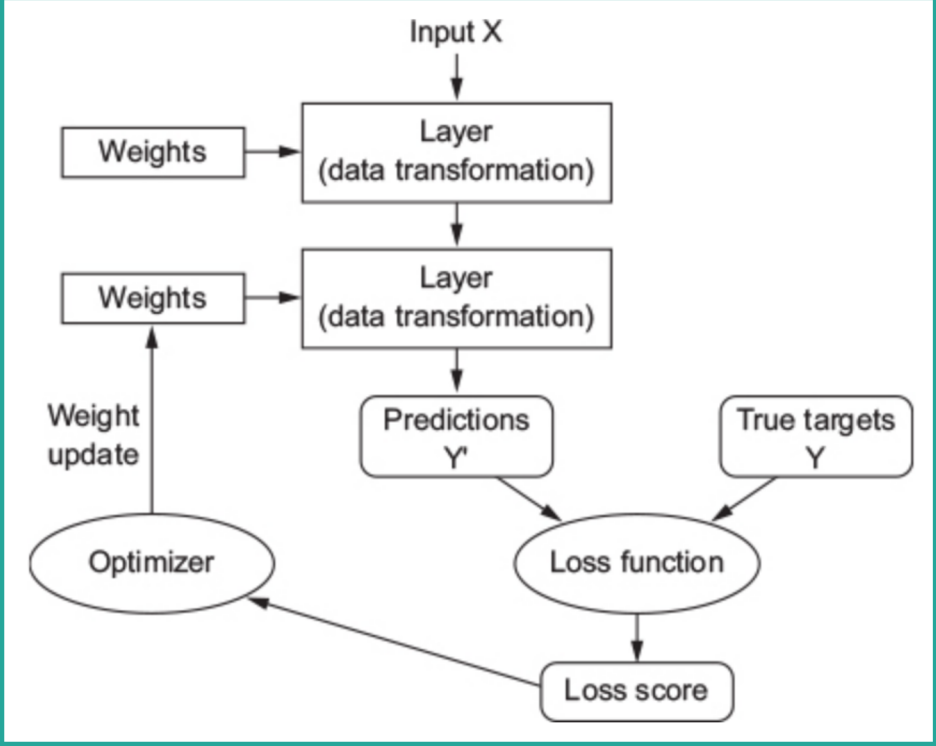
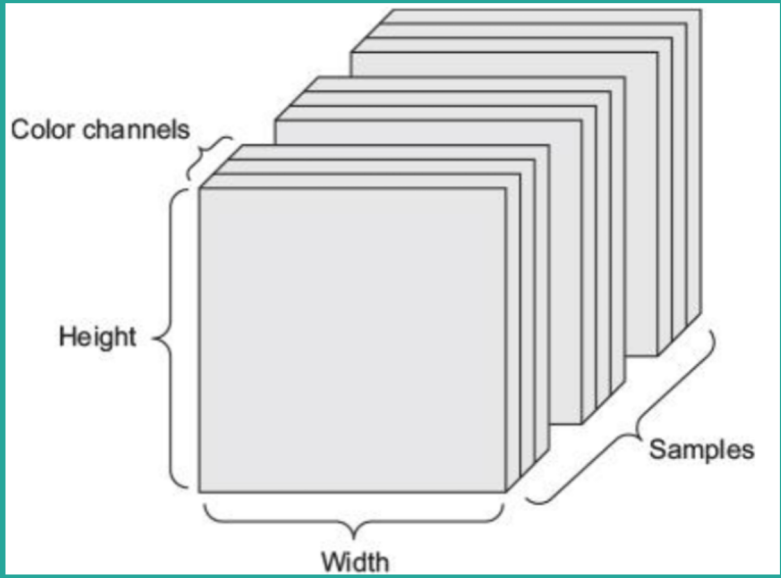
Neural Networks Architectures & Practical Learning Strategies

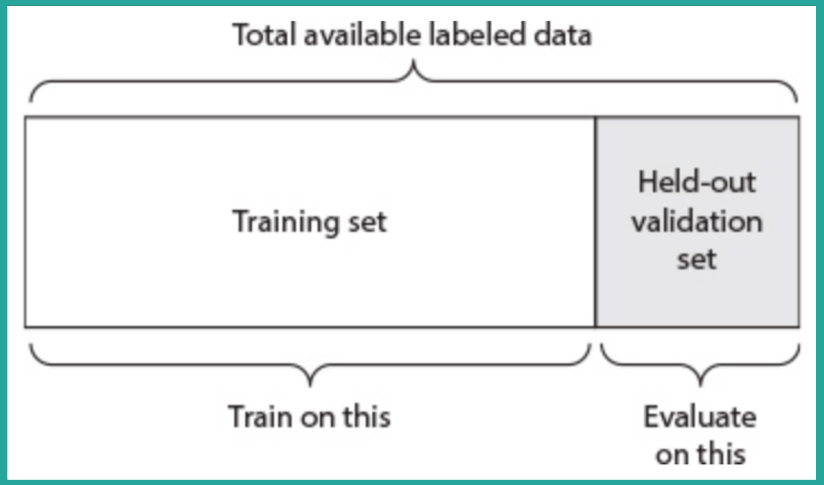
Julian Baer, Mathilde Boumasmoud and Almut Luetge

Recap from week 1





Recap from week 2

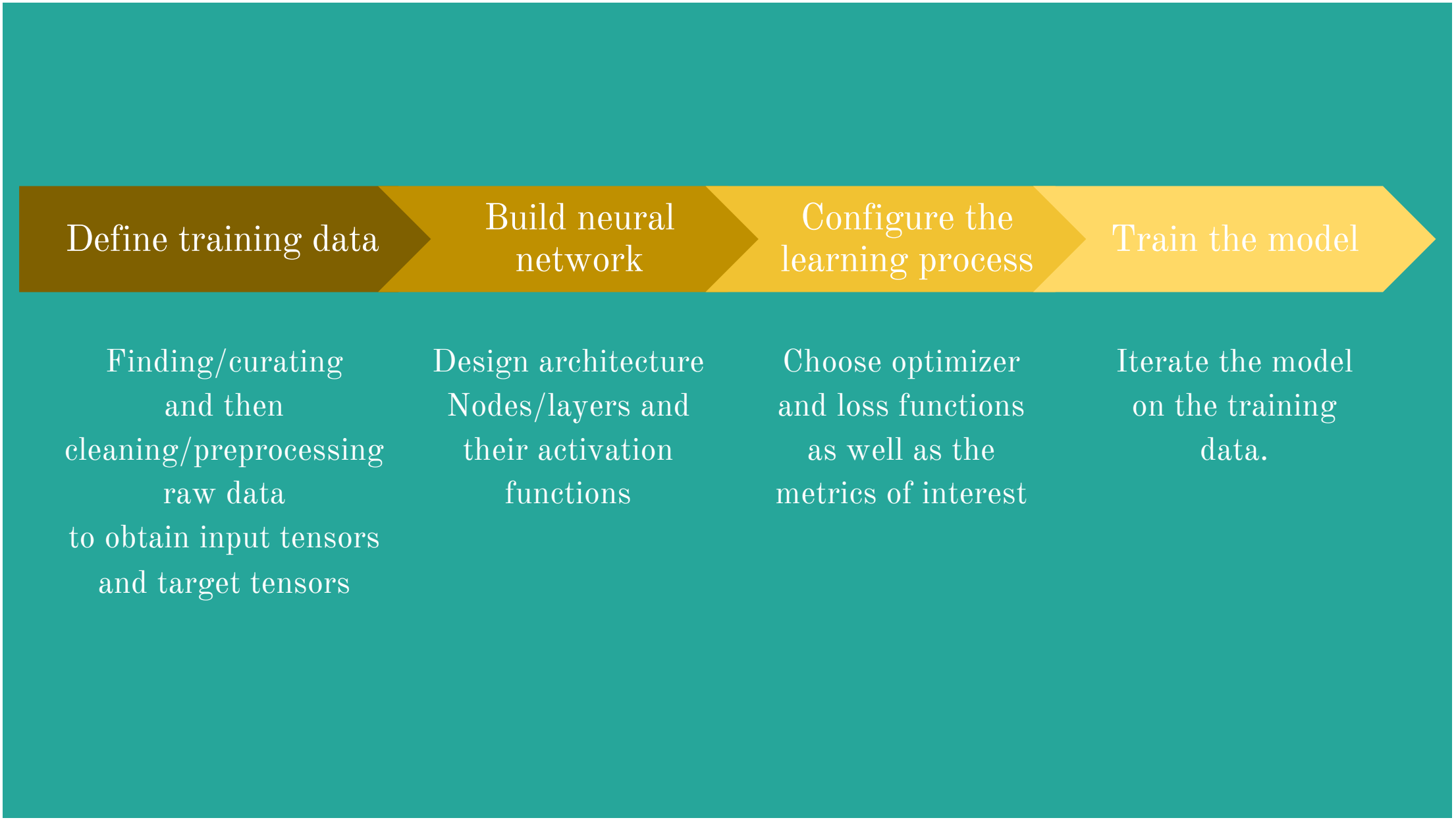


+ Test data

- Data distribution
- Internal Covariate Shift
  Normalization
- Vanishing/Exploding Gradients

} Batch

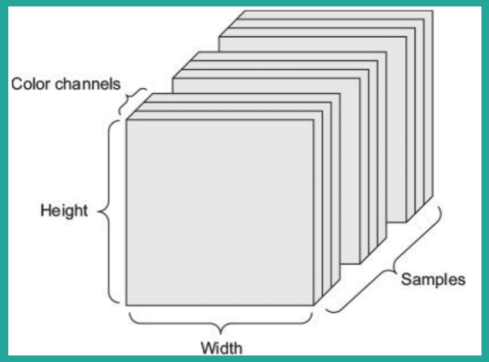
- Raw data preprocessing and feature engineering
- Network architectures: which type/topology is appropriate?
- Model hyperparameters, how to tune them?
- Optimizers
- Overfitting and how to avoid it

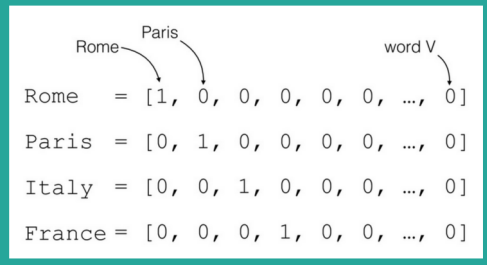

Define training data
Build neural network
Configure the learning process
Train the model
Finding/curating and then cleaning/preprocessing raw data to obtain input tensors and target tensors
Design architecture Nodes/layers and their activation functions
Choose optimizer and loss functions as well as the metrics of interest
Iterate the model on the training data.

Raw data preprocessing

- Data vectorization

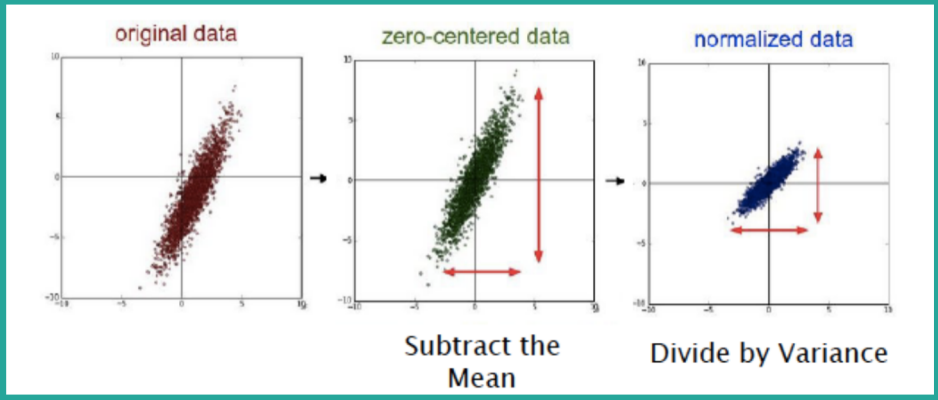
Whatever input—sound, images, text—needs to be turned into tensors In an image, pixel values
How about text? One-hot encoding

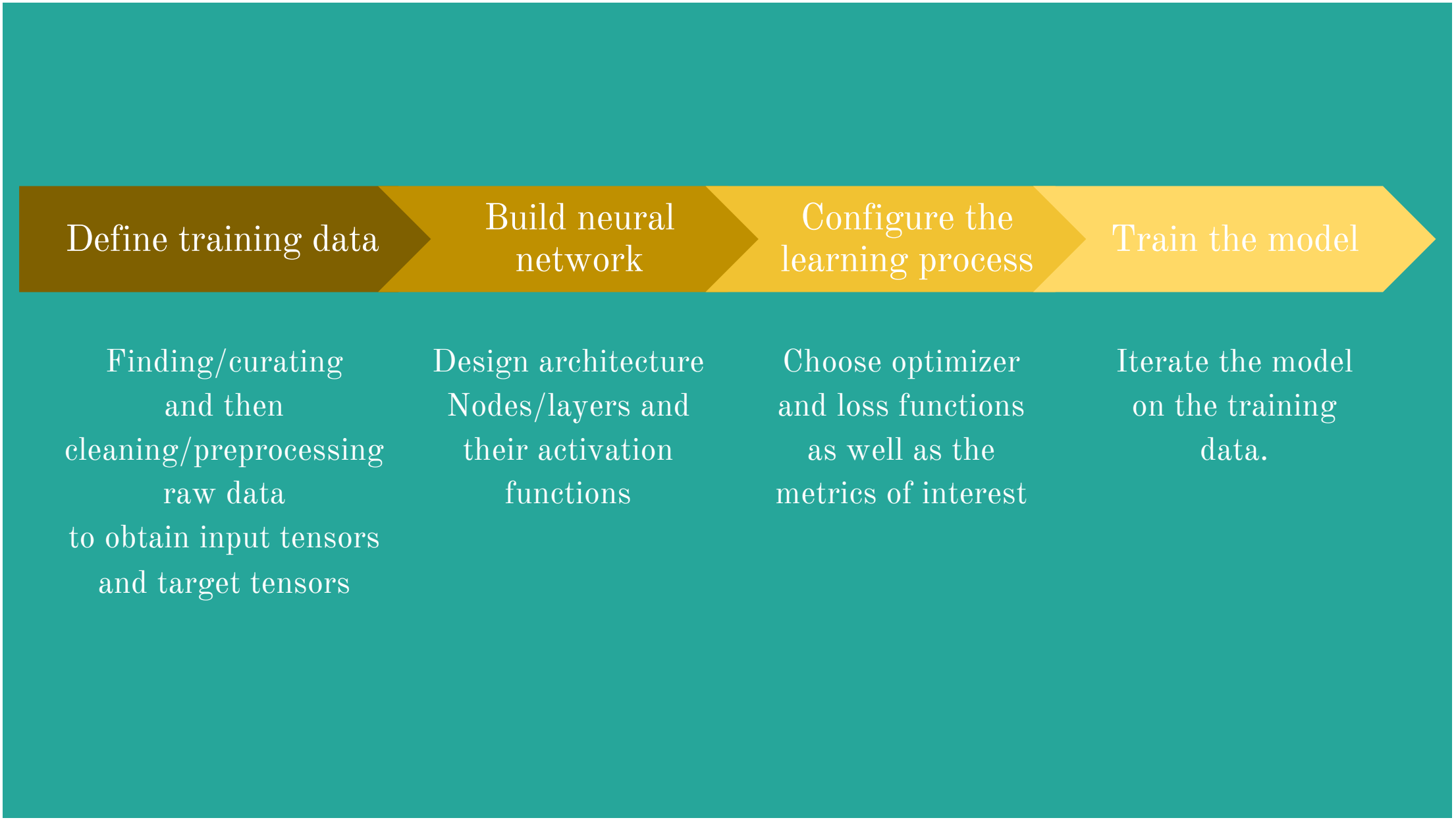




- Data normalization

The tensors must contain floating-point data




Define training data
Build neural network
Configure the learning process
Train the model
Finding/curating and then cleaning/preprocessing raw data to obtain input tensors and target tensors
Design architecture Nodes/layers and their activation functions
Choose optimizer and loss functions as well as the metrics of interest
Iterate the model on the training data.

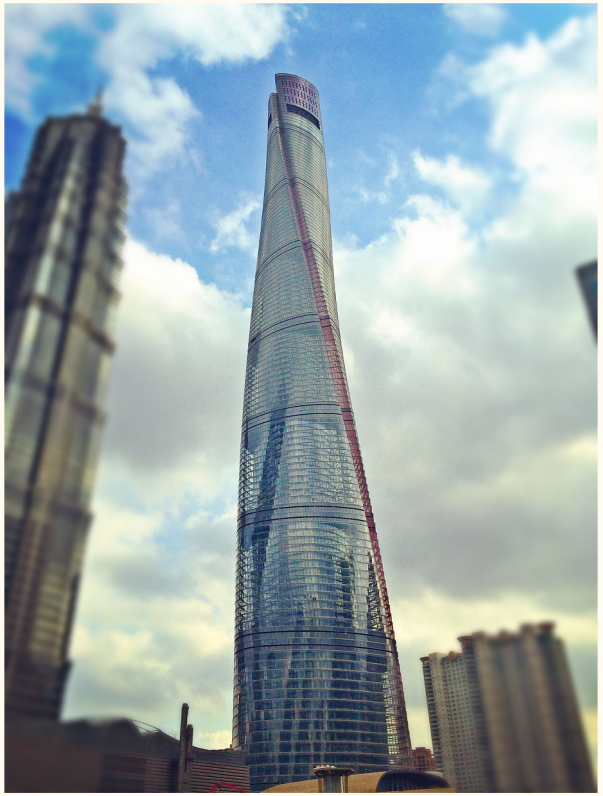
Feature engineering

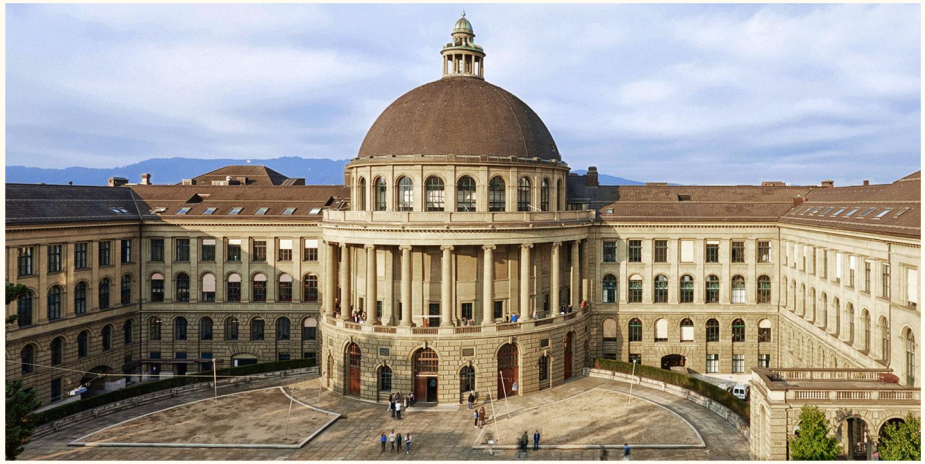
Is the preprocessing of data into more meaningful features, using our knowlegde about the data

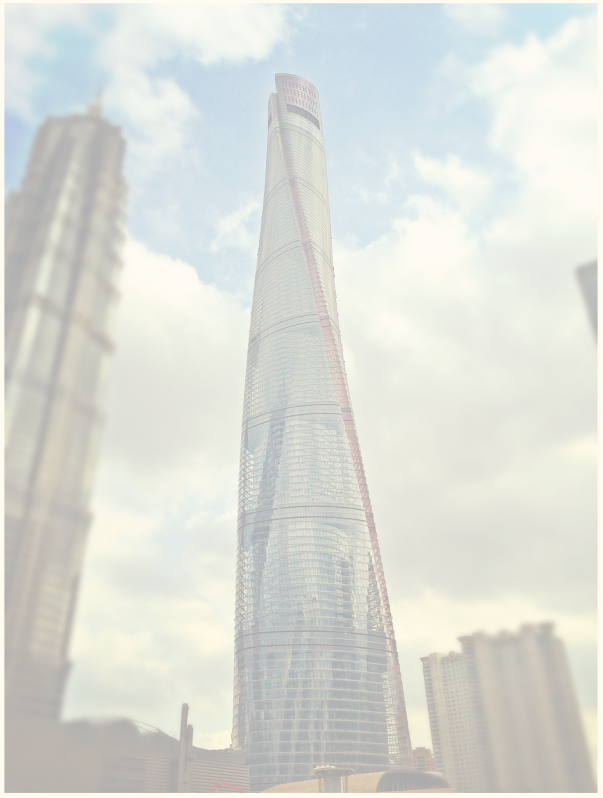
Example

Raw data: pixel grid		
Better features: clock hands' coordinates	{x1: 0.7, y1: 0.7} {x2: 0.5, y2: 0.0}	{x1: 0.0, y2: 1.0} {x2: -0.38, 2: 0.32}
Even better features: angles of clock hands	theta1: 45 theta2: 0	theta1: 90 theta2: 140

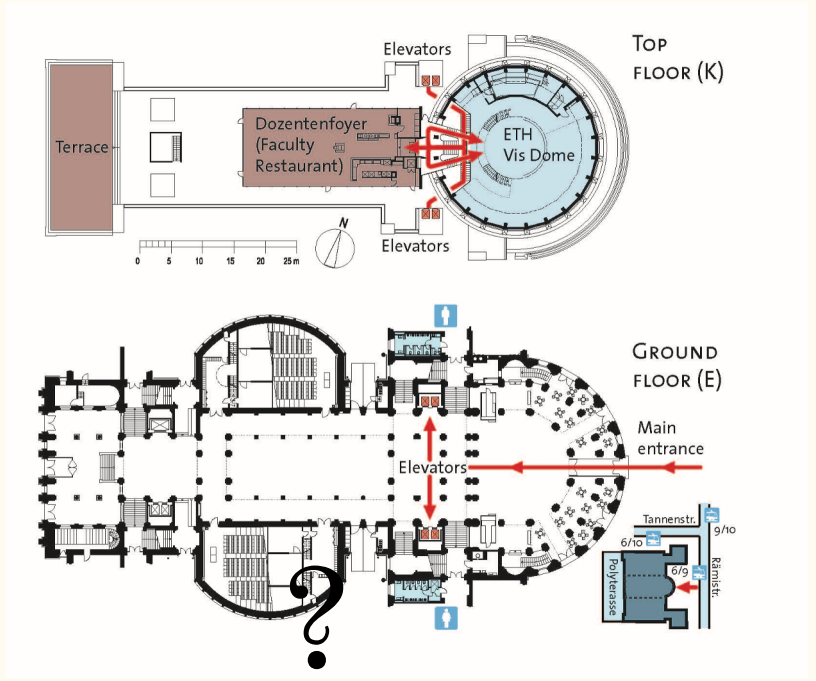
Network architectures

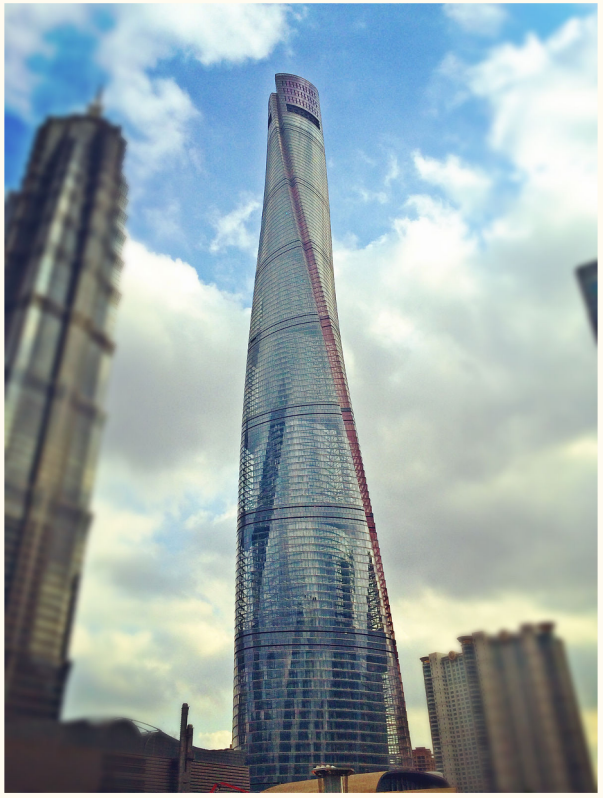


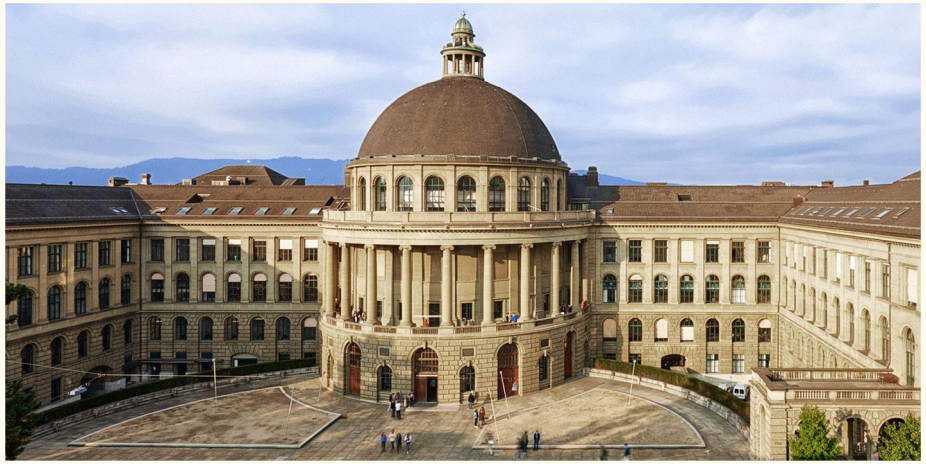


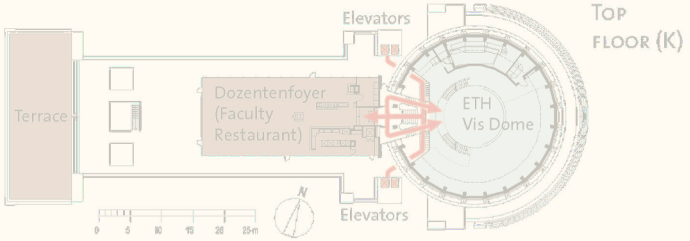


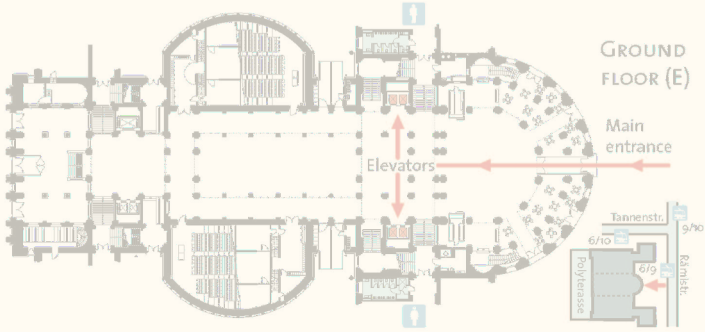
OR


Top floor (K)
Elevators
Terrace
Dozentenfoyer (Faculty Restaurant)
ETH Vis Dome
Elevators
0 5 10 15 20 25 m
Ground floor (E)
Main entrance
Elevators
Tannenstr.
9/10
6/10
6/9
Polyterasse
Rämistr.
?



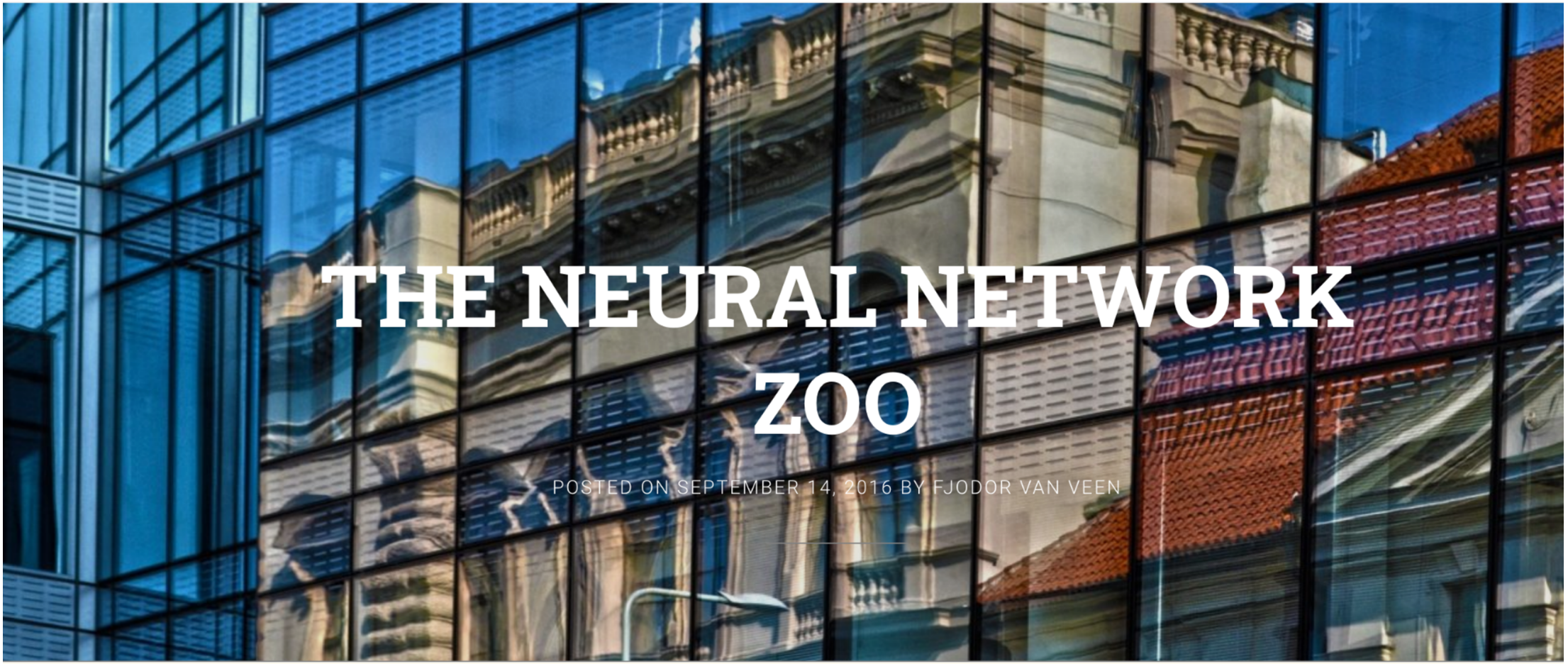






Let’s start with different model types

THE ASIMOV INSTITUTE



THE NEURAL NETWORK ZOO

POSTED ON SEPTEMBER 14, 2016 BY FJODOR VAN VEEN

https://www.asimovinstitute.org/neural-network-zoo/

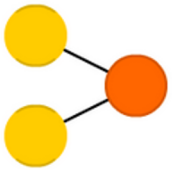
THE ASIMOV INSTITUTE

THE NEURAL NETWORK ZOO

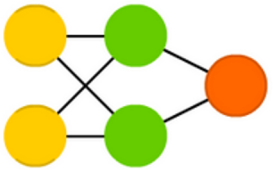
POSTED ON SEPTEMBER 14, 2016 BY FJODOR VAN VEEN

THE ASIMOV INSTITUTE

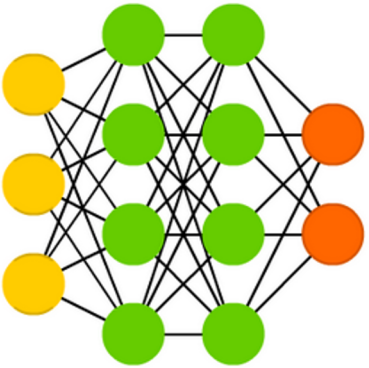
Perceptron (P)



Feed Forward (FF)



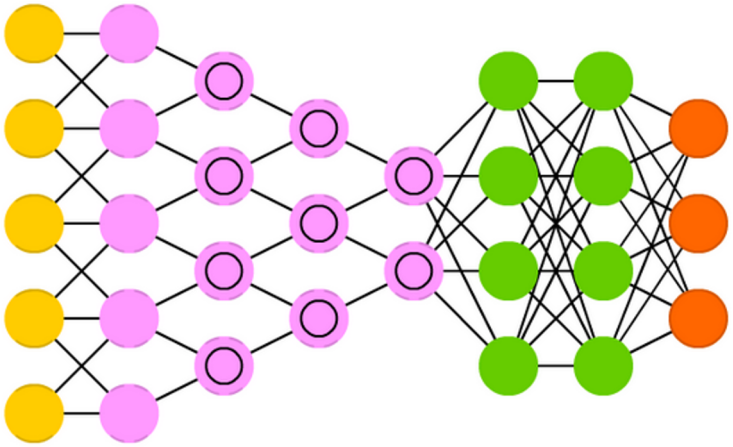
Deep Feed Forward (DFF)



THE NEURAL NETWORK ZOO

POSTED ON SEPTEMBER 14, 2016 BY FJODOR VAN VEEN

Deep Convolutional Network (DCN)

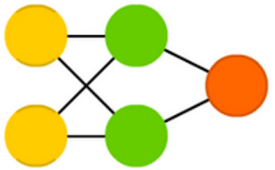


THE ASIMOV INSTITUTE

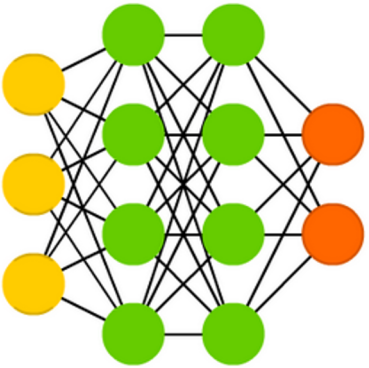
THE NEURAL NETWORK ZOO

Perceptron (P)

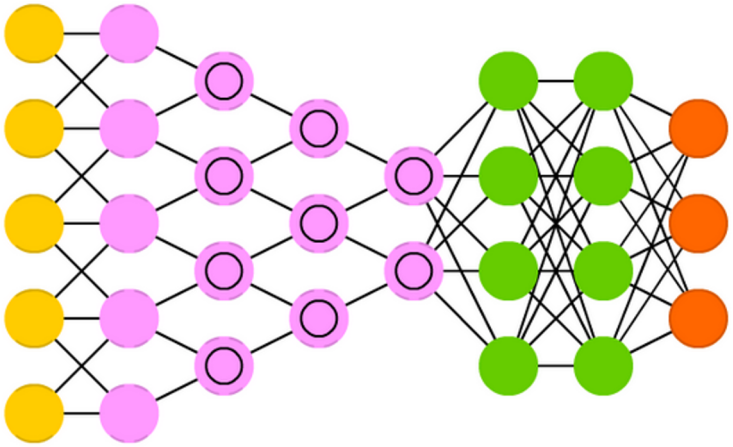
Feed Forward (FF)



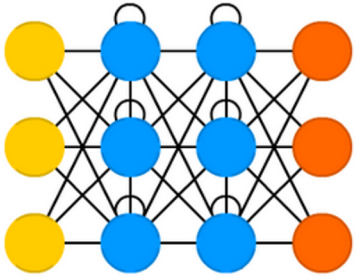
Deep Feed Forward (DFF)



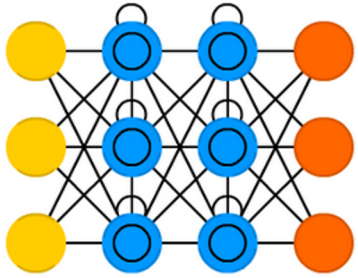
Deep Convolutional Network (DCN)



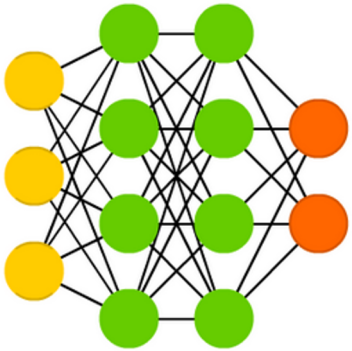
Recurrent Neural Network (RNN)



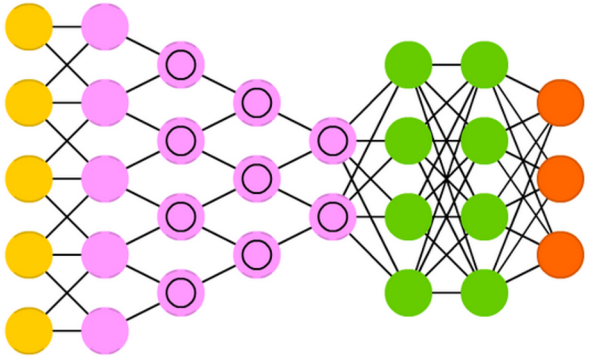
Long / Short Term Memory (LSTM)



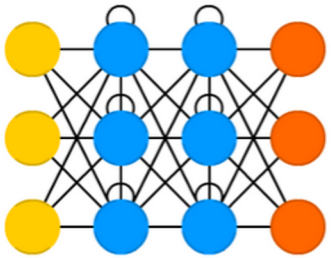
Deep Feed Forward (DFF)



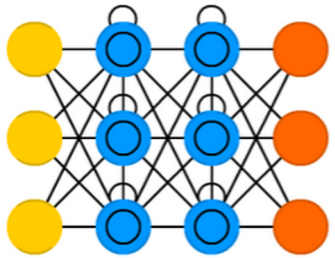
Deep Convolutional Network (DCN)

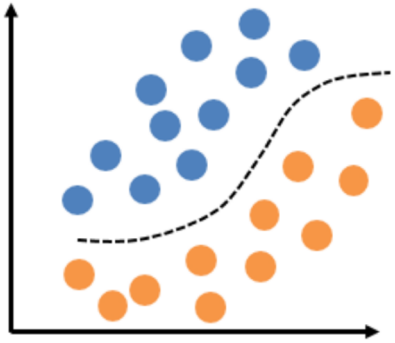


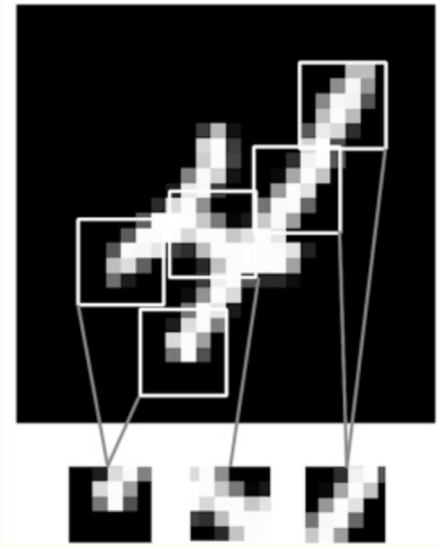
Recurrent Neural Network (RNN)

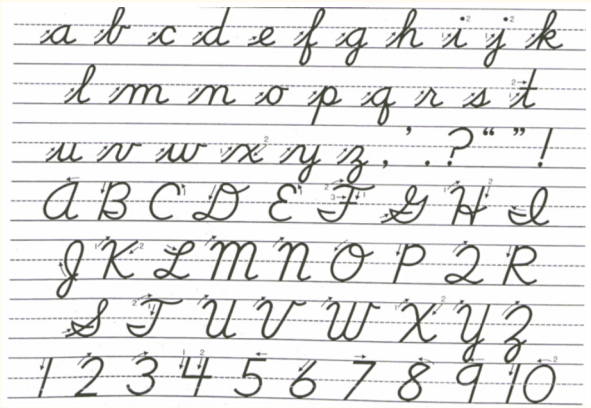


Long / Short Term Memory (LSTM)

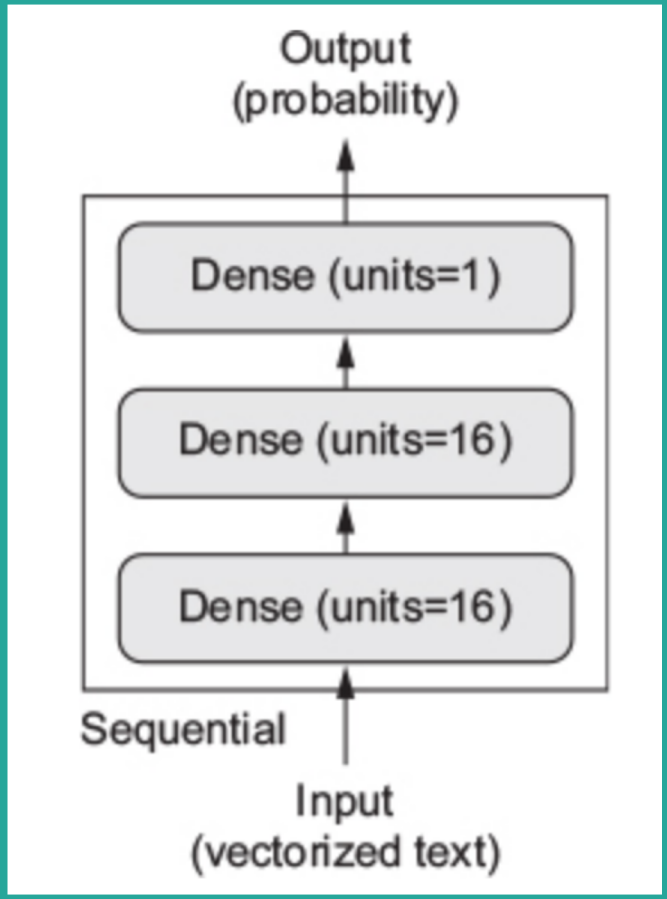




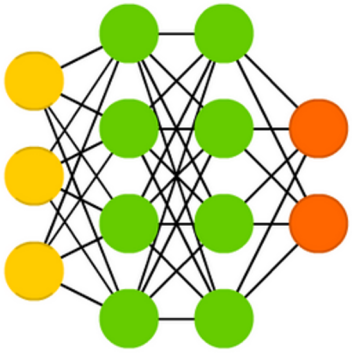




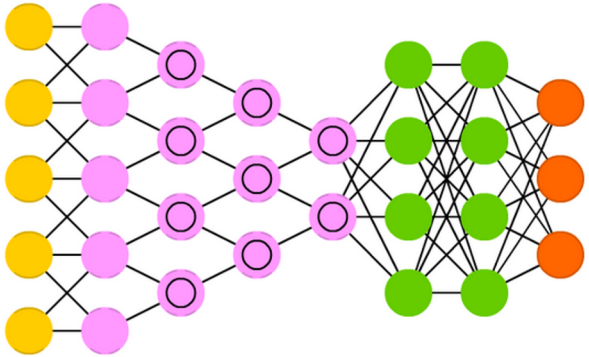
Sequential networks



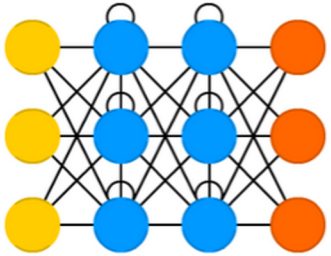
Deep Feed Forward (DFF)



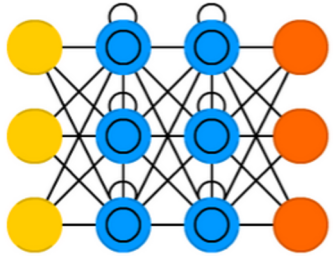
Deep Convolutional Network (DCN)



Recurrent Neural Network (RNN)



Long / Short Term Memory (LSTM)



Dense

```
keras.layers.Dense
```

Conv1D

```
keras.layers.Conv1D
```

RNN

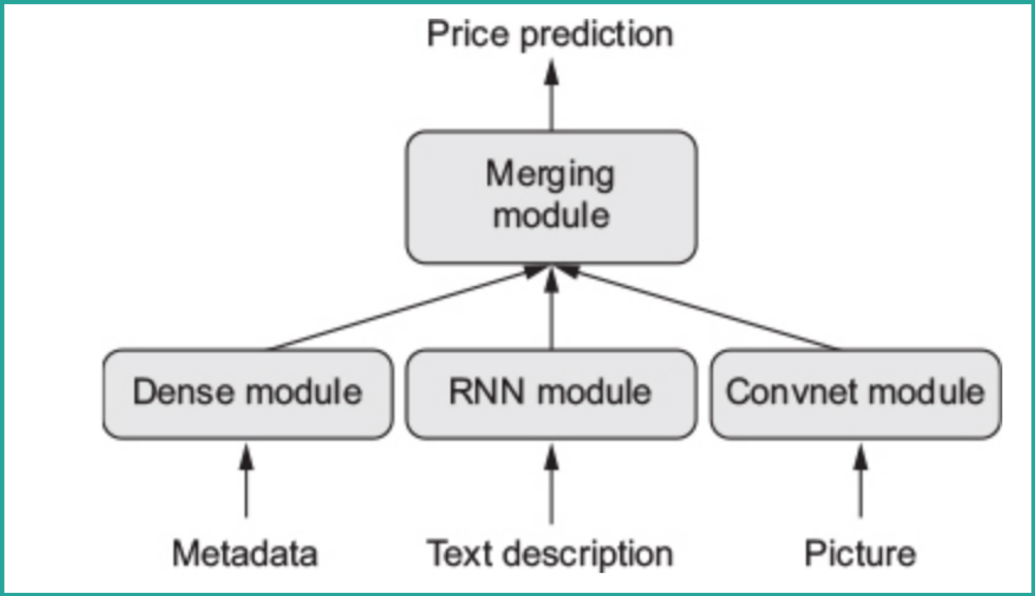
```
keras.engine.base_layer.wrapped_fn()
```

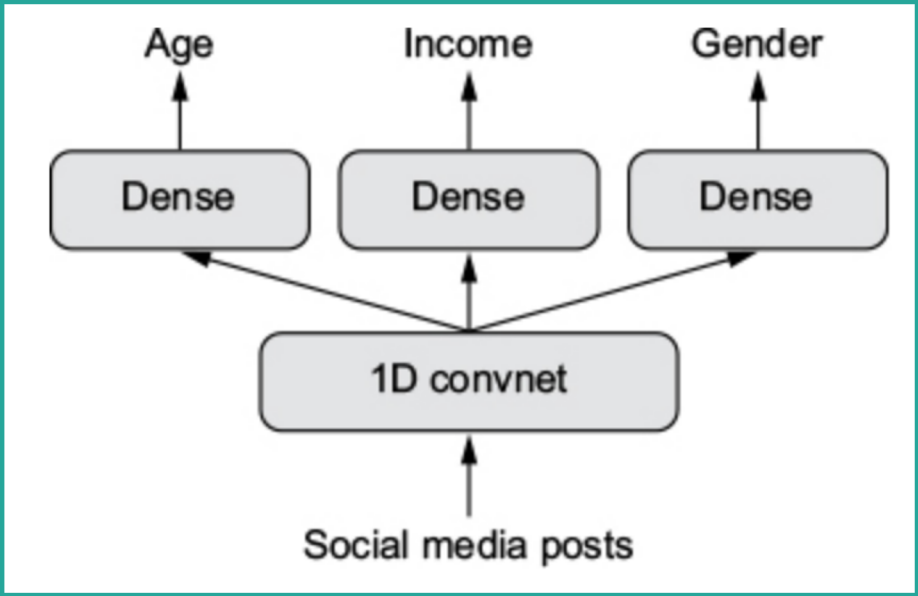
LSTM

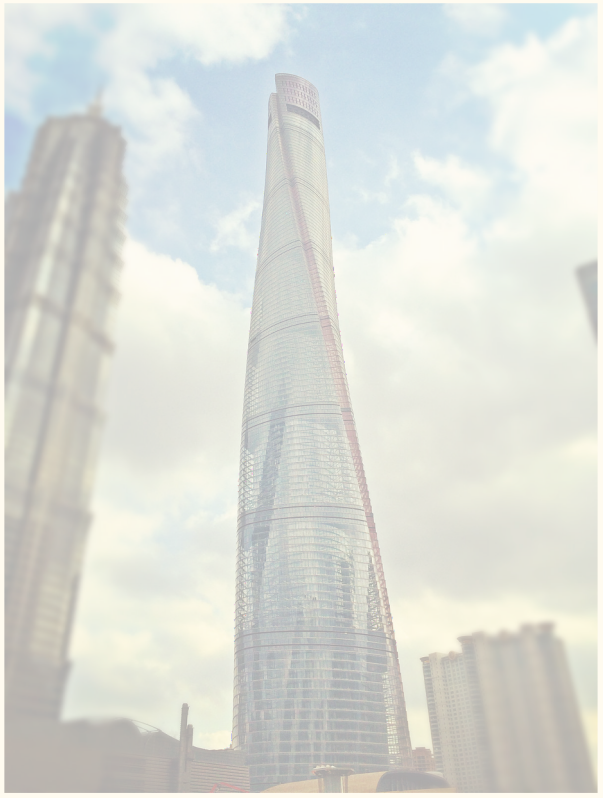
```
keras.layers.LSTM
```

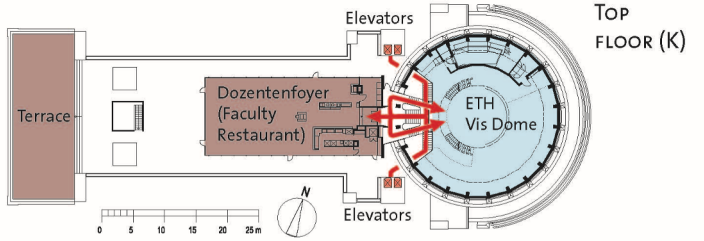
many more can be found: https://keras.io/layers/

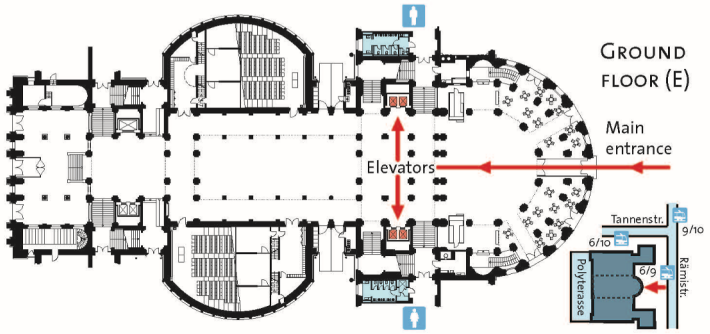
More complex networks





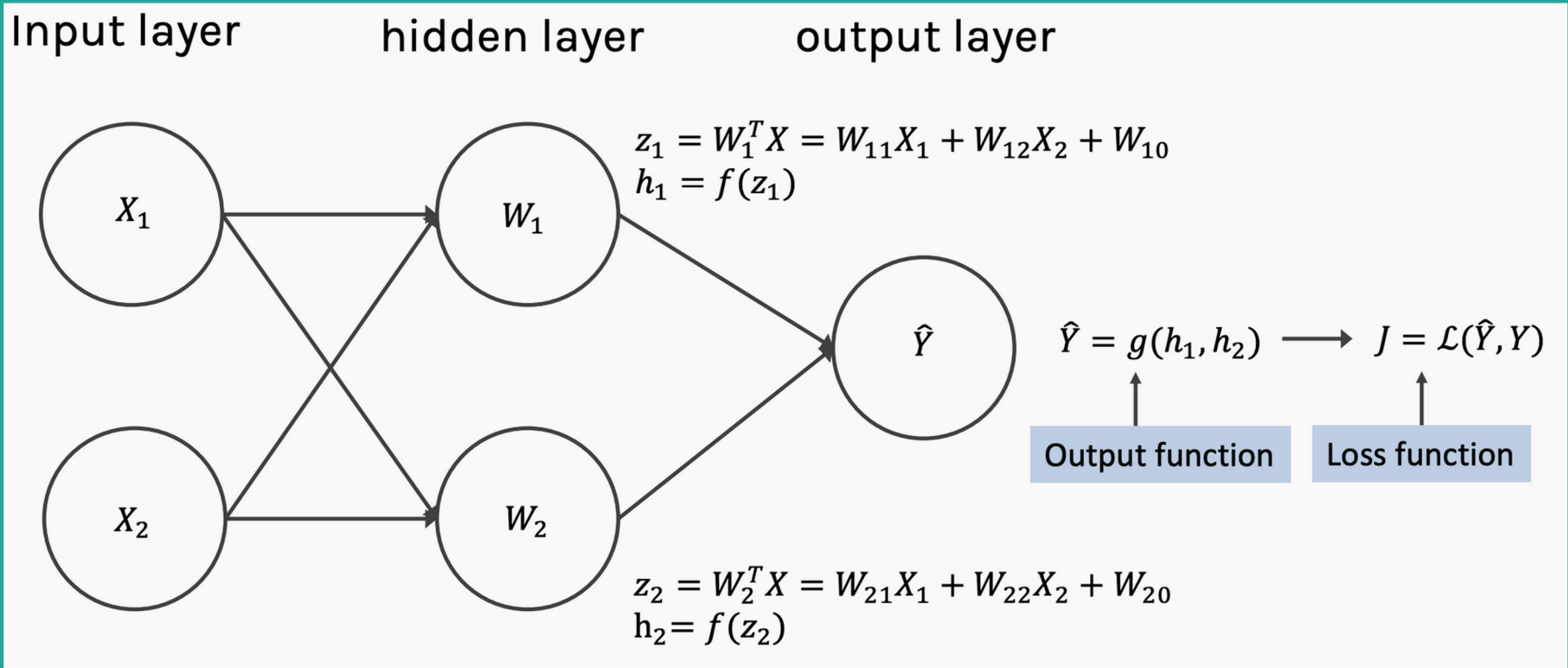






Now what about their topology?

Basic topology



A neural network with one hidden layer and two features (the simplest possible multi-layer multi-feature network).

https://towardsdatascience.com/comprehensive-introduction-to-neural-network-architecture-c08c6d8e5d98

Table 4.1. Choosing the right last-layer activation and loss function for your model

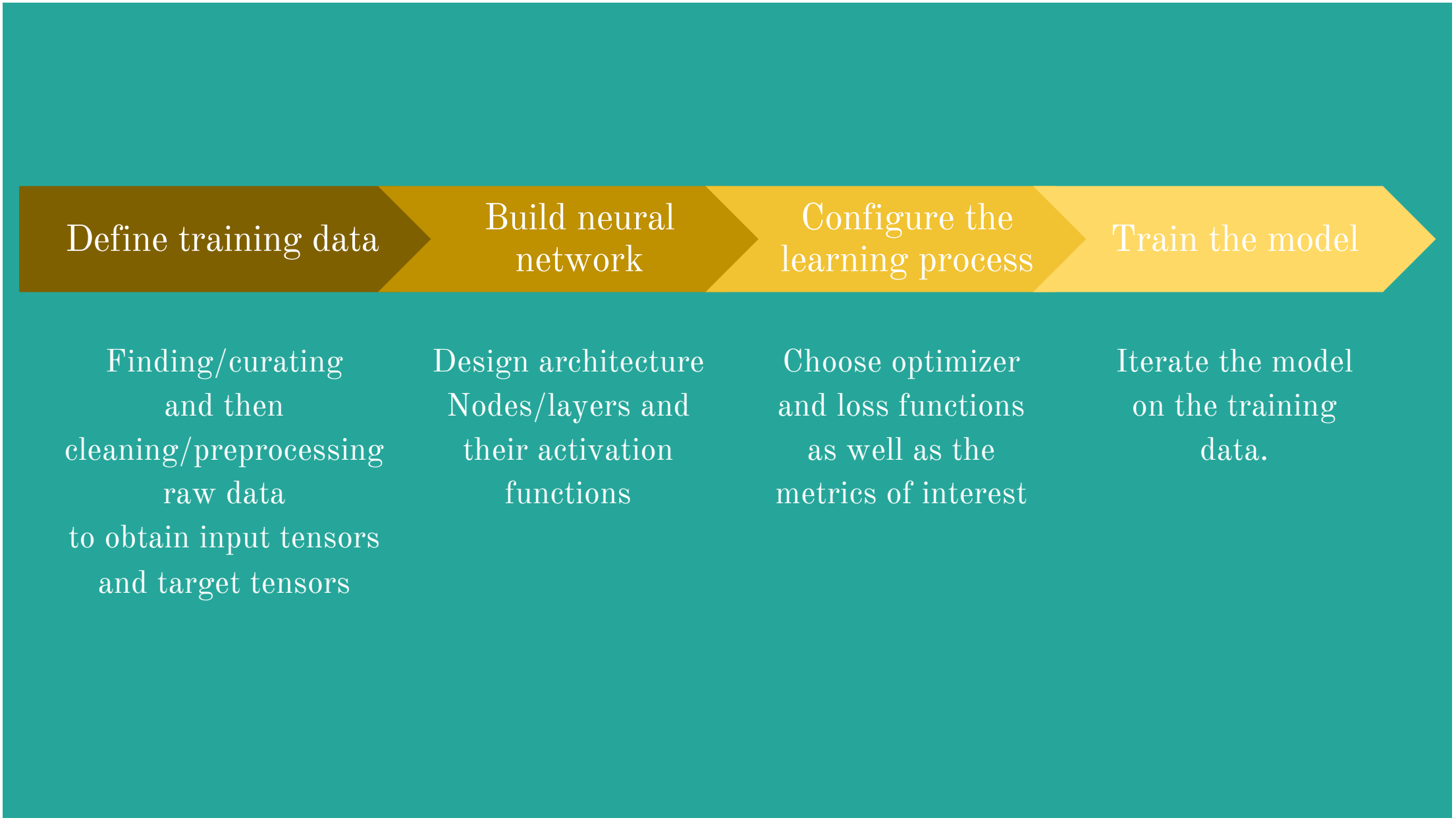
Problem type	Last-layer activation	Loss function
Binary classification	sigmoid	binary_crossentropy
Multiclass, single-label classification	softmax	categorical_crossentropy
Multiclass, multilabel classification	sigmoid	binary_crossentropy
Regression to arbitrary values	None	mse
Regression to values between 0 and 1	sigmoid	mse or binary_crossentropy

List of available activation functions: https://keras.io/activations/

Practical Learning Strategies

Hyperparameters

- Number of hidden layers
- Number of nodes
- Learning Rate
- Regularization parameters
- Momentum parameter
- Batch Size
- Number of epochs


Define training data
Build neural network
Configure the learning process
Train the model
Finding/curating and then cleaning/preprocessing raw data to obtain input tensors and target tensors
Design architecture Nodes/layers and their activation functions
Choose optimizer and loss functions as well as the metrics of interest
Iterate the model on the training data.

Define training data → Build neural network → Configure the learning process → Train the model

Finding/curating and then cleaning/preprocessing raw data to obtain input tensors and target tensors

Design architecture Nodes/layers and their activation functions

Choose optimizer and loss functions as well as the metrics of interest

Iterate the model on the training data.

```
baseline_model <-
  keras_model_sequential() %>%
  layer_dense(units = 16, activation = "relu", input_shape = 10000) %>%
  layer_dense(units = 16, activation = "relu") %>%
  layer_dense(units = 1, activation = "sigmoid")
```

- Number of hidden layers
- Number of nodes
- Regularization parameter
- Learning Rate
- Momentum parameter
- Batch Size
- Number of epochs

```
baseline_model %>% compile(
  optimizer = "adam",
  loss = "binary_crossentropy",
  metrics = c("accuracy")
)
```

Define training data	Build neural network	Configure the learning process	Train the model
Finding/curating and then cleaning/preprocessing raw data to obtain input tensors and target tensors	Design architecture Nodes/layers and their activation functions	Choose optimizer and loss functions as well as the metrics of interest	Iterate the model on the training data.

```
baseline_model <-
  keras_model_sequential() %>%
  layer_dense(units = 16, activation = "relu", input_shape = 10000) %>%
  layer_dense(units = 16, activation = "relu") %>%
  layer_dense(units = 1, activation = "sigmoid")
```

- Number of hidden layers
- Number of nodes
- Regularization parameters
- Learning Rate
- Momentum parameter
- Batch Size
- Number of epochs

```
baseline_model %>% compile(
  optimizer = "adam",
  loss = "binary_crossentropy",
  metrics = c("accuracy")
)
```

Define training data | Build neural network | Configure the learning process | Train the model

Finding/curating and then cleaning/preprocessing raw data to obtain input tensors and target tensors

Design architecture Nodes/layers and their activation functions

Choose optimizer and loss functions as well as the metrics of interest

Iterate the model on the training data.

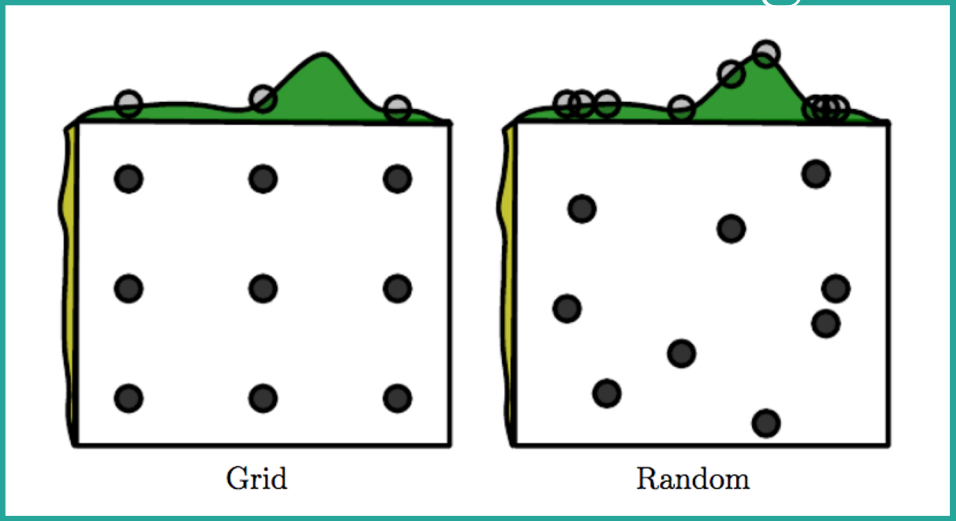
```
baseline_model <-
  keras_model_sequential() %>%
  layer_dense(units = 16, activation = "relu", input_shape = 10000) %>%
  layer_dense(units = 16, activation = "relu") %>%
  layer_dense(units = 1, activation = "sigmoid")
```

```
history <- baseline_model %>% fit(
  partial_train_data,
  partial_train_labels,
  epochs = 20,
  batch_size = 512,
  validation_data = list(val_data, val_labels),
  verbose=1
)
```

- Number of hidden layers
- Number of nodes
- Regularization parameters
- Learning Rate
- Momentum parameter
- Batch Size
- Number of epochs

How to tune hyperparameters?

- Manual tuning
- Automatic tuning

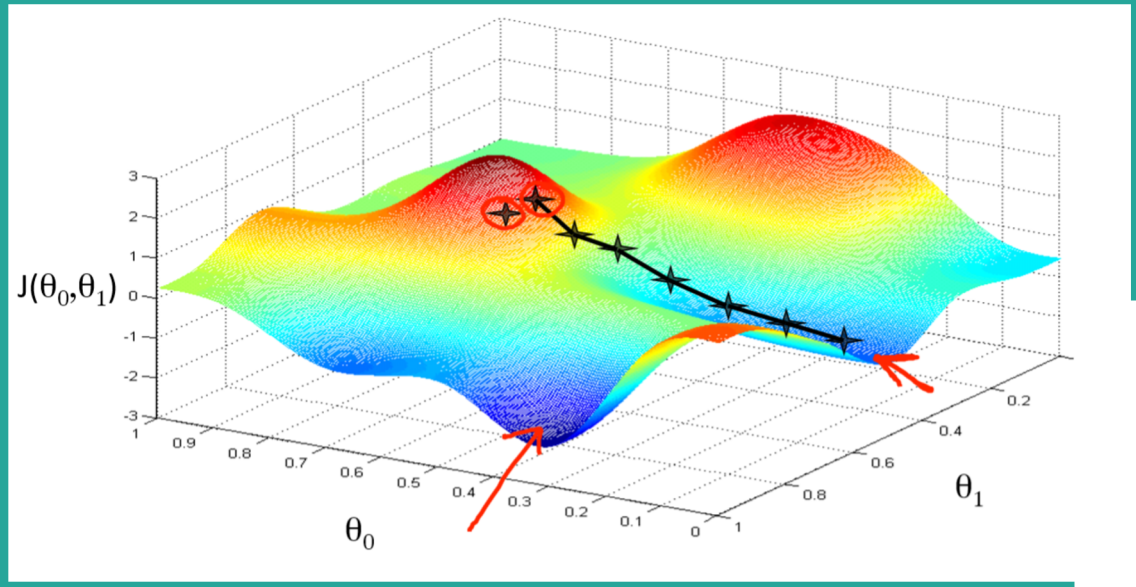


Optimizers

are defined methods of how to propagate a model (update the weights)

learning rate and momentum are parameters of the optimizer

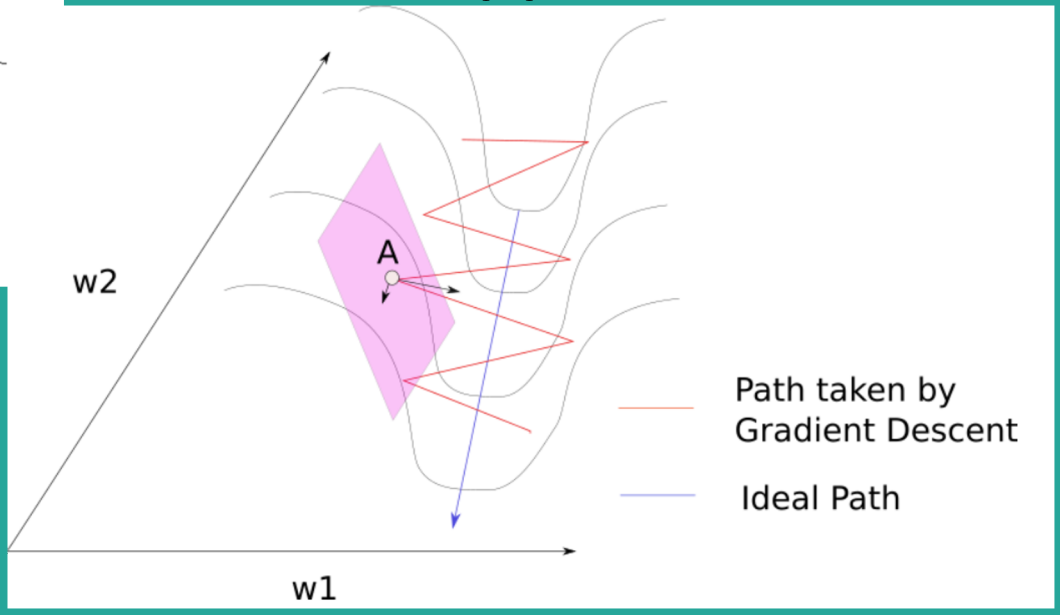
Gradient descent



https://hackernoon.com/gradient-descent-aynk-7cbe95a778da

First-order iterative optimization algorithm for finding a local minimum of a differentiable function.

https://blog.paperspace.com/intro-to-optimization-momentum-rmsprop-adam/



Optimizers: SGD, Learning rate

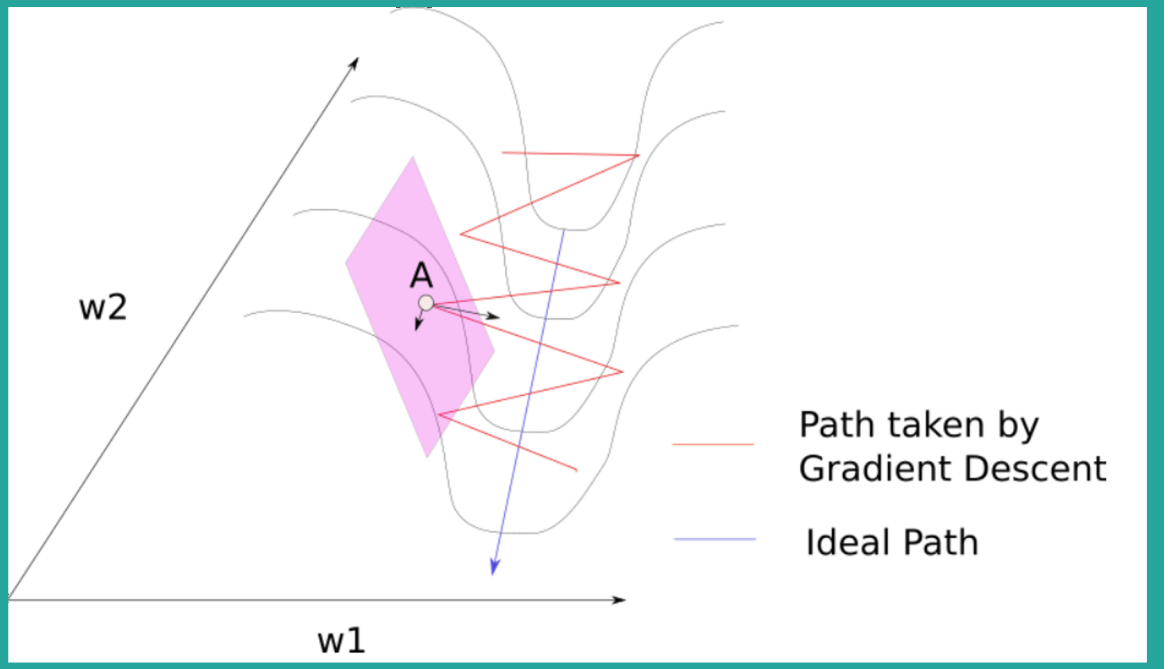
Controls how much model is changed after every update of weights

Range from 0 to 1

Step size = gradient * learning rate

Large values require less epochs for convergence and may lead to unstable results

Small values require more epochs for convergence and may lead to failure of training



https://blog.paperspace.com/intro-to-optimization-momentum-rmsprop-adam/

Optimizers: SGD, Learning rate

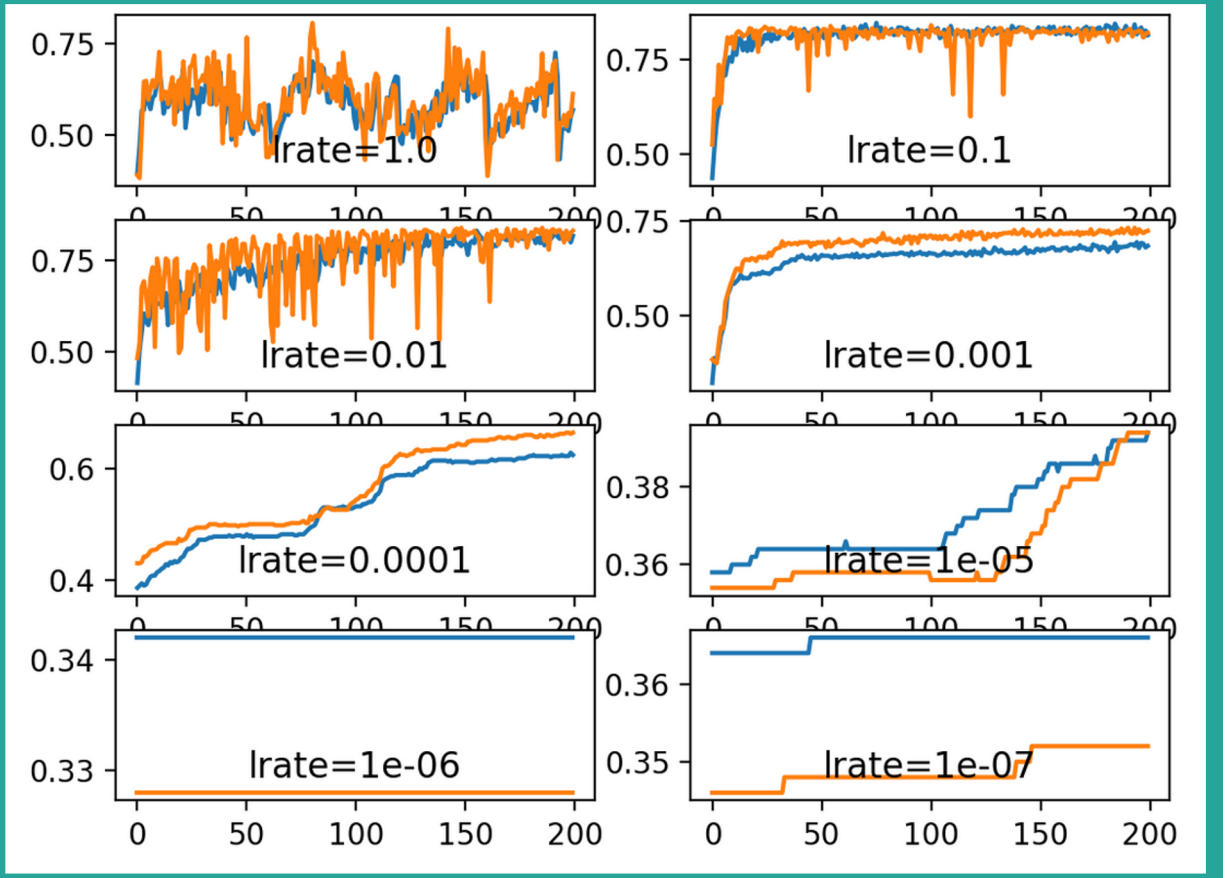
Controls how much model is changed after every update of weights

Range from 0 to 1

Step size = gradient * learning rate

Large values require less epochs for convergence and may lead to unstable results

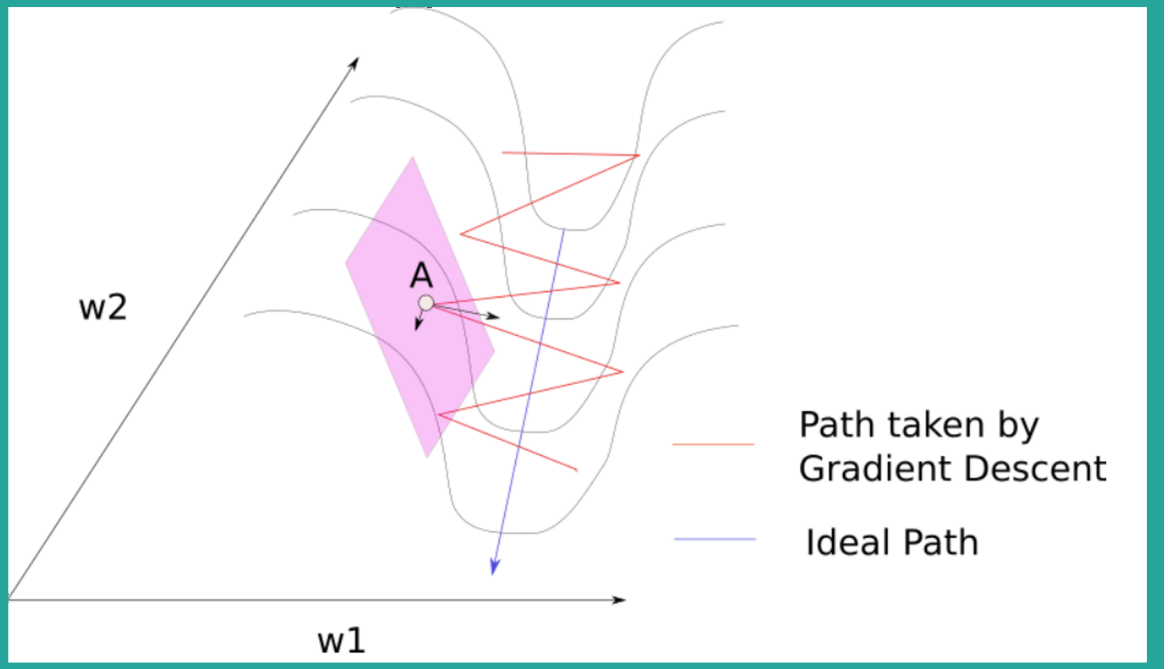
Small values require more epochs for convergence and may lead to failure of training



https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/

Optimizers: SGD, Momentum

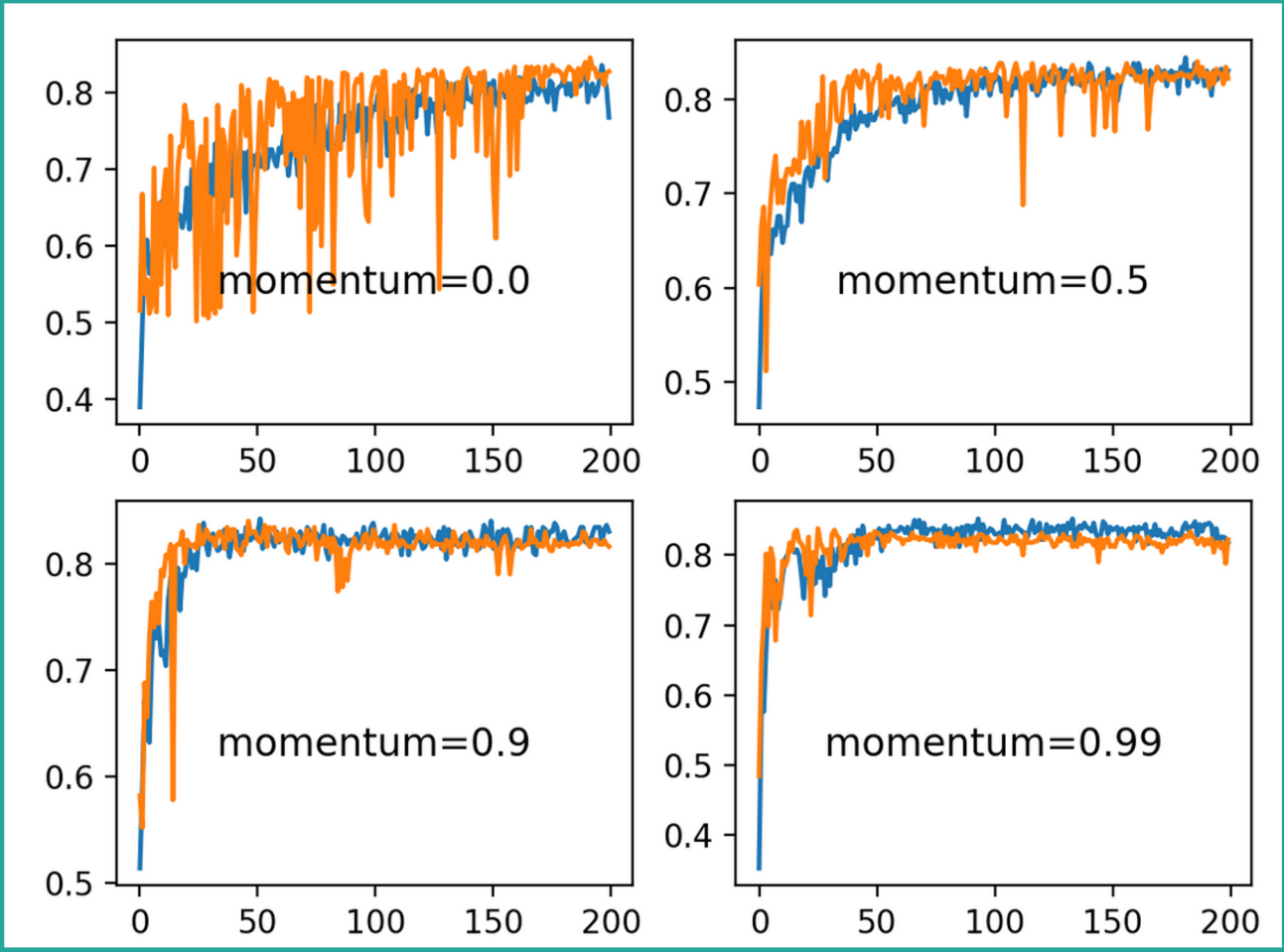
Accelerates learning in direction of minimum by taking into account the last epochs



https://blog.paperspace.com/intro-to-optimization-momentum-rmsprop-adam/

Optimizers: SGD, Momentum

Accelerates learning in direction of minimum by taking into account the last epochs



https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/

But: use adaptive optimizers!

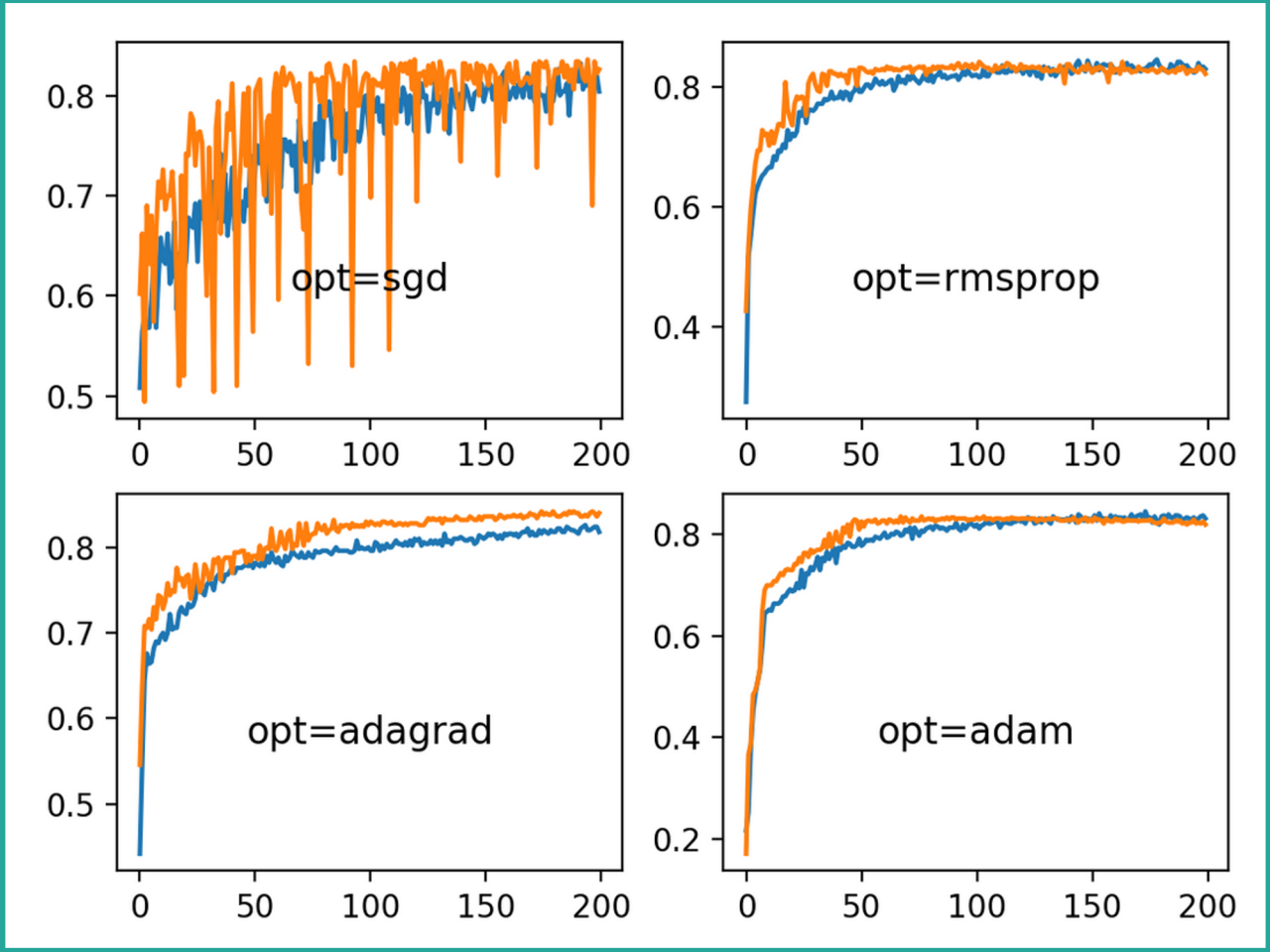
Variable learning rate

RMSprop:
Root Mean Square propagation

ADAM

and a lot more...

https://keras.io/optimizers/



https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/

Optimization vs Generalization

Concepts

Overfitting and Underfitting

Practical demo & exercise on limiting overfitting

Definitions

- Optimized model

Fits exceptionally well to training data but fares poorly on validation data and never-before-seen test data

→ This is a model which **overfits**

Conceptually, model is able to predict training data based on inherent patterns of the training data

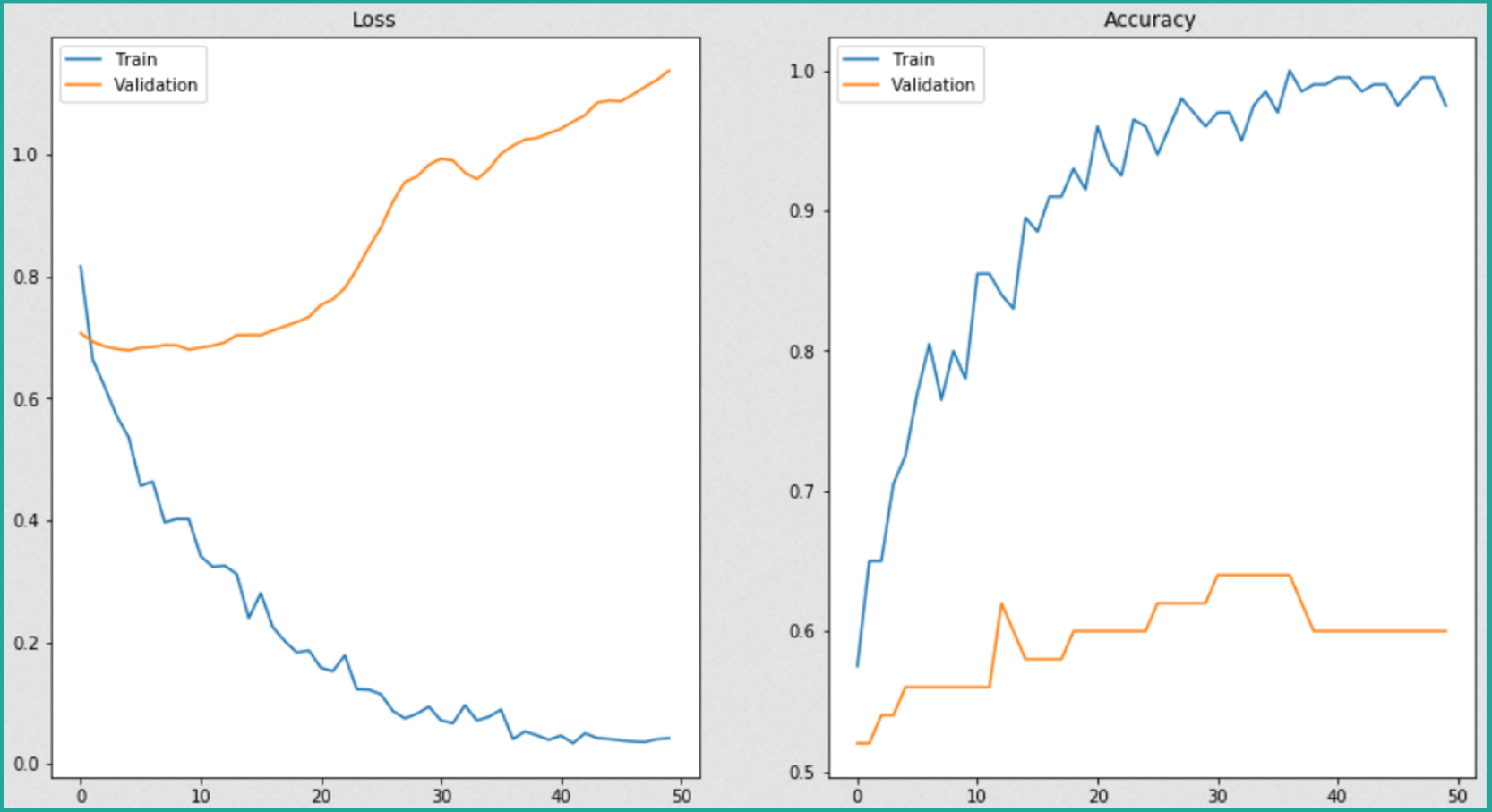
- Generalizable model

Is able to predict new test data successfully based on reasonable amount of training

If a model is kept too general, it may fare worse on the training data than on the validation data

→ This is a model which **underfits**

How to visualize overfitting?

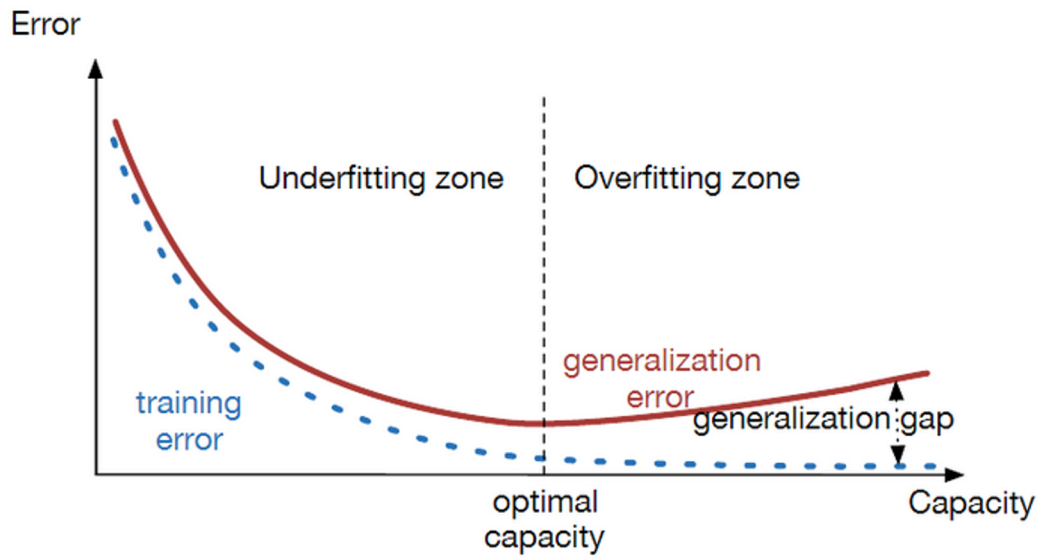


https://towardsdatascience.com/dont-overfit-how-to-prevent-overfitting-in-your-deep-learning-models-63274e552323

Cheatsheet ;)

http://www.deeplearningbook.org/
chapter 11, page 426

Hyperparameter	Increases capacity when...	Reason	Caveats
Number of hidden units	increased	Increasing the number of hidden units increases the representational capacity of the model.	Increasing the number of hidden units increases both the time and memory cost of essentially every operation on the model.
Learning rate	tuned optimally	An improper learning rate, whether too high or too low, results in a model with low effective capacity due to optimization failure.	
Weight decay coefficient	decreased	Decreasing the weight decay coefficient frees the model parameters to become larger.	
Dropout rate	decreased	Dropping units less often gives the units more opportunities to "conspire" with each other to fit the training set.	



How to avoid overfitting?

- Get more training data
- Reduce network capacity
- Dropout
- Stopping rules
- Regularization

Demo overfitting in R

Exercises: hyperparameters tuning

References

- http://www.deeplearningbook.org (Chapter 11)
- https://livebook.manning.com/book/deep-learning-with-r (Chapter 3 and 4)
- https://srdas.github.io/DLBook (Training Neural Networks, Part 1, 2 &3)
- https://www.asimovinstitute.org/neural-network-zoo/
- https://keras.rstudio.com/articles/tutorial_overfit_underfit.html
- https://blog.paperspace.com/intro-to-optimization-momentum-rmsprop-adam/
- https://towardsdatascience.com/dont-overfit-how-to-prevent-overfitting-in-your-deep-learning-models-63274e552323
- https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/