# Machine Learning for Marine Scientists

## Part 1: Introduction

28.09.2021

**Frauke Albrecht, Caroline Arnold, Jakob Lüttgau, Tobias Weigel**

Helmholtz AI Consultants for Earth and Environment

# Logistics

- Virtual room
  - Please share your cameras :-)
  - Please post questions in the shared google doc
  - Breakouts for the tutorial part in the afternoon
- Timeline for today
  - 10:00-12:00 - Lecture
  - 13:00-15:15 - Tutorial with jupyterlab

- Your trainers for this course:
  - Frauke Albrecht
  - Caroline Arnold
  - Jakob Lüttgau
  - Tobias Weigel

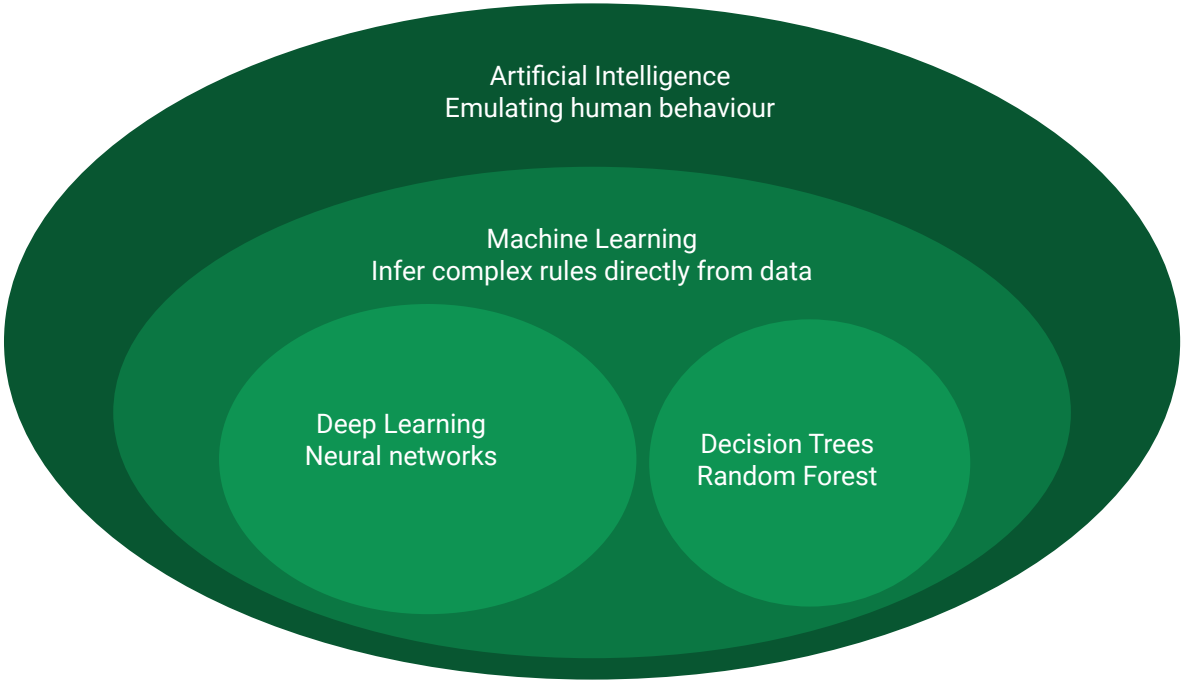
- If you need help after the course with an individual project, contact us
- Also, in need of further training - contact Enno/us

# Artificial intelligence and machine learning

Standard programming: explicit set of rules → algorithm

```
if stock > 5:
   lower_price()
else:
   raise_price()
```

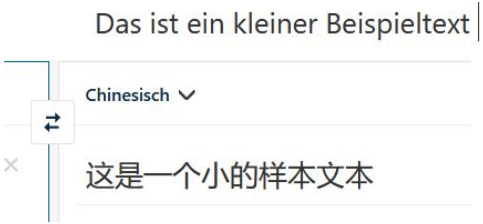
Artificial intelligence: no given set of rules, algorithm “learns” on its own



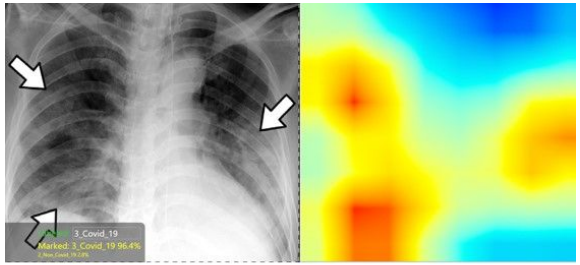
# Machine learning examples

Earth system & climate science

Sequence to sequence translation




Medical scans



Automated satellite image analysis

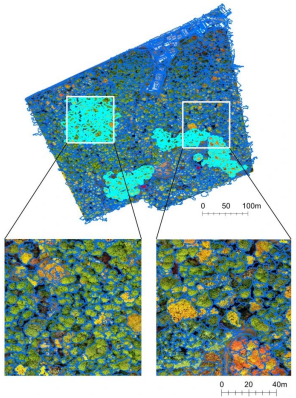
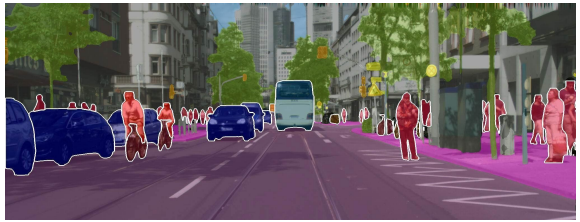
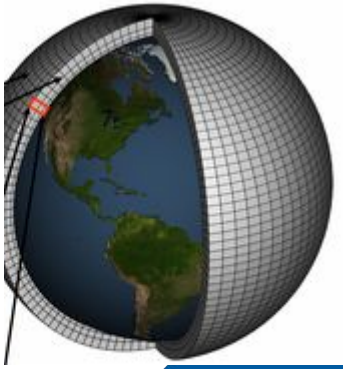


Image segmentation for autonomous driving





Improving climate models

# Successful machine learning

Data

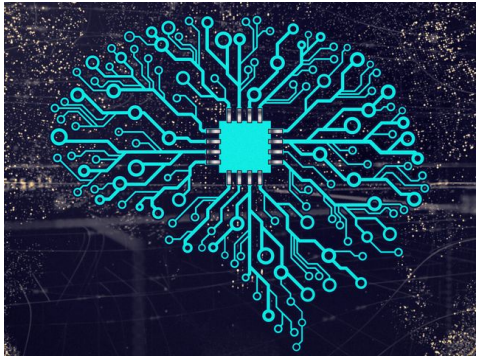
- Most important resource
- High quality
- From simulations, observations, benchmark datasets, ...

Model

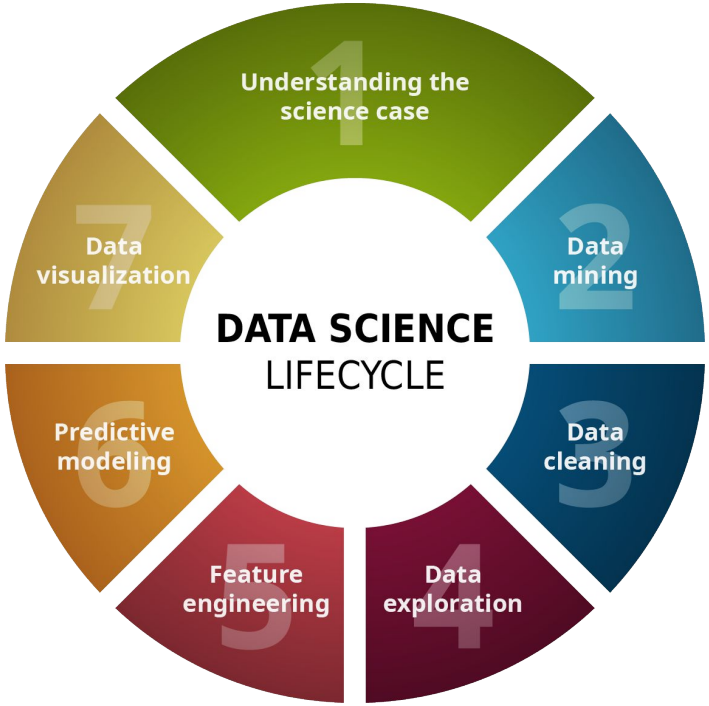
- Algorithm that can learn from data
- Programmed and trained

Hardware

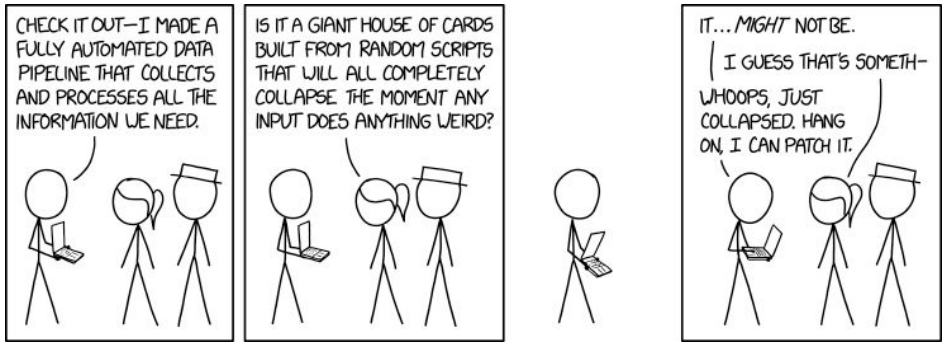
- ML often uses GPUs
- Efficient for the typical calculations
- Depends on your problem size / dataset



# The workflow is more than “just the ML model”



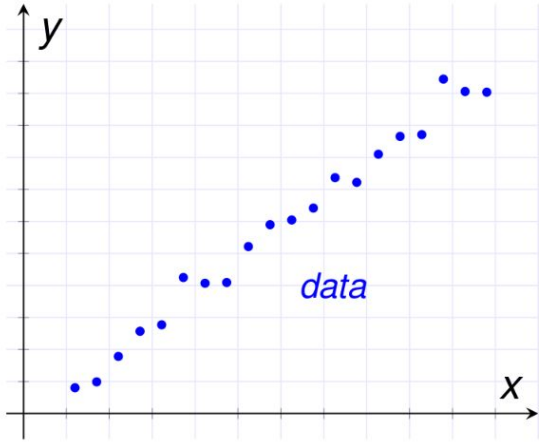

- All steps in the workflow are relevant
- You will likely iterate the whole workflow many times until you reach a satisfying solution




https://xkcd.com/2054/

# Linear Regression

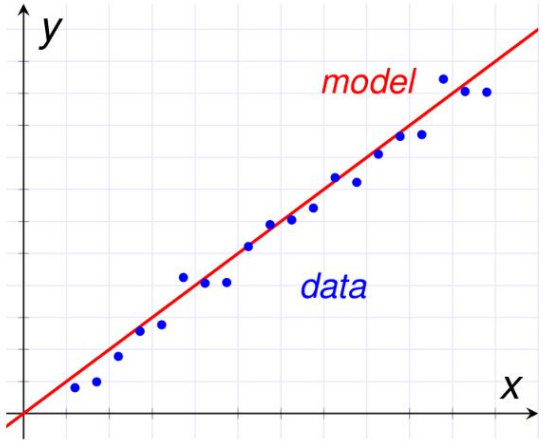
Slides by Dr. Markus Götz (KIT)




- **Data set:** $\{samples, labels\} = \{x, y\}$

# Linear Regression

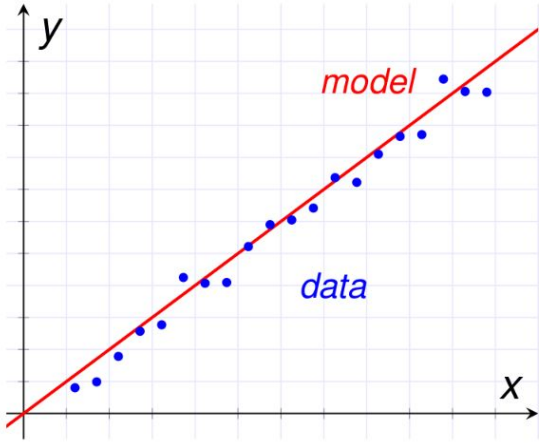
Slides by Dr. Markus Götz (KIT)




- **Data set:** $\{samples, labels\} = \{x, y\}$
- **Model:** definition $\hat{y} = wx + b$ with $w$ and $b$ trainable parameters

# Linear Regression

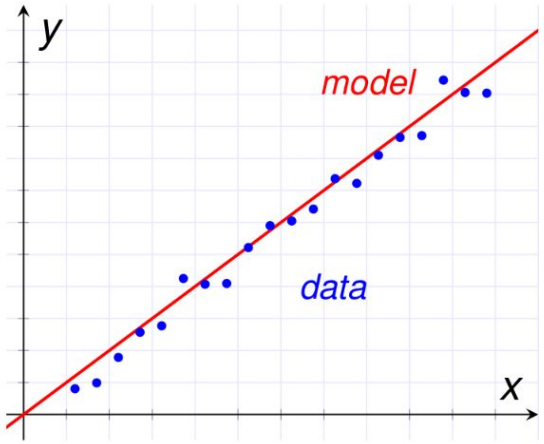
Slides by Dr. Markus Götz (KIT)




- **Data set:** $\{samples, labels\} = \{x, y\}$
- **Model:** definition $\hat{y} = wx + b$ with $w$ and $b$ trainable parameters
- **Loss function:** or cost/error/objective $J(w, b) = MSE(w, b) = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$

# Linear Regression

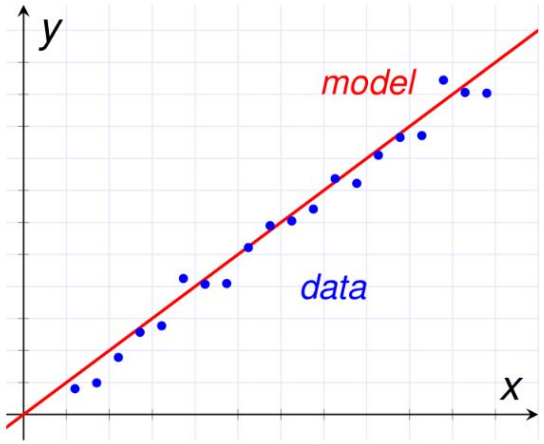
Slides by Dr. Markus Götz (KIT)




- **Data set:** $\{samples, labels\} = \{x, y\}$
- **Model:** definition $\hat{y} = wx + b$
  with $w$ and $b$ trainable parameters
- **Loss function:** or cost/error/objective
  $J(w, b) = MSE(w, b) = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$
- **Train:** the model, e.g. optimization
  $\hat{w}, \hat{b} = \arg\min J(w, b)$

# Linear Regression

Slides by Dr. Markus Götz (KIT)




- **Data set:** $\{samples, labels\} = \{x, y\}$
- **Model:** definition $\hat{y} = wx + b$ with $w$ and $b$ trainable parameters
- **Loss function:** or cost/error/objective $J(w, b) = MSE(w, b) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$
- **Train:** the model, e.g. optimization $\hat{w}, \hat{b} = \arg\min J(w, b)$

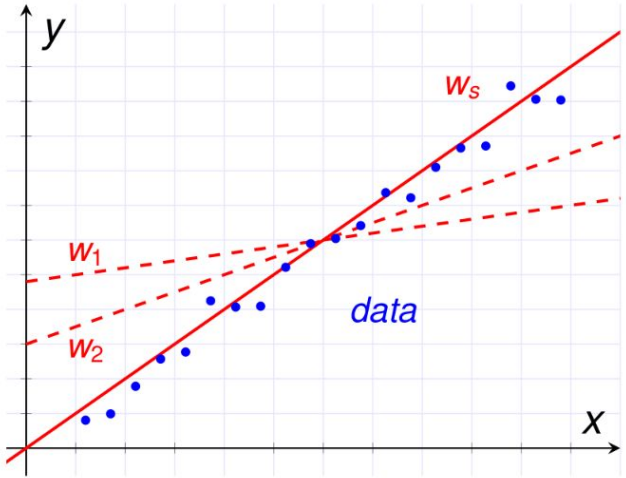
- **Basic recipe for most machine learning algorithms**

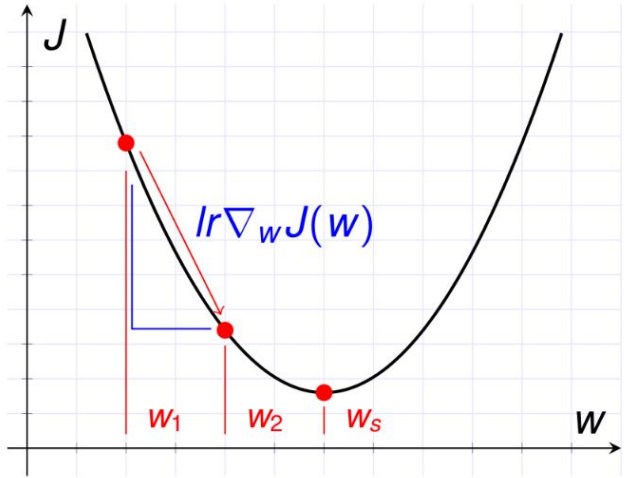
# Optimization: Gradient Descent

Slides by Dr. Markus Götz (KIT)

- Iterative optimization technique, weight update in direction of negative gradient

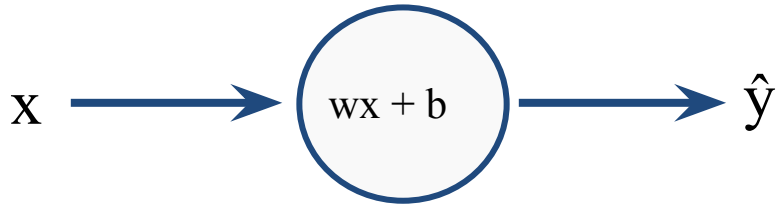
$$w_{i+1} = w_i - lr \nabla_{w_i} J(w_i)$$




- $lr$ is learning rate, gradient update factor
- **Stochastic gradient descent (SGD)**, sample subset (**batch**) updates

# Linear Regression - Visualization



# Linear Regression in Python

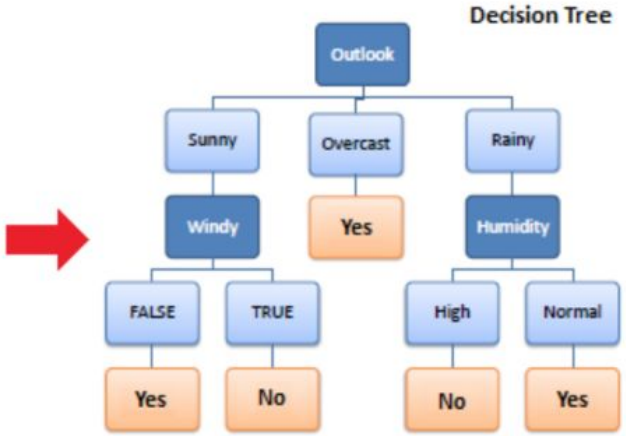
```
>>> from sklearn.linear_model import LinearRegression
>>> linReg = LinearRegression()
>>> reg = linReg.fit(X, y)
>>> w = reg.coef_
>>> b = reg.intercept_
```

# Random Forests - Decision Trees

- Random Forests are based on Decision Trees
- Tree-like model and diagram to find decisions
- Decision Trees can be used for regression and classification

Predictors | Target

| Outlook | Temp | Humidity | Windy | Play Golf |
|---|---|---|---|---|
| Rainy | Hot | High | False | No |
| Rainy | Hot | High | True | No |
| Overcast | Hot | High | False | Yes |
| Sunny | Mild | High | False | Yes |
| Sunny | Cool | Normal | False | Yes |
| Sunny | Cool | Normal | True | No |
| Overcast | Cool | Normal | True | Yes |
| Rainy | Mild | High | False | No |
| Rainy | Cool | Normal | False | Yes |
| Sunny | Mild | Normal | False | Yes |
| Rainy | Mild | Normal | True | Yes |
| Overcast | Mild | High | True | Yes |
| Overcast | Hot | Normal | False | Yes |
| Sunny | Mild | High | True | No |




Source: https://www.saedsayad.com/decision_tree.htm

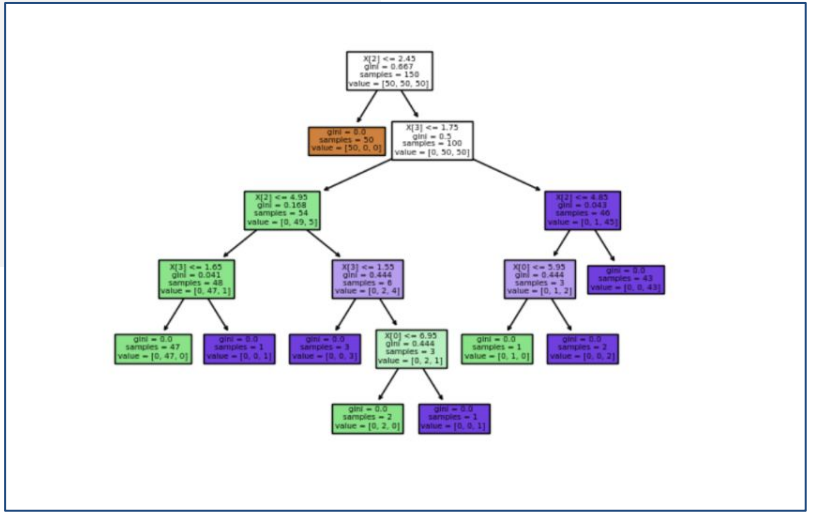
# Random Forests - Decision Trees in Python

```
>>> from sklearn.dataset import load_iris
>>> from sklearn import tree
>>> X, y = iris.data, iris.target
>>> clf = tree.DecisionTreeClassifier()
>>> clf = clf.fit(X, y)
>>> tree.plot_tree(clf)
```

Source: https://scikit-learn.org/stable/modules/tree.html#classification

# Random Forests - Decision Trees in Python

```
>>> from sklearn.dataset import load_iris
>>> from sklearn import tree
>>> X, y = iris.data, iris.target
>>> clf = tree.DecisionTreeClassifier()
>>> clf = clf.fit(X, y)
>>> tree.plot_tree(clf)
```



Source: https://scikit-learn.org/stable/modules/tree.html#classification

# Random Forests - Decision Trees in Python

```
>>> from sklearn import tree
>>> reg = tree.DecisionTreeRegressor()
>>> reg = reg.fit(X, y)
>>> reg.predict(X, y)
>>> tree.plot_tree(reg)
```

Source: https://scikit-learn.org/stable/modules/tree.html#regression

# Random Forests - Construction

A Random Forest is a set of Decision Trees:

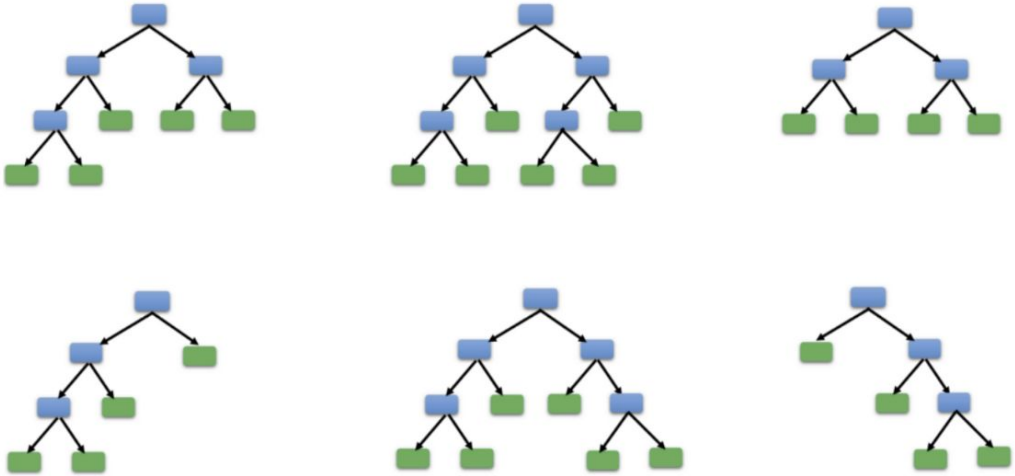
1. Create a bootstrapped dataset
   (select a subset of the dataset, multiple selection allowed)

2. Create a Decision Tree from this bootstrapped dataset
   using a random subset of variables in each split

→ **REPEAT**

# Random Forests

Using a bootstrapped sample and considering only a subset of the variables at each step results in a wide variety of trees



Source: https://towardsai.net/p/machine-learning/why-choose-random-forest-and-not-decision-trees

# Random Forests - Usage

- Classification:
  - Apply all Decision Trees created to the data
  - Count results
  - The class that got most counts is the result

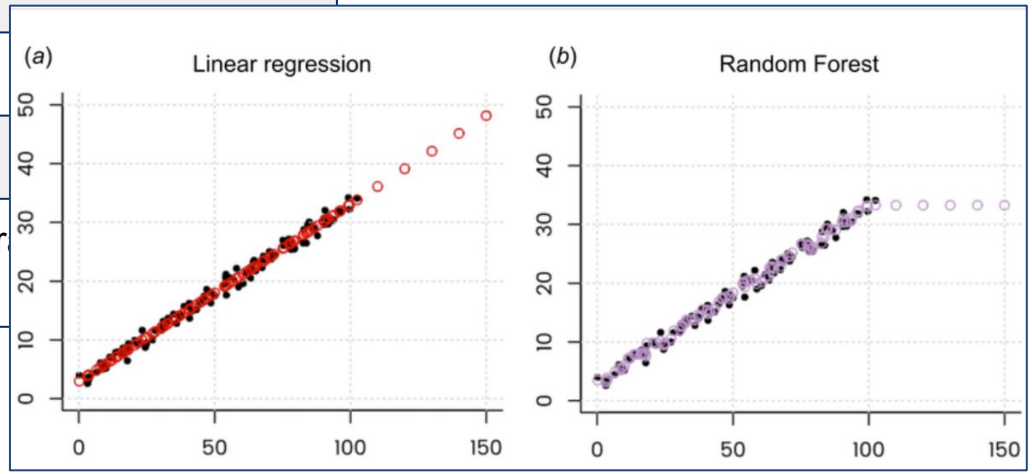
- Regression:
  - Apply all Decision Trees created to the data
  - Average results

# Random Forests - Pros & Cons

| Pro | Con |
| --- | --- |
| relatively easy to build | expensive to train |
| less prone to overfitting than Decision Trees | still prone to overfitting |
| normalization not necessary | |
| feature importances included | |
| cannot extrapolate / robust for outliers | cannot extrapolate |

# Random Forests - Pros & Cons

| Pro | Con |
| --- | --- |
| relatively easy to build | expensive to train |
| less prone to overfitting than Decision Trees | still prone to overfitting |
| normalization not necessary | |
| feature importances included | |
| cannot extrapolate / robust for outliers | cannot extr |




Source: https://www.researchgate.net/figure/Illustration-of-the-extrapolation-problem-of-Random-Forest-Even-though-Random-Forest-is_fig13_327298817

# Random Forests in Python

```
>>> from sklearn.ensemble import RandomForestClassifier
>>> clf = RandomForestClassifier()
>>> clf = clf.fit(X, y)
>>> clf.predict(X, y)
```

```
>>> from sklearn.ensemble import RandomForestRegressor
>>> reg = RandomForestRegressor()
>>> reg = reg.fit(X, y)
>>> reg.predict(X, y)
```

# Random Forests - Further Topics

- AdaBoost
  - A Random Forest, but the trees have a fixed length (**stumps**) (usually 1 node, 2 leaves -> weak learner)
  - Stumps have different weight for the final decision
  - Stumps are made by taking the previous stump's mistakes into account
- Gradient Boost
  - Starts with a single node
  - Builds trees influenced by the previous errors (predict residuals)
  - Trees are in general larger than a stump, but with restricted size
- XGBoost
  - Based on Gradient Boost, but differences in modelling, especially more regularization
  - Faster than Gradient Boost

# Further Sources

Linear Regression with PyTorch:

- https://www.youtube.com/watch?v=YAJ5XBwlN4o

Gradient Descent

- https://en.wikipedia.org/wiki/Gradient_descent

Decision Trees

- https://youtu.be/7VeUPuFGJHk
- https://youtu.be/jVh5NA9ERDA

Random Forests

- https://youtu.be/J4Wdy0Wc_xQ
- https://youtu.be/nyxTdL_4Q-Q (Missing Data)

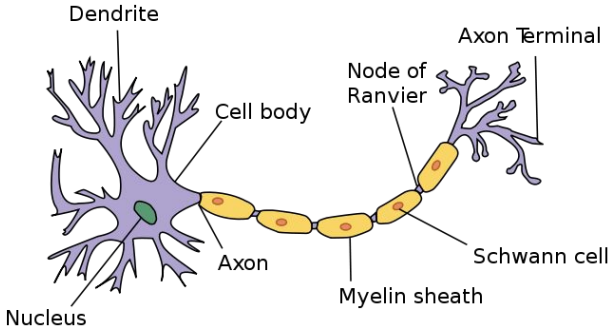
Further Topics

- https://youtu.be/LsK-xG1cLYA (AdaBoost)
- https://youtu.be/3CC4N4z3GJc (Gradient Boost)
- https://youtu.be/OtD8wVaFm6E (XGBoost)

# BREAK (10 min)

# Artificial Neural Networks

- Cullock & Pitt 1943
- Perceptron - “neuron”
  - React to inputs
  - Send outputs to next neuron

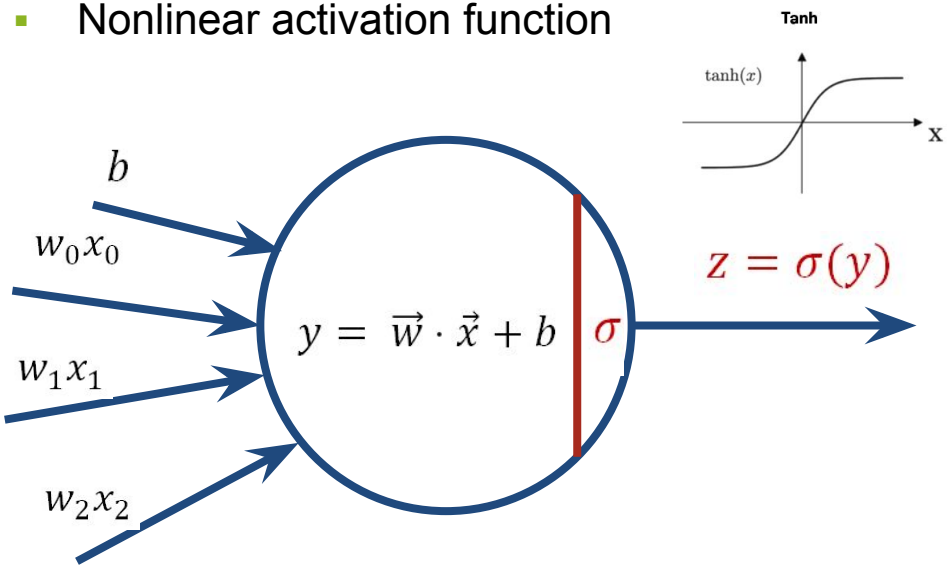



INPUTS OUTPUTS

- Multivariate linear regression

$$\vec{w} = (w_0, w_1, \dots, w_k) \qquad \vec{x} = (x_0, x_1, \dots, x_k)$$

- Nonlinear activation function

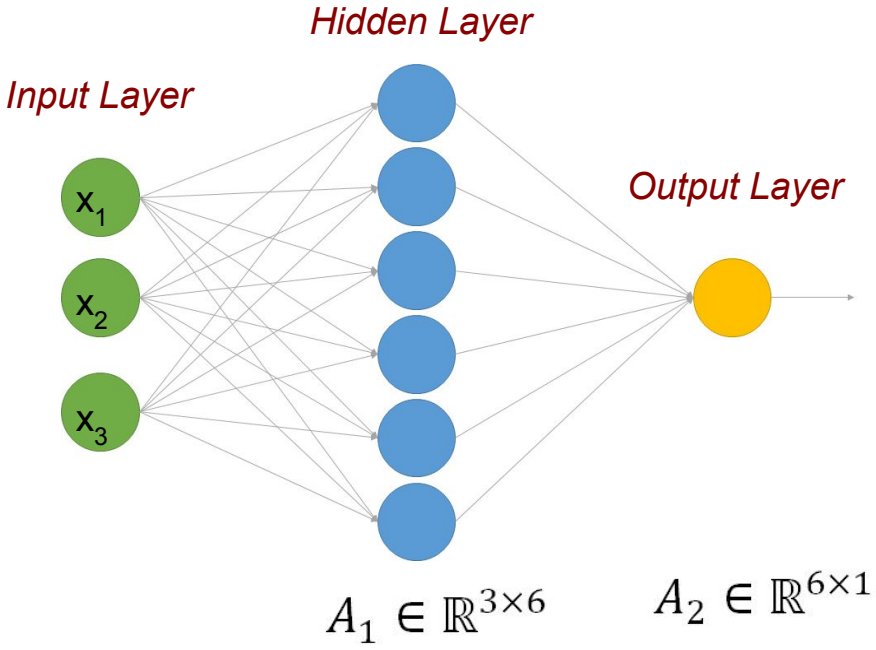


# Simple artificial neural network

- Combine neurons in layers + stack
- Matrix vector multiplication + nonlinear activation function composition

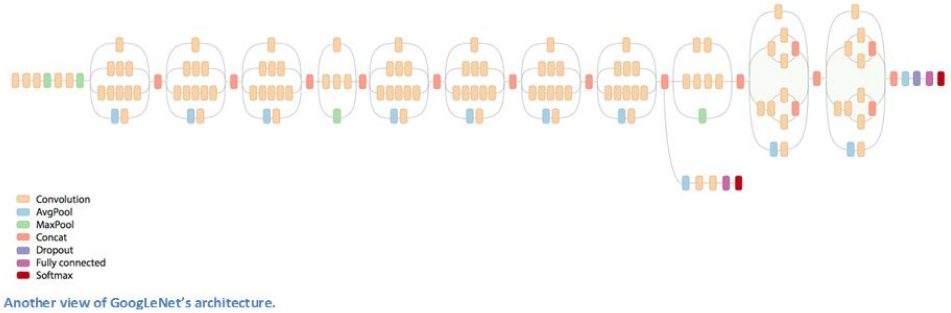
$$z = \sigma_2(A_2\sigma_1(A_1\vec{x}))$$

- Universal approximation theorem: ANN can approximate any well-behaved function
- Efficient with high-dimensional data
- Efficient for learning representations
- Can do classification and regression

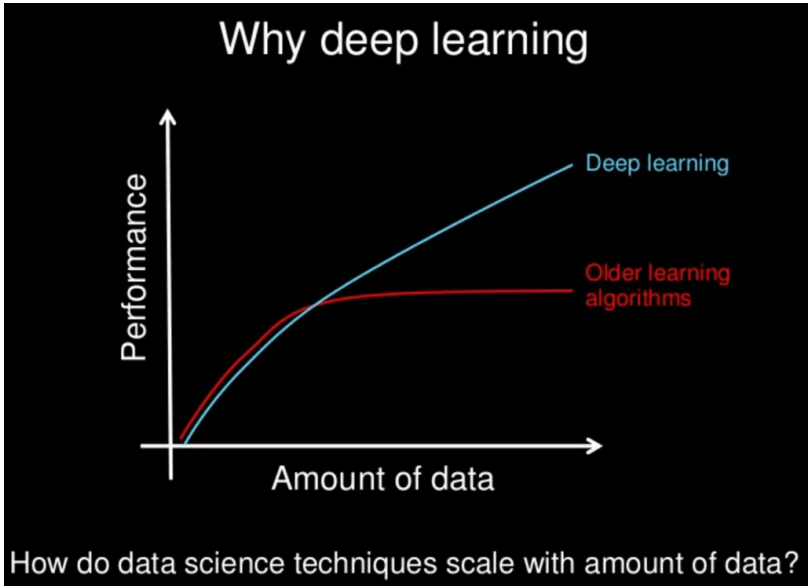


# Deep Learning

- Deep networks: many hidden layers
- → up to $10^9$ network parameters
- Express very complex functions
- Deal with noisy input data
- First layers: extract basic features
- Later layers: combine to higher-level features




Another view of GoogLeNet's architecture.




Growing field with Big Data!

GoogLeNet // Andrew Ng via https://machinelearningmastery.com/what-is-deep-learning/

# Training a neural network

- Supervised learning

$$\mathcal{D} = \{(x_i, y_i)\}_{i=1\ldots N}$$

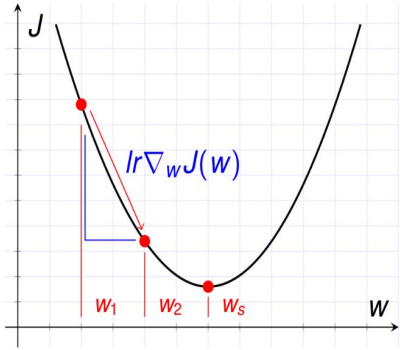
- Predict output with initial guess for w

$$\hat{y} = M(x; w)$$

- Loss function: quantify error

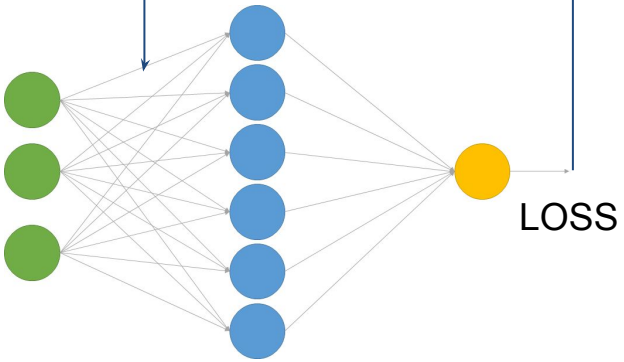
$$\mathrm{J}_w(\hat{y}, y) = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2$$

$10^3$ - $10^9$ weight parameters!




- Backpropagation algorithm
  - Computationally efficient
  - Finds local minimum

$$w_{ij} = lr \cdot \frac{\partial J}{\partial w_{ij}}$$



# Training a neural network

- Stochastic gradient descent
- Minibatch of *k* samples

$$m = 1 \ldots N/k$$

$$J_w(\hat{y}, y) = \frac{1}{k}\sum_{i=1}^{k}(\hat{y}_i - y_i)^2$$

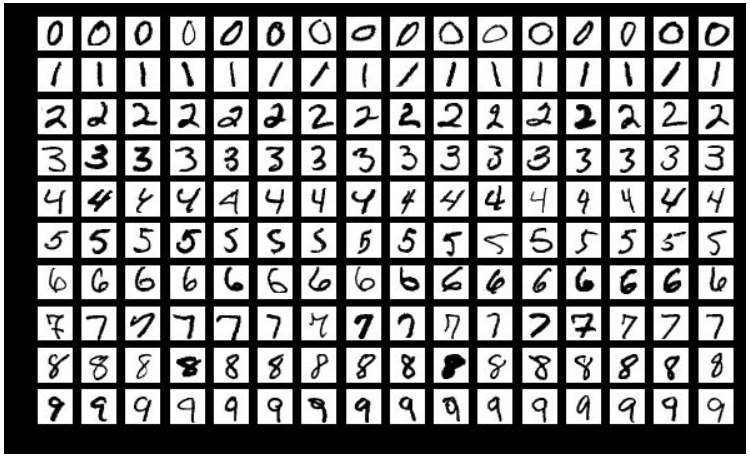
- All minibatches → one **training epoch**
- Shuffle before forming minibatches
- Avoids saddle points
- Speeds up calculation

Training with the PyTorch optimizer

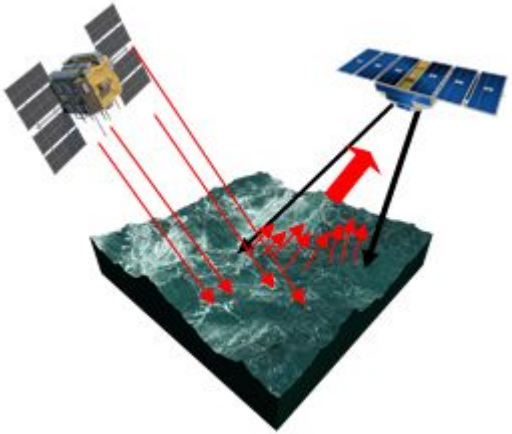
```
for batch in dataset:
    features, labels = batch
    preds = model(features)
    loss = loss_fn(labels, preds)
    # reset gradients
    optimizer.zero_grad()
    # calculate weight updates
    loss.backward()
    # do the updates
    optimizer.step()
```

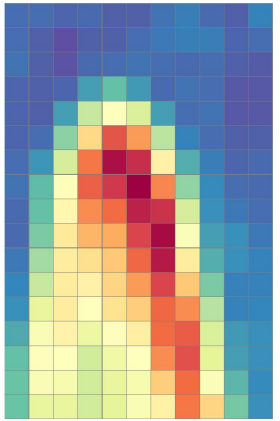
# Getting started with neural networks

**Tutorial MNIST:** Recognizing hand-written numbers → classification



**Tutorial CyGNSS:** Predicting wind speed from satellite observations → regression





# Implementing your own neural network

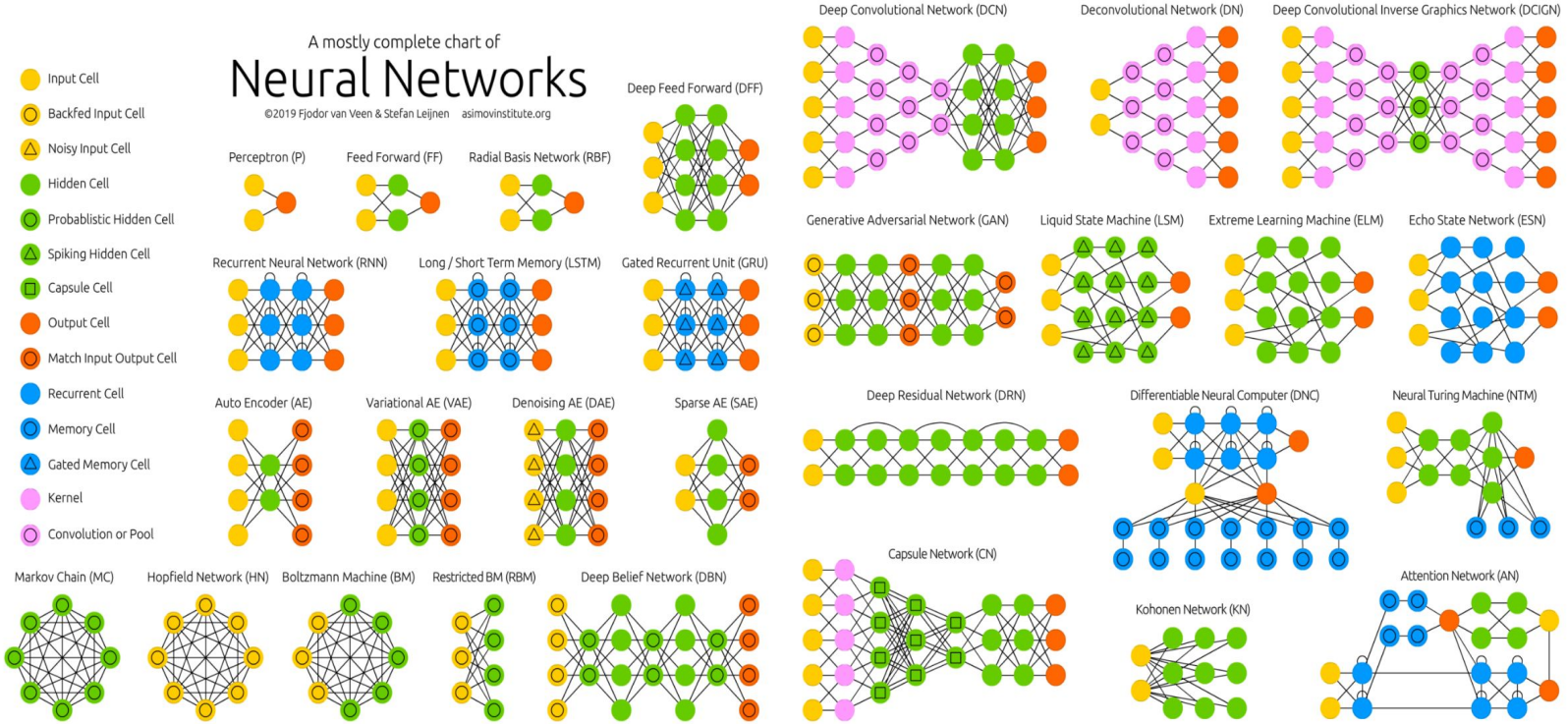
- You don’t have to start from scratch
- Open source frameworks
  - PyTorch (our preference)
  - Keras
  - Tensorflow




- Hardware: frameworks use GPUs as they are efficient for the matrix-vector multiplications
- HAICORE ressources if you are at a Helmholtz facility

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

# Neural Network Zoo

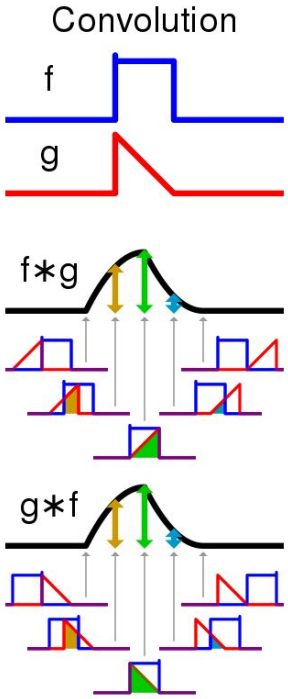


# A close look at some noteable architectures:

- **FFNN**: *Feed-Forward Neural Networks* we already learned about. Some basic concepts of these are also found in all the other architectures, but each of the following architectures introduces some bias which helps them excel at certain tasks.
- **CNN**: *Convolutional Neural Networks*, are especially useful for pattern recognition with localised features. Very well researched for computer vision and regular gridded data.
- **RNN**: Recurrent Neural Networks preserve some state/information as they are being invoked repeatedly. A prediction changes based on what was presented earlier.
- **LSTM:** *Long-Short-Term-Memory*, similar to RNNs are especially useful for time series data as they can preserve some information from earlier invocations. But the “gated” mechanisms are more feasible for more distant temporal relationships.
- **AE:** *Auto-Encoders* primarily learn an efficient representation, which often turn out to be useful for adaptation to other concrete tasks.

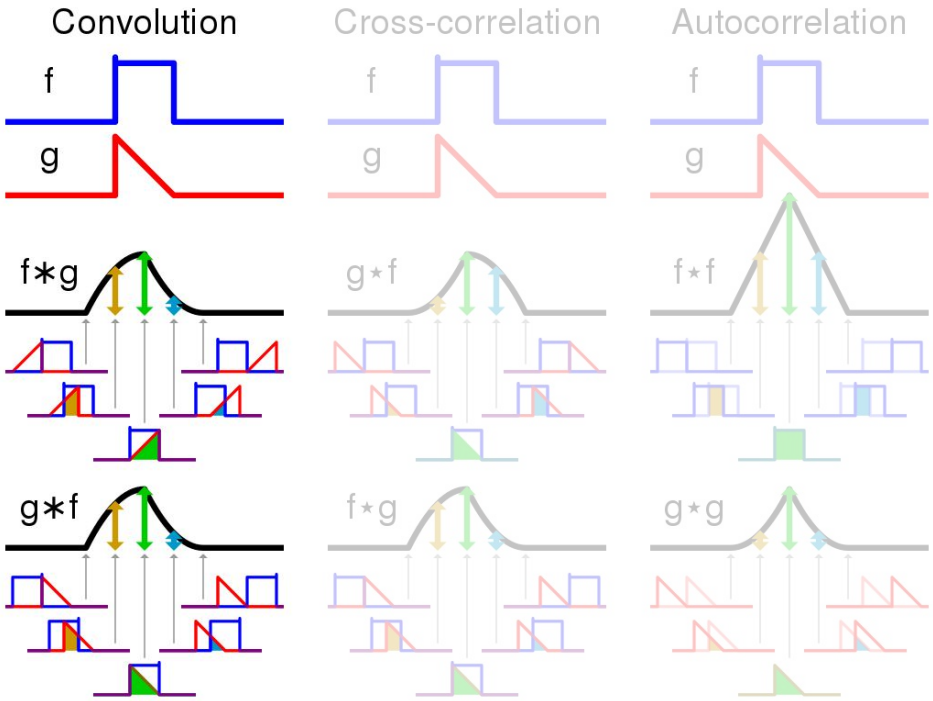
https://en.wikipedia.org/wiki/Convolution

# CNNs: Convolutions Basics and Applied



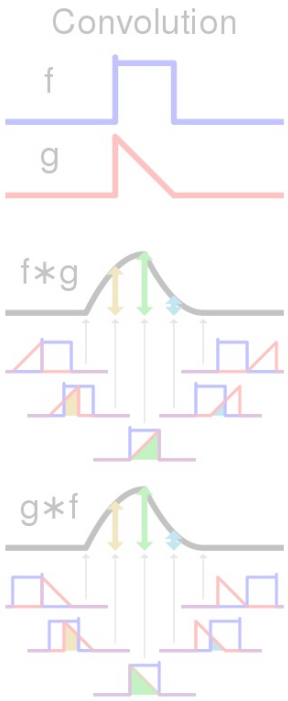
https://en.wikipedia.org/wiki/Convolution

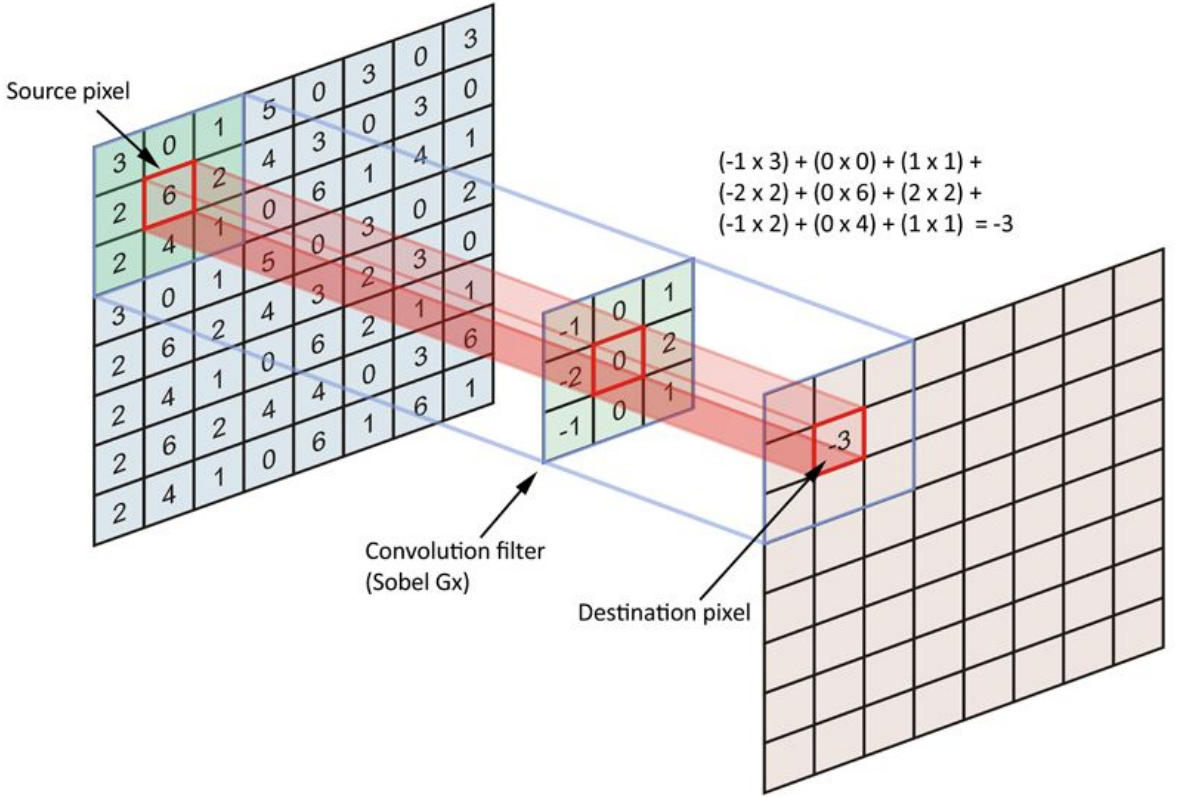
# CNNs: Convolutions Basics and Applied



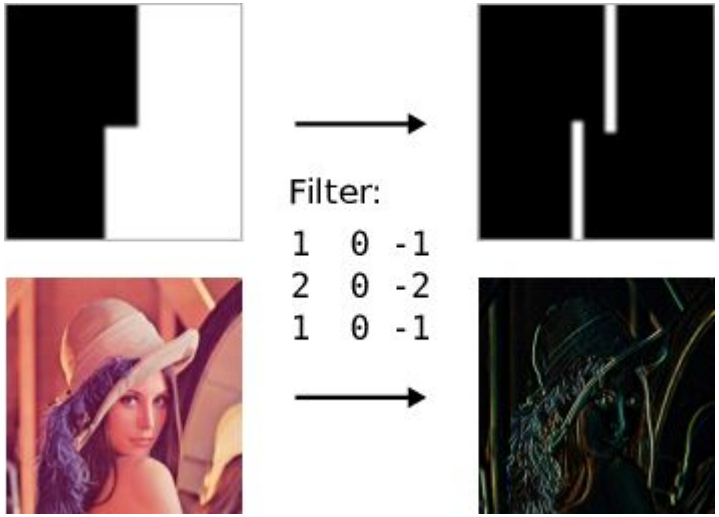
https://datascience.stackexchange.com/questions/23183/

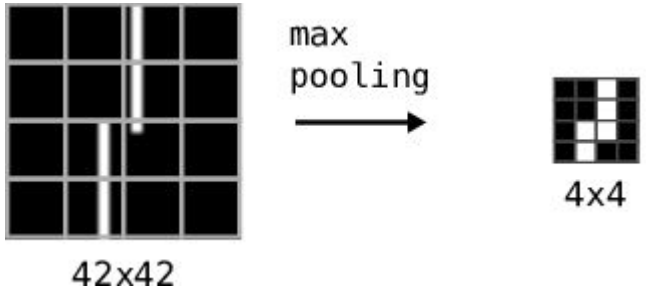
# CNNs: Convolutions Basics and Applied in 2D



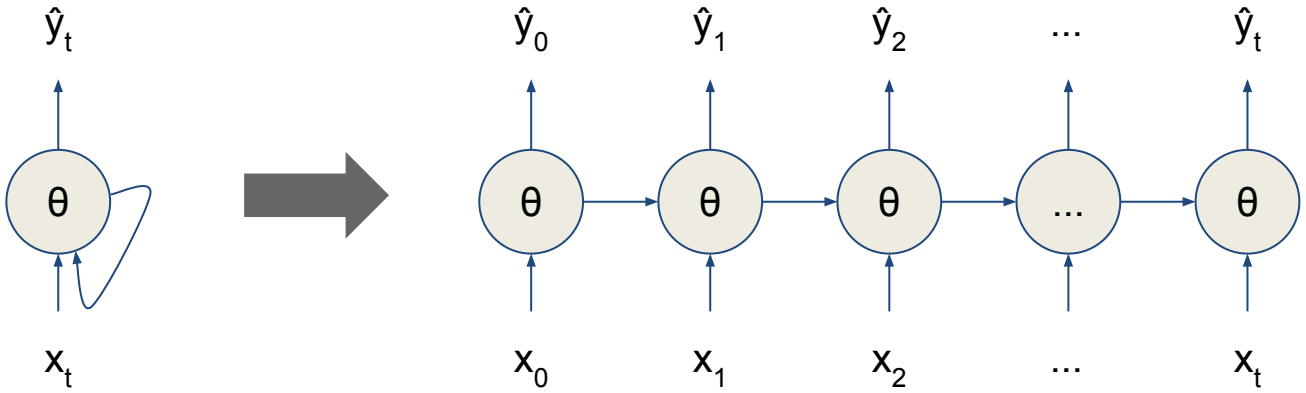
# CNNs: 2D Filters and Max-Pooling



# RNNs: Recurrent Neural Networks

Network is repeatedly invoked with output of previous invocation also becoming an input for subsequent executions:

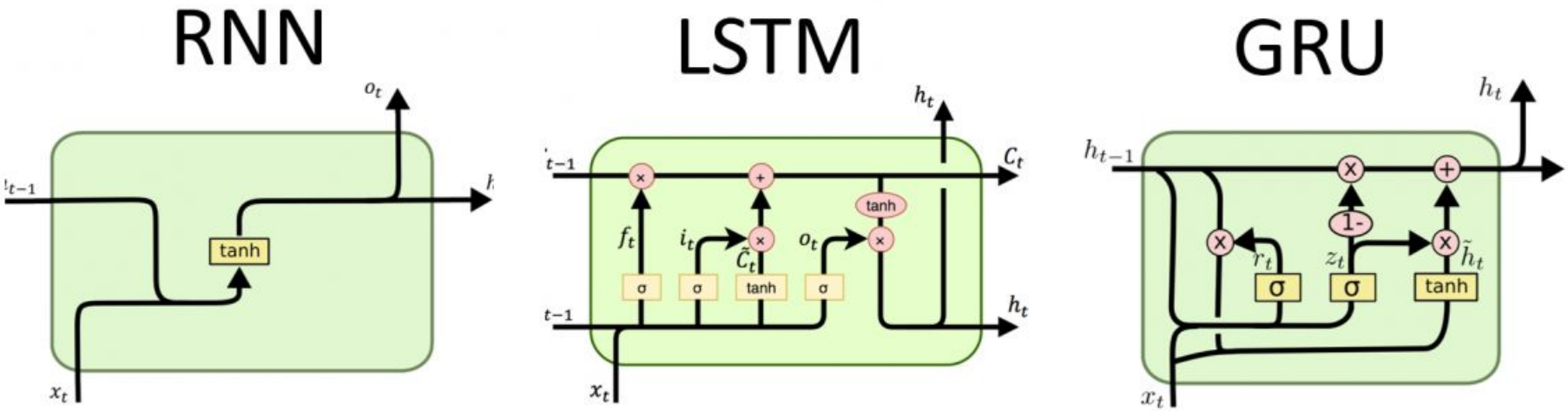



θ: Model
$x_i$: input at time/position i
$\hat{y}_i$: prediction at time/position i

https://medium.com/analytics-vidhya/recurrent-neural-network-and-its-variants-de75f9ee063

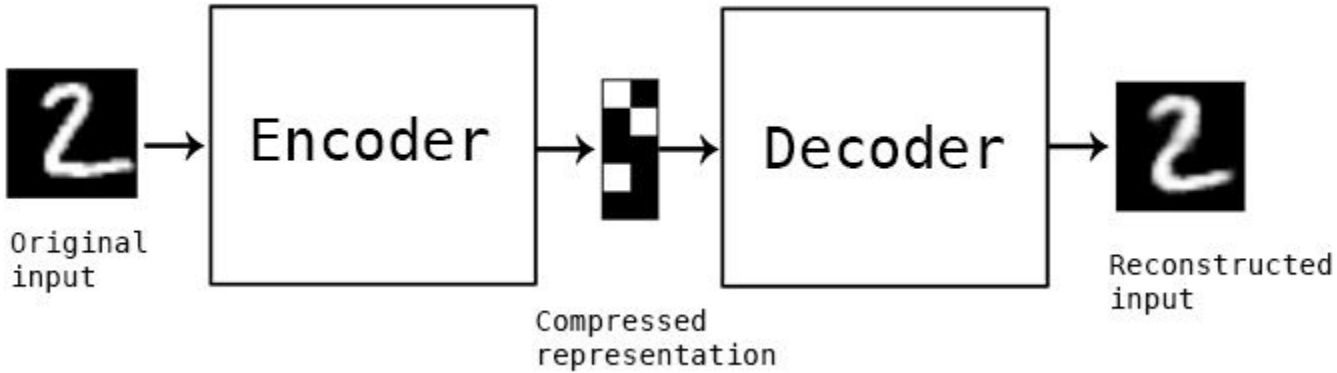
# RNNs: Recurrent Neural Networks

Overcoming vanishing gradients?



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

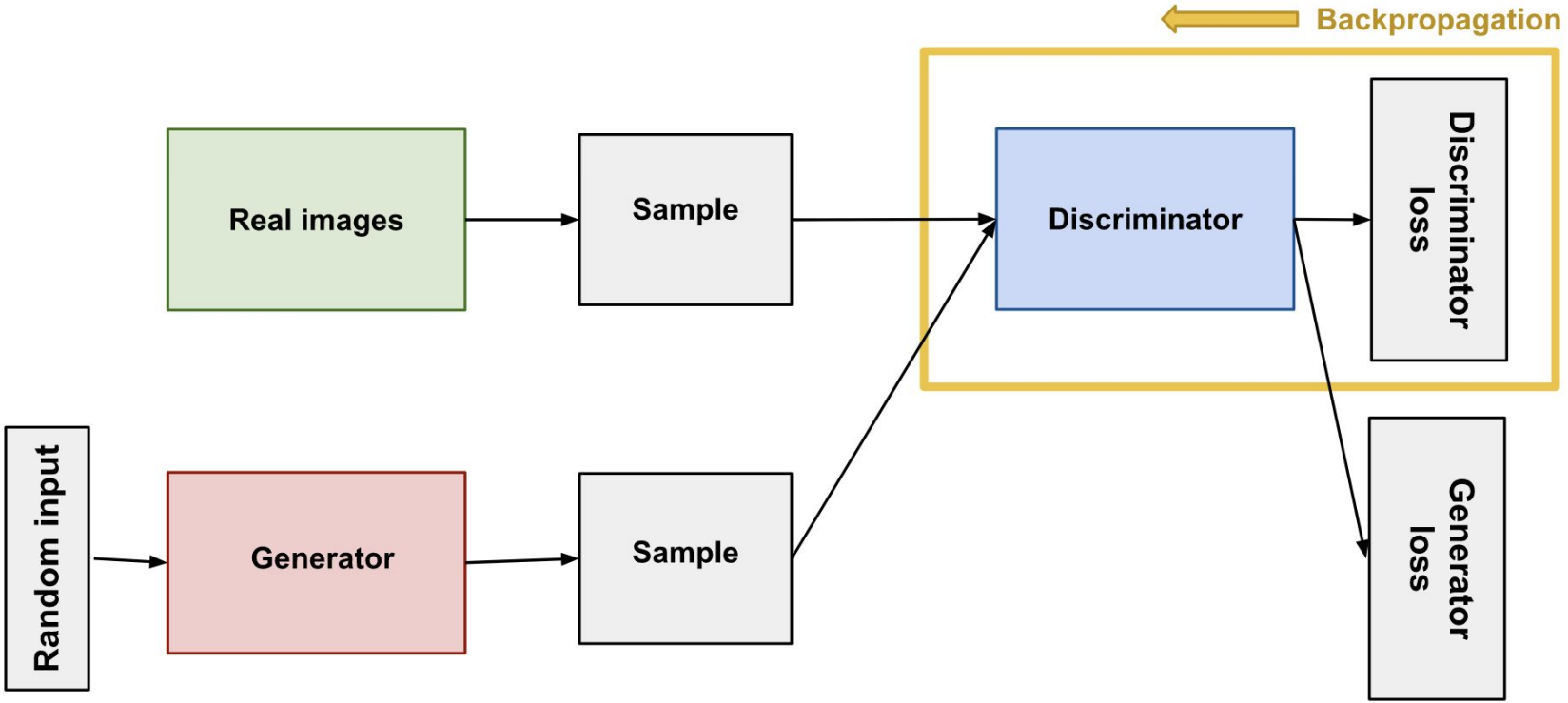
# AE: Auto-Encoders in a Nutshell



# https://thispersondoesnotexist.com/

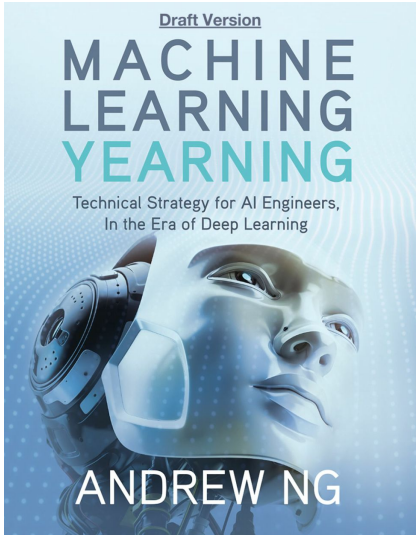
https://developers.google.com/machine-learning/gan/discriminator

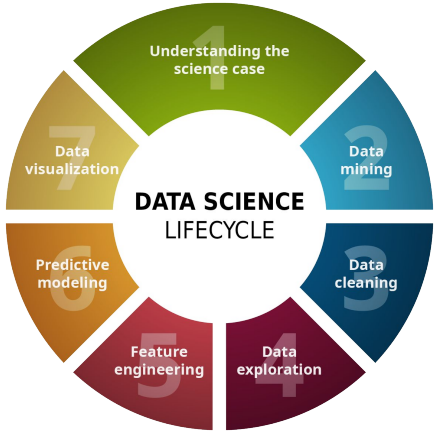
# GANs: Generative Adversarial Networks



# Error analysis, general improvements

(full recommended reference: Andrew Ng, Machine Learning Yearning)




https://github.com/ajaymache/machine-learning-yearning

# Introducing training and test sets

Divide your data in two parts:

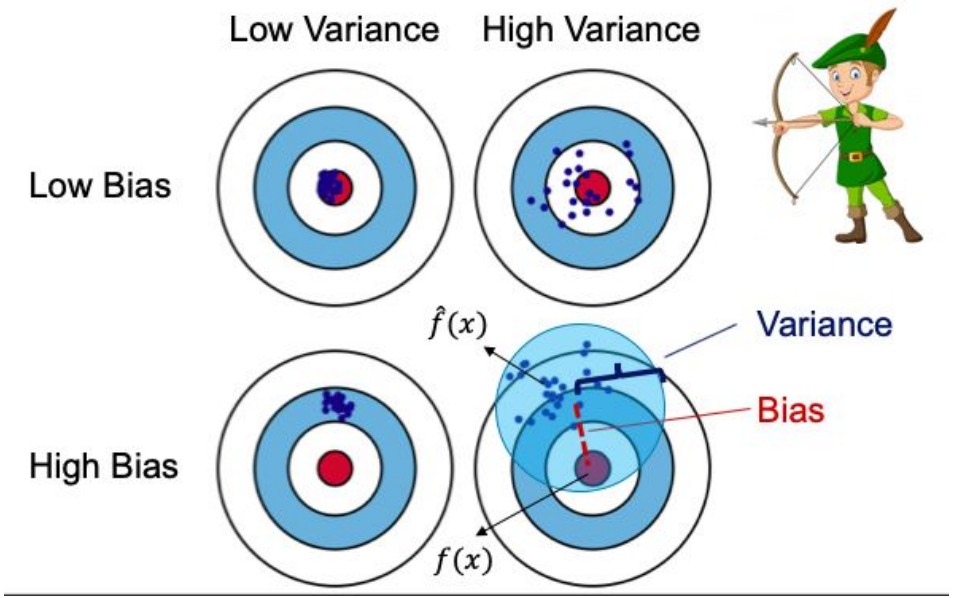
- Training set: a subset to train your model
- Test set: a subset to independently test the trained model

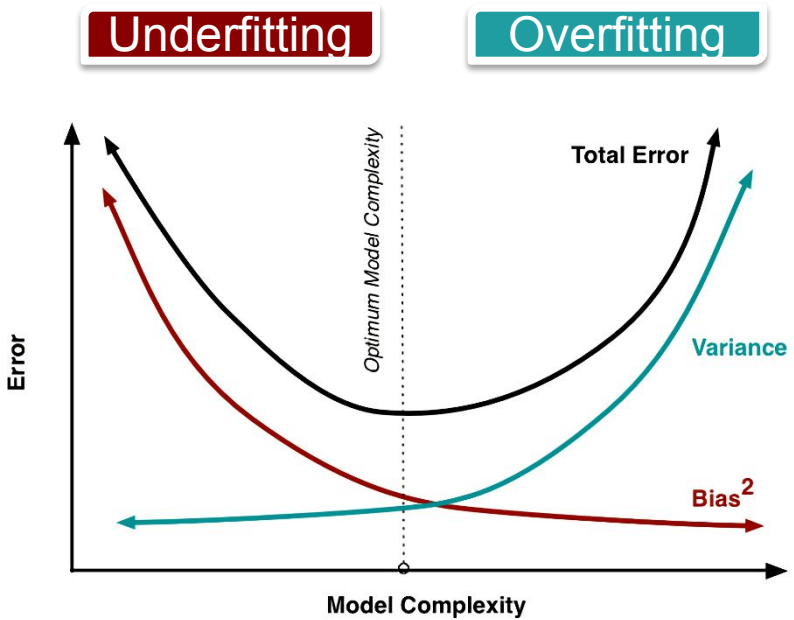
... often, you also create a third set - we will introduce that later.

# Best Practices in ML

## Underfitting and Overfitting

Slides by Dr. Rıdvan Salih Kuzu (DLR)



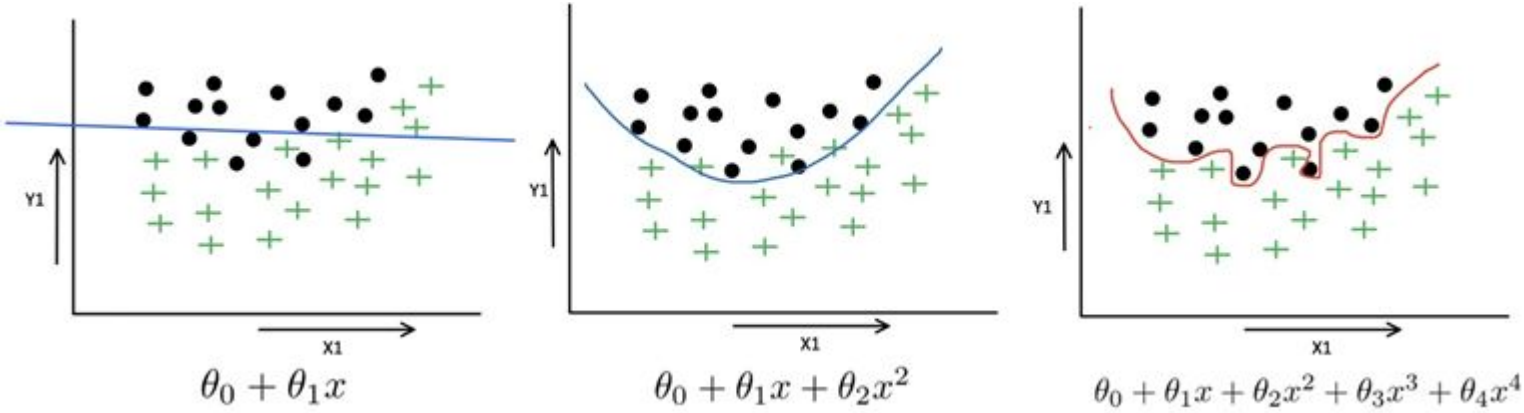




- **Bias:** the difference between the average prediction of our model and the correct value
- **Variance:** the variability of model prediction for a given data point which tells us spread of our data

# Best Practices in ML

Slides by Dr. Rıdvan Salih Kuzu (DLR)

## Underfitting and Overfitting




### Underfitting

- **High** training error
- Training and test errors are **close**
- High **bias**

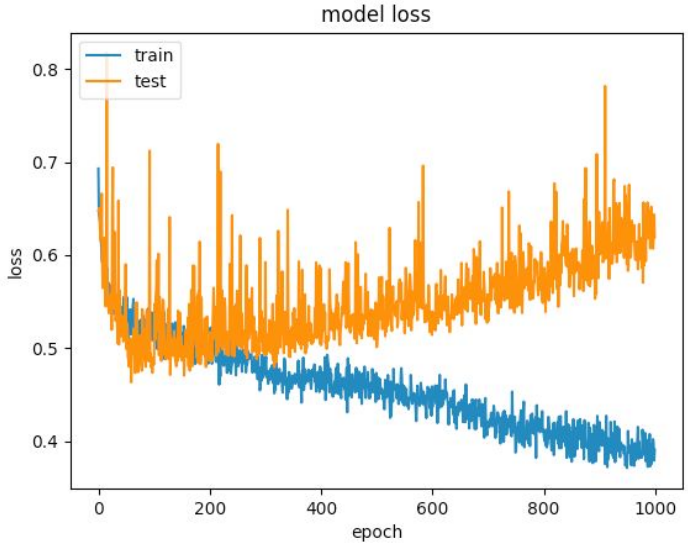
### Just Right

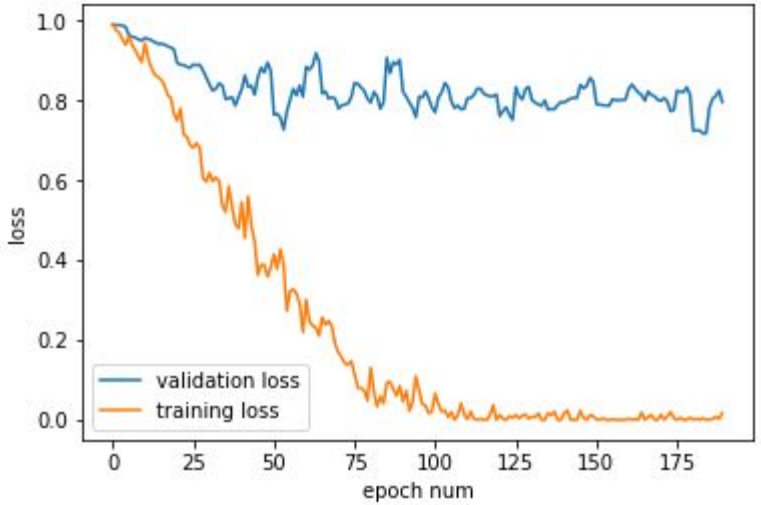
- Training error slightly lower than test error

### Overfitting

- **Low** training error
- Test error is much **bigger** than training errors
- High **variance**

# Overfitting manifestations



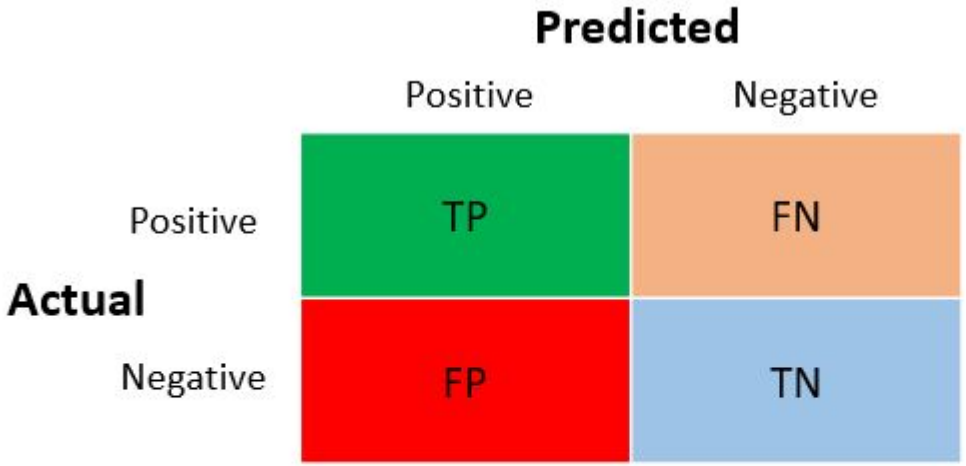




What to do about this?

# Best Practices in ML

## Model Performance Assessment

Slides by Dr. Rıdvan Salih Kuzu (DLR)

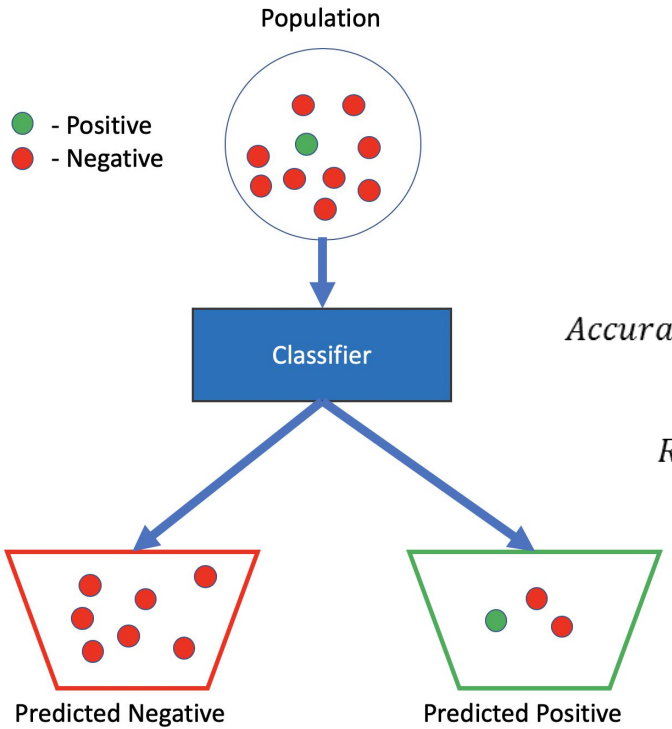



$$\text{Accuracy} = \frac{\blacksquare + \blacksquare}{\blacksquare + \blacksquare + \blacksquare + \blacksquare} \qquad \text{Precision} = \frac{\blacksquare}{\blacksquare + \blacksquare} \qquad \text{Recall} = \frac{\blacksquare}{\blacksquare + \blacksquare}$$

# Best Practices in ML

## Model Performance Assessment

Slides by Dr. Rıdvan Salih Kuzu (DLR)




| | | Real | |
|---|---|---|---|
| | | Positive | Negative |
| Predicted | Positive | 1 | 2 |
| | Negative | 0 | 7 |

$$Accuracy = \frac{Correctly\ Predicted}{Total\ Observation} = \frac{1+7}{10}$$

$$Recall = \frac{Real\ Positive}{Predicted\ Results} = \frac{1}{1+0}$$

$$Precision = \frac{Real\ Positive}{Real\ Results} = \frac{1}{1+2}$$

$$F_1 = 2x\frac{Precision\ x\ Recall}{Precision + Recall} = 0.5$$

**Precision** is the ability of a classification model to identify only the relevant data points

**Recall** can be thought as of a model's ability to find all the data points of interest in a dataset

**$F_1$-score**: harmonic mean of precision and recall

# Choose an evaluation metric

Decide which metric to use early on.

Typical metrics:

- classification: Accuracy
- regression: MSE

$$Accuracy = \frac{Correctly\ predicted}{Total\ Observation}$$

Correctly predicted = TP + TN

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

There are more metrics, but these work for a start.

Rule of thumb: Use the metric typical for your field / similar (non-ML) work.

# Choosing train/validation/test sets

- 60/20/20 split or similar - but not necessarily randomly:

  It may be wise to select intentionally from different distributions:

  - Choose validation and test sets to reflect data you expect to get *in the future* and want to do well on.
- Carefully consider because once set up, the sets should not be changed.

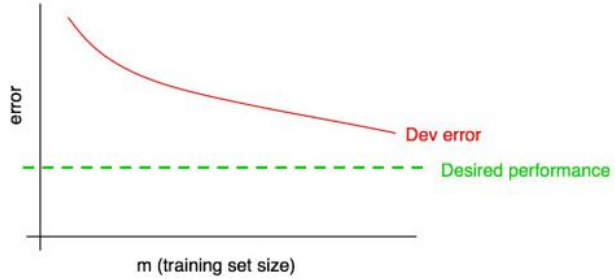
# Error analysis - step by step

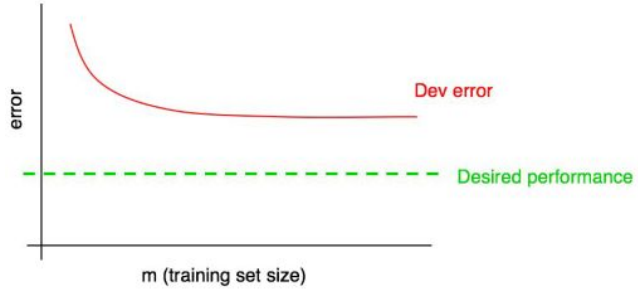
1. Build and train a first, imperfect system quickly and iterate.
2. Address under- and overfitting.
3. Manually examine a few misclassified/noisy samples
4. Prioritize and focus on fixing most prominent error category
5. Consider what could be an unavoidable bias (that you will not be able to fix)

**Unavoidable bias (optimal error rate):** Even the theoretically best possible system will be unable to beat this (e.g., because of fraction of noisy samples).
**Avoidable bias:** Training error larger than desired performance
**Variance:** difference between error on validation and test set

from Machine Learning Yearning by Andrew Ng

# Learning curve plots



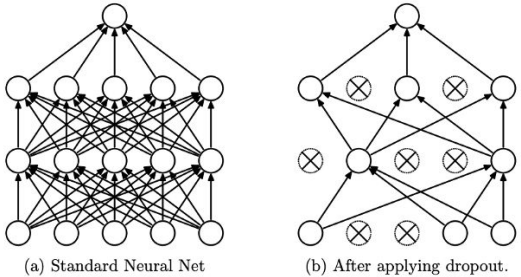

Right plot: Adding more dev data will probably not help.

from Machine Learning Yearning by Andrew Ng

# Dropouts and regularization

Further methods to deal with overfitting in NNs:

- Add dropout layers and tune the dropout factors



(a) Standard Neural Net (b) After applying dropout.

- Regularization: Additional term for the cost function
  - L1 (Lasso regression): sum over absolute weights
  - L2 (Ridge regression): sum over squared weights
    - pytorch: weight_decay parameter to Adam optimizer
  - Tune the scaling factor (lambda)

$$Cost\ function = Loss + \lambda \sum w^2$$

https://medium.com/analytics-vidhya/understanding-regularization-with-pytorch-26a838d94058

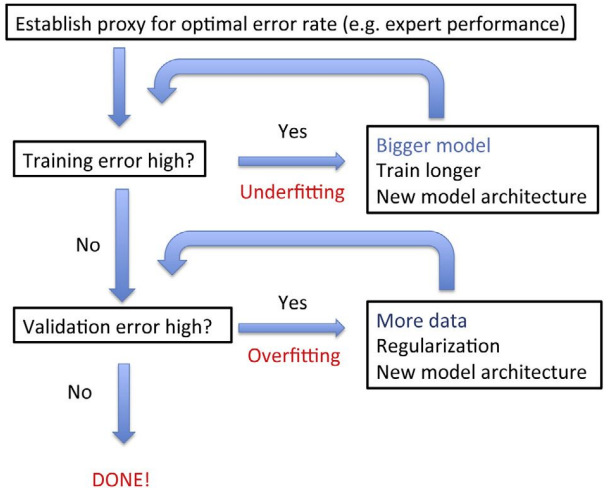
# How to find the optimal network parameters?

- Each ML algorithm has a number of hyperparameters that need to be adapted for the specific problem at hand
- Examples: optimizer learning rate, choice of activation function, number of hidden layers, number of neurons in each layer …

- Tune the parameters in a systematic way, testing against your validation set
  - Random search
  - Autotuners like NNI
  - … this would be a seperate tutorial ...

# ML workflow (possibly redundant with former / later sections)

- General ML workflow for a supervised learning problem (Mehta 2019)
  - Collect and pre-process the data
  - Define model & architecture
  - Choose cost function & optimizer
  - Train the model
  - Use validation data to adjust the hyperparameters etc
  - Finally! Evaluate and study the model performance on the test data
- Never let what you learn on the test set guide your selection for the training procedure (you want to judge the generalization capability!!)
- Obtaining good quality data to answer your (research) question can be where you spend most of your time



# Sources


1. Mehta, P. *et al.* A high-bias, low-variance introduction to Machine Learning for physicists. *Physics Reports* **810**, 1–124 (2019).
2. Andrew Ng. Machine Learning Yearning. https://github.com/ajaymache/machine-learning-yearning
3. The Neural Network Zoo https://www.asimovinstitute.org/neural-network-zoo/
4. Neural Netowrk Zoo Prequel: Cells and Layers https://www.asimovinstitute.org/neural-network-zoo-prequel-cells-layers/
5.