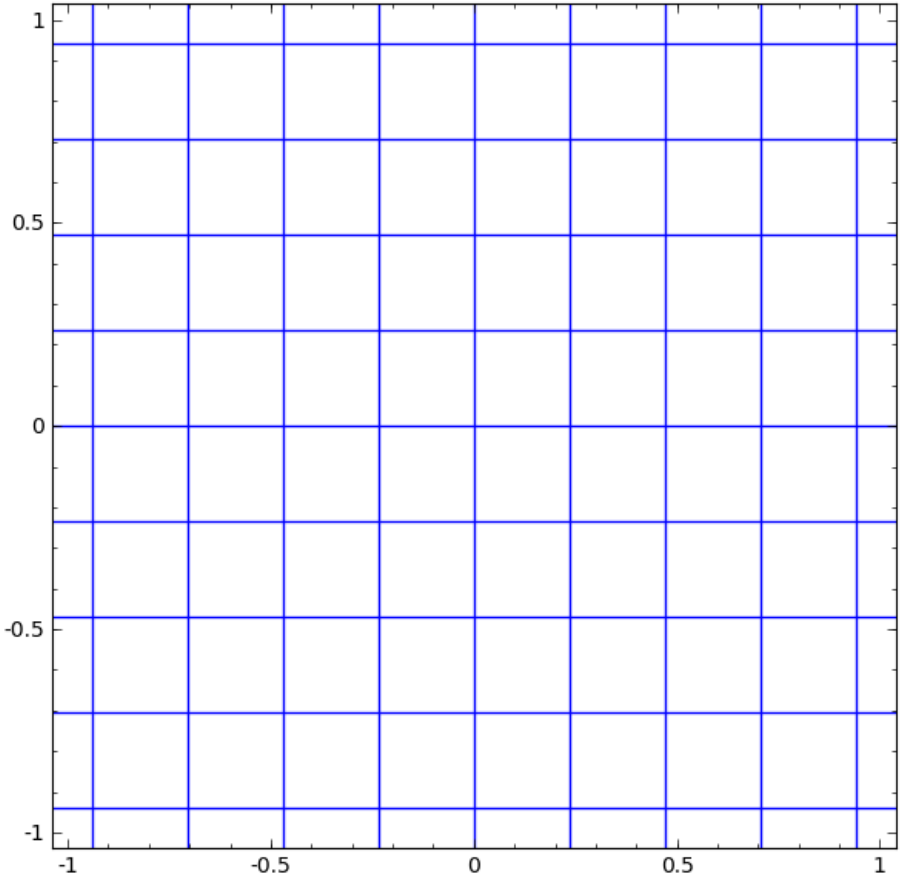
- A compressing rotation of the vector
- stretching = negative compression.
- "independently", e.g. if in dimension x is stretched with 2, then in dimension y can be compressed with -0.5
- if you do this more often, it still remains linear
- Can you then actually combine many such multiplications into one?
  - Yes, multiply the matrices
- Bias adds translation

Activation function

Desired properties

- non-linear for hiddden layers (because otherwise everything remains linear even with many layers)
- good gradients that do not vanish and also do not explode
- cheap to calculate

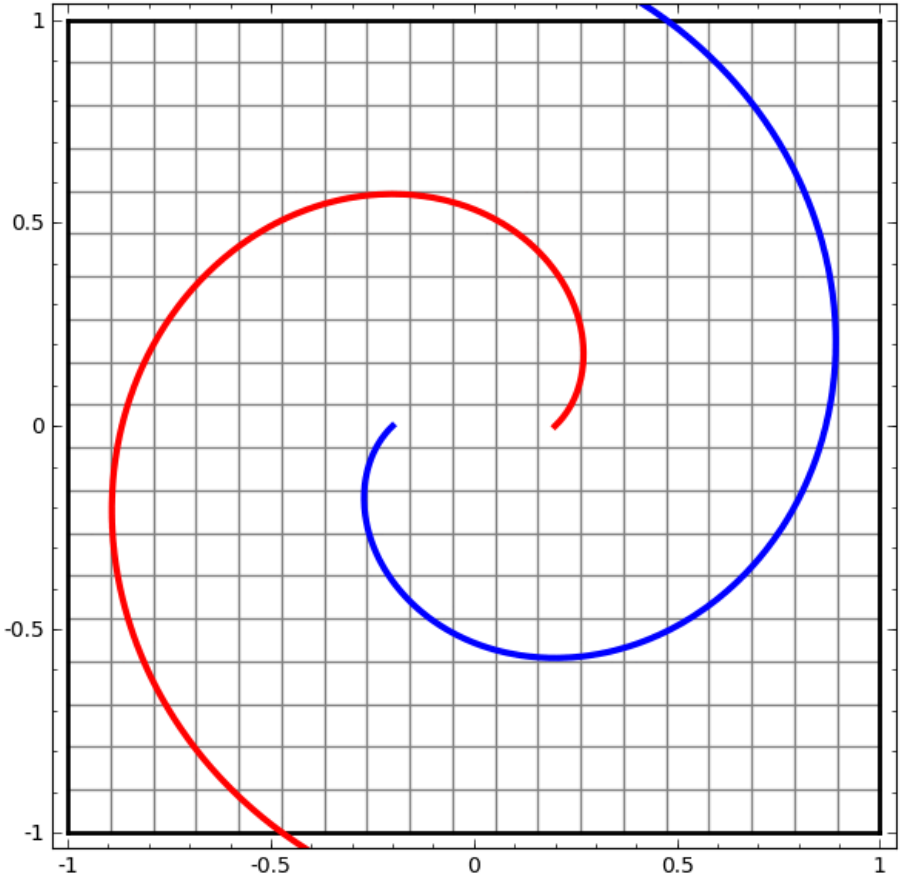
Visual explanation, single layer



https://colah.github.io/posts/2014-03-NN-Manifolds-Topology/

https://www.youtube.com/watch?v=vdqu6fvjc5c

Complex network: making data linearly separable



https://colah.github.io/posts/2014-03-NN-Manifolds-Topology/

https://www.youtube.com/watch?v=vdqu6fvjc5c

ML Models are restricted by their training data

Jacob Menick
@jacobmenick

As soon as you realise an ML model is just a view of the data it's been trained on you start getting real interested in datasets

9:50 PM · May 13, 2020 · Twitter Web App

https://twitter.com/jacobmenick/status/1260658763687538688

Models will only learn what you feed them

ME: Hey want to play an imagination game? I tell you to imagine something and then you answer a question, ok?

BOT: That sounds like a lot of fun. I would love to do that. I love imagination games.

ME: Great. Imagine I placed wet bread on a large diamond. What will happen to the diamond?

BOT: It will get soggy and you will have to throw it away. What else do you like to do?

https://twitter.com/TomerUllman/status/1259834838829465601 https://twitter.com/mikiobraun/status/1260115051417124869

Practical advice from a master of his craft

Challenges of training neural nets

1. Neural net training is a leaky abstraction - you need to understand what is going on
2. Neural net training fails silently - the possible error surface is large

The recipe

1. Understand your data
2. Make one simple experiment after the other
3. Make your model good / large enough to overfit on a batch
4. Regularize on full data set
5. Tune and scrape the barrel

https://karpathy.github.io/2019/04/25/recipe/

ML researchers work with fixed benchmark datasets, and spend all of their time searching over the knobs they do control: architecture & optimization. In applied ML, you're likely to spend most of your time on data collection and annotation -- where your investment will pay off.

https://twitter.com/fchollet/status/1353421758699687942

Challenges of Deep Learning

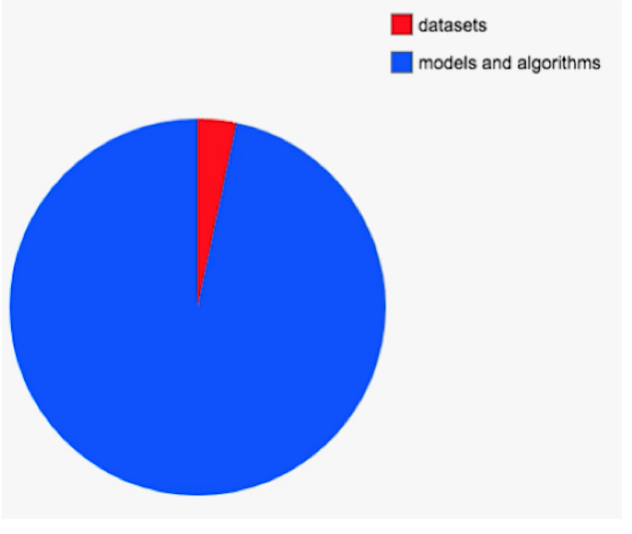
- deep learning is more an art than a science
- we can provide guidelines as to what is likely to work or not
- ultimately every problem is unique
- you will have to try and evaluate different strategies empirically
- currently there is no theory that will tell you in advance precisely what you should do to optimally solve a problem
- you must try and iterate

François Chollet in Deep Learning with Python (Manning Publications)

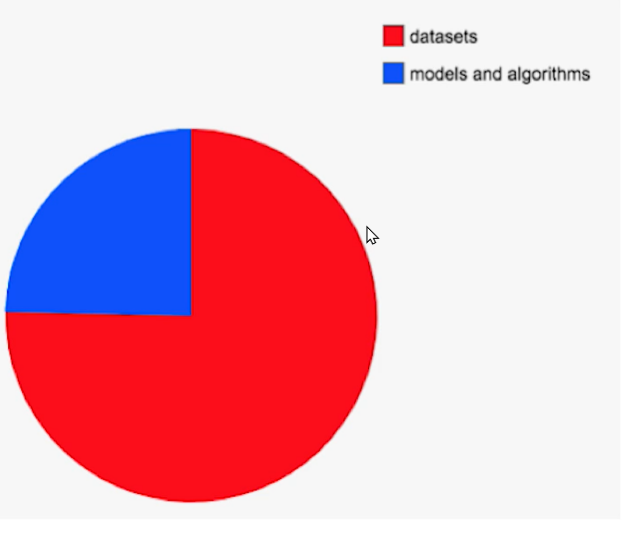
Data is King

Amount of lost sleep over...

PhD



Tesla



Andrej Karpathy - TRAIN AI 2018 - Building the Software 2.0 Stack

https://vimeo.com/272696002

Making samples cover reality is more important than the model architecture

We show a counter-intuitive result that adding more sources of variation to an imperfect estimator approaches better the ideal estimator at a 51× reduction in compute cost.

https://proceedings.mlsys.org/paper/2021/hash/cfecdb276f634854f3ef915e2e980c31-Abstract.html

https://twitter.com/rasbt/status/1425489844407517184

Why doesn't my model train properly?

- Bug in training code (typically silent)
- Model does train, but just not what you expect it to learn
- Pretrained model just does not match your data
- Architecture or capacity not a good fit
- You need to train for longer (sometimes even a converging loss curve is a false friend)
- Optimizer or learning rate is not appropriate

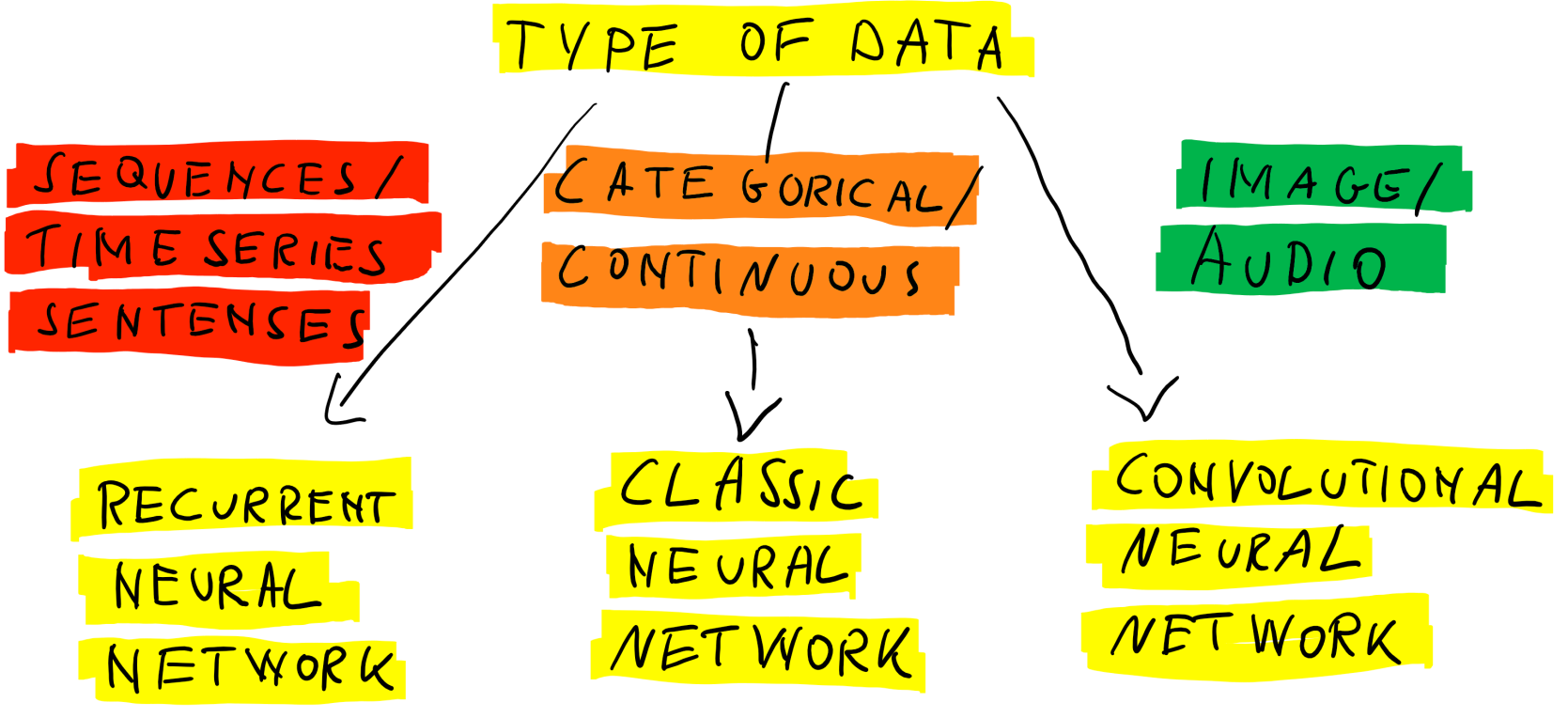
Actions to take when model doesn't train

don't give up before you know what the problem is, have trust in your abilities

- Make sure there is no bug
  - Code review
- Simplify code or learning task until it works
  - e.g. just one category
- Make sure prediction data/output match training data
- Make sure you can overfit on small data set

Schedule

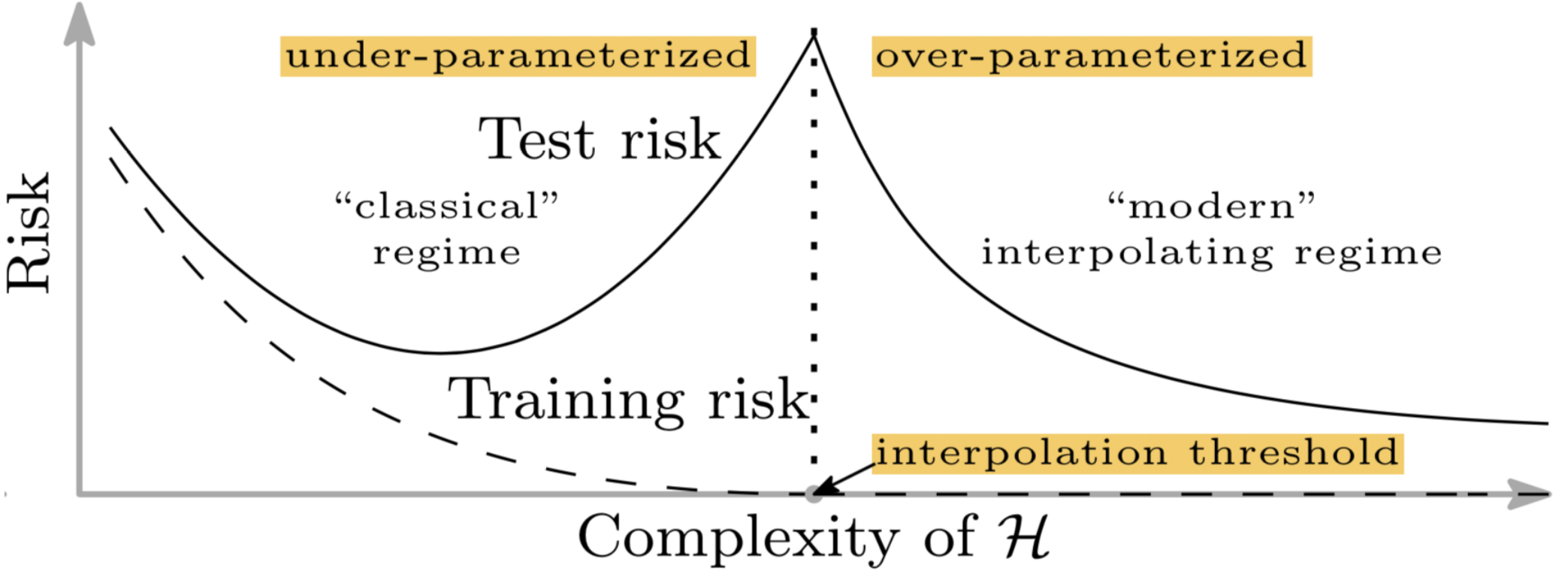
- Day 1: Introduction
    1. TensorFlow
    2. Basics of Supervised Learning
    3. Deep Learning best practices
- Day 2: Applications
    1. ***Tabular Data***
    2. Image Recognition
    3. Sequences
- Day 3: Advanced
    1. Unsupervised Deep Learning
    2. Deep Reinforcement Learning


TYPE OF DATA
SEQUENCES / TIME SERIES SENTENSES
CATEGORICAL / CONTINUOUS
IMAGE / AUDIO
RECURRENT NEURAL NETWORK
CLASSIC NEURAL NETWORK
CONVOLUTIONAL NEURAL NETWORK

Standard Models: Tabular Data

- Fully-Connected Layers, but how many and how big?
- And what acitivation?
- There exists a two-layer neural network with ReLU activations and 2n+d weights that can represent any function on a sample of size n in d dimensions. (https://arxiv.org/abs/1611.03530)
- interatctive experiments in the browser: https://playground.tensorflow.org/

Why have larger models then?



https://arxiv.org/abs/1812.11118

Composing Neural Networks

- input and output need to match training data and loss function
- some parts have clear function
  - mapping between dimensions, linear layer
  - Flatten, Reshape, BatchNorm, Dropout, Up-/Down-Sample
- the rest:
  - soviet approach to aviation: add enough power until things become airborne

Oriol Vinyals - The Deep Learning toolkit in 2020: https://youtu.be/-fdexQBpRas

Losses

need to match final layer and output of training data

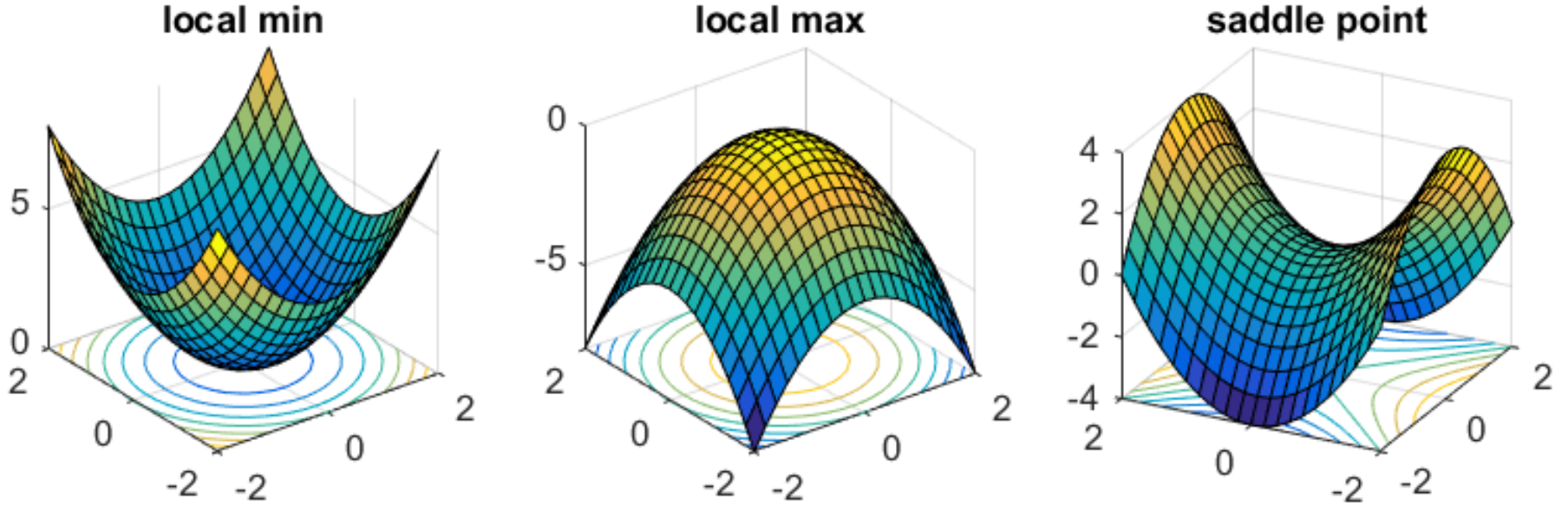
- regression: linear (or any other activation if the range of values is limited) activation and MSE
- two class: sigmoid activation and binary cross entropy
- multi class: softmax activation with categorical cross entropy
- multi-label: sigmoid activation and binary cross entropy, each output node would encode an hypothesis of its own

https://stats.stackexchange.com/questions/260505/should-i-use-a-categorical-cross-entropy-or-binary-cross-entropy-loss-for-binary

https://towardsdatascience.com/cross-entropy-for-classification-d98e7f974451

Local minima?

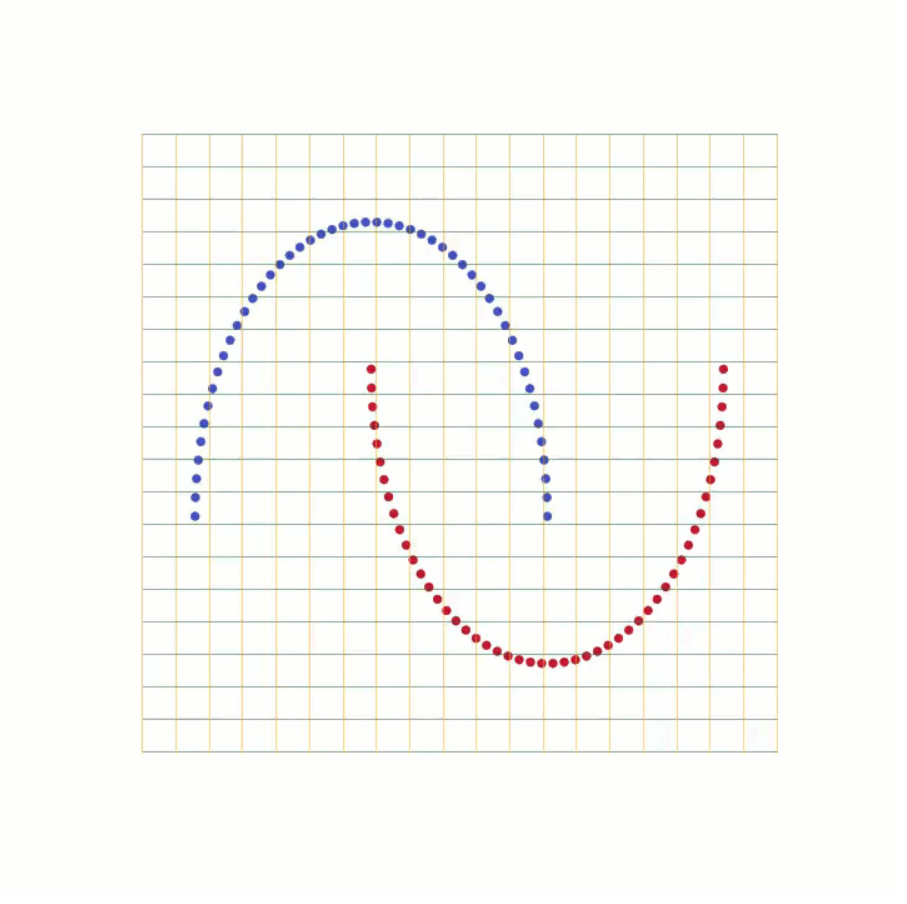
- local optimal points in the objective landscape almost always lay in saddle-points or plateaus rather than valleys
- there is always a subset of dimensions containing paths to leave local optima and keep on exploring



https://lilianweng.github.io/lil-log/2019/03/14/are-deep-neural-networks-dramatically-overfitted.html
https://arxiv.org/abs/1406.2572
https://www.offconvex.org/2016/03/22/saddlepoints/

Intuition for the learning process

network stretches and folds the paper until it can find a line to separate red from blue

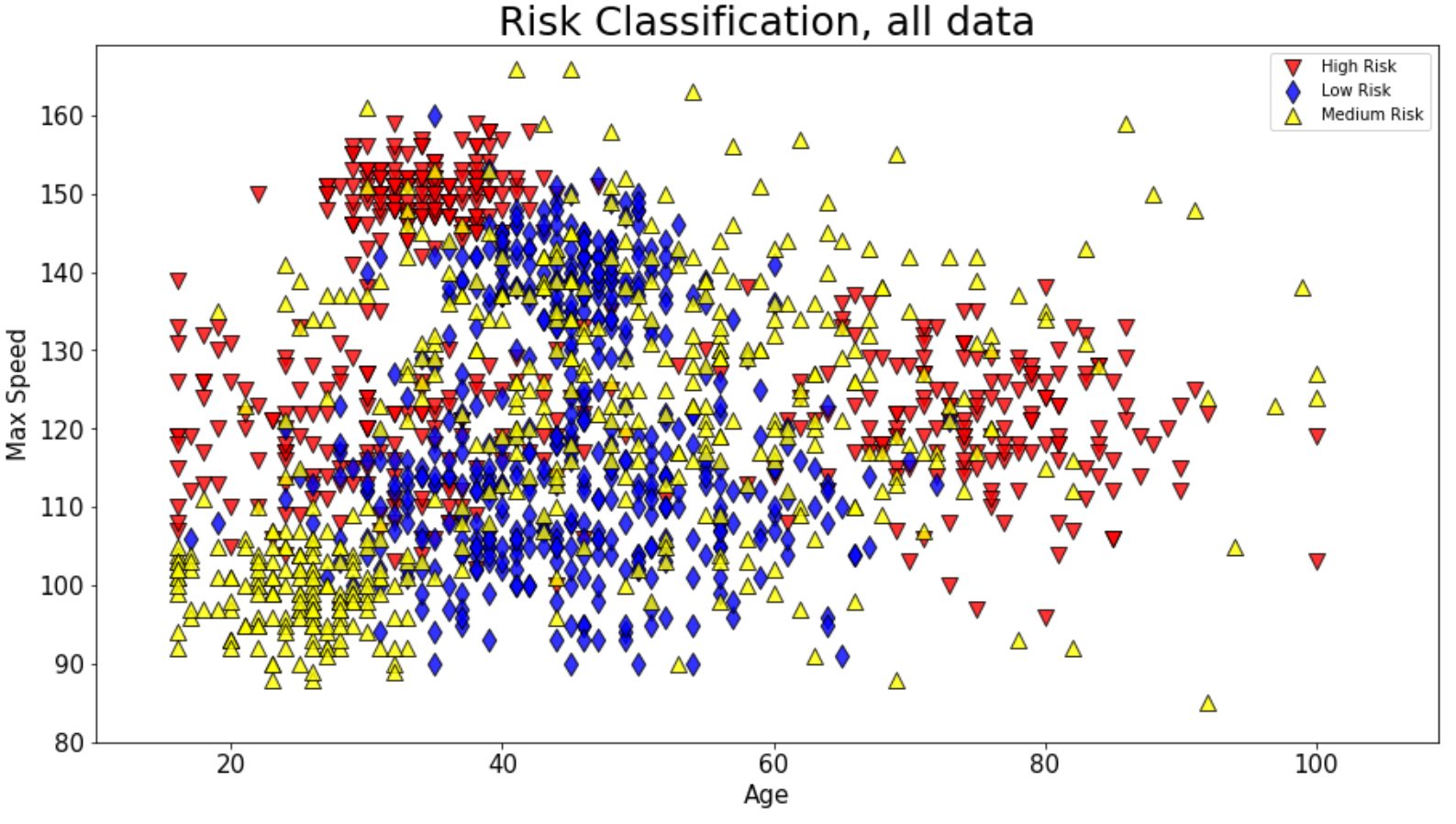


https://twitter.com/random_forests/status/1084618439602298881
http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/
https://cs.stanford.edu/people/karpathy/convnetjs/
https://brohrer.github.io/what_nns_learn.html

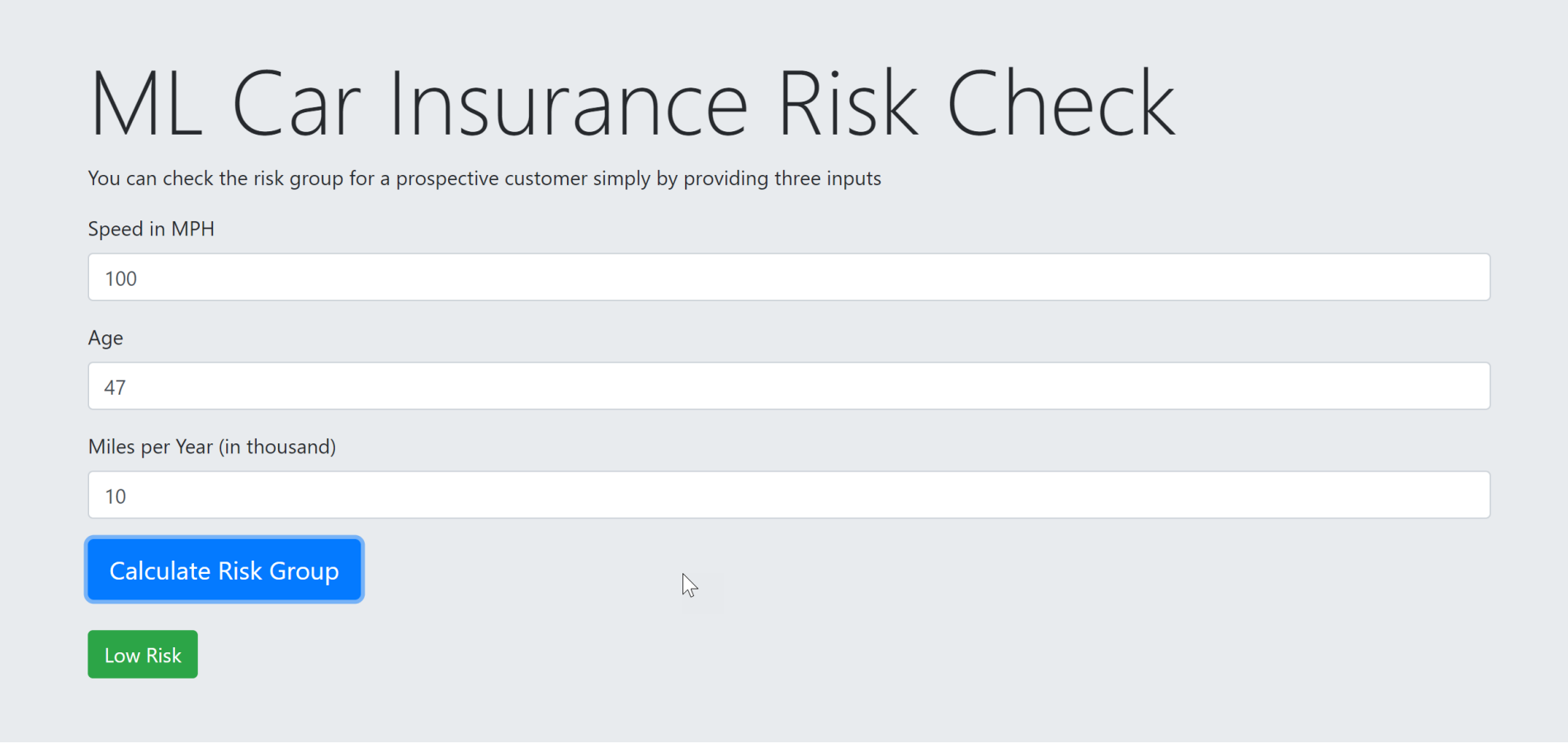
Scenario: Predicting Risk

- We are CTO of a highly innovative Car Insurance Company
- Different from other insurance companies we determine the rate by the actual number of accidents per customer
- ***Objective: how many accidents will prospective customers have?***

Classification based on known data

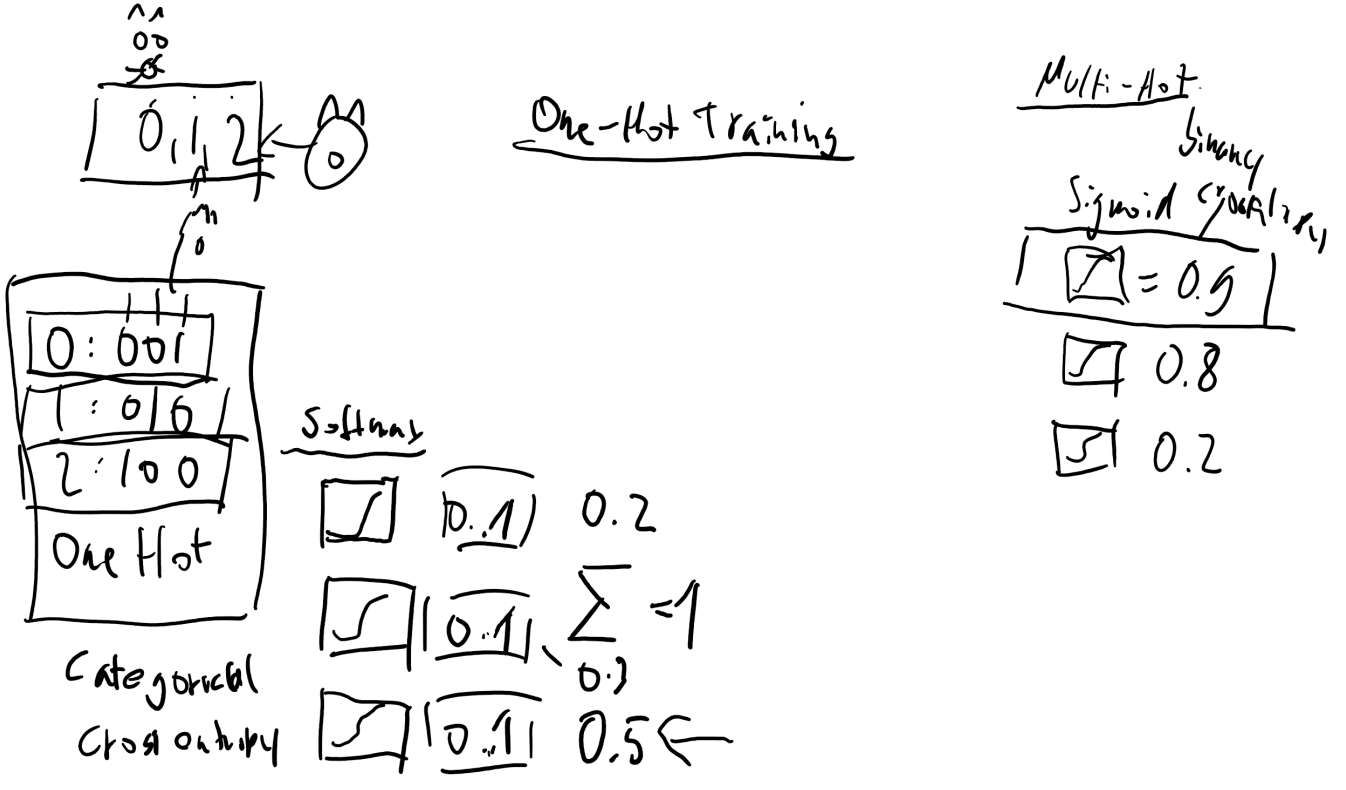


Car Insurance: Predicting Crash Risk for prospective customers



https://embarced.github.io/ml-concepts/html/calculator.html

One-Hot-Encoding



Disclaimer - just a sketch done during the workshop

Network for this practical example

https://colab.research.google.com/github/embarced/notebooks/blob/master/deep/tabular.ipynb?hl=en

Solutions using all kinds of approaches, including classic ML as a reference:
https://colab.research.google.com/github/djcordhose/ml-workshop/blob/master/notebooks/classic/strategies.ipynb?hl=en

make a copy of the notebook to your Google Drive

Is 80% accuracy good?

you need to understand the business problem to decide

- How accurate does my model really need to be? What kind of false positive rate is acceptable?
- What data can I use? If you're predicting flight days tomorrow, you can look at weather data, but if someone is buying a flight a month from now then you'll have no clue.

https://jvns.ca/blog/2014/06/19/machine-learning-isnt-kaggle-competitions/
https://towardsdatascience.com/measuring-model-goodness-part-1-a24ed4d62f71

Words from François Chollet

François Chollet 
@fchollet

If your classifier is "99% accurate", either you're using the wrong metric (a metric this high is not informative), or you have an overfitting or leakage problem.

Metrics are feedback points on the way towards better models. Not trophies to show off. They should be actionable.

12:34 AM · Sep 25, 2019 · Twitter for Android

https://twitter.com/fchollet/status/1176625911036145666

Schedule

- Day 1: Introduction
    1. TensorFlow
    2. Basics of Supervised Learning
    3. Deep Learning best practices
- Day 2: Applications
    1. Tabular Data
    2. ***Image Recognition***
    3. Sequences
- Day 3: Advanced
    1. Unsupervised Deep Learning
    2. Deep Reinforcement Learning

Image Recognition is hard - even seemingly simple tasks are so far unsolved

recognizing digits and numbers in natural scene images

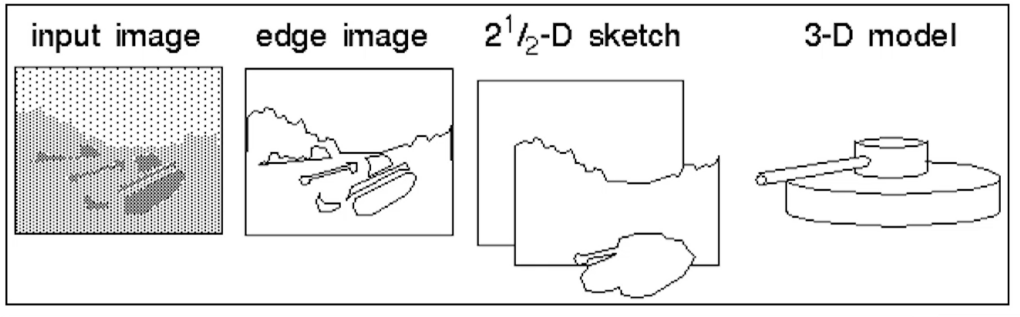
http://ufldl.stanford.edu/housenumbers/

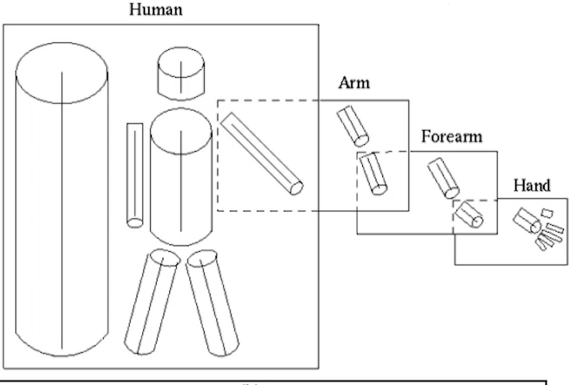
Image Recognition is everywhere

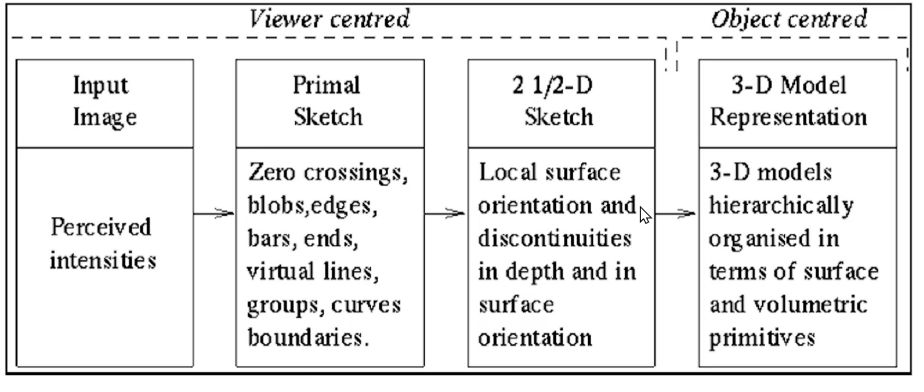
Kyunghyun Cho (@kchonyc) twitterte um 3:10 AM on Fr., Aug. 20, 2021: i was looking for a low room divider (which i've failed to find) on amazon and have accidentally learned they use a conv net for product retrieval.
https://t.co/qGOA4ycL1h
(https://twitter.com/kchonyc/status/1428524822586437639?s=03)

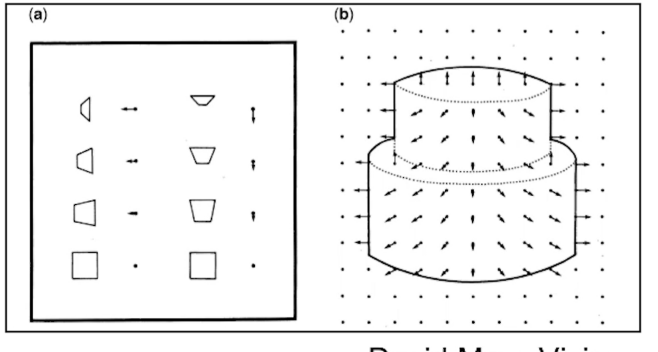
History of Image Recognition: 1990s

Visual Recognition: 1980 ~ 1990







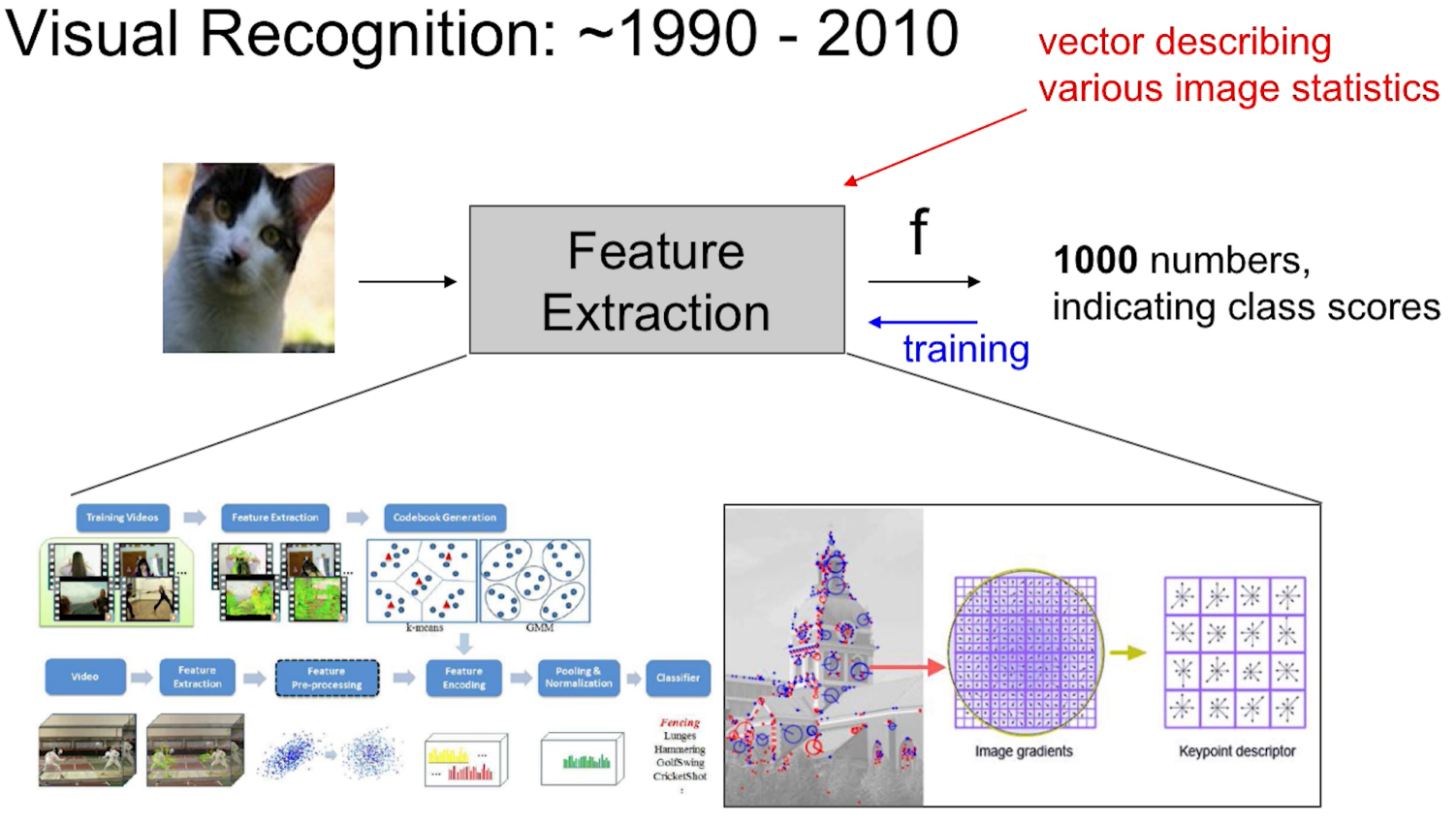


David Marr, Vision

Andrej Karpathy - TRAIN AI 2018 - Building the Software 2.0 Stack

https://vimeo.com/272696002

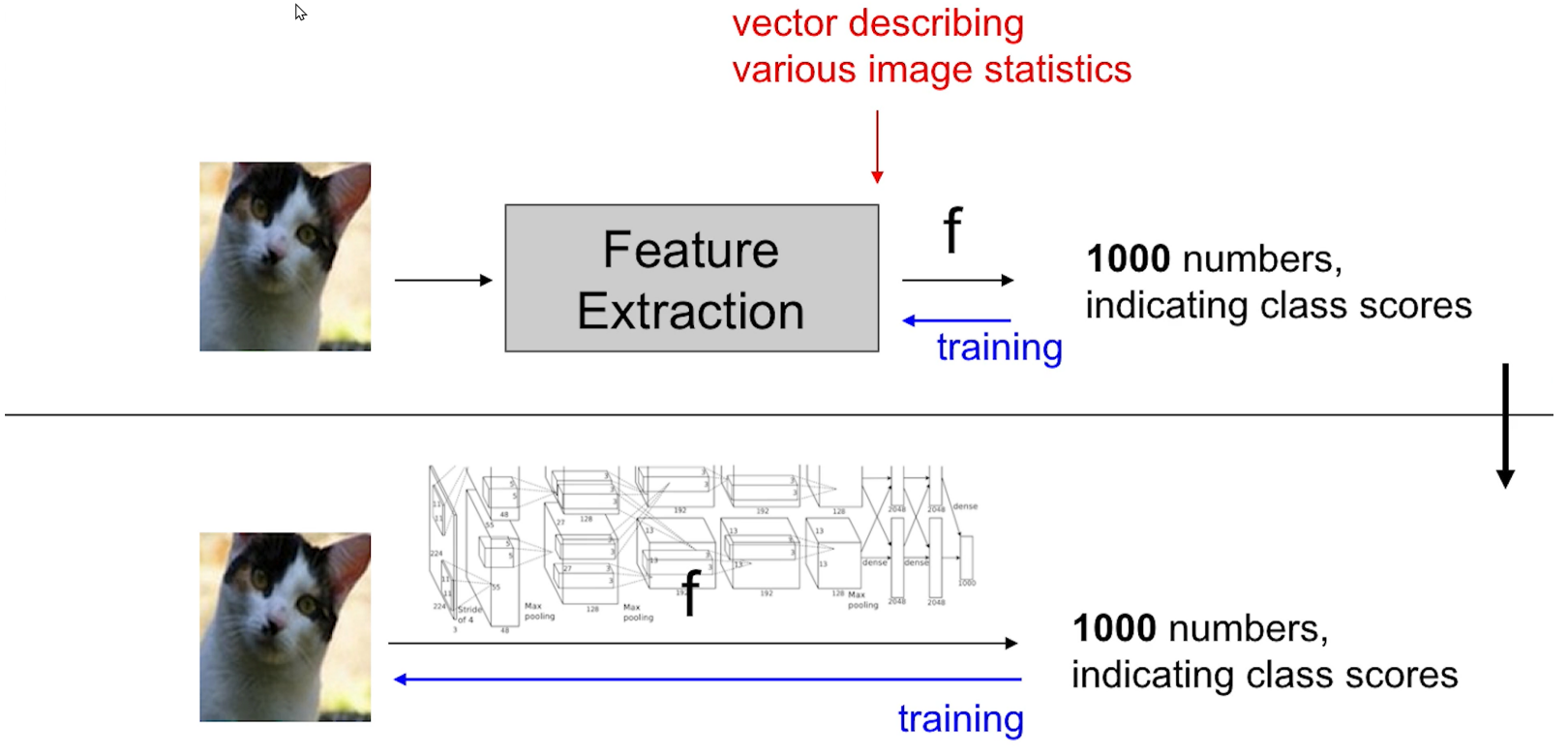
History of Image Recognition: 2000s



Andrej Karpathy - TRAIN AI 2018 - Building the Software 2.0 Stack

https://vimeo.com/272696002

History of Image Recognition: Deep Learning



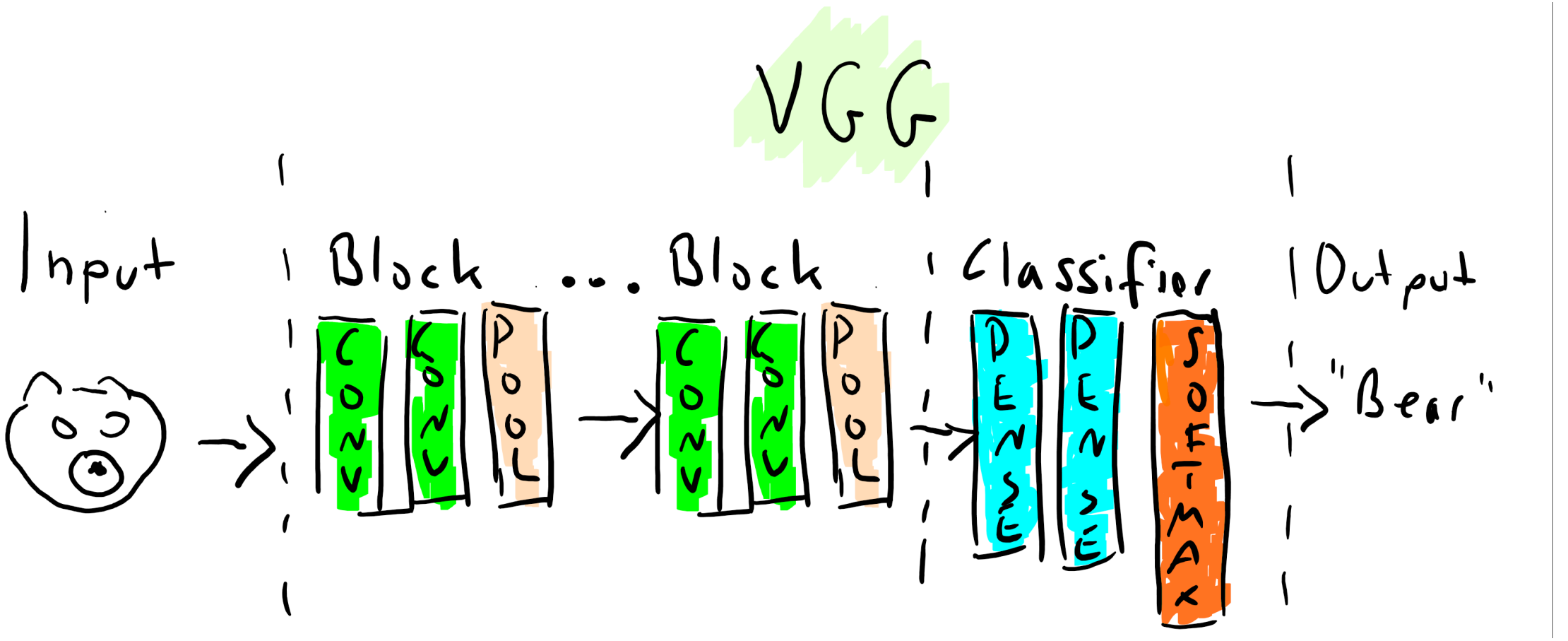
Andrej Karpathy - TRAIN AI 2018 - Building the Software 2.0 Stack

https://vimeo.com/272696002

Images Recognition using Neural Networks

- Feeding all pixels into Dense Layers will work, but
- Images are 2-d (or n-d) and generalize much better with a network that aknowledges this
- CNNs add this as a geometric prior (https://towardsdatascience.com/geometric-foundations-of-deep-learning-94cdd45b451d)
- use well known feature detectors (convolutions)
- filter parameters are trainable
- Convolutional networks will learn feature extraction before passing few features to Dense Layer Classifiers (https://towardsdatascience.com/simple-introduction-to-convolutional-neural-networks-cdf8d3077bac)

VGG: a basic network architecture



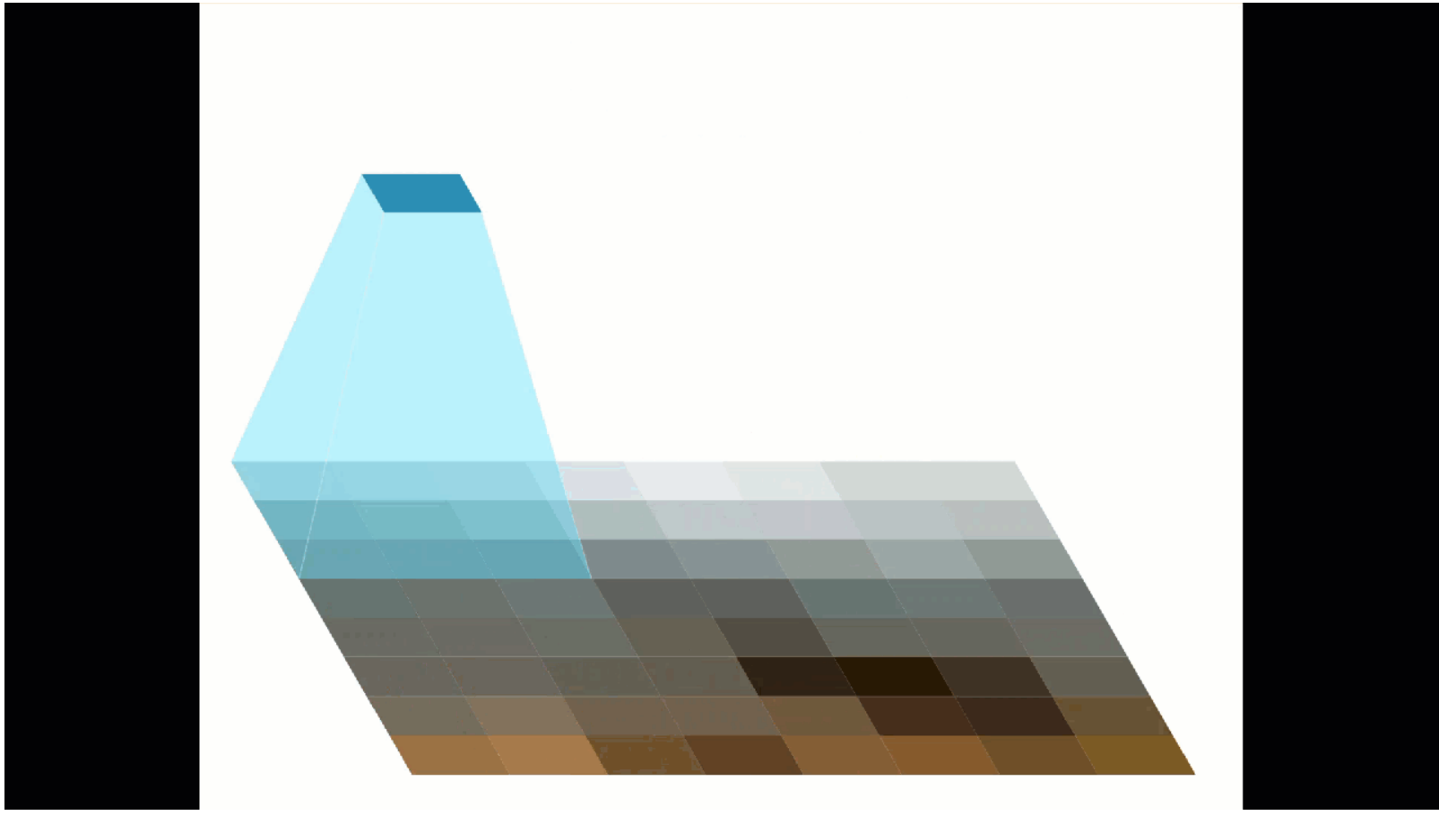
VGG starts with a number of convolutional blocks for feature extraction and ends with a fully connected classifier

VGG contains a number of specialized neural network layers

Most important: Convolutional Layers - filtering for features

- applying a number of filter kernels
- result of a filter kernel is like a manual feature extraction
- by stacking filter operations we hope to extract meaningful features
- parameters of the kernerls are initialized randomly
- parameters are learned during training

How do they work?



https://sigmoidprime.com/post/the-inner-workings-of-convolutional-nets/

https://twitter.com/wster/status/1079741301418049537

Interactive exploration

- https://github.com/okdalto/VisualizeMNIST
- https://transcranial.github.io/keras-js/#/mnist-cnn
- https://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html
- http://setosa.io/ev/image-kernels/

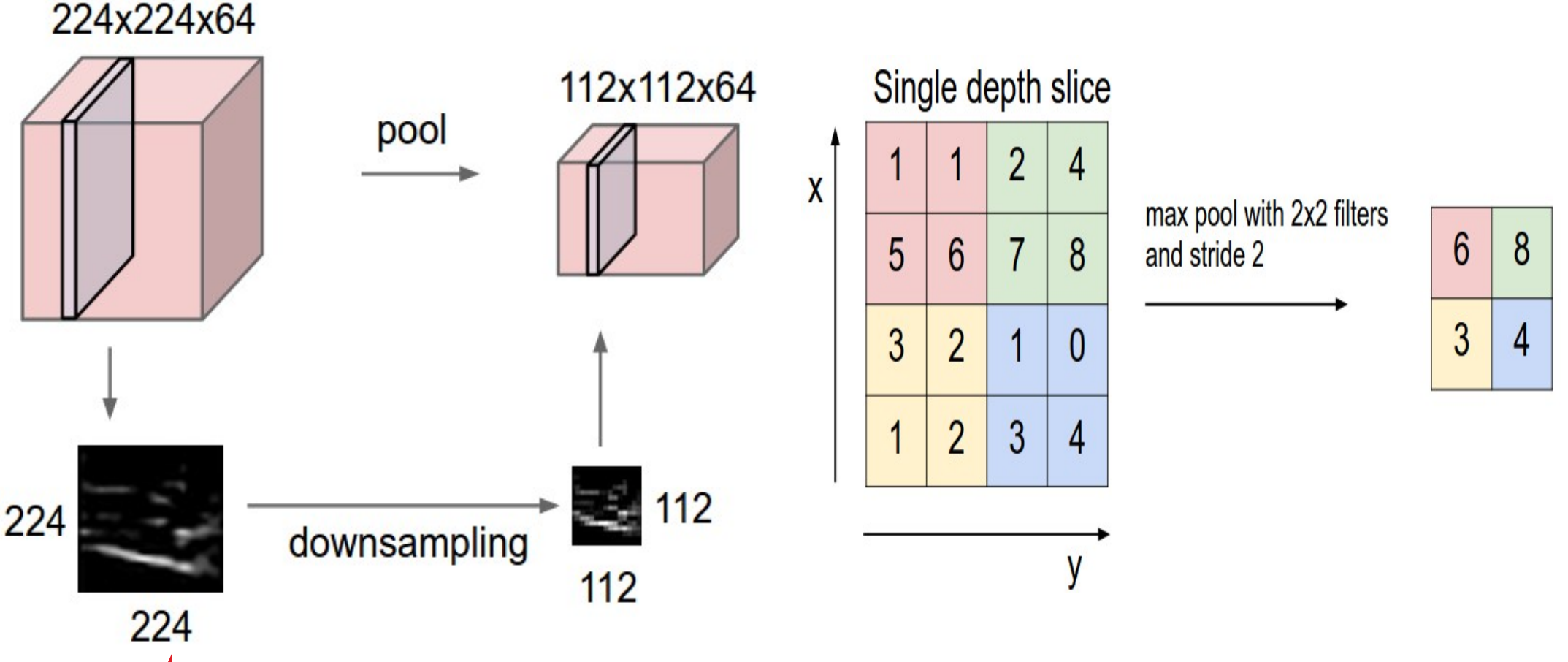
Keras Implementation

```
model.add(Conv2D(filters=32))
```

https://www.tensorflow.org/api_docs/python/tf/keras/layers/Conv2D

Pooling - reducing size

- reduces computational load
- prevents overfitting by removing details



Keras Implementation

Pooling

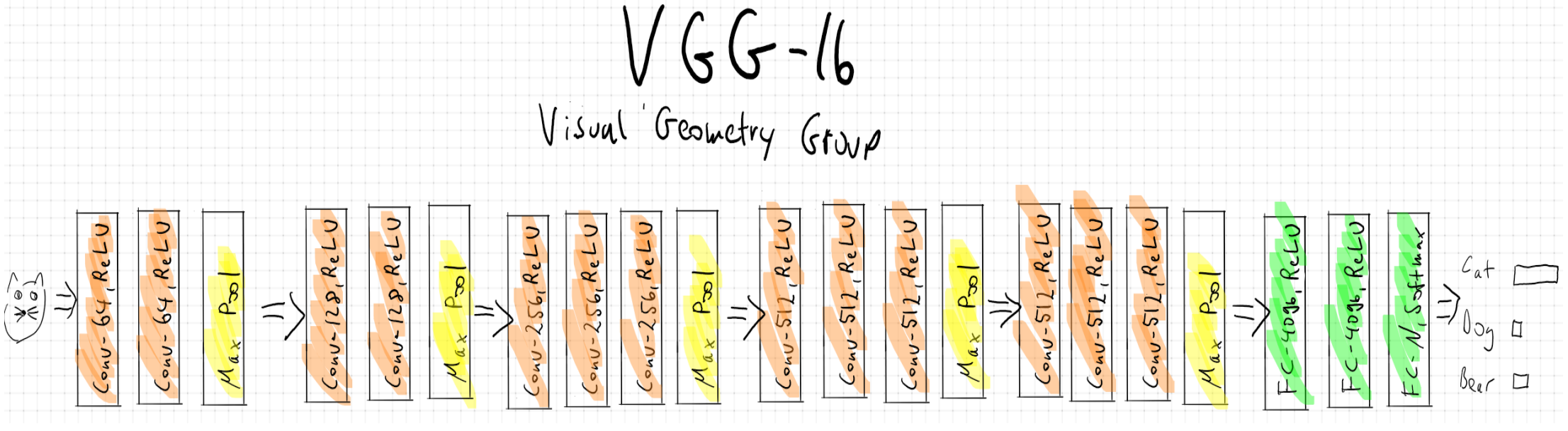
```
model.add(MaxPooling2D())
```

Flatten 2d to make it accessible to Dense layers

```
model.add(Flatten())
```

https://www.tensorflow.org/api_docs/python/tf/keras/layers/MaxPool2D

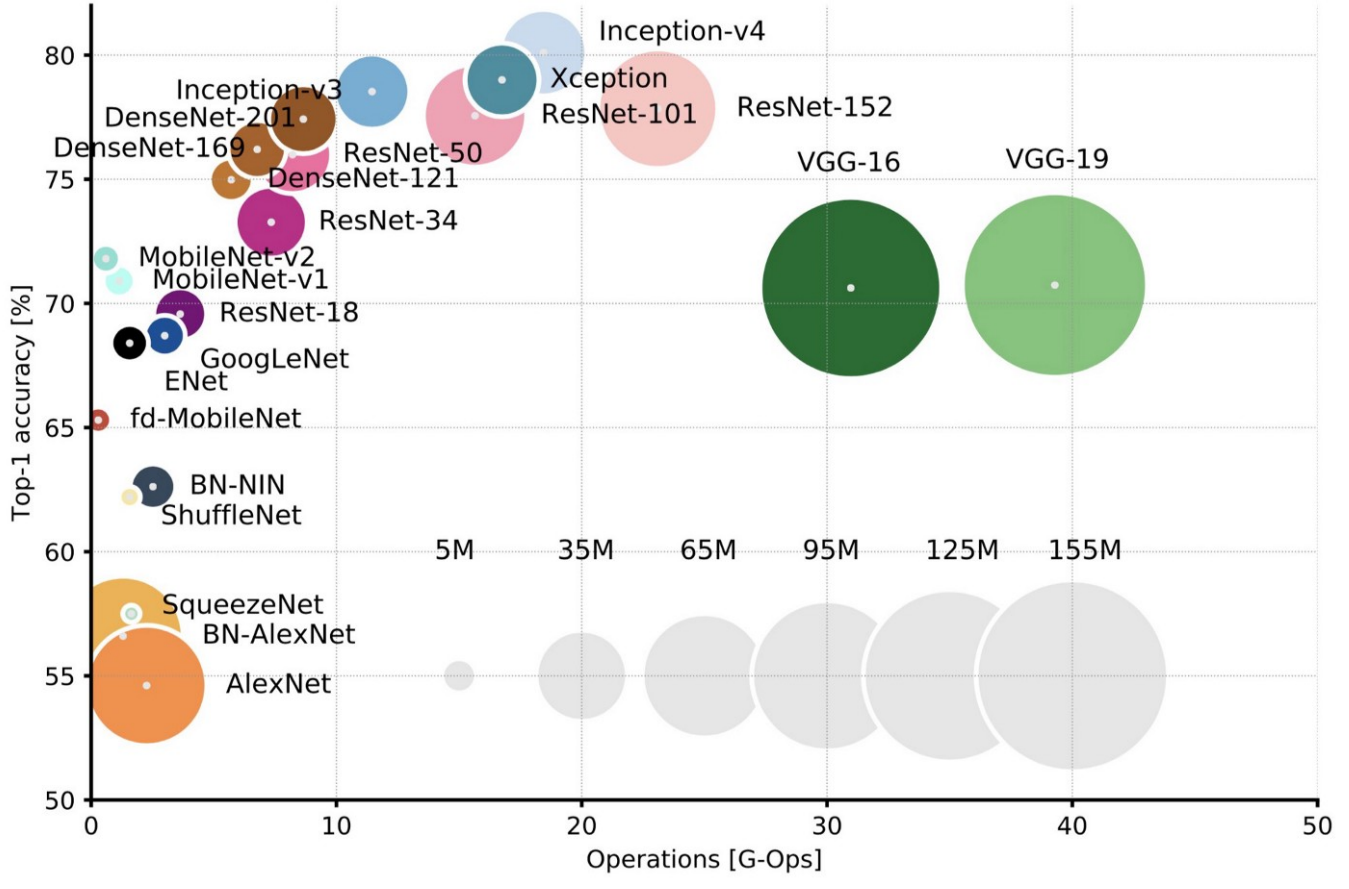
Complete original VGG-16



Costly, but straight forward, often simplified

https://arxiv.org/abs/1409.1556

Comparing Standard CNN Architectures



https://towardsdatascience.com/neural-network-architectures-156e5bad51ba

Networks for Images

A much more realistic example. How do you approach a challenge like this and how to you make it generalize to the real world.

https://colab.research.google.com/github/embarced/notebooks/blob/master/deep/intro_cnn.ipynb?hl=en

Outdated version unsuccessfully trying transfer learning

https://colab.research.google.com/github/DJCordhose/ml-workshop/blob/master/notebooks/tf2/images-intro.ipynb?hl=en

(just for the idea how to do that)

make a copy of the notebook to your Google Drive

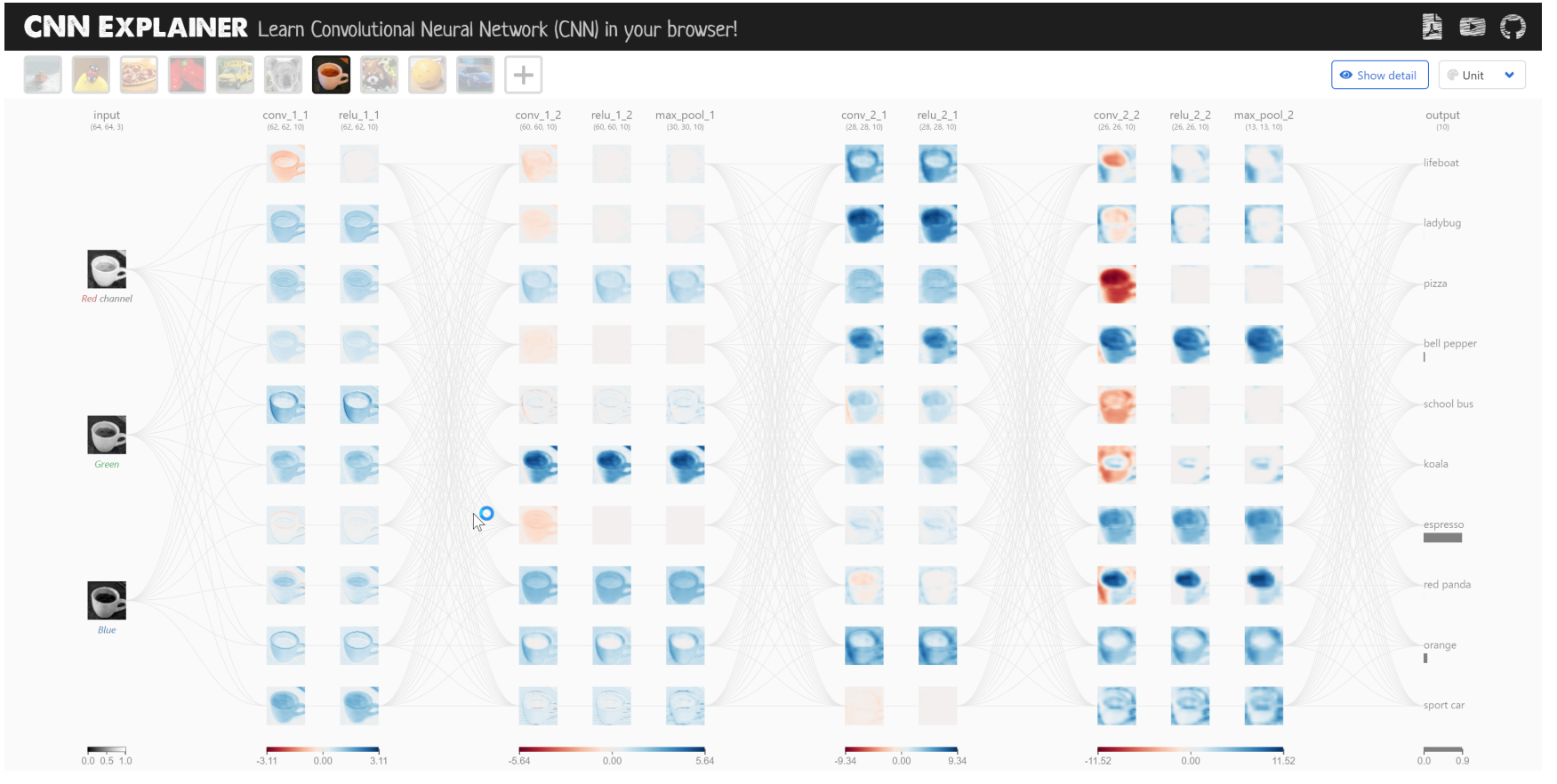
Best Practices - Image Data

- try a couple of standard architectures
  - ResNet 50 can be a good starting point
  - transfer learning often does not work
    - might only work if your image data is similar to ImageNet
  - use large batch size and SGD as a means to force regularization
    - https://twitter.com/DJCordhose/status/1376140631555371008
- if standard model seems too complex or does not generalize try custom model
  - VGG subset with dropout and Batch Normalization might be a good candidate
- Augmentation with small sample size might help

https://keras.io/api/applications/
https://keras.io/api/applications/resnet/#resnet50v2-function
https://www.tensorflow.org/api_docs/python/tf/keras/layers/experimental/preprocessing

Understanding CNNs

Network to process images



https://poloclub.github.io/cnn-explainer/

Schedule

- Day 1: Introduction
    1. TensorFlow
    2. Basics of Supervised Learning
    3. Deep Learning best practices
- Day 2: Applications
    1. Tabular Data
    2. Image Recognition
    3. ***Sequences***
- Day 3: Advanced
    1. Unsupervised Deep Learning
    2. Deep Reinforcement Learning

Networks for Sequences, Time Series and Texts

How does this sequence continue?

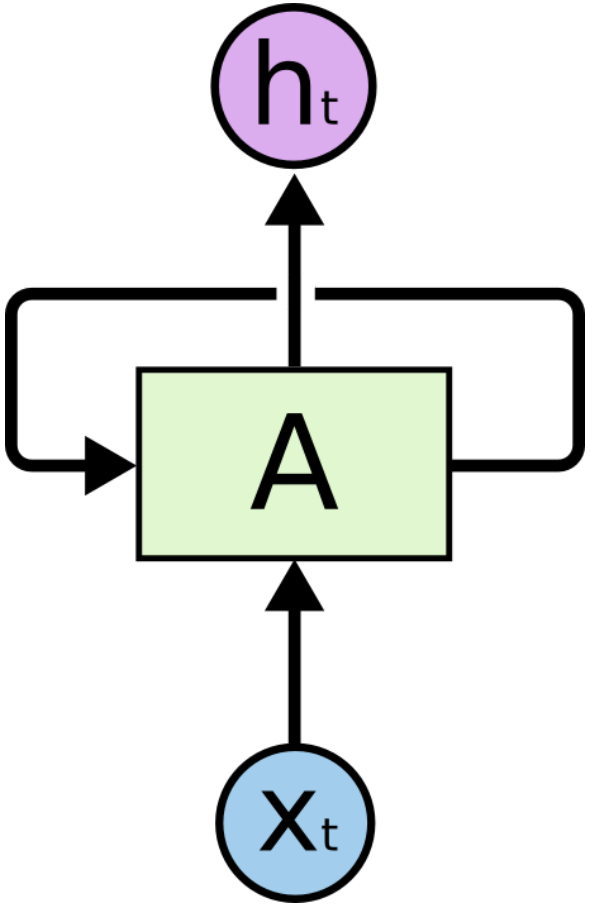
```
[10, 20, 30, 40, 50, 60, 70, 80, 90]
```

Question: How do we train a network to predict the next number?

Challenge: Dense Networks have no notion and memory of previous events

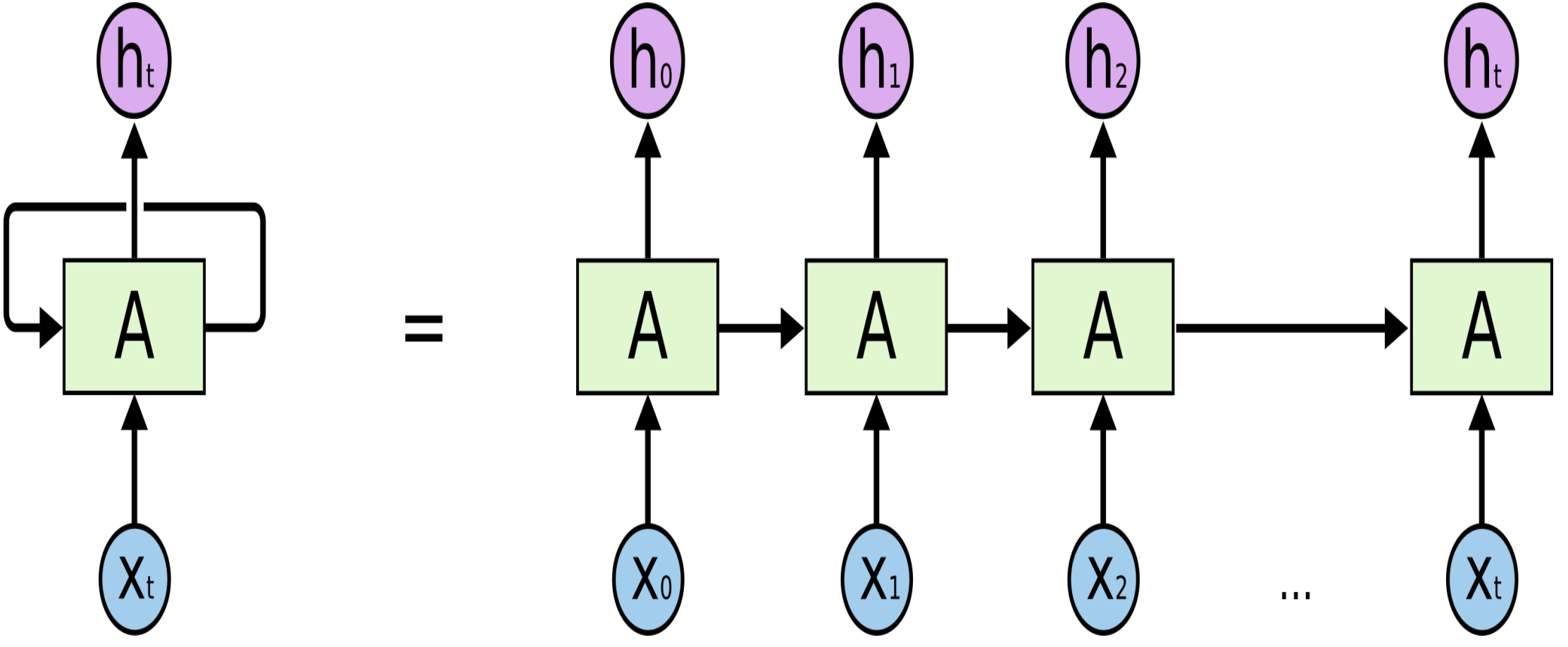
- lack capability to deal with sequential data, which is required to predict time series or "understand" text
- they could still be used to spit out a number of values based on a number of inputs
- the information that things follow each other would be lost, though
- this would take the prior of things being a sequence
- we might need a similar trick as we applied with CNNs

Solution: Recurrent Neural Networks - Networks with Loops



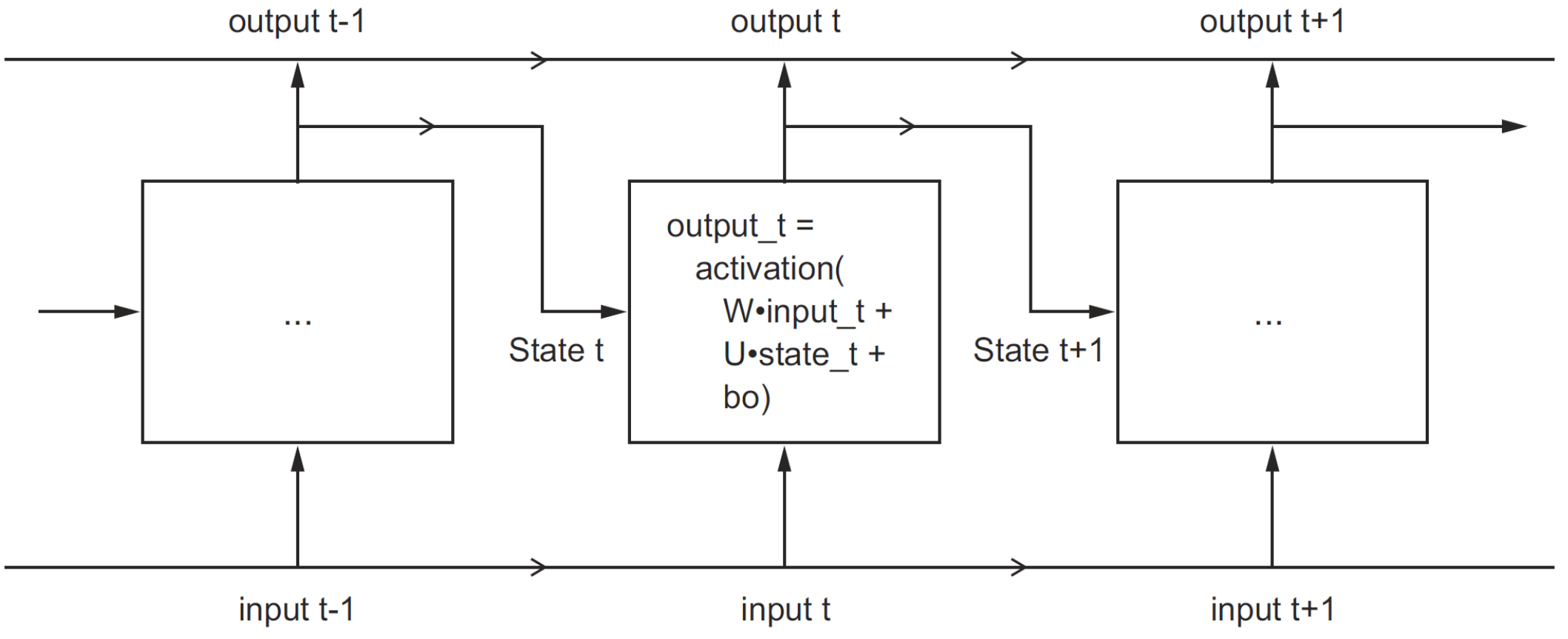
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Unrolling the loop



http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Simple RNN



$$output_t = activation(W input_t + U output_{t-1} + b)$$

Deep Learning with Python, Chapter 6, François Chollet, Manning

Question

Even having a network that can deal with time sequences, how do you train it using our data?

First Step: Slice and Splice data to have a training set

```
[10, 20, 30, 40, 50, 60, 70, 80, 90]
```

Training Data, sliced to only use 3 past events

```
[10, 20, 30] => 40
[20, 30, 40] => 50
[30, 40, 50] => 60
[40, 50, 60] => 70
[50, 60, 70] => 80
[60, 70, 80] => 90
```

Sequence Forecasting with RNNs

The Model

```
model.add(SimpleRNN(units=50, activation='relu', name="RNN_Input"))
model.add(Dense(units=1, name="Linear_Output"))
model.compile(optimizer='adam', loss='mse')
model.fit(X, y)
```

Predictions

```
[10, 20, 30] => 39.767338
[70, 80, 90] => 100.001076
[100, 110, 120] => 130.40291
[200, 210, 220] => 231.74236
[200, 300, 400] => 489.32404
```

Hands-On: Sequence Prediction

https://colab.research.google.com/github/embarced/notebooks/blob/master/deep/intro_rnn_basics.ipynb?hl=en

make a copy of the notebook to your Google Drive

Main issues with simple RNNs: Vanishing or exploding gradient problem

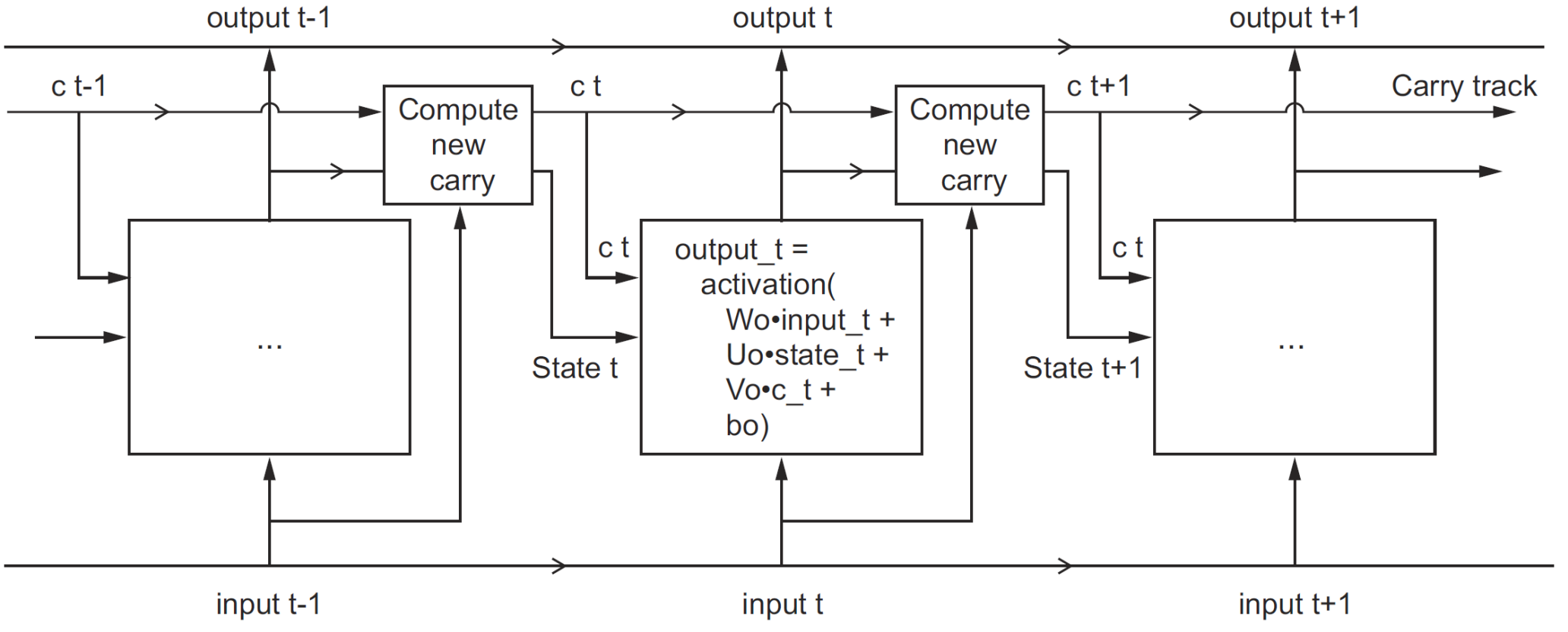
Effectively long term memory does not work:

- there is no training, because you are either on a plateau or in front of a wall
- RNNs experiences difficulty in memorizing values from far away in the sequence
- Predictions based on most recent values only

LSTM (Long short-term memory) / GRU (Gated Recurrent Unit)

allow past information to be selectively reinjected at a later time, thus fighting the vanishing-gradient problem

LSTM



Deep Learning with Python, Chapter 6.2.2, François Chollet, Manning

Advanced Keras RNN Layers

```
# LSTM and GRU both allow for long term memory

model.add(LSTM(units=rnn_units))
# less expensive, but often as good
model.add(GRU(units=rnn_units))
```

```
# Passes all outputs of all timesteps (not only the last one) to the next layer
# also allows for stacking RNN layers

model.add(GRU(units=rnn_units, return_sequences=True))
```

```
# Adds Dropout inside feedback loop

model.add(GRU(units=rnn_units, return_sequences=True, recurrent_dropout=0.2))
```

https://www.tensorflow.org/api_docs/python/tf/keras/layers

A practical application

Time Series Forecasting

https://colab.research.google.com/github/DJCordhose/ml-workshop/blob/master/notebooks/tf-intro/2020-01-time-series.ipynb

Hands-On

How does the number of past steps used for prediction change the outcome?

Make experiments with a number of training steps reduced to 5 or 10.

Before you get tempted, read this

François Chollet ✓
@fchollet

There are lots of blog posts out there claiming to teach you how to use deep learning (typically RNNs) to predict stock prices, FX rates, BTC price, from past price data. Is it actually possible?

Well, mostly not. A thread.

7:17 PM · Sep 27, 2019 · Twitter Web App

https://twitter.com/fchollet/status/1177633367472259072

Open Question

How would you map

- any number of input time steps
- with any number of features
- to any number of output time steps
- with any number of features

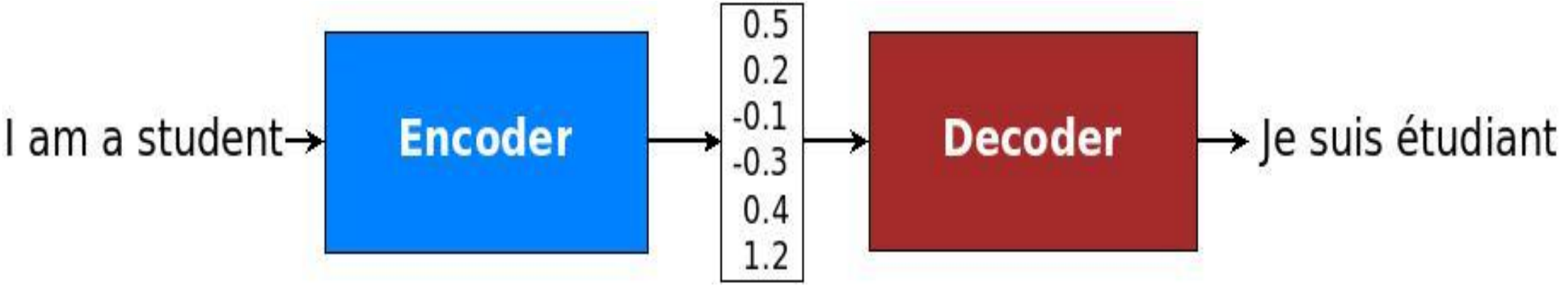
Surprising Solution: RNN Encoder-Decoder

- RNN Encoder-Decoder consists of two recurrent neural networks (RNN)
- they act as an encoder and a decoder pair
- encoder maps a variable-length source sequence to ***a fixed-length vector***, the ***latent representation***
- all the temporal information as lost in the latent representation
- decoder maps the vector representation back to a variable-length target sequence

https://arxiv.org/abs/1406.1078
https://machinelearningmastery.com/encoder-decoder-long-short-term-memory-networks/

Example Application: Sequence to Sequence translations

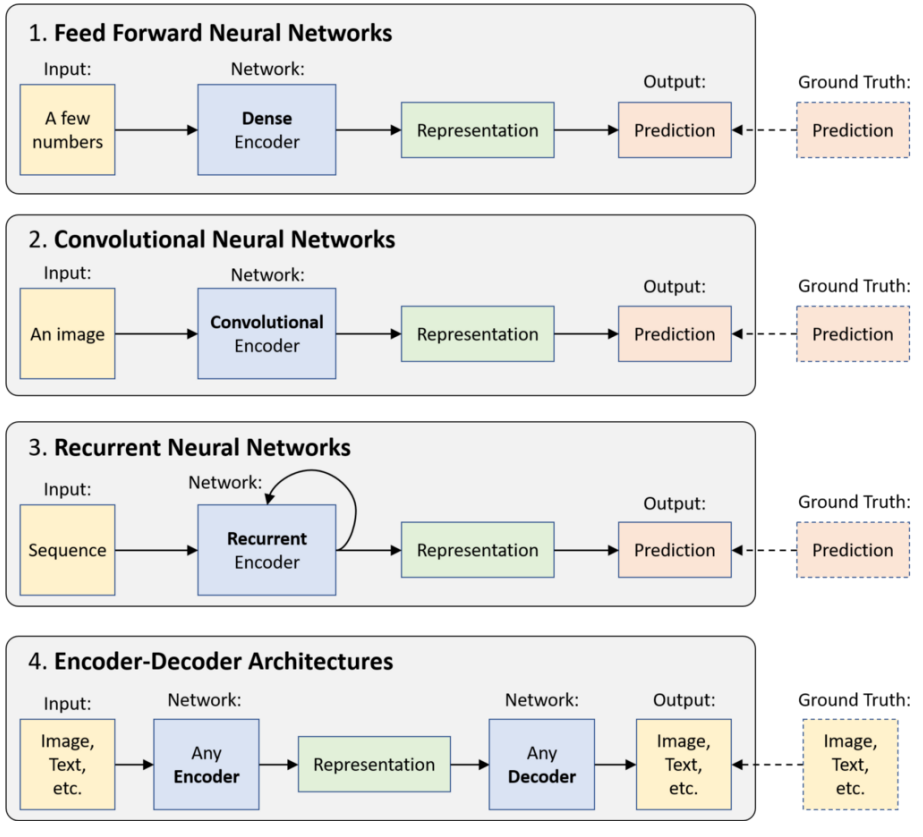
- could be interpreted as sequential embedding
- fixed-length vector between decoder and encoder is the latent space



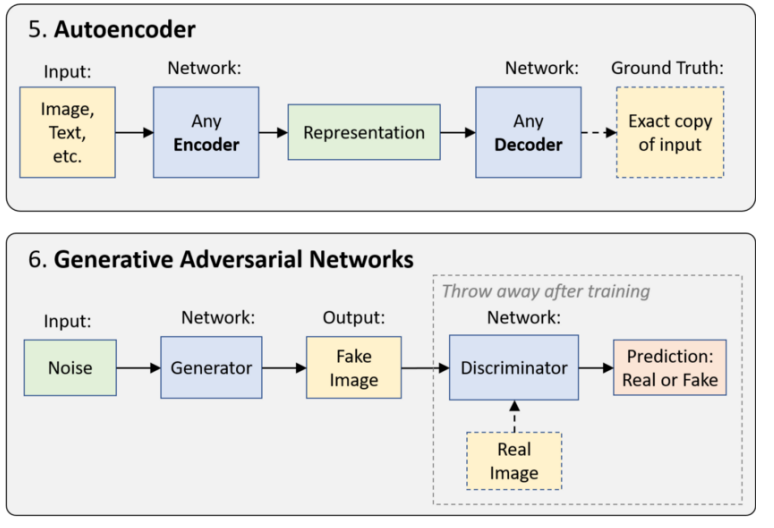
https://github.com/tensorflow/nmt

You can even describe all networks as a combination of encoder / decoder

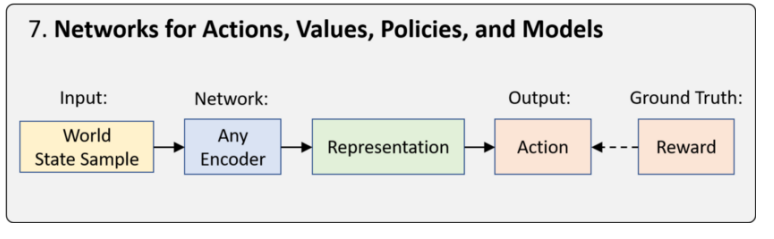
Supervised Learning



Unsupervised Learning



Reinforcement Learning



https://medium.com/tensorflow/mit-deep-learning-basics-introduction-and-overview-with-tensorflow-355bcd26baf0

Encoder / Decoder Architectures

This is mostly about the sequence of layers

```
# Encoder (nothing new)

model.add(Input(shape=(n_steps_in, n_features)))
model.add(LSTM(units=ENCODER_SIZE, activation='relu'))
```

```
# Latent Space (nothing new)

model.add(Dense(units=ENCODING_DIM, activation='relu'))
model.add(Dense(units=ENCODING_DIM, activation='relu'))
```

```
# Decoder

# rolls out the latent space for each of the desired output steps
model.add(RepeatVector(n_steps_out))
model.add(LSTM(units=DECODER_SIZE, activation='relu', return_sequences=True))
model.add(Dense(units=1)) # just to combine
```

https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/keras/layers

Using an encoder/decoder architecture

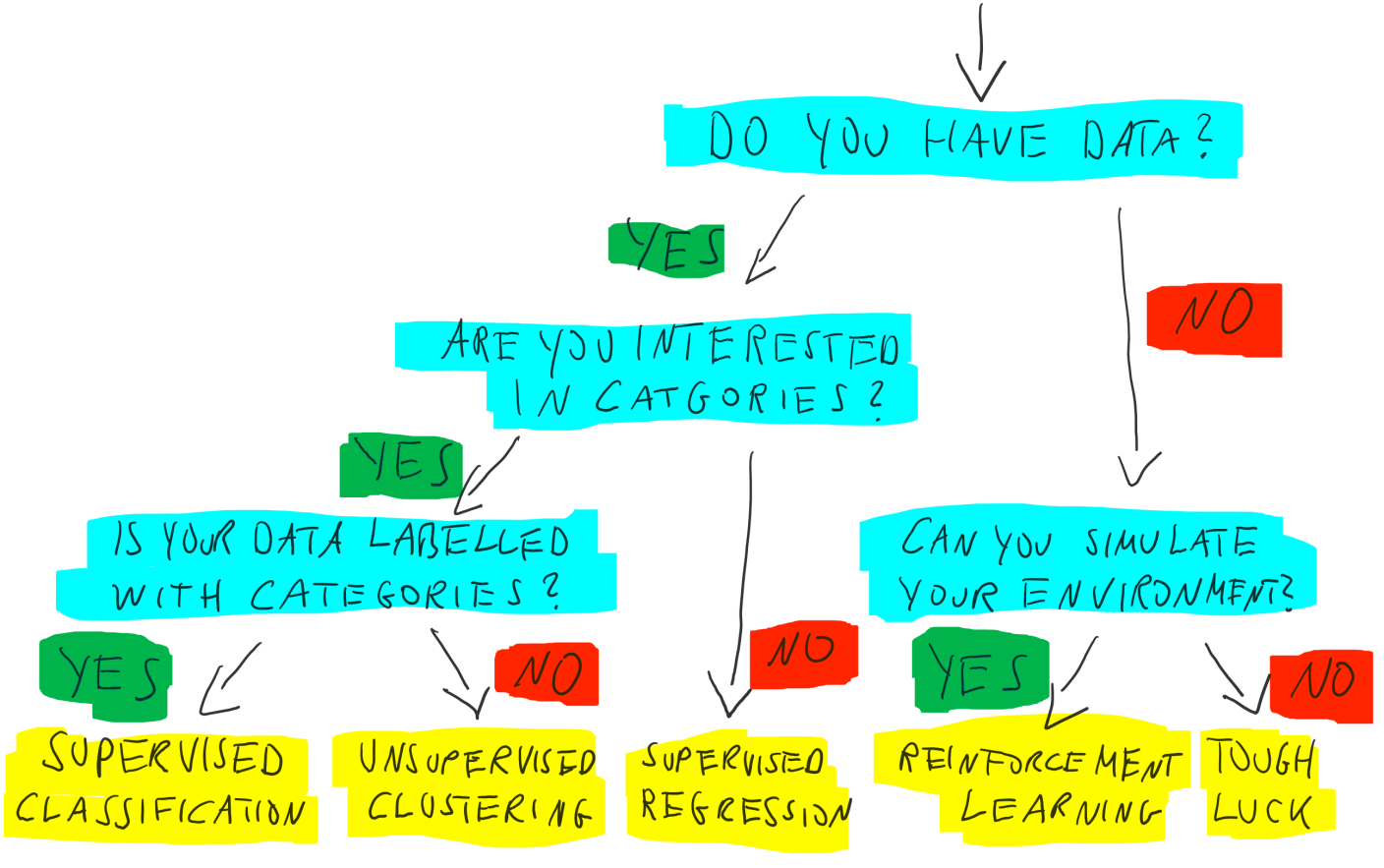
Time Series Forecasting

https://colab.research.google.com/github/DJCordhose/ml-workshop/blob/master/notebooks/tf2/time-series-encoder-decoder.ipynb

Schedule

- Day 1: Introduction
    1. TensorFlow
    2. Basics of Supervised Learning
    3. Deep Learning best practices
- Day 2: Applications
    1. Tabular Data
    2. Image Recognition
    3. Sequences
- Day 3: Advanced
    1. ***Unsupervised Deep Learning***
    2. Deep Reinforcement Learning

There is more than Supervised Learning



Machine Learning has data as its core

- Supervised Machine Learning: Learn to imitate relation between input and output
    - Time Series data already is special in that respect as we are only constructing pairs of input and output
    - sometimes called self-supervised training
- Unsupervised Machine Learning: Objective needs to be described in more detail as we are not only having a simple immitation
- Reinforcement Learning: objective even more abstract, given as a reward in a given environment

Working with Text

A special case of sequence

… however …

Attention is all you need

- earlier sequence transduction models are based on complex recurrent neural networks that include an encoder and a decoder
- best performing models also connect the encoder and decoder through an attention mechanism
- Transformer models are based solely on attention mechanisms, dispensing with recurrence and convolutions
- more parallelizable

https://arxiv.org/pdf/1706.03762.pdf

Transformer Core ideas

1. self-attention for encoding long range dependencies
2. self-supervision for leverating large unlabeled datasets (aka unsupervised pre-training)
3. additional supervised training for downstream tasks, e.g.
    - translation (lang1 & lang2 pairs)
    - question answering (Q&A pairs)
    - sentiment analysis (text & mood pairs)
    - etc.

https://www.youtube.com/watch?v=iFhYwEi03Ew

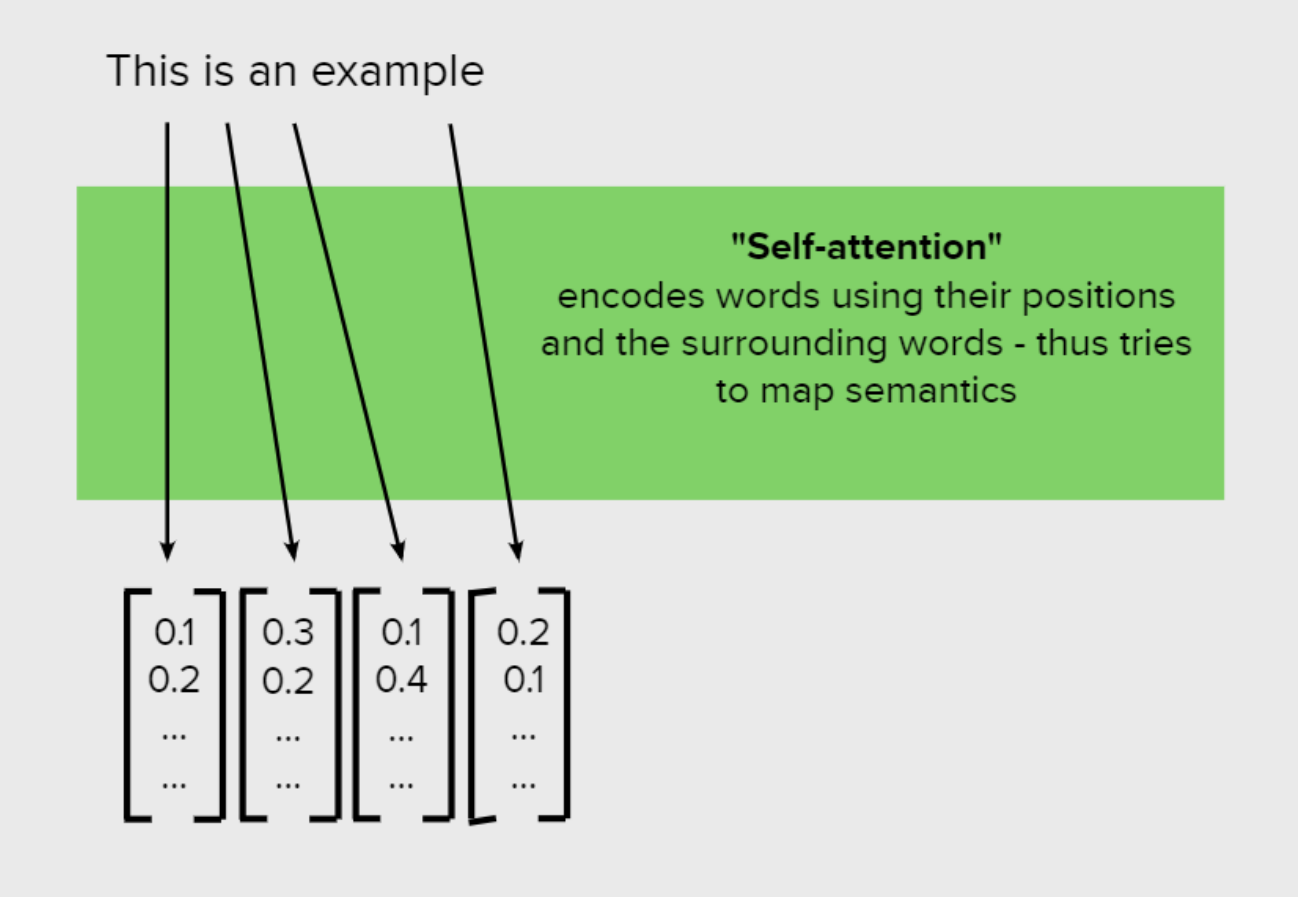
Huggincface is the place to go

Models out of the box:
https://colab.research.google.com/github/embarced/notebooks/blob/master/deep/transformers-pipelines.ipynb?hl=en

https://huggingface.co/transformers/ main framework is Pytorch, but supports TensforFlow and JAX as well:
https://huggingface.co/transformers/#supported-frameworks

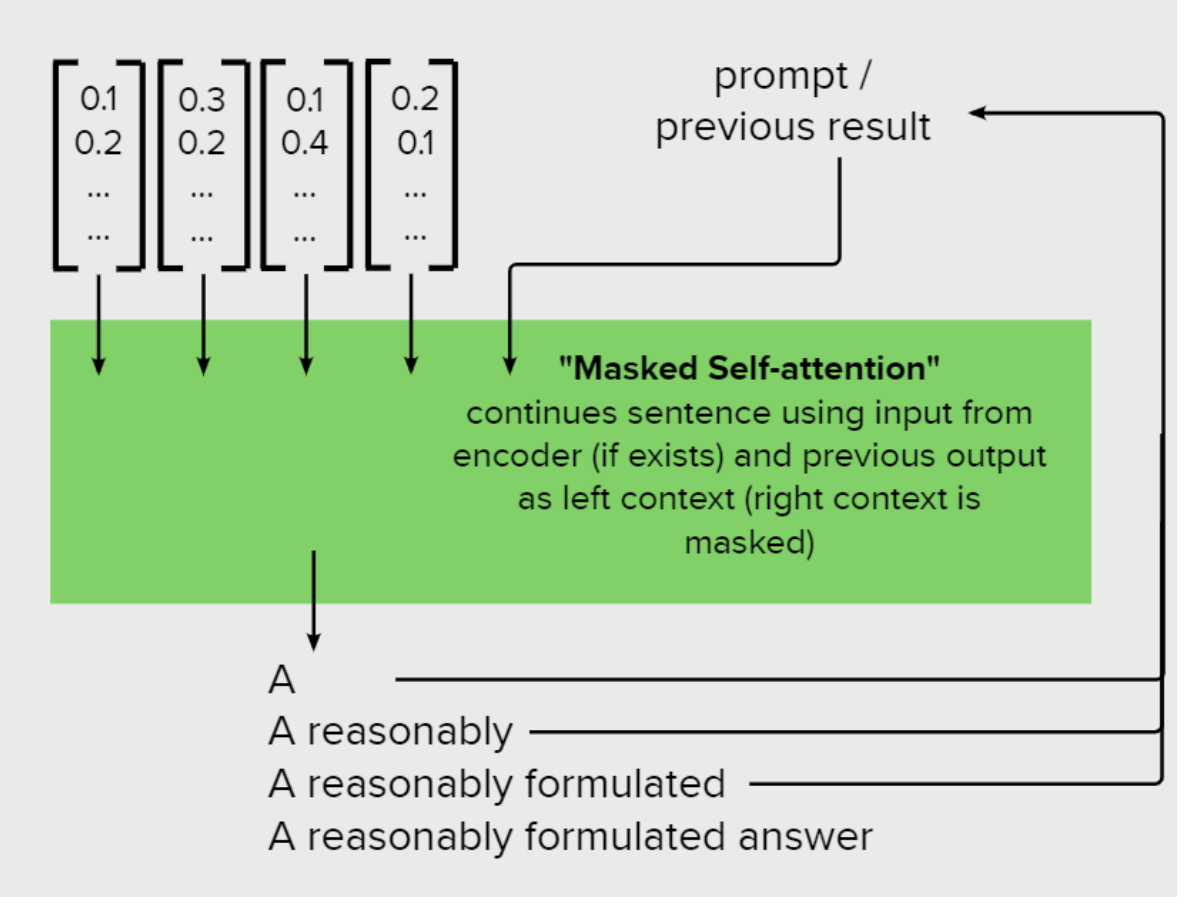
Encoder

encodes words into vectors (latent representation)

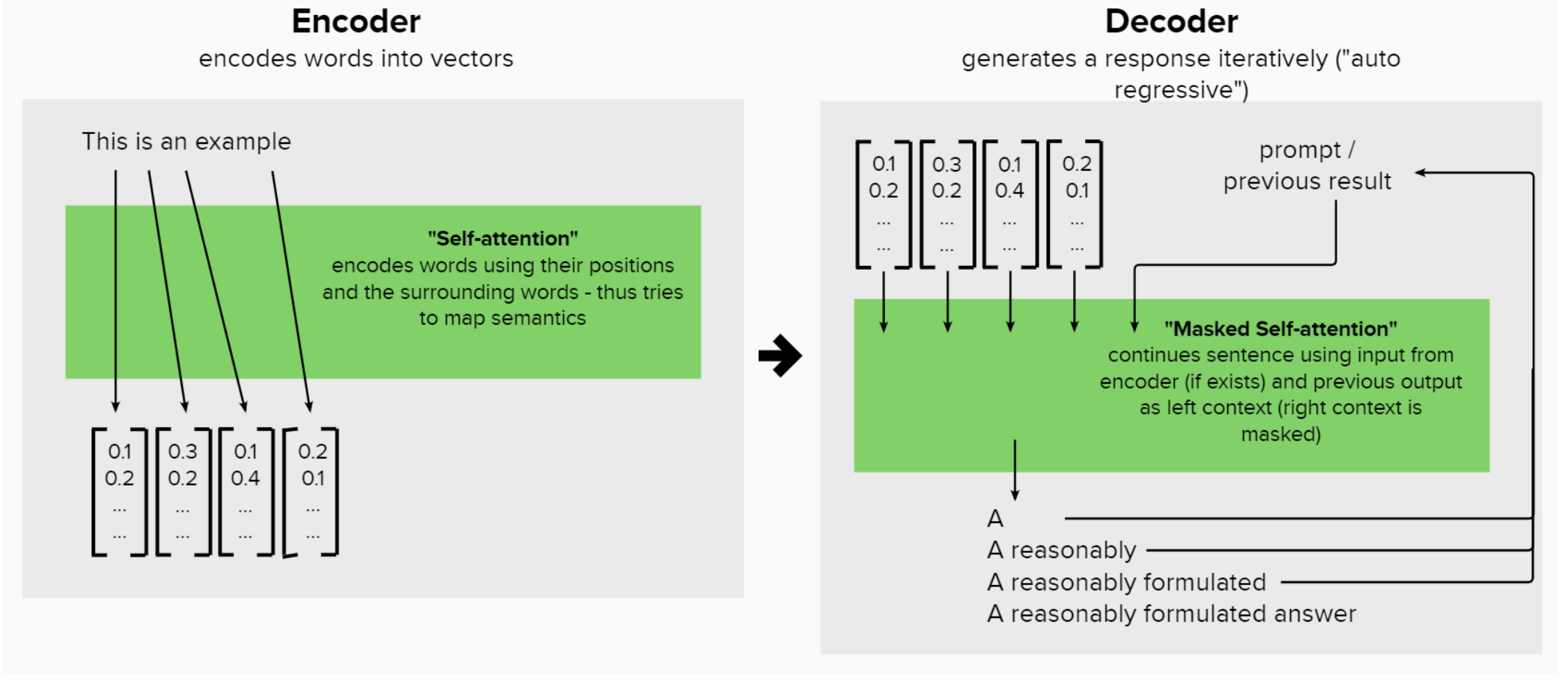


Decoder

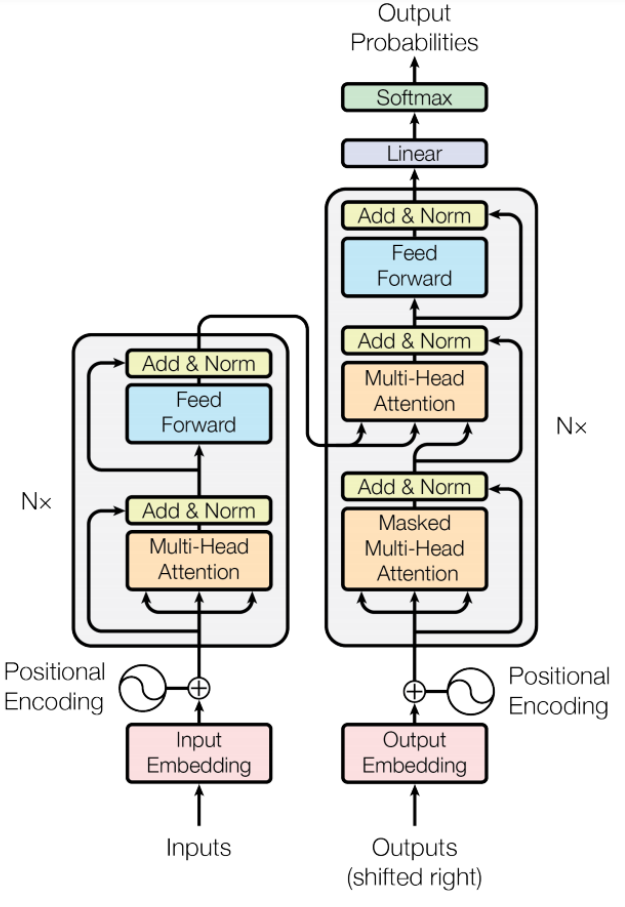
generates a response iteratively ("auto regressive")



Encoder / Decoder playing togehter



Transformer Architecture



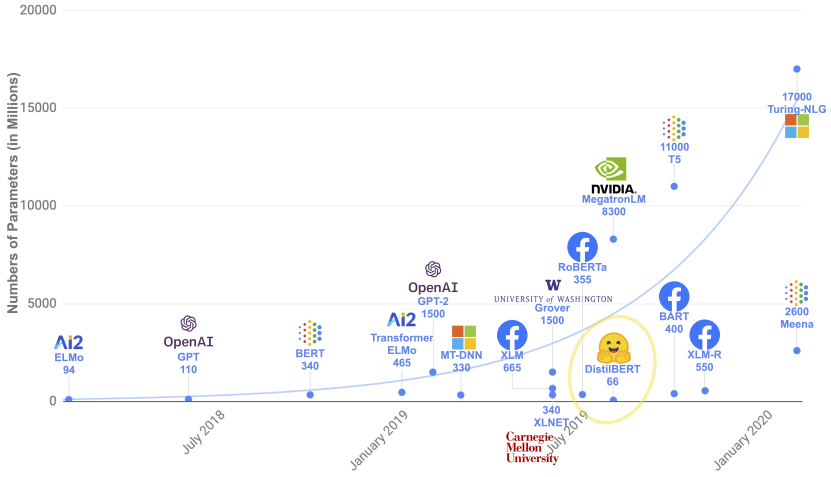
https://arxiv.org/pdf/1706.03762.pdf

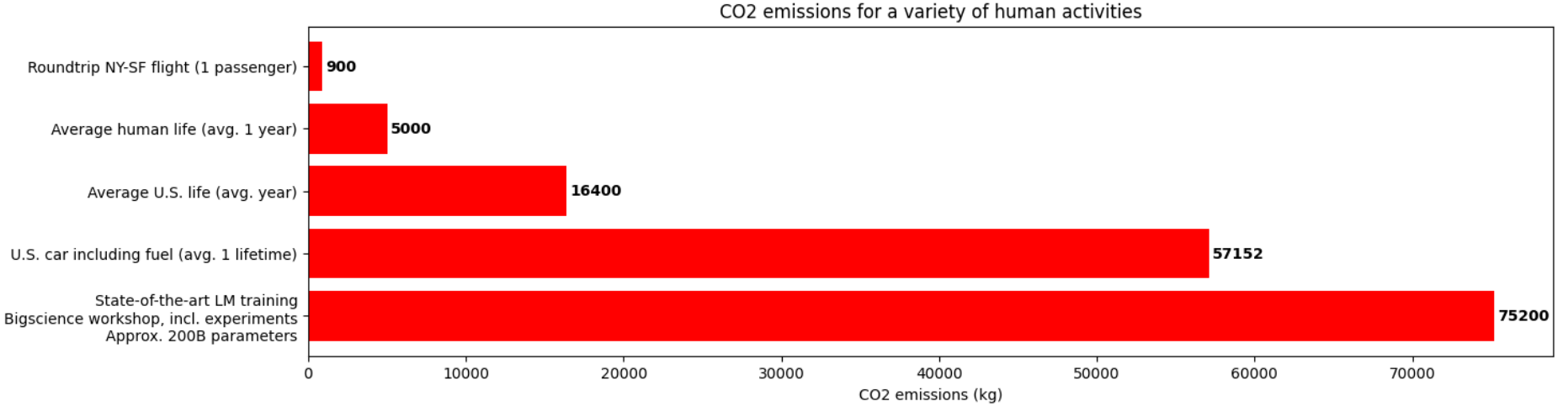
Transformer Zoo

- the original transformer was meant for translation tasks
- usage has broadened ever since
- spawning a whole zoo of transformers
- some use encoder only
- some use decoder only
- some use a combination of encoder/decoder just like the original transformer

https://huggingface.co/transformers/model_summary.html
https://huggingface.co/transformers/#supported-models

Evolution of Transformers (original paper is from 2017)





https://huggingface.co/course/chapter1/4?fw=pt

Complete transformer (BART/T5-like)

also called sequence-to-sequence Transformer models

- combined use of encoder and decoder, as in the original transformer approach
- allows summaries of texts in addition to translations
- name is probably most appropriate as texts really get transformed
- models like T5 and BART are most common here
- to be able to operate on all NLP tasks, it transforms them into text-to-text problems by using specific prefixes:
  - summarize:
  - question:
  - translate English to German:
  - etc.

https://arxiv.org/abs/1910.10683

Encoder only (BERT-like)

also called auto-encoding Transformer models

- some problems only need the encoder part of the original transformer
- "understanding" texts and their semantics is sufficient to e.g.
  - answer questions about a text with individual original text passages or to
  - sort the mood of a text into "positive" or "negative",
  - filling individual word gaps in texts
- in all three cases, no "complex" answer is required for which a decoder would be needed.
- BERT and derived models are famous off-the-shelf representatives for encoder only models

Example for encoder only BERT: Classifier

Email Classifier, urgent oder not urgend:

- https://twitter.com/ClementDelangue/status/1409728768915214337
- https://huggingface.co/clem/autonlp-test3-2101787?text=I+would+be+nice+if+this+is+done+by+next+year vs
- https://huggingface.co/clem/autonlp-test3-2101787?text=I+would+be+nice+if+this+is+done+by+yesterday

Decoder only (GPT-like)

also called auto-regressive Transformer models

- the decoder part can transform given inputs into complete sentences
- e.g. useful in itself, to complete started sentences
- GPT would be an example for this kind of application
  - unidirectional: trained to predict next word
  - by OpenAI

Evolution of GPT

GPT: Generative Pre-Trained Transformer

- GPT-1: 2018, 110 million parameters (https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf), https://www.youtube.com/watch?v=LOCzBgSV4tQ
- GPT-2: 2019, 1.5 billion parameters (https://d4mucfpksywv.cloudfront.net/better-language-models/language_models_are_unsupervised_multitask_learners.pdf), https://www.youtube.com/watch?v=BXv1m9Asl7I
- GPT-3: 2020, 175 billion parameters (https://arxiv.org/abs/2005.14165), https://www.youtube.com/watch?v=wYdKn-X4MhY

Typical example for decoder only GPT: completing a text

Title: United Methodists Agree to Historic Split
Subtitle: Those who oppose gay marriage will form their own denomination
Article: **After two days of intense debate, the United Methodist Church has agreed to a historic split - one that is expected to end in the creation of a new denomination, one that will be "theologically and socially conservative," according to The Washington Post. The majority of delegates attending the church's annual General Conference in May voted to strengthen a ban on the ordination of LGBTQ clergy and to write new rules that will "discipline" clergy who officiate at same-sex weddings. But those who opposed these measures have a new plan: They say they will form a separate denomination by 2020, calling their church the Christian Methodist denomination.**
The Post notes that the denomination, which claims 12.5 million members, was in the early 20th century the "largest Protestant denomination in the U.S.," but that it has been shrinking in recent decades. The new split will be the second in the church's history. The first occurred in 1968, when roughly 10 percent of the denomination left to form the Evangelical United Brethren Church. The Post notes that the proposed split "comes at a critical time for the church, which has been losing members for years," which has been "pushed toward the brink of a schism over the role of LGBTQ people in the church." Gay marriage is not the only issue that has divided the church. In 2016, the denomination was split over ordination of transgender clergy, with the North Pacific regional conference voting to ban them from serving as clergy, and the South Pacific regional conference voting to allow them.

Figure 3.14: The GPT-3 generated news article that humans had the greatest difficulty distinguishing from a human written article (accuracy: 12%).

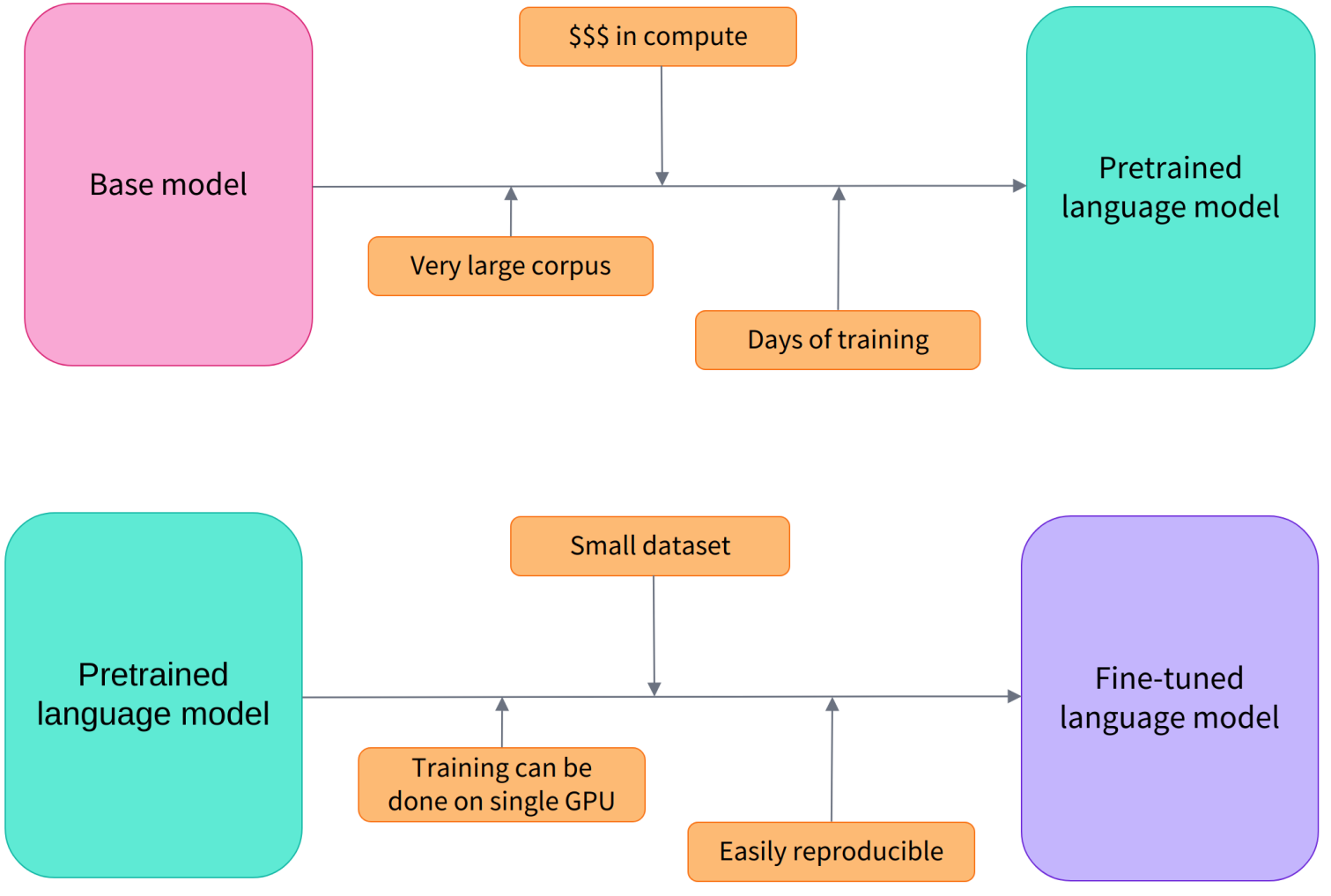
https://arxiv.org/pdf/1706.03762.pdf

Example for decoder only GPT: DALL·E generating images from text

two red sails opposing each other, schematic

https://openai.com/blog/dall-e/
https://huggingface.co/spaces/flax-community/dalle-mini

Fine-tuning using transfer learning



https://huggingface.co/course/chapter1/4?fw=pt

Fine tune sentiment on IMDB

https://colab.research.google.com/github/embarced/notebooks/blob/master/deep/transformers-fine-tuning.ipynb?hl=en

- make a copy of the notebook to your Google Drive
- make the model on whatever GPU you will get
- make up some movies reviewsa and evaluate them using the model

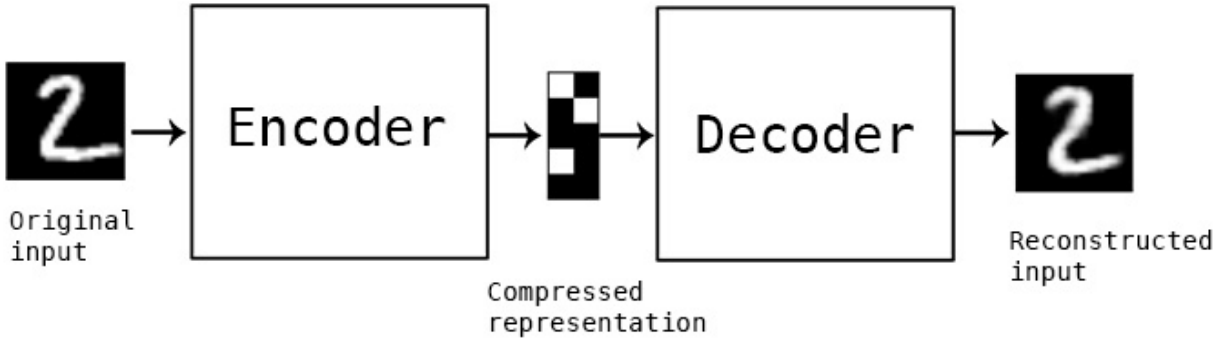
On the Opportunities and Risks of Foundation Models

- foundational models: trained on broad data at scale and are adaptable to a wide range of downstream tasks
- ML is undergoing a paradigm shift with the rise of these models
- their scale results in new emergent capabilities
- defects of the foundation model are inherited by all the adapted models downstream
- lack of clear understanding of how they work, when they fail, and what they are even capable of

https://arxiv.org/abs/2108.07258

Autoencoder for Unsupervised Deep Learning

- Data
  - Input: **All types of data are possible**
  - Output: **The same data**
- Learning type: **Unsupervised**
- Model architecture: **Neural network, symmetric with bottleneck**
- Loss: **typically MSE, all NN losses possible**
- Optimization Method: **Backpropagation**



https://blog.keras.io/building-autoencoders-in-keras.html

Why Autoencoders?

- Compression
- Data denoising
- Dimensionality reduction
- Building an abstract representation for further use
- Clustering (also for data visualization)
- Outlier detection

Example Application #1: Data Denoising

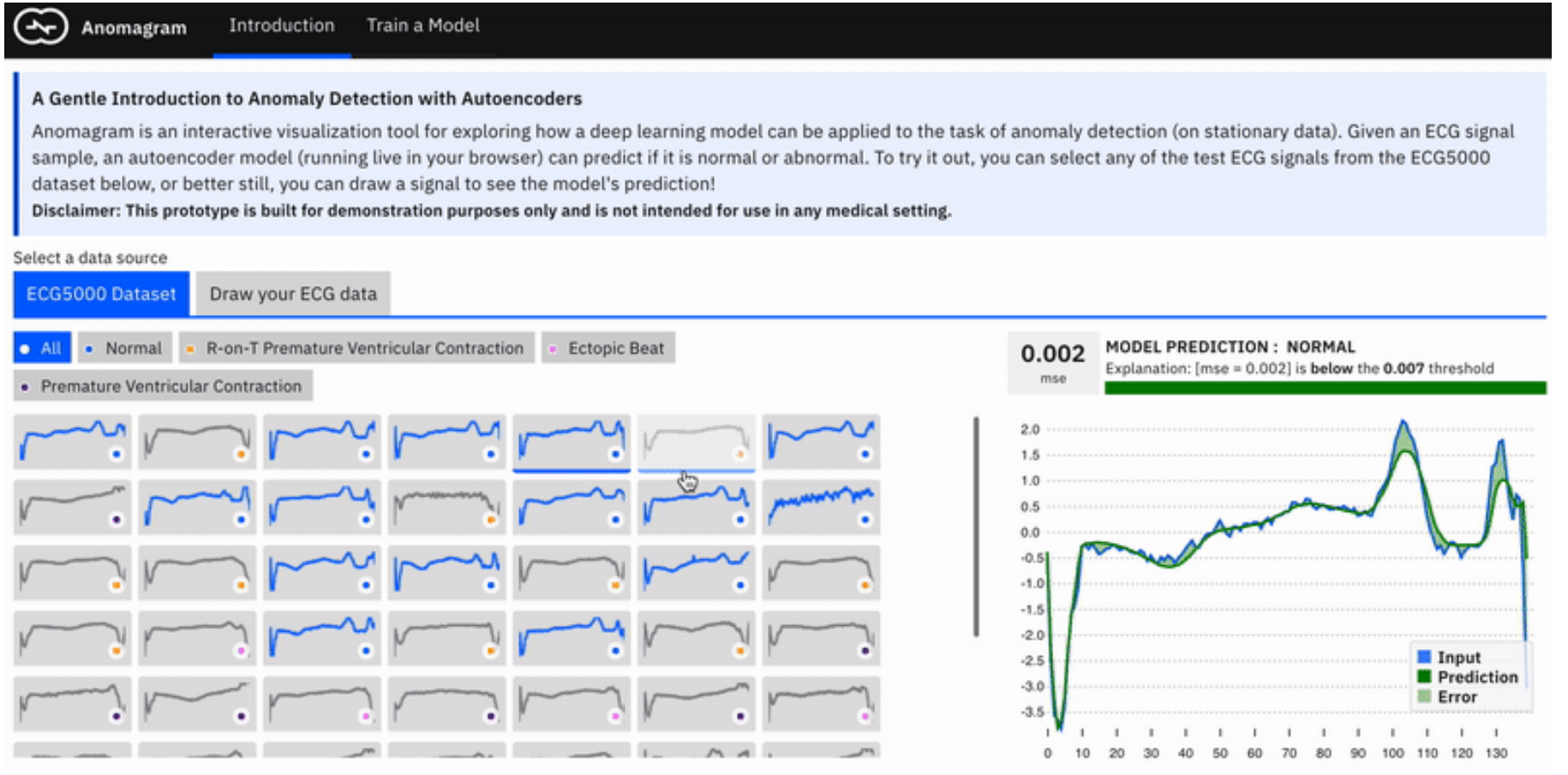
https://colab.research.google.com/github/embarced/notebooks/blob/master/deep/autoencoder-denoising.ipynb?hl=en

Example Application #2: Outlier Detection

An anomaly (outlier, abnormality) is defined as “an observation which deviates so much from other observations as to arouse suspicions that it was generated by a different mechanism” - Hawkins 1980.

https://www.springer.com/gp/book/9789401539968

Demo: Anomagram

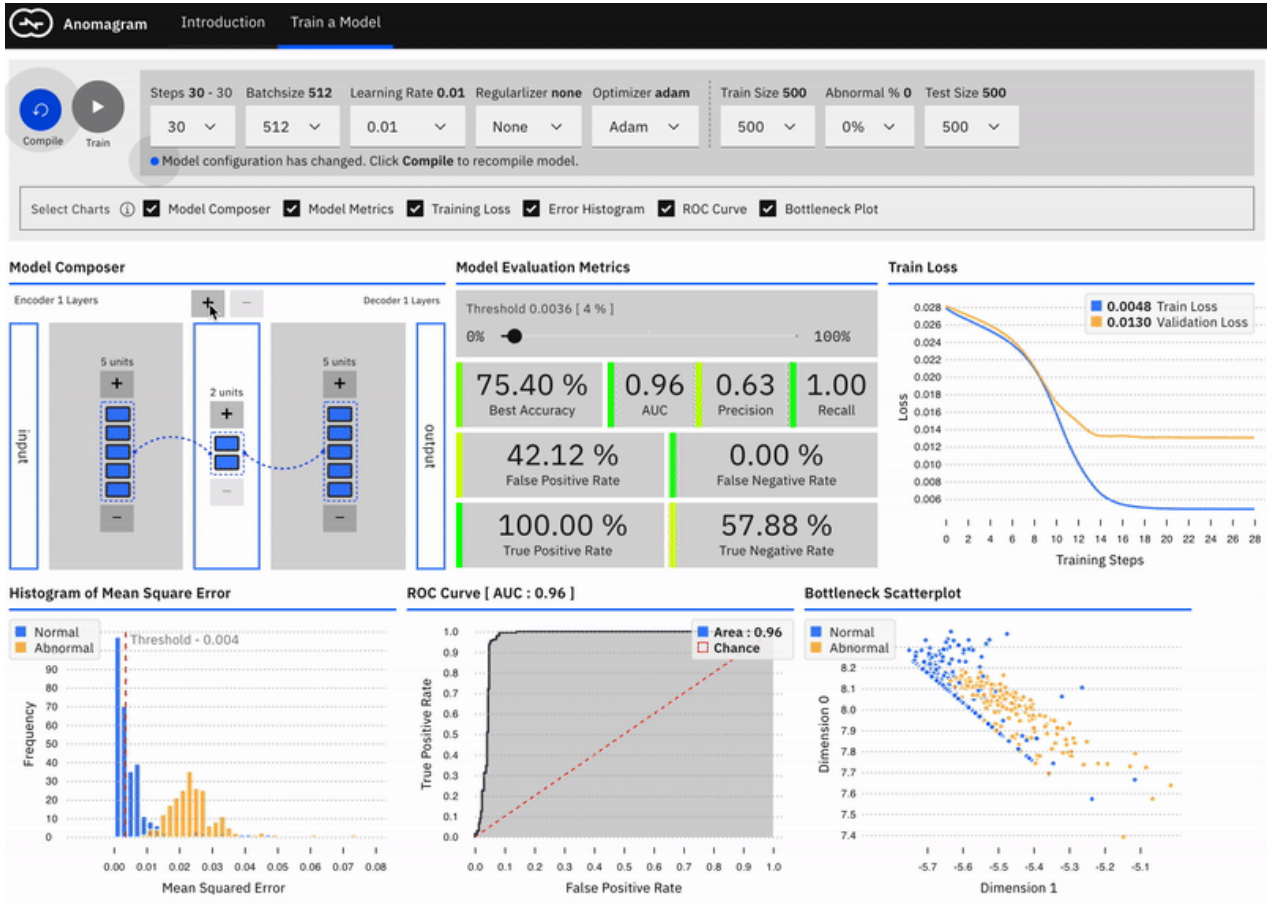


https://github.com/victordibia/anomagram
https://victordibia.github.io/anomagram/#/

Detecting Anomalies in electrocardiograms (ECG)

- Train with normal data to low reconstruction loss
- Abnormal data will not be reproduced well, i.e. will have high reconstruction loss
- Set a threshold on loss by maximizing a metric (accuracy)
- Does not need abnormal data for training (but can tolerate some abnormal data)
- Can do without any labelling

Exercise: Train your first Autoencoder



https://victordibia.github.io/anomagram/#/train

Our example: Regression testing

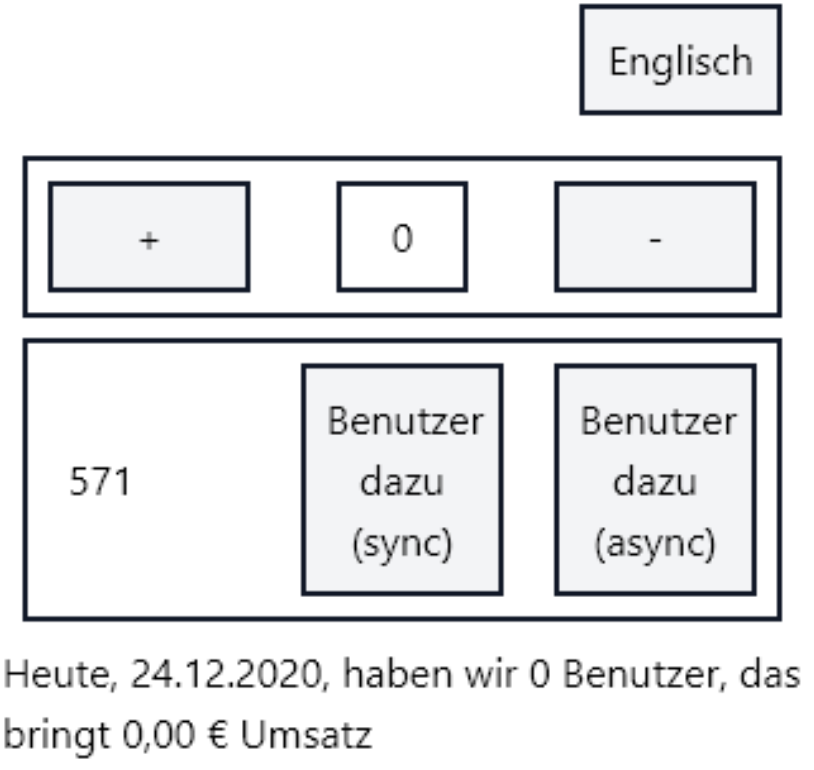
Regression testing is re-running functional and non-functional tests to ensure that previously developed and tested software still performs after a change.

If not, that would be called a regression. Changes that may require regression testing include bug fixes, software enhancements, configuration changes, and even substitution of electronic components.

As regression test suites tend to grow with each found defect, test automation is frequently involved. Sometimes a change impact analysis is performed to determine an appropriate subset of tests.

https://en.wikipedia.org/wiki/Regression_testing

Our sample application



Simple, but complex enough to show typical error cases

https://djcordhose.github.io/react-showcase/

Autoencoder for Outlier Detection

https://colab.research.google.com/github/embarced/notebooks/blob/master/deep/autoencoder-outlier-detection.ipynb?hl=en

Hands-On:

1. train model and tweak threshold to identify broken schreenshots

Optional:

1. experiment with dimensions of latent space, L1 and L2 regularization, and batch size and epochs
2. change resolution of images
3. experiment with color
4. can you come up with a better architecture of the network?
5. do other losses work for you?
6. do the plots of the latent space make sense?

More links

- More autoencoder examples: https://colab.research.google.com/github/DJCordhose/ml-workshop/blob/master/notebooks/tf-intro/2020-01-autoencoder.ipynb
- Wide range of examples from autoencoder, our architecture is derived from it: https://colab.research.google.com/github/ageron/handson-ml2/blob/master/17_autoencoders_and_gans.ipynb
- Academic considerations
  - https://fleuret.org/dlc/#lecture-7
    - https://twitter.com/francoisfleuret/status/1382708204468060171
  - https://www.deeplearningbook.org/contents/autoencoders.html
  - Deconvolution explained: https://arxiv.org/pdf/1603.07285v1.pdf, p. 18ff

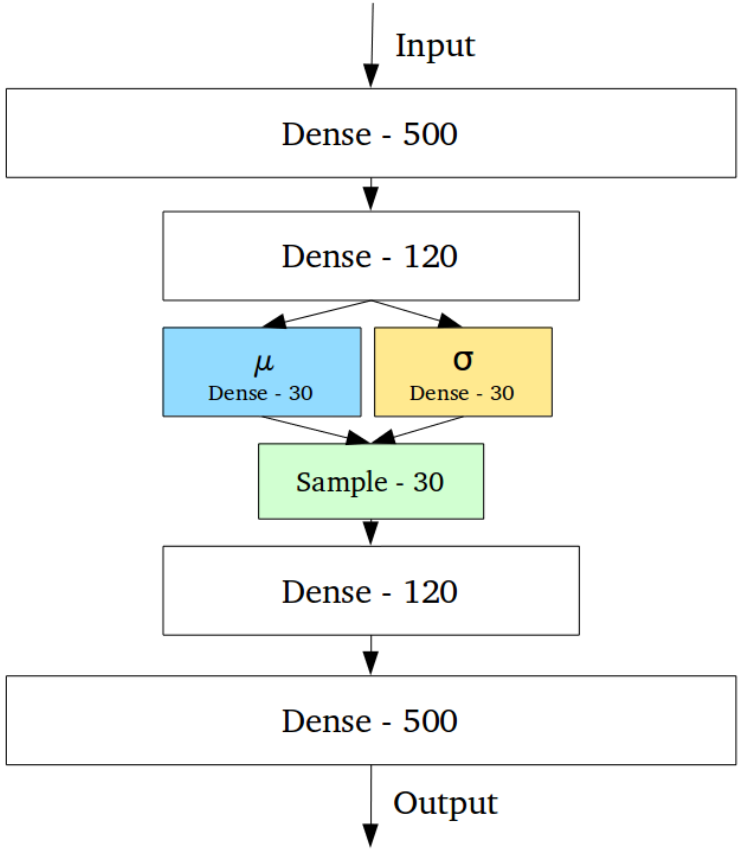
What else counts as Deep Unsupervised Learning?

Variational Auto Encoders (VAE)

- VAE is a generative model
- latent space learns a probability distribution modelling your data
- actually learning mean and standard deviation of distribution
- sampling from it can generate new data

https://blog.keras.io/building-autoencoders-in-keras.html
https://colab.research.google.com/github/DJCordhose/ml-workshop/blob/master/notebooks/tf2/autoencoder-mnist-vae-gan.ipynb
https://github.com/keras-team/keras/blob/master/examples/variational_autoencoder.py

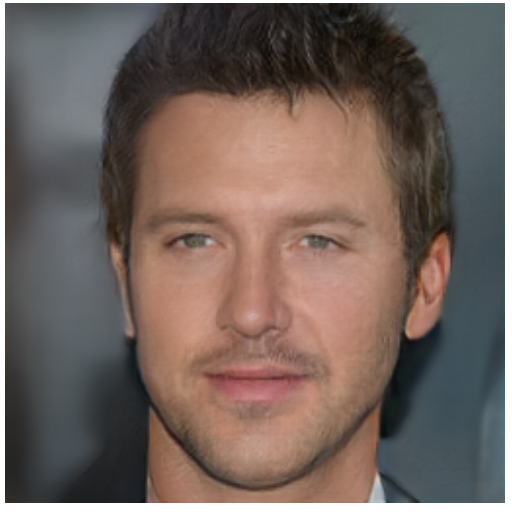
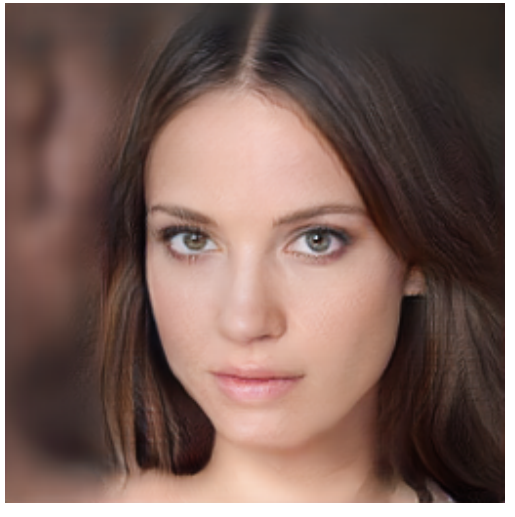
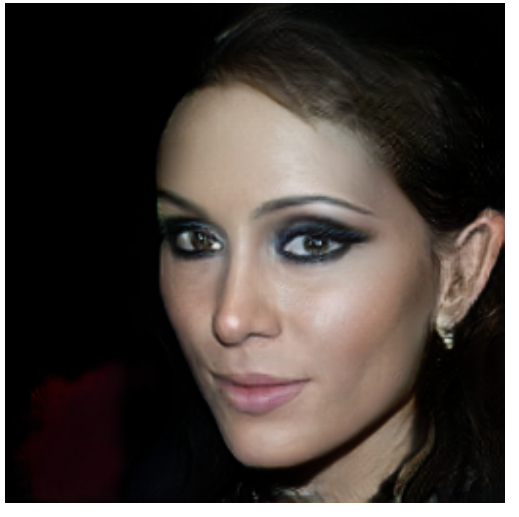
VAE illustrated



https://towardsdatascience.com/intuitively-understanding-variational-autoencoders-1bfe67eb5daf

Celebrities

Do you know any of them?


Images of Celebrities have been generated

Trained for two weeks on a single high-end GPU on CelebA-HQ data set (images of celebreties)

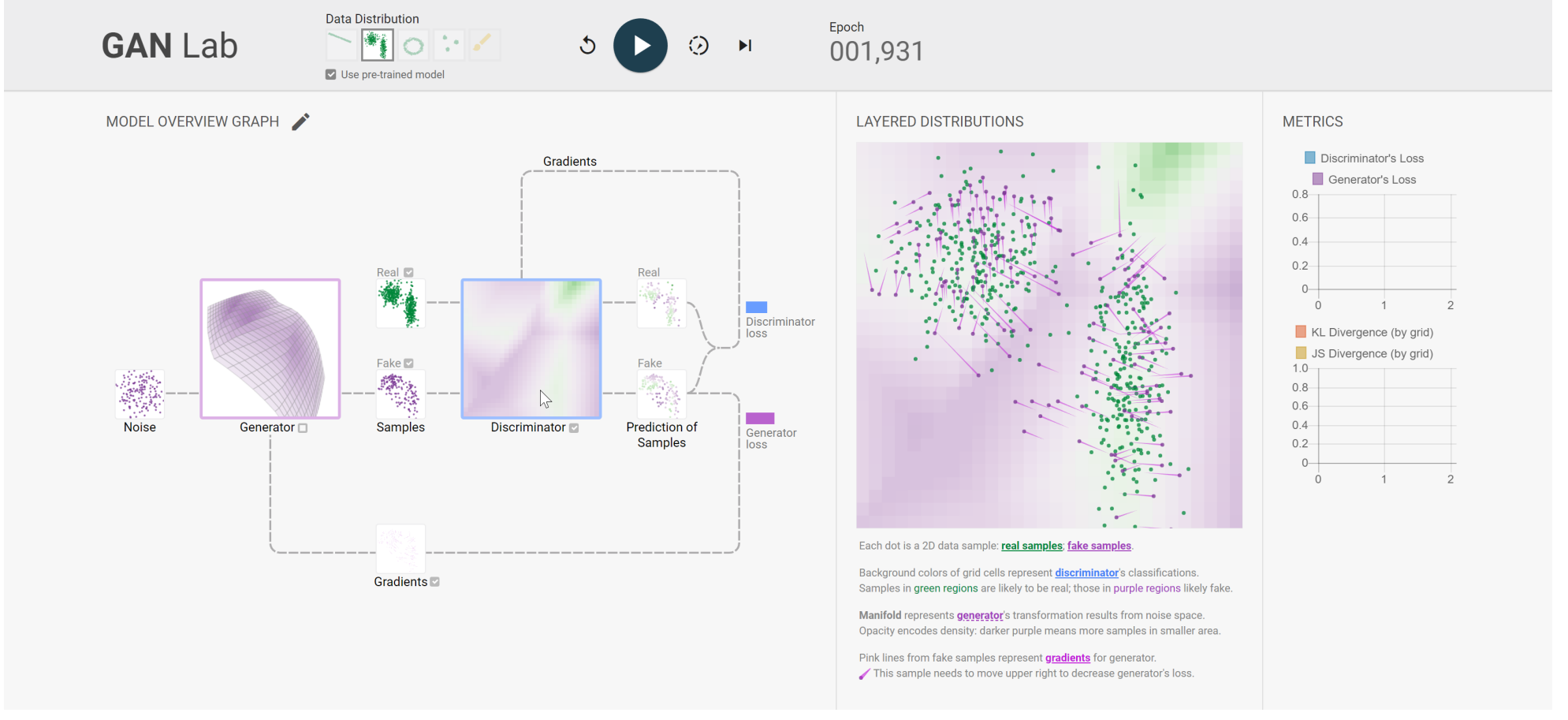
https://alantian.net/ganshowcase/
https://github.com/alantian/ganshowcase
https://twitter.com/alanyttian/status/988242167998148608

GANs

- Generative adversarial networks
- uses two neural networks, working against each other
- one network tries to create artefacts that look real
- the other one tries to classify which artefact presented is real and which is fake
- typically superior to VAE

https://pathmind.com/wiki/generative-adversarial-network-gan
https://arxiv.org/abs/1406.2661

Understanding GANs



https://twitter.com/minsukkahng/status/1037016214575505409
https://poloclub.github.io/ganlab/
https://minsuk.com/research/papers/kahng-ganlab-vast2018.pdf

Restricted Boltzmann Machines (RBMs)

- RBMs were founded 1986 by Geoffrey Hinton
- Two-Layer networks like half an autoencoder where data flows back and forth
- interesting for historical reasons, but surpassed by more up-to-date models: AE, VAE, GANS
- Embeddings/Latent Space/Hidden Representation originally called 'Distributed representations'

https://en.wikipedia.org/wiki/Boltzmann_machine
https://dl.acm.org/citation.cfm?id=104287
http://www.cs.toronto.edu/~hinton/absps/families.pdf
https://pathmind.com/wiki/restricted-boltzmann-machine
https://towardsdatascience.com/restricted-boltzmann-machines-simplified-eab1e5878976

Schedule

- Day 1: Introduction
    1. TensorFlow
    2. Basics of Supervised Learning
    3. Deep Learning best practices
- Day 2: Applications
    1. Tabular Data
    2. Image Recognition
    3. Sequences
- Day 3: Advanced
    1. Unsupervised Deep Learning
    2. ***Deep Reinforcement Learning***

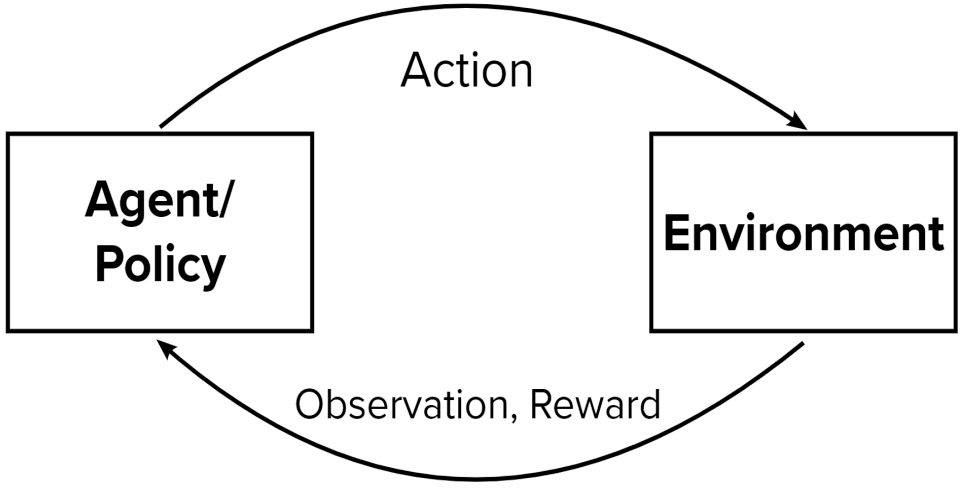
Reinforcement Learning

- special position in the field of machine learning
- does not rely on existing data
- generates data in experiments
- experiments conducted in simulation or in the real world, if possible safely and in large numbers
- experiments are guided by desired outcome
- desired outcome is formally defined by rewards
- corresponds most closely to what many people intuitively imagine as intelligent behavior

State of Reinforcement Learning

- still rather academic
- computer or board games are typical academic application
- algorithm is central and important for academics
- practical applications include
  - optimization
  - robotics
  - multi-armed bandits as a more general form of A/B testing

Real challenge is to model the problem for RL



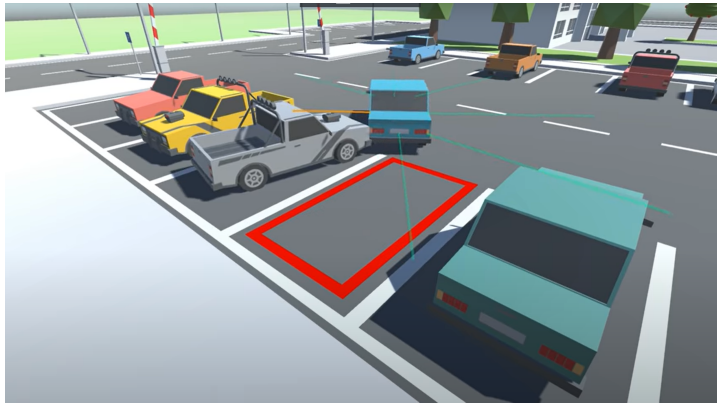
1. Based on ***Observations*** an ***Agent*** executes
2. ***Actions*** within a given
3. ***Environment*** which lead to positive or negative
4. ***Rewards***.

The Agent's job is to maximize the cumulative Reward

http://gym.openai.com/docs/

Sample Modelling

- input eight depth sensors, current speed and position, relative position to the target
- outputs engine force, braking force and turning force
- Policy as NN with 3 hidden layers of 128 neurons each

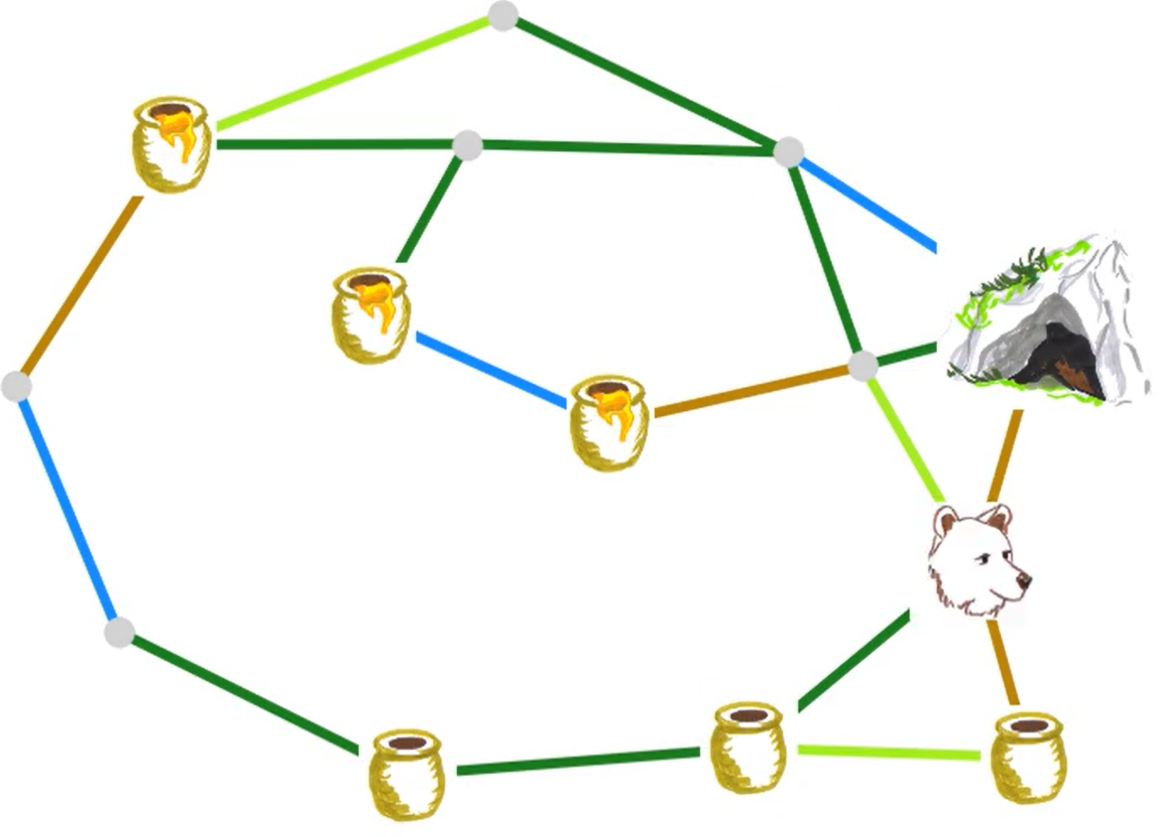


- Setting: https://youtu.be/VMp6pq6_QjI
- 20k trials, first success: https://youtu.be/VMp6pq6_QjI?t=140
- 300k trials, starts looking good: https://youtu.be/VMp6pq6_QjI?t=572

More fun parking

- two AI Agents fighting for the same parking spot: https://twitter.com/SamuelArzt/status/1175738904055562240
- AI Learns to parallel park: https://twitter.com/SamuelArzt/status/1248642920875528194
- bits of code: https://twitter.com/SamuelArzt/status/1176487746195603457

Our Application



Structure of Observation and Reward are crucial

agents can learn short cuts and unexpected behavior

What machines learned as opposed what the designers intended them to learn

- Reward shaping a soccer robot for touching the ball caused it to learn to get to the ball and vibrate touching it as fast as possible
- Robot hand pretending to grasp an object by moving between the camera and the object
- Simulated pancake making robot learned to throw the pancake as high in the air as possible in order to maximize time away from the ground
- Agent pauses the game indefinitely to avoid losing
- In an artificial life simulation where survival required energy but giving birth had no energy cost, one species evolved a sedentary lifestyle that consisted mostly of mating in order to produce new children which could be eaten (or used as mates to produce more edible children).
- Agent kills itself at the end of level 1 to avoid losing in level 2

What machines learned as opposed what the designers intended them to learn - cont'd

- Evolved player makes invalid moves far away in the board, causing opponent players to run out of memory and crash
- Creatures exploited physics simulation bugs by twitching, which accumulated simulator errors and allowed them to travel at unrealistic speeds
- Reward-shaping a bicycle agent for not falling over & making progress towards a goal point (but not punishing for moving away) leads it to learn to circle around the goal in a physically stable loop.
- ... algorithm learns to bait an opponent into following it off a cliff, which gives it enough points for an extra life, which it does forever in an infinite loop.
- The PPO algorithm discovers that it can slip through the walls of a level to move right and attain a higher score.

https://vkrakovna.wordpress.com/2018/04/02/specification-gaming-examples-in-ai/
https://arxiv.org/abs/1803.03453
https://docs.google.com/spreadsheets/u/1/d/e/2PACX-1vRPiprOaC3HsCf5Tuum8bRfzYUiKLRqJmbOoC-32JorNdfyTiRRsR7Ea5eWtvsWzuxo8bjOxCG84dAg/pubhtml
https://twitter.com/mogwai_poet/status/1060286856493813760

Exercise on paper: How to model as an RL problem?

- Objective: What should be achieved? When is an episode over?
- Environment: Who or what provides observations and rewards? Typically a simulation or the real world.
- Actions: What are the possible actions of the agent?
- Agent / Policy: Who or what chooses actions?
- Observations: What does the environment indicate to the agent?
- Rewards: What are the rewards and how much are they?

Sample/Reference Solution: Objective

- find the least strenuous way through the territory
- eat all the honey
- finally get back to the cave
- after a certain number of steps we abort the episode without success

Sample/Reference Solution: Environment

Software simulation of the bear's territory

Sample/Reference Solution: Actions

- 4 discrete actions in the style of an Atari game:
  - left
  - right
  - up
  - down
- doing nothing is not an option
  - the bear never learns that staying in the cave and just sleeping on might be an interesting solution to the problem

Sample/Reference Solution: Agent:

The bear

- how this looks like technically we have to discuss later

Sample/Reference Solution: Observations

- we show the bear only the part of its territory that it can see from its current position (partially observed)
- such an observation makes the problem more challenging
- just as we would have to explore the game situation in such a game, the bear would have to do the same in many playthroughs
- we choose this variant to demonstrate the power of reinforcement learning

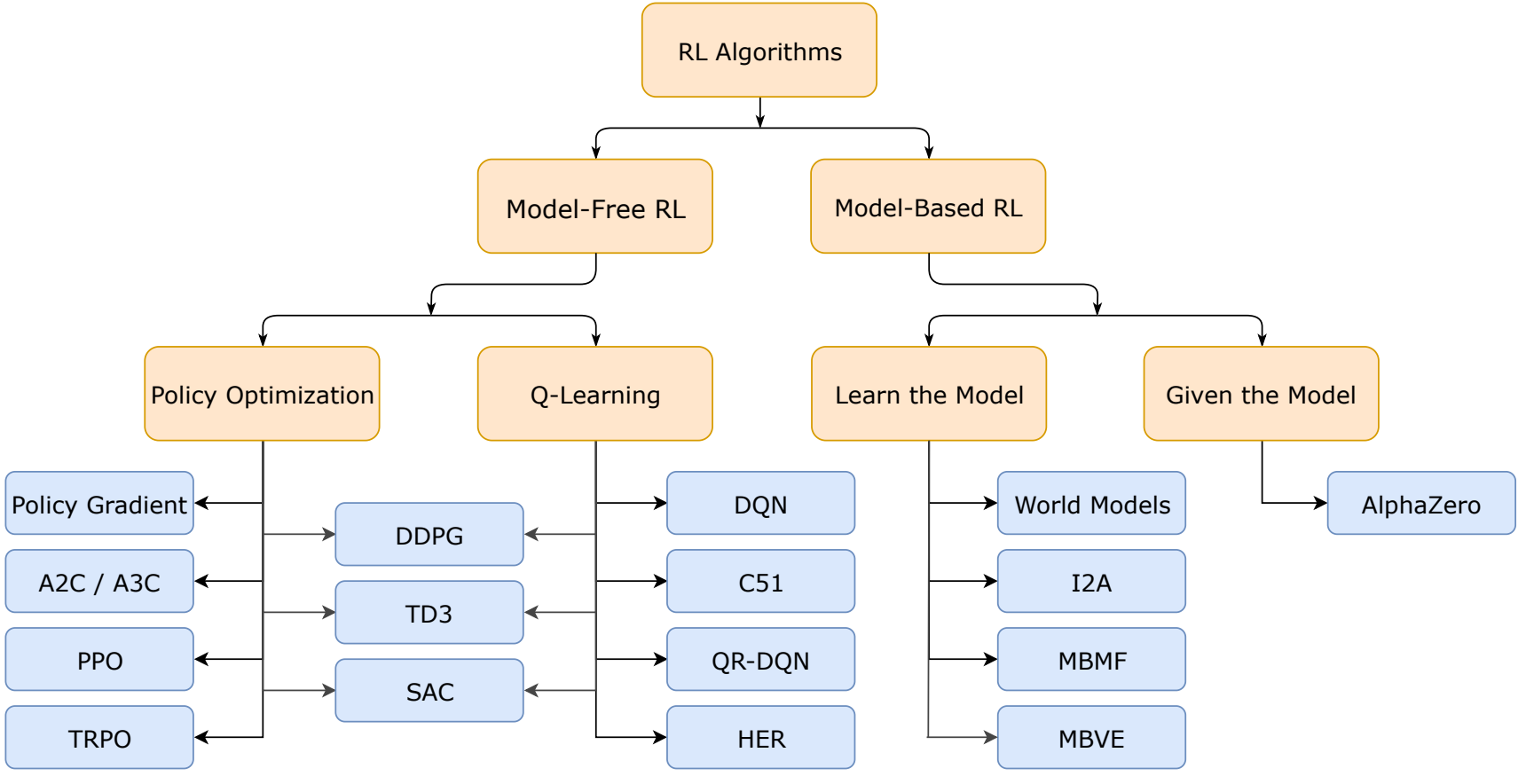
More details in Notebook

Sample/Reference Solution: Rewards

- for each step we combine different rewards
- normalize them by the theoretical number of points to be achieved
- namely the sum of the rewards of all honey pots.
- reward always below 1
- if it is negative there was only cost, no reward.
- different costs per path are modeled
- honey is attractive to collect even if the path is expensive
- otherwise the incentive for the bear is not there

Now we have formulated the challenge as an RL problem, we need to choose a learning algorithm

There are a lot of algorithms



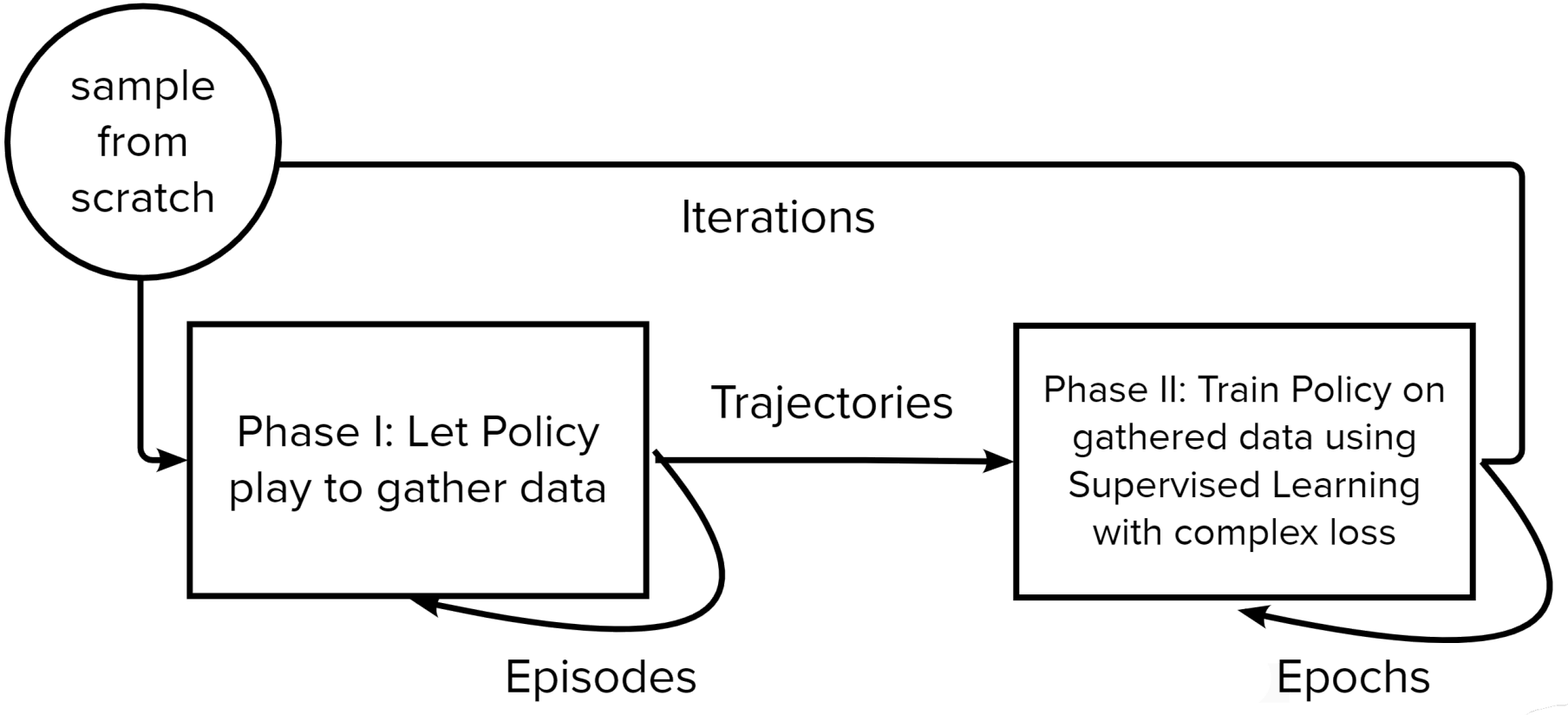
https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html

For practical work PPO is the algorithm to choose when creating samples is inexpensive

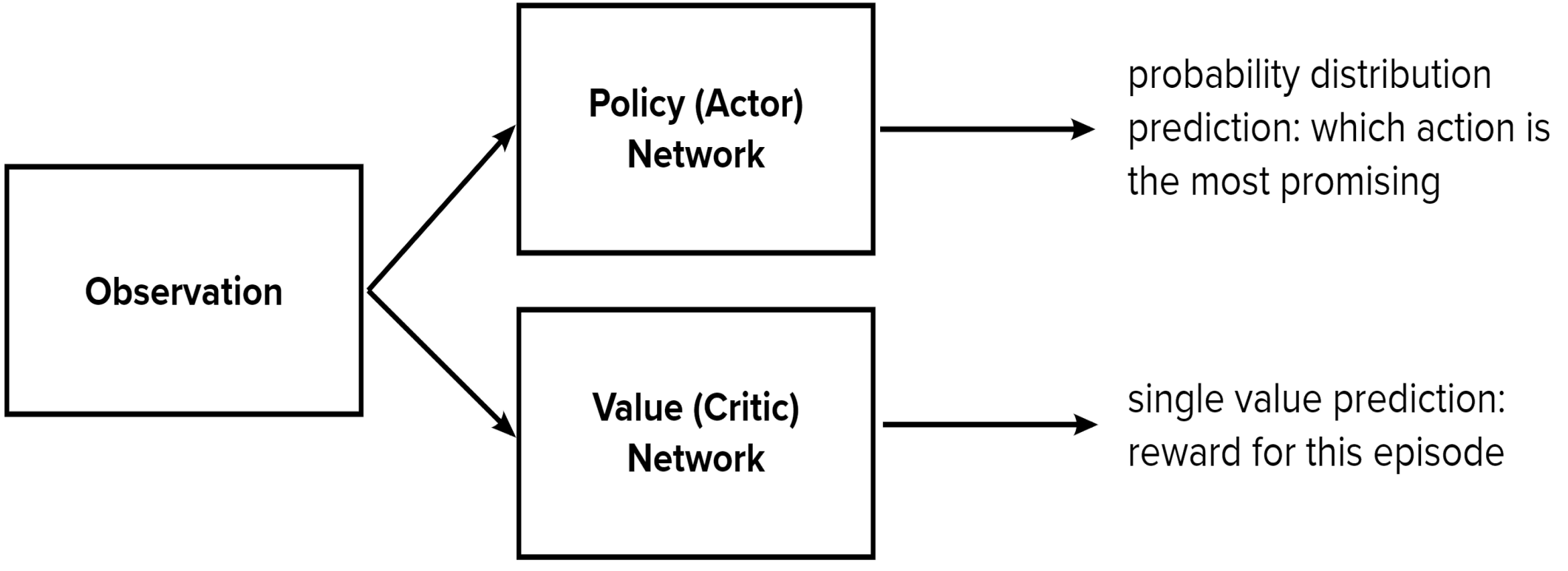
- if samples are expensive SAC might be best choice
- for our example sampling is cheap

https://spinningup.openai.com/en/latest/algorithms/ppo.html

PPO (Proximal Policy Optimization)



Deep Reinforcement Learning with PPO



PPO uses two networks

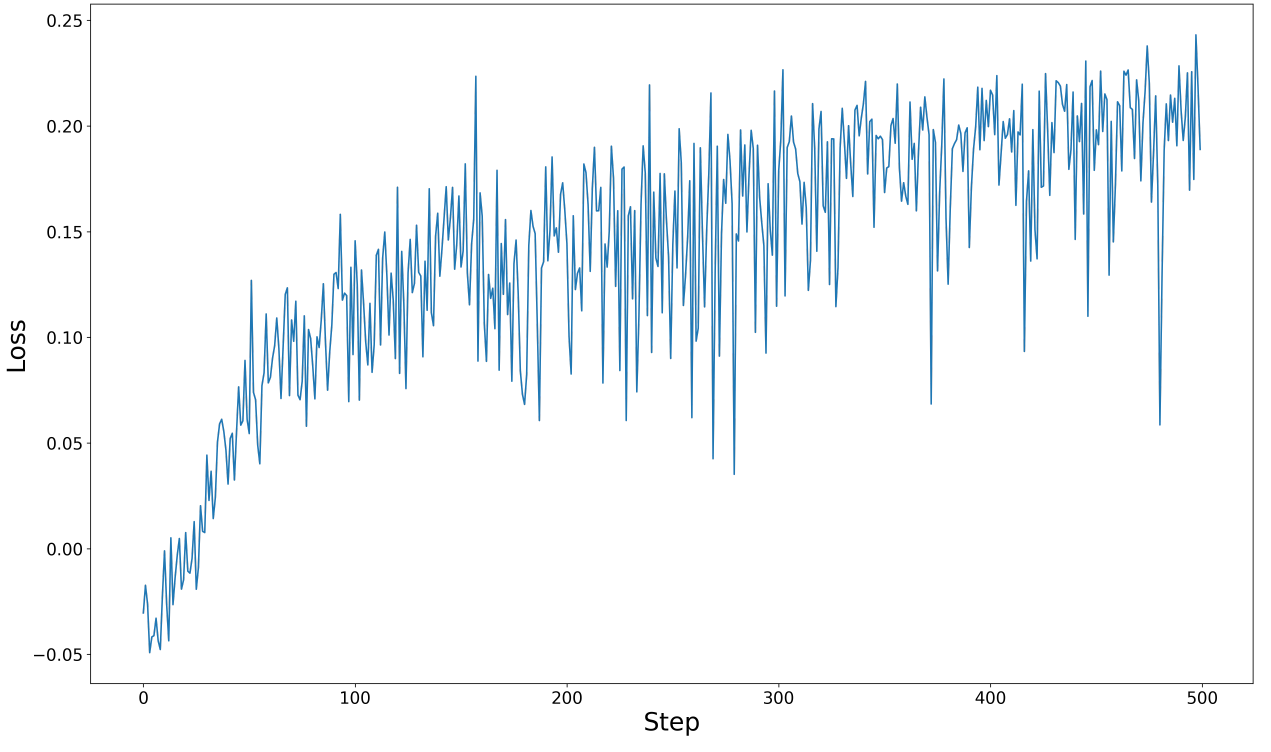
Special Loss in PPO

- most advanced and practical variant of policy gradient branch
- uses vanilla backpropagation for training
- has special loss consisting of many parts
- turns hard constraints into penalties
- uses value function to have better generalization

More details in Notebook

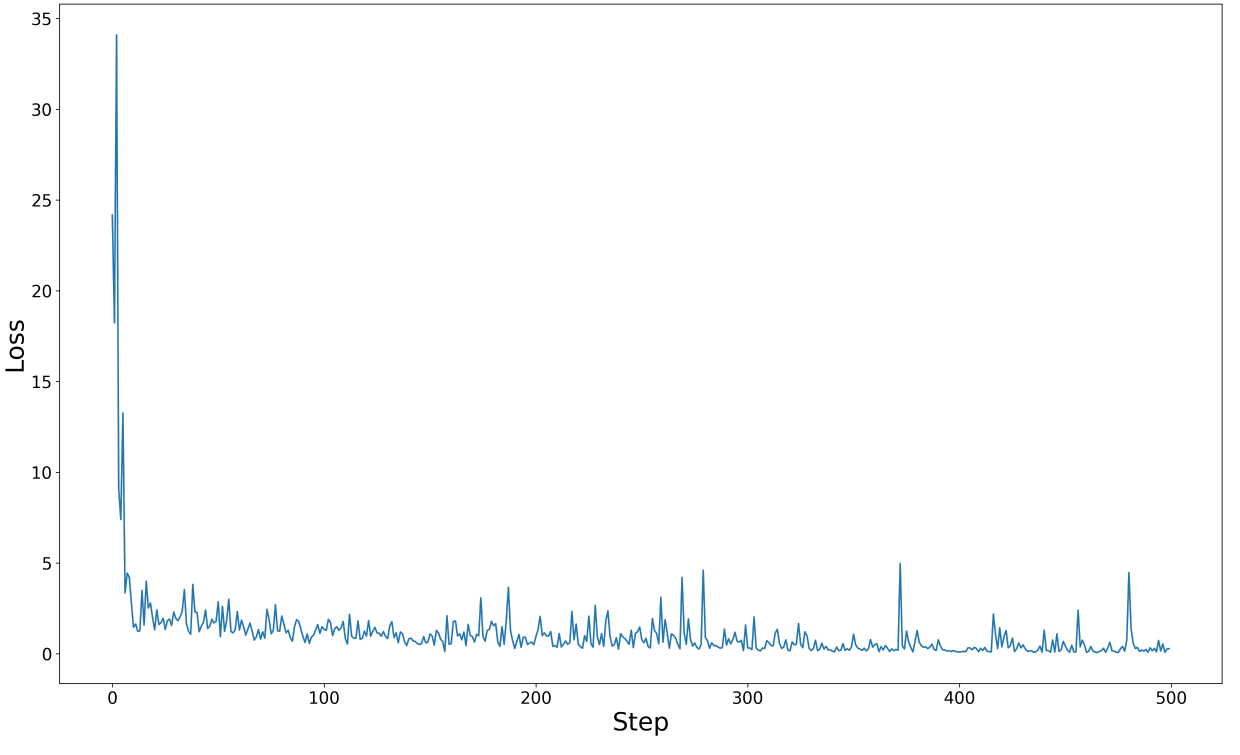
Policy Loss

which action to perform



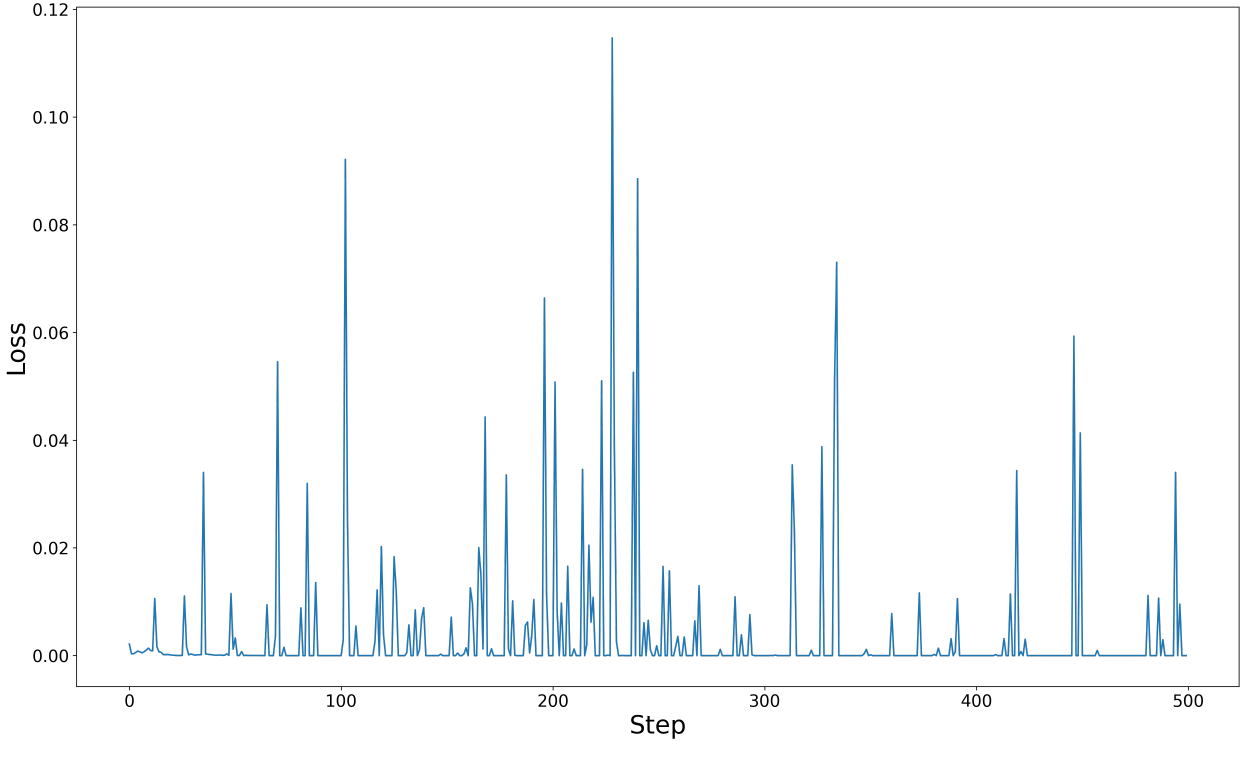
Value Loss

prediction of reward



KL Loss

prevent catatrophic updates



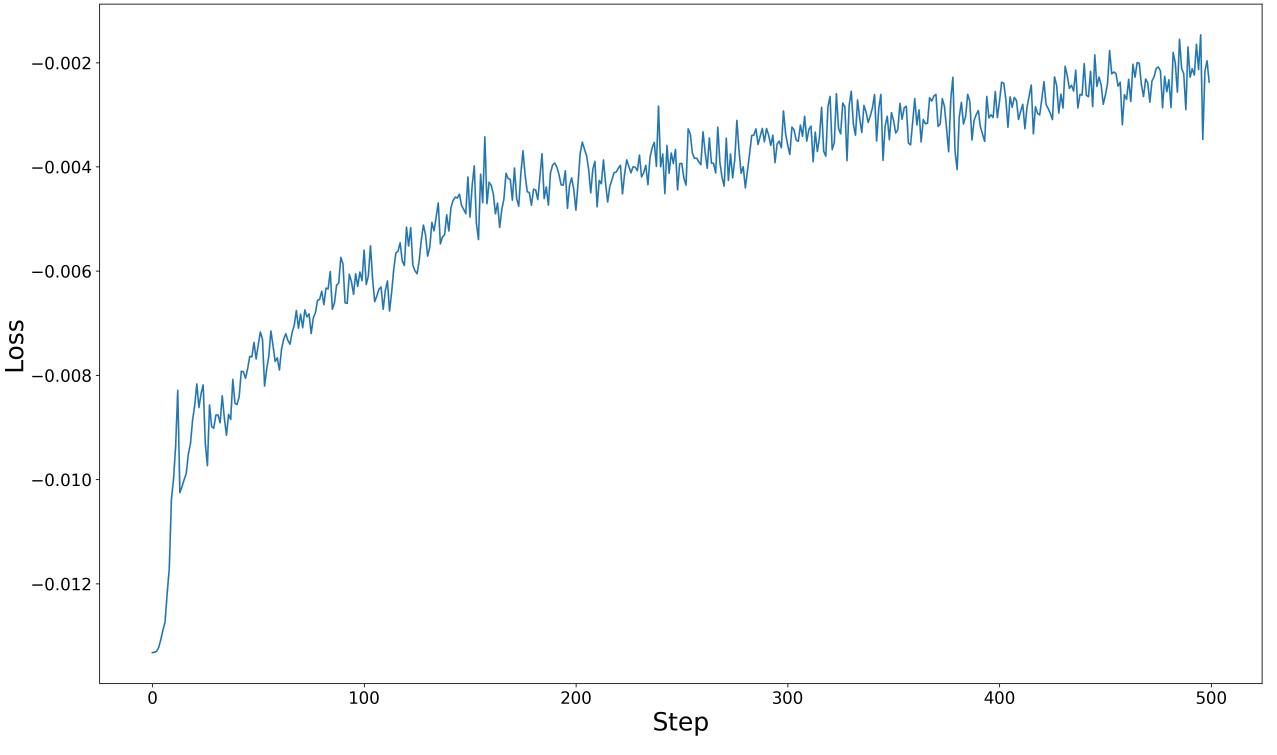
KL Divergence in PPO

we don't want any new policy to be too different from the current one

- a measure of how difference of two probability distributions
  - https://en.wikipedia.org/wiki/Kullback%E2%80%93Leibler_divergence
- Penalize big changes in the action distribution from policy network
- Adding KL Divergence as part of the loss function
- Makes it accessible to standard first order optimizers in NN backpropagation
  - https://medium.com/@jonathan_hui/rl-proximal-policy-optimization-ppo-explained-77f014ec3f12

Entropy Loss

trade exploitation for exploration over time



Interactive introduction to Deep Reinforcement Learning

https://colab.research.google.com/github/embarced/notebooks/blob/master/deep/intro_rl.ipynb?hl=en

Hands-On train your own RL model

Optional Hands-On: tweak agent and training

1. Tweak Value and Policy Network
2. Tweak PPO parameters including entropy and loss
3. Change training time

Alternative Hands-On: experiment with different modeling

1. staying home becomes an option - "no move" is a valid action
2. no penalty for invalid move (walk out of world)
3. change rewards
4. change observation

How good can you get? What is the optimum?

- generally you just do not know
- the higher the score the better
- baselines can help in Reinforcement Learning as well
- in our special case, there is a deterministic baseline
  - best first search (https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm)
  - guarentees to find optimal solution
  - exponential complexity
  - but works fine for our size of the problem
  - optimal score between ***.73*** and ***.74*** with very low variance
  - number of steps to pass through complete turf around ***17***
- our reinforcement learning approach trades the perfect solution for
  - nearly perfect solution
  - linear time complexity

Applying Reinforcement Learning to a problem

Is your problem approachable by Reinforcement Learning?

- Can you define what would be the agent and what the environment?
- You can simulate your environment or are able to safely perform a large number of experiments in the real world
- You need to be able to define your problem as a Markov Decision Process (MDP)
- Good sanity check: Would you as a human be able to play the game based on the observation and reward you get?
- there are different kinds of observations
- fully observed environment (e.g. Jump'n'Run Game)
- partially observed environment (e.g. Ego Shooter)
- Choose a proper Reinforcement Learning Algorithm

More Resources

- Open-Ended Learning Leads to Generally Capable Agents - Deepming getting closer to AGI
  - https://deepmind.com/blog/article/generally-capable-agents-emerge-from-open-ended-play
  - https://twitter.com/maxjaderberg/status/1420024605721432074
  - https://www.youtube.com/watch?v=lTmL7jwFfdw
- Alternative Lib: https://github.com/thu-ml/tianshou/
- PPO from scratch
  - https://keras.io/examples/rl/ppo_cartpole/
  - https://github.com/higgsfield/RL-Adventure-2/blob/master/3.ppo.ipynb
- MuZero: http://www.furidamu.org/blog/2020/12/22/muzero-intuition/
- The scientific paper for PPO: https://arxiv.org/abs/1707.06347
- A3C paper, explains several basic techniques such as the utility of the Critic and the additional entropy loss-term: https://arxiv.org/abs/1602.01783

Finally

More Teaching Resources

- https://www.tensorflow.org/guide
- https://github.com/parrt/fundamentals-of-deep-learning
- Short and practical introduction to data science: https://github.com/rasbt/data-science-tutorial
- Fundamentals of Machine Learning and Deep Learning in Python using Scikit-Learn, Keras and TensorFlow 2: https://github.com/ageron/handson-ml2
- CS231n: Convolutional Neural Networks for Visual Recognition: http://cs231n.stanford.edu/
- CS224n: Natural Language Processing with Deep Learning: http://web.stanford.edu/class/cs224n/index.html#schedule

All Notebooks

- Intro:
    - Quickstart: https://colab.research.google.com/github/embarced/notebooks/blob/master/deep/intro_quickstart.ipynb?hl=en
    - Regression: https://colab.research.google.com/github/embarced/notebooks/blob/master/deep/intro-regression.ipynb?hl=en
        - Extended Version: https://colab.research.google.com/github/embarced/notebooks/blob/master/deep/intro-regression-detail.ipynb?hl=en
    - Classification: https://colab.research.google.com/github/embarced/notebooks/blob/master/deep/intro-classification.ipynb?hl=en
        - Extended Version: https://colab.research.google.com/github/embarced/notebooks/blob/master/deep/intro-classification-detail.ipynb?hl=en
    - Generalization: https://colab.research.google.com/github/embarced/notebooks/blob/master/deep/intro-regularization.ipynb?hl=en
    - Metrics: https://colab.research.google.com/github/embarced/notebooks/blob/master/deep/binary-metrics.ipynb
- Tabular: https://colab.research.google.com/github/embarced/notebooks/blob/master/deep/tabular.ipynb?hl=en
- Image/CNN:
    - https://colab.research.google.com/github/embarced/notebooks/blob/master/deep/intro_cnn.ipynb?hl=en
    - Outdated version unsuccessfully trying transfer learning (just for the idea how to do that): https://colab.research.google.com/github/DJCordhose/ml-workshop/blob/master/notebooks/tf2/images-intro.ipynb?hl=en
- Sequences, RNNs:
    - Basics: https://colab.research.google.com/github/embarced/notebooks/blob/master/deep/intro_rnn_basics.ipynb?hl=en
    - Complete Example: https://colab.research.google.com/github/DJCordhose/ml-workshop/blob/master/notebooks/tf-intro/2020-01-time-series.ipynb?hl=en
    - Encoder-Decoder: https://colab.research.google.com/github/DJCordhose/ml-workshop/blob/master/notebooks/tf2/time-series-encoder-decoder.ipynb
    - Transformers
        - https://colab.research.google.com/github/embarced/notebooks/blob/master/deep/transformers-pipelines.ipynb?hl=en
        - https://colab.research.google.com/github/embarced/notebooks/blob/master/deep/transformers-fine-tuning.ipynb?hl=en
- Autoencoder/Unsupervised:
    - https://colab.research.google.com/github/embarced/notebooks/blob/master/deep/autoencoder-denoising.ipynb?hl=en
    - https://colab.research.google.com/github/embarced/notebooks/blob/master/deep/autoencoder-outlier-detection.ipynb?hl=en
- Deep Reinforcement Learning: https://colab.research.google.com/github/embarced/notebooks/blob/master/deep/intro_rl.ipynb?hl=en

All Playgrounds and interactive Visualizations

- Classic
  - https://ml-playground.com
- Deep Learning in general
  - https://okai.brown.edu
  - https://playground.tensorflow.org/
- Classification:
  - https://teachablemachine.withgoogle.com/
- CNNs
  - https://transcranial.github.io/keras-js/#/mnist-cnn
  - https://poloclub.github.io/cnn-explainer/
  - https://setosa.io/ev/image-kernels/
- GANs
  - https://poloclub.github.io/ganlab/
- RL????
- Embeddings and Autoencoder
  - https://projector.tensorflow.org/
  - https://victordibia.github.io/anomagram
- NLP
  - https://projector.tensorflow.org/
  - https://storage.googleapis.com/tfjs-models/demos/mobilebert-qna/index.html
- Metrics
  - https://zackakil.github.io/precision-recall-playground/

Finding data sets to play with

- Google released a search engine for datasets
  - Search: https://toolbox.google.com/datasetsearch
  - Launch blog post: https://www.blog.google/products/search/making-it-easier-discover-datasets/
- Kaggle Datasets: https://www.kaggle.com/datasets
- TensorFlow Datasets: https://medium.com/tensorflow/introducing-tensorflow-datasets-c7f01f7e19f3
- https://www.openml.org/search?type=data
- https://github.com/jbrownlee/Datasets

Where to go from here

- https://sebastianraschka.com/blog/2021/dl-course.html
- https://learning.oreilly.com/library/view/hands-on-machine-learning/9781491962282/
- https://www.manning.com/books/deep-learning-with-python-second-edition
  - https://github.com/fchollet/deep-learning-with-python-notebooks
- https://keras.io

Verified Demo Code

https://keras.io/examples/

Standard Models: Assorted

TensorFlow Hub is a repository of trained machine learning models

https://www.tensorflow.org/hub

https://tfhub.dev/

Standard Models: Language

https://huggingface.co/transformers/

Standard Models: Recommendation

https://github.com/tensorflow/recommenders

Final words

François Chollet 

@fchollet

I'd guess I'd summarize it as, being a "deep learning expert" in 2021 is like being a "medicine expert" in 1800. You know a lot less than you think, and most of what you think you know is wrong. Just keep learning and experimenting, and don't play stupid status games.

7:34 PM · Apr 25, 2021 · Twitter Web App

https://twitter.com/fchollet/status/1386373123889528835