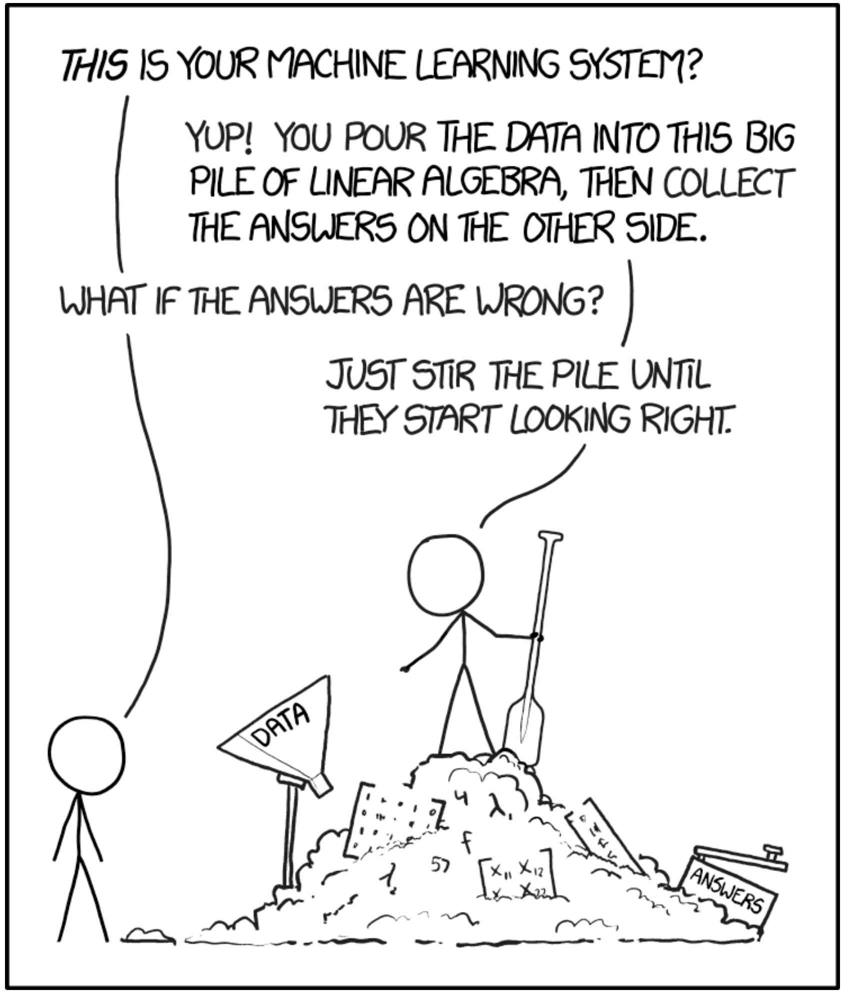


xkcd

Part 3:
Best Practices, Pitfalls & Tricks

An Inconvenient Truth

- Deep neural networks comprise millions of parameters: we don’t know what these parameters mean
- Most of the time, we don’t know what a NN learns
- NN are not suitable to gain “understanding”

<u>Neural networks are black boxes: treat them as such!</u>

An Inconvenient Truth: Example

Audi (82%)

BMW (91%)

Ferrari (79%)

An Inconvenient Truth: Example

Ferrari (95%)

Ferrari (79%)

An Inconvenient Truth: Example

- Training data selection is critical
- The NN “learns” your interpretation based on the training data, including observational/operator bias (NN are not unbiased!)
- If all Ferraris in the training data are red, and all other cars are not red, then all red objects must be Ferraris!

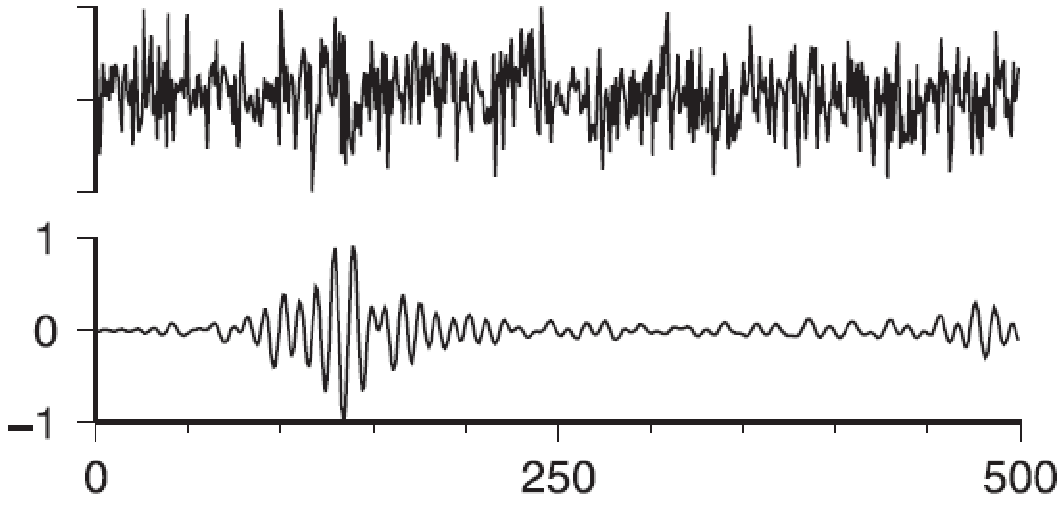
An Inconvenient Truth

- Machine Learning is mostly based on trial-and-error
- There is no recipe for good performance, only guidelines
- But: more theory is (slowly) being developed

Pitfalls

1. Bias and class imbalance in training set
2. Overfitting
3. Extrapolation beyond training data range
4. Improper weight initialisation
5. Excessive learning rates

Pitfalls: Class Imbalance

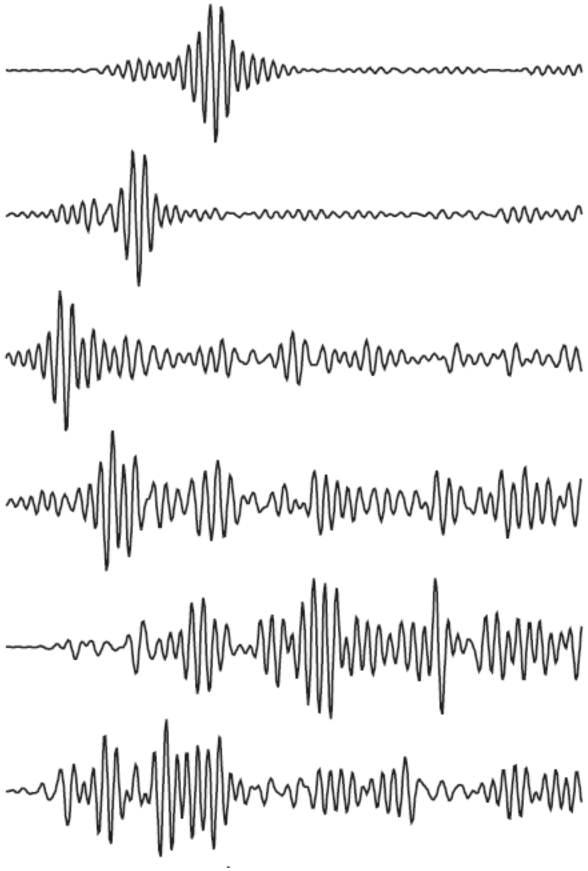


Valentine & Trampert (2012)

Earthquake detection:

1. Noise
2. Earthquake

Pitfalls: Class Imbalance



Valentine & Trampert (2012)

→ Prediction: noise

→ Prediction: noise

→ Prediction: noise

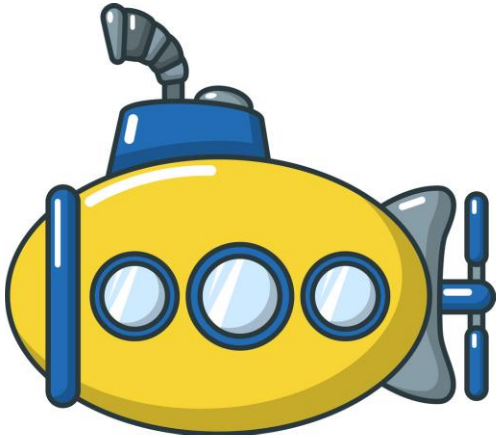
→ Prediction: noise

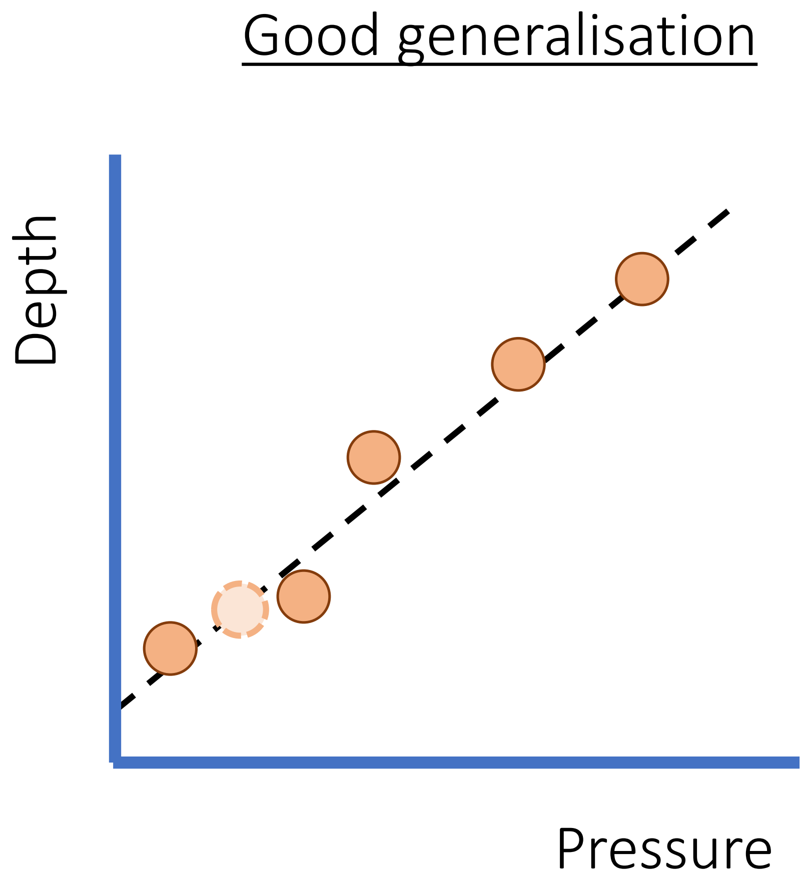
→ Prediction: noise

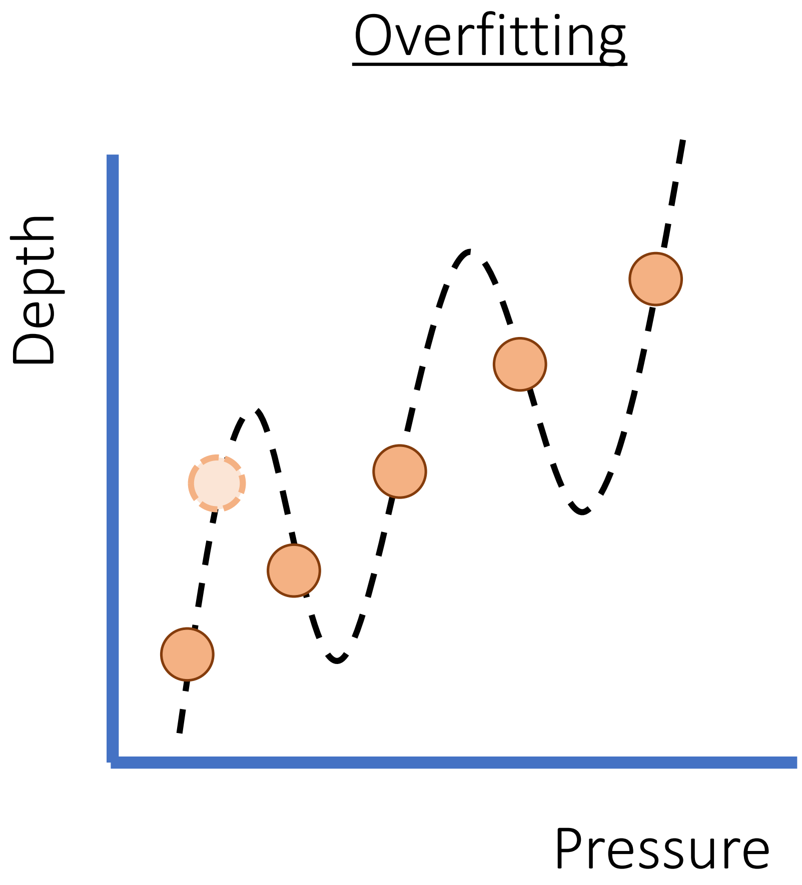
→ Prediction: noise

99.9% accuracy!

Pitfalls: Overfitting

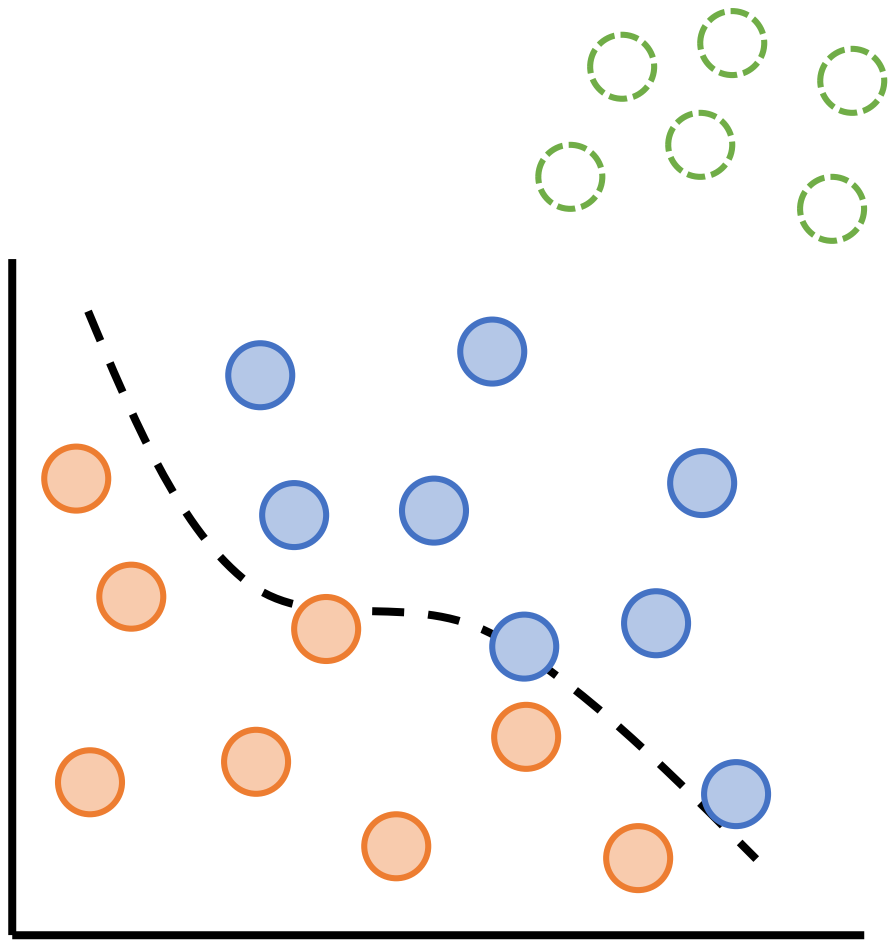




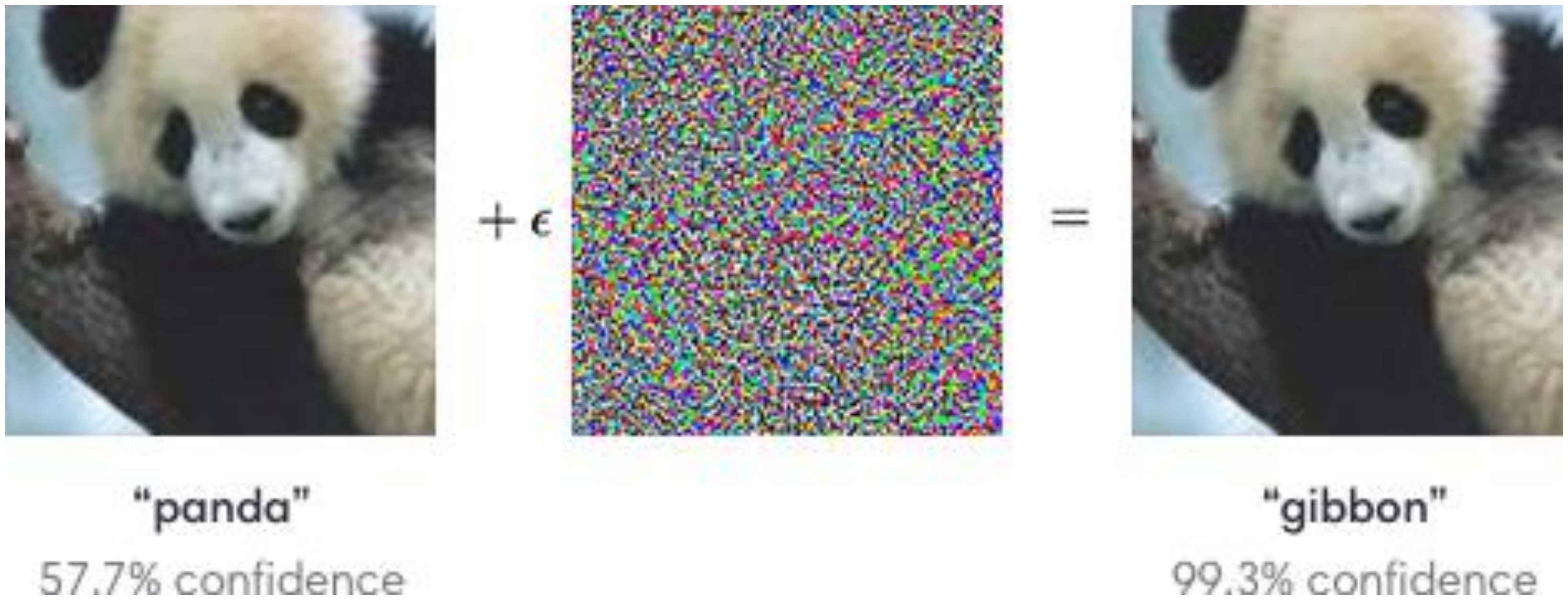


Pitfalls: Extrapolation

- Most NN architectures have a monotonic response
- Beyond the data range the network confidence increases, whereas it should decrease!
- Example: predicting large earthquakes based on small ones

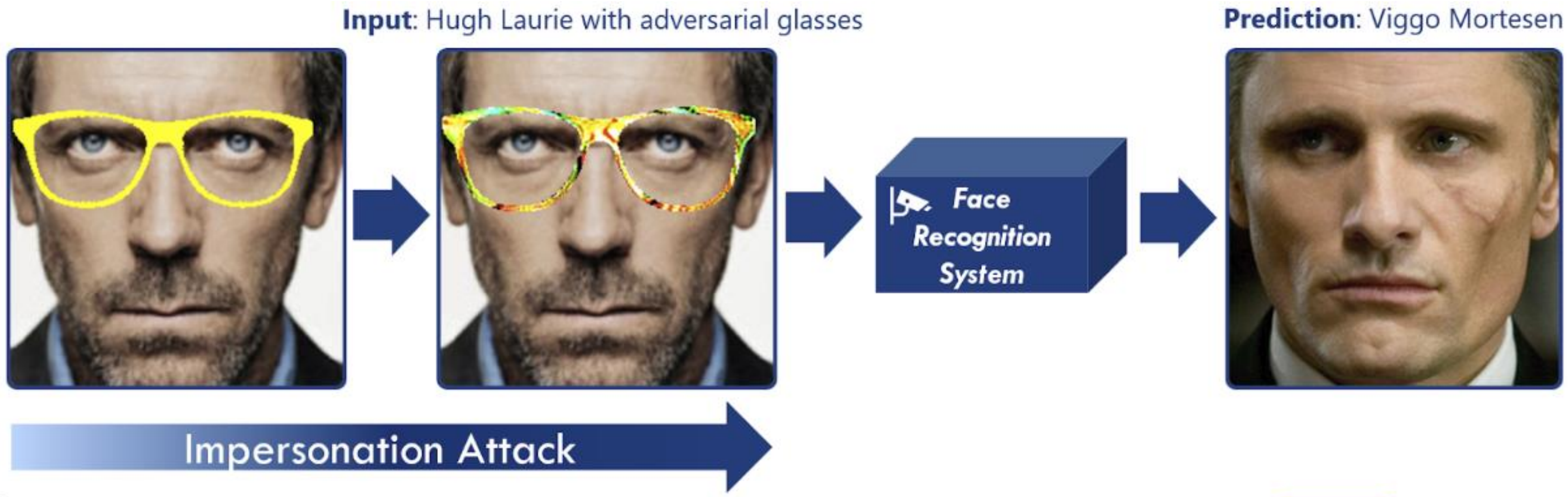


Pitfalls: Extrapolation (Adversarials)



https://openai.com/blog/adversarial-example-research/

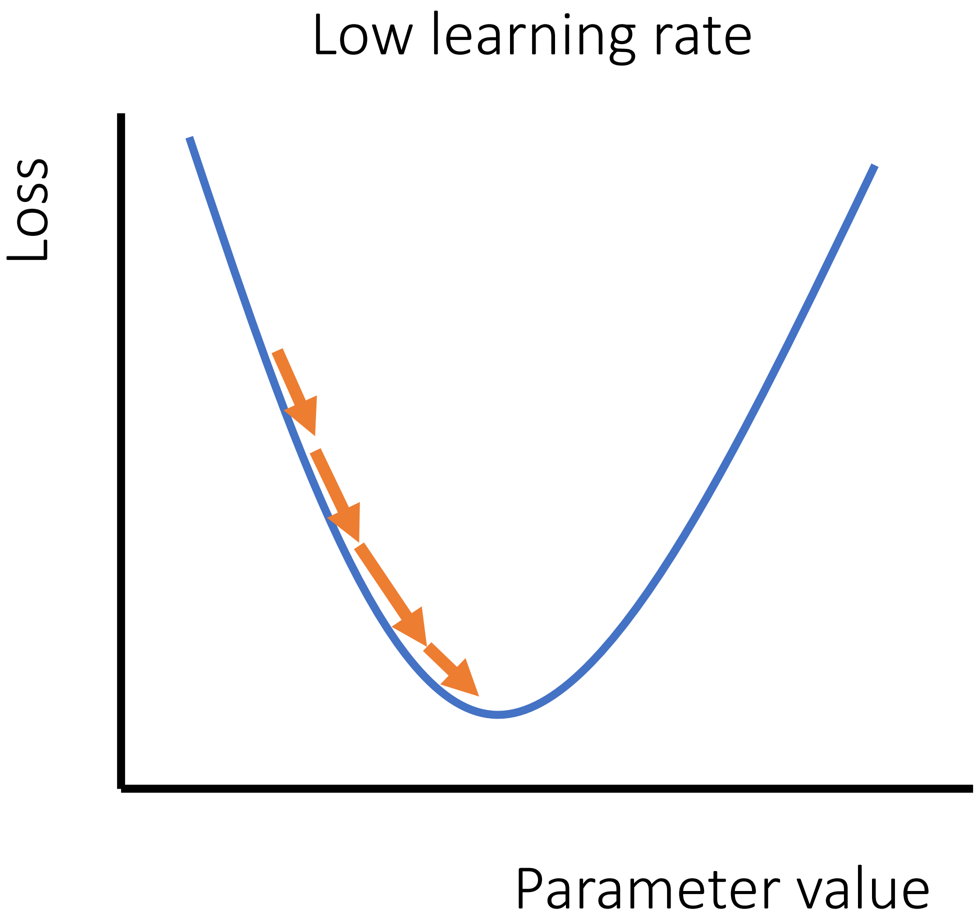
Pitfalls: Extrapolation (Adversarials)

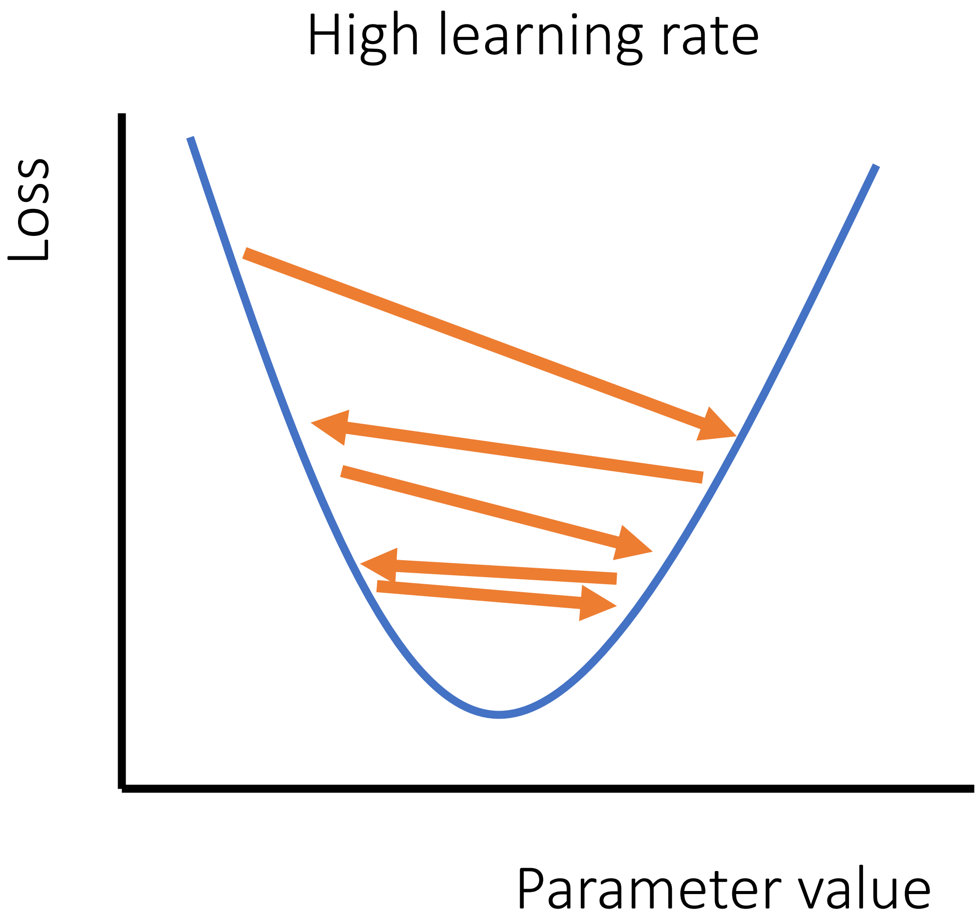


Pitfalls: Initialisation

- Weights are initialised by sampling from a random distribution
- If variance of every layer output < 1: vanishing gradients
- If variance of every layer output > 1: exploding gradients
- **Solution**: sample from random distribution with variance inversely proportional to layer input. This depends on the activation function!
    - ReLU: “He Normal initialisation” (He et al., 2015)
    - Sigmoid/tanh: “Xavier/Glorot initialisation” (Glorot & Bengio, 2010)

Pitfalls: Learning Rates





Guidelines

1. Data representation and network architecture are most important
2. Bigger networks require more data = manual labour
3. Training data should be balanced, test data should be representative for real-world application
4. Training a NN is like turning a key in a lock: it only works if all components fall into place

Best Practices (1/2)

1. Start with a small network architecture
2. Before anything else, <u>verify that training/test data is correct!</u>
3. Try overfitting your data. If that doesn't work, something is fundamentally wrong (e.g. initialisation)
4. Scale/shift the input data to have zero mean and a variance of around 1 (see basic MNIST tutorial)
5. Monitor train/test loss: if training loss decreases but test loss increases, the network is overfitting

Best Practices (2/2)

6. Monitor training process using TensorBoard. Make quantitative comparison between different “experiments” (architectures, hyperparameters, etc.)
7. Use Adam’s optimiser, ReLU activation (arguable)
8. Experiment with regularisation: batch normalisation, layer normalisation, dropout, noise layers (not covered today)
9. Be patient: if the network/dataset is large, training can take days on a decent GPU

Resources

- YouTube
  - Lectures by Ian Goodfellow, Andrew Ng
  - Conference talks: e.g. NeurIPS (previously NIPS)
- Udacity course (free): "*Intro to TensorFlow for Deep Learning*"
- Competitions: Kaggle.com, DrivenData.org

Time to get really dirty…