Last time..

Today:

Generalization

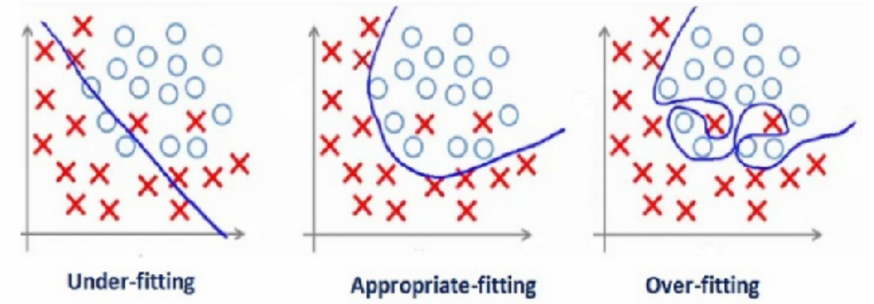
Overfitting, regularization, initialization, hyper-parameters, vanishing gradients / unstable gradients and deep NNs etc.

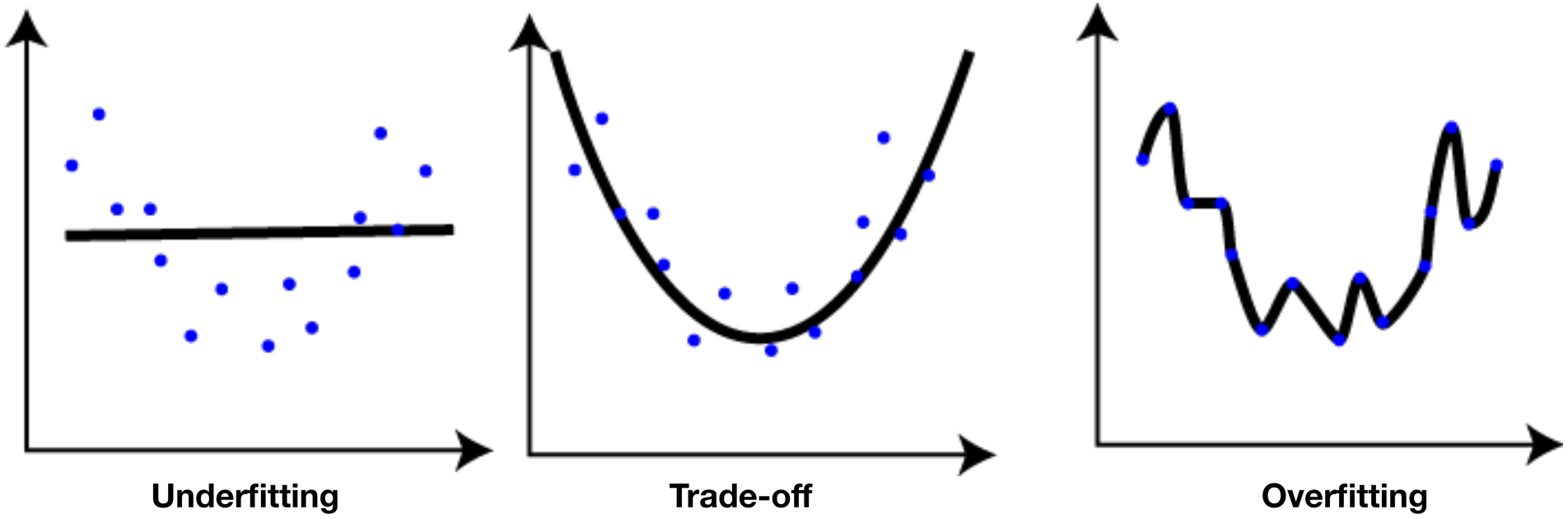
Agenda

- Generalization vs. Memorization
  - Under-fitting, over-fitting and tradeoff
  - Slitting the dataset
  - Model capacity vs. data
  - Loss-curves
- Preventing overfitting
  - Early-stopping, Data augmentation (or more labeled data), Regularization, Dropout, Ensemble learning, Multi-task learning, Preprocessing of input-data, Batch Normalization, Parameter initialization, Hyper-parameters, Transfer-learning, Challenges DNN

Generalization (vs. memorization)

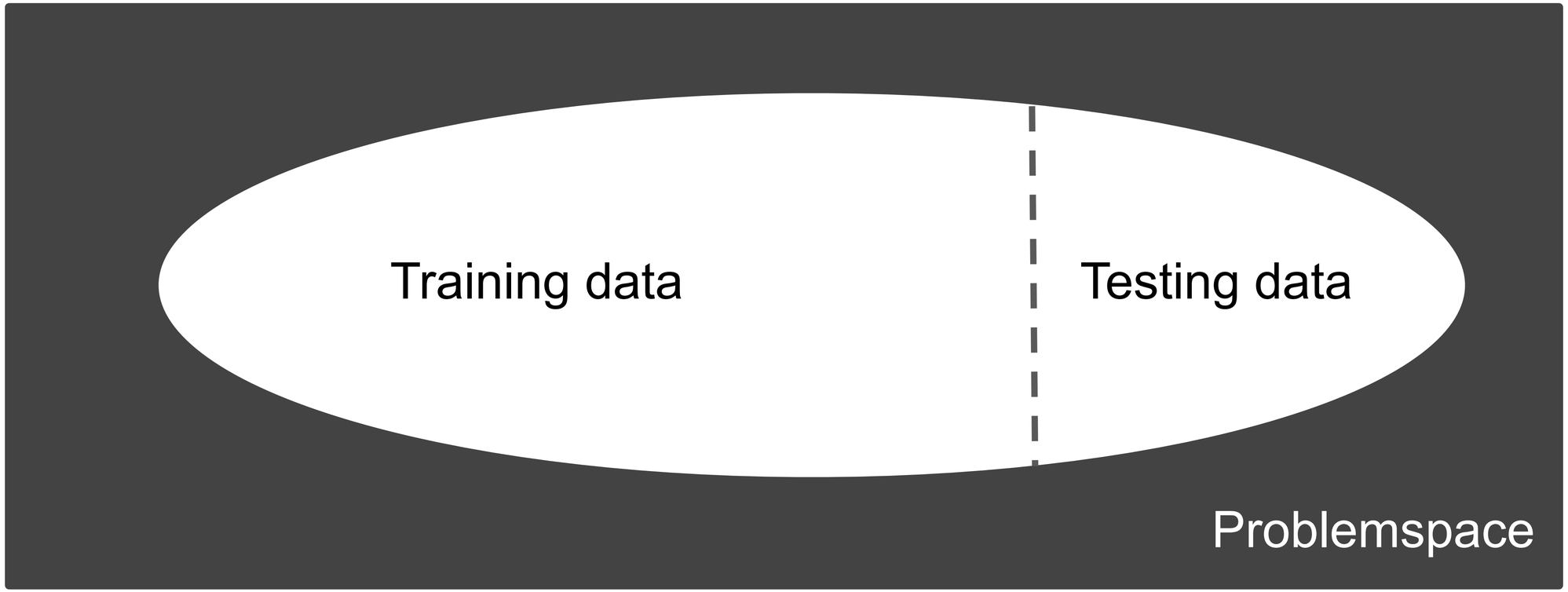
- Goal: model that perform well on unseen examples (**generalization**)
- Overfitting = too specialized on training data (**memorization**)
- Underfitting = not able to learn important patterns in training data



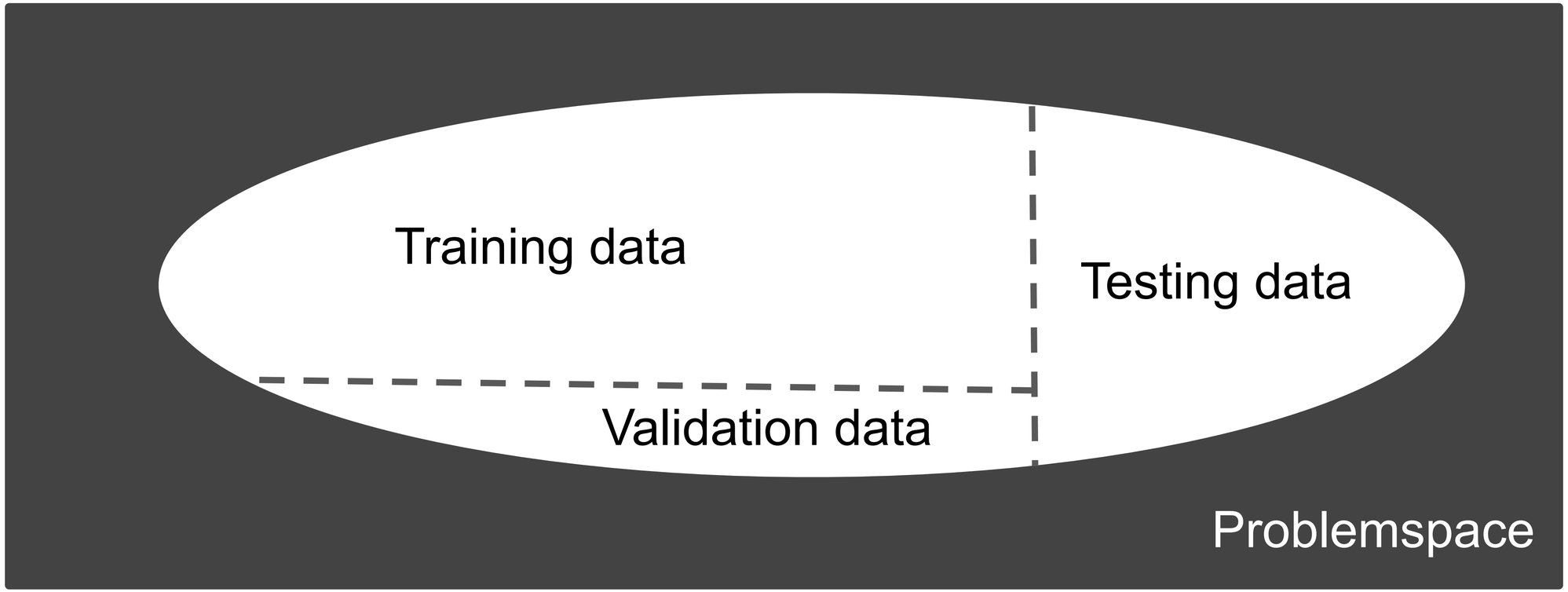


Splitting the dataset
Training, Validation and Test data

Training, Validation and Test data

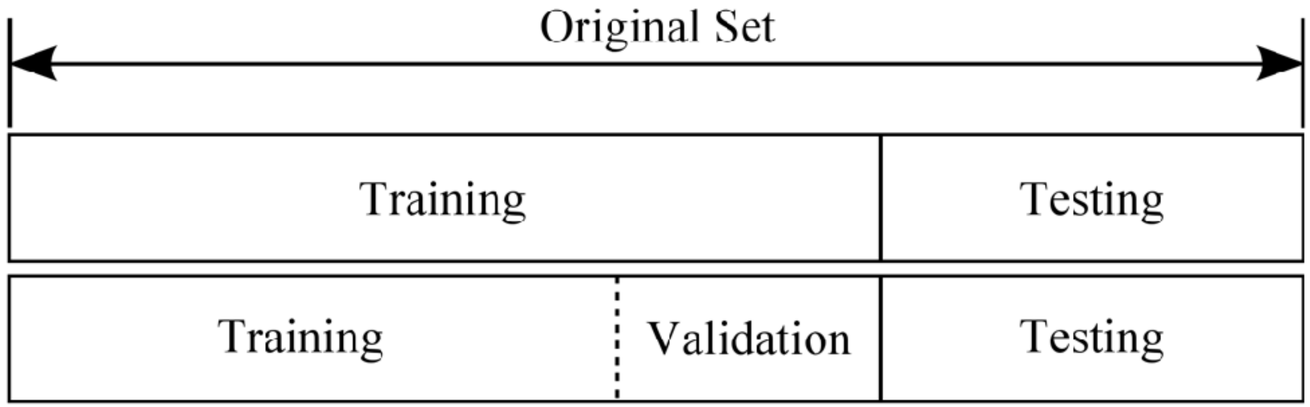


Training, Validation and Test data

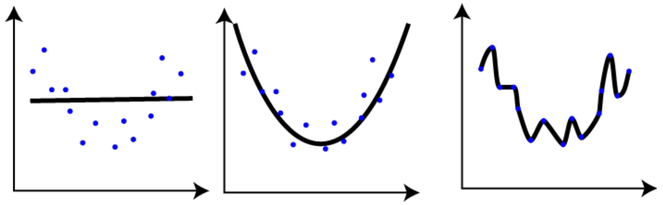


Splitting the dataset

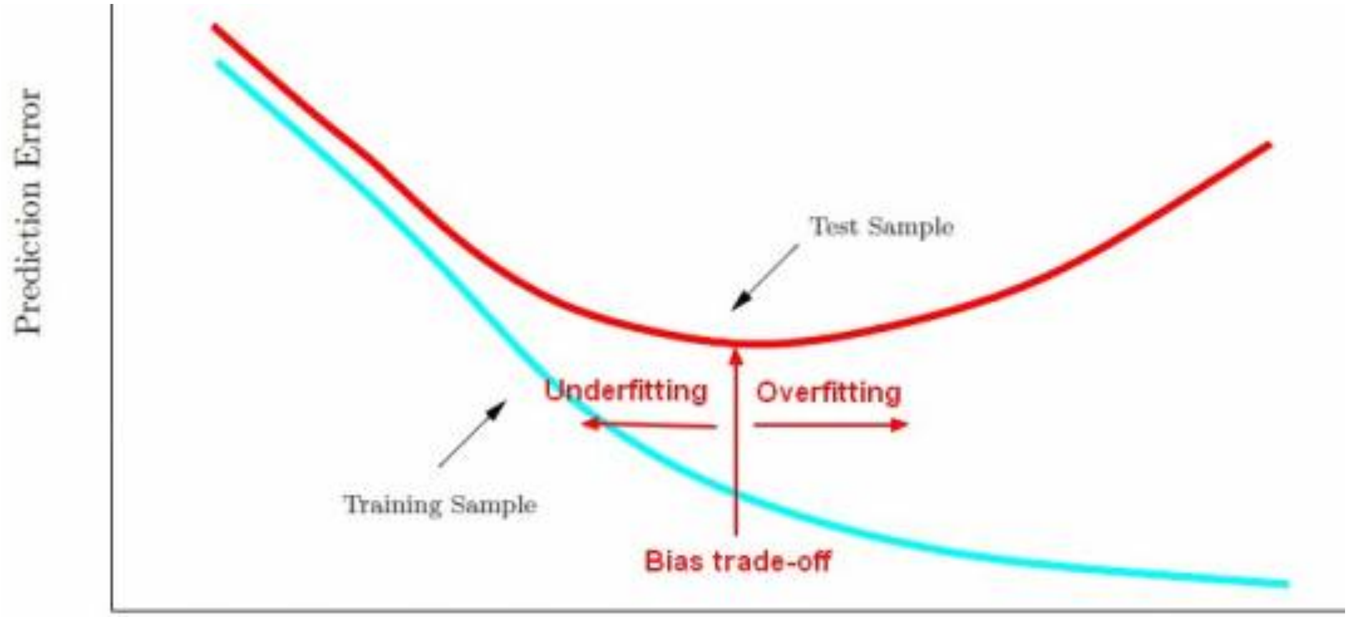
- **Fit** parameters to minimize C over **training data**
- Use **test set** as a measure how well the method will **generalize to unseen data**
- Want to **minimize** training error and test error
- Generalization error:



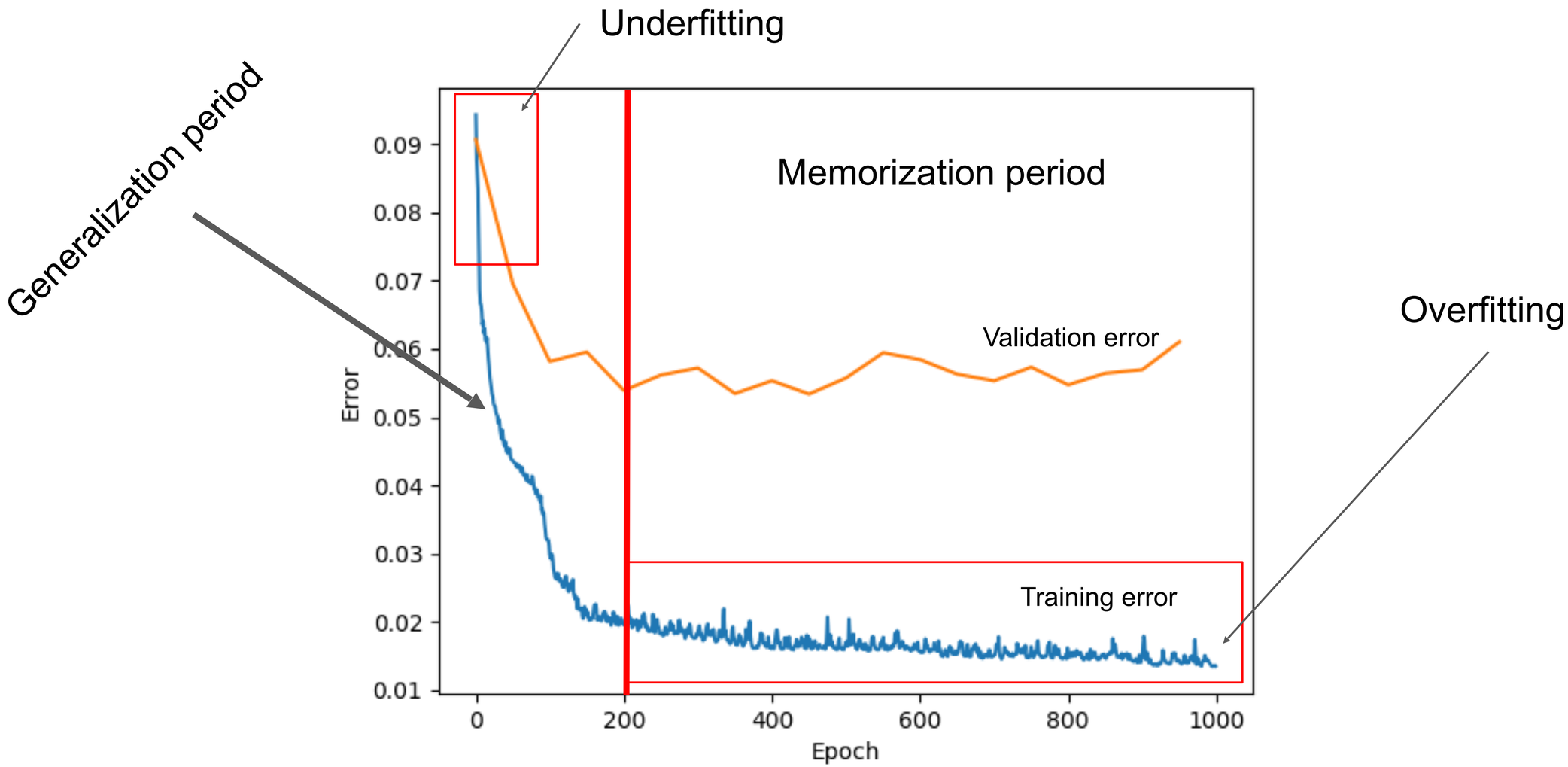
Overfitting



- Why overfitting?
  - **Many** parameters (high model capacity) - Trade off: not more than what's needed
  - **Little** training data
    - Typically a **combination** of the two
  - A neural network typically becomes overfitted if **training** proceeds for a **long time**
  - +++

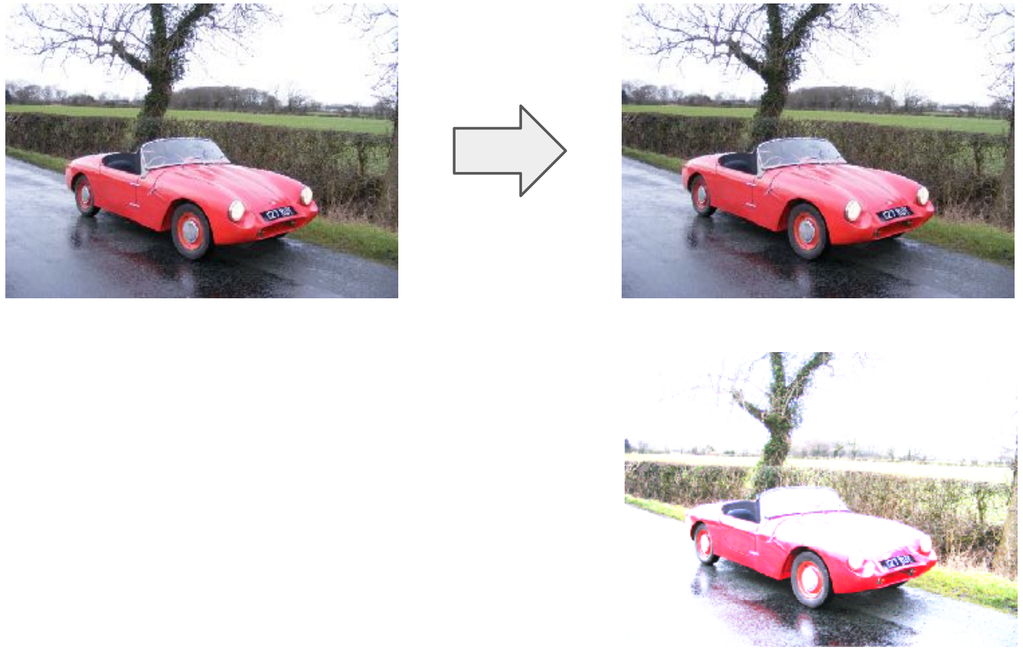


Overfitting

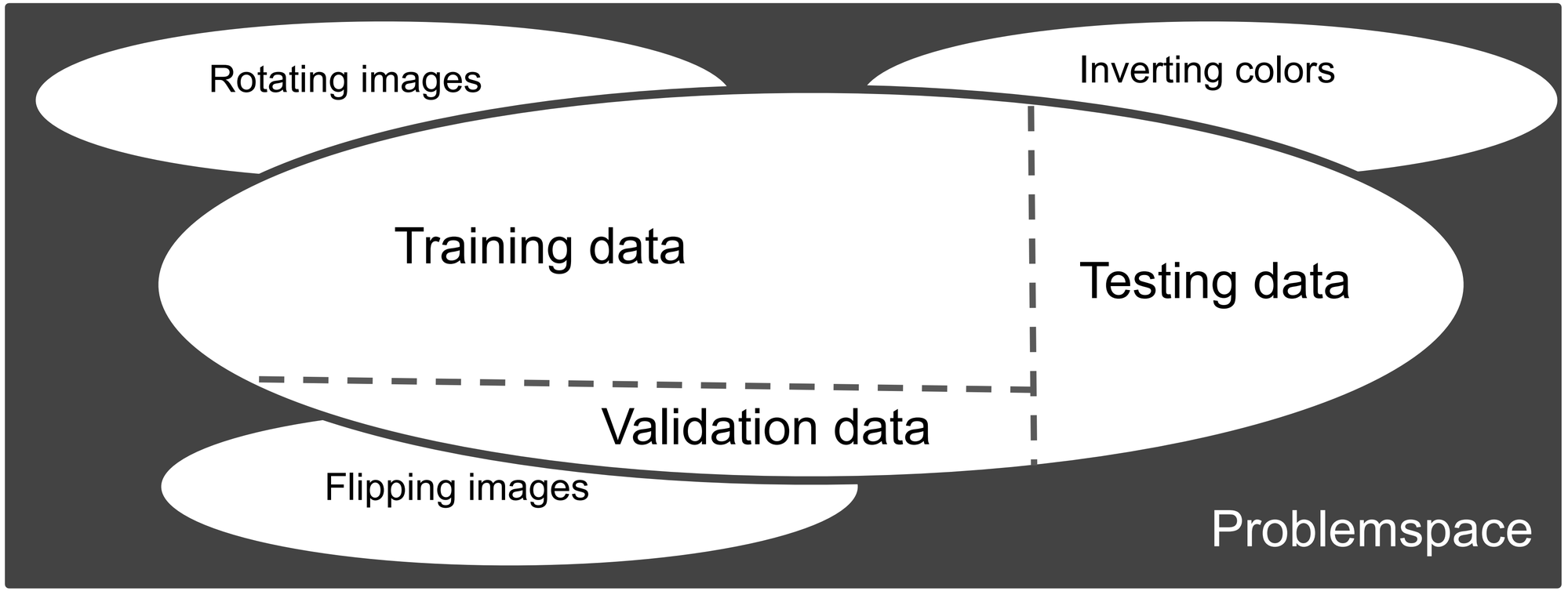


Increase training set & Data augmentation

- **Larger** training set reduce the probability of overfitting
  - **Gather / acquire** more data of the same type
  - **Extend** training set by performing data augmentation
    - **Alter** existing examples in a way that should not affect the target value (the label)
    - For images:
      - Use **transformations** on the examples that does not change the label
      - Mirroring, rotation, brightness etc.

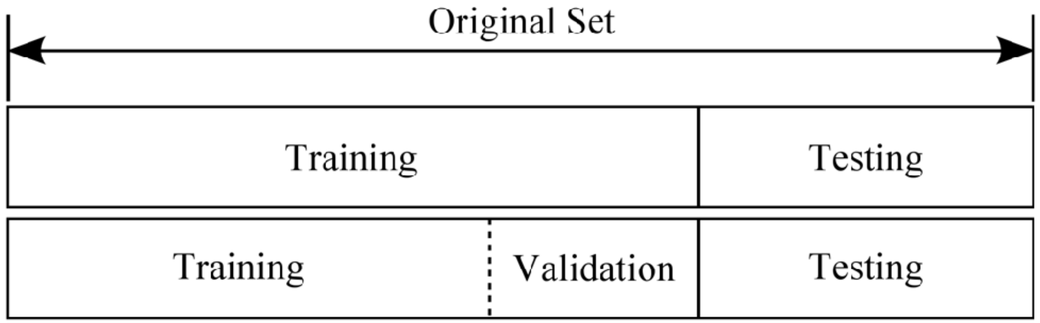


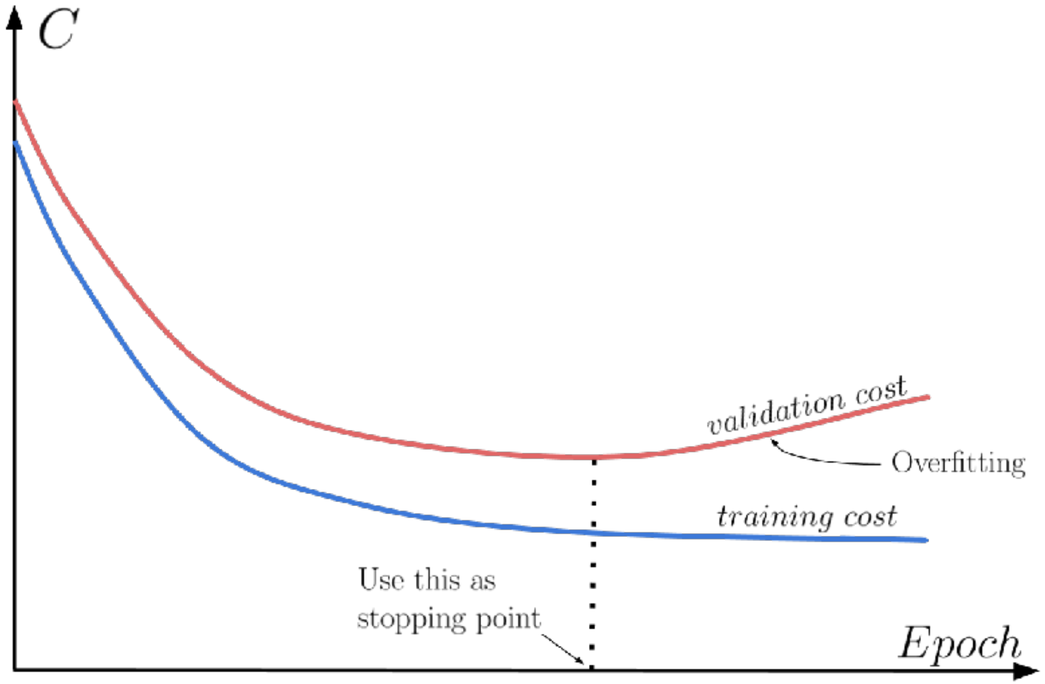
Data augmentation



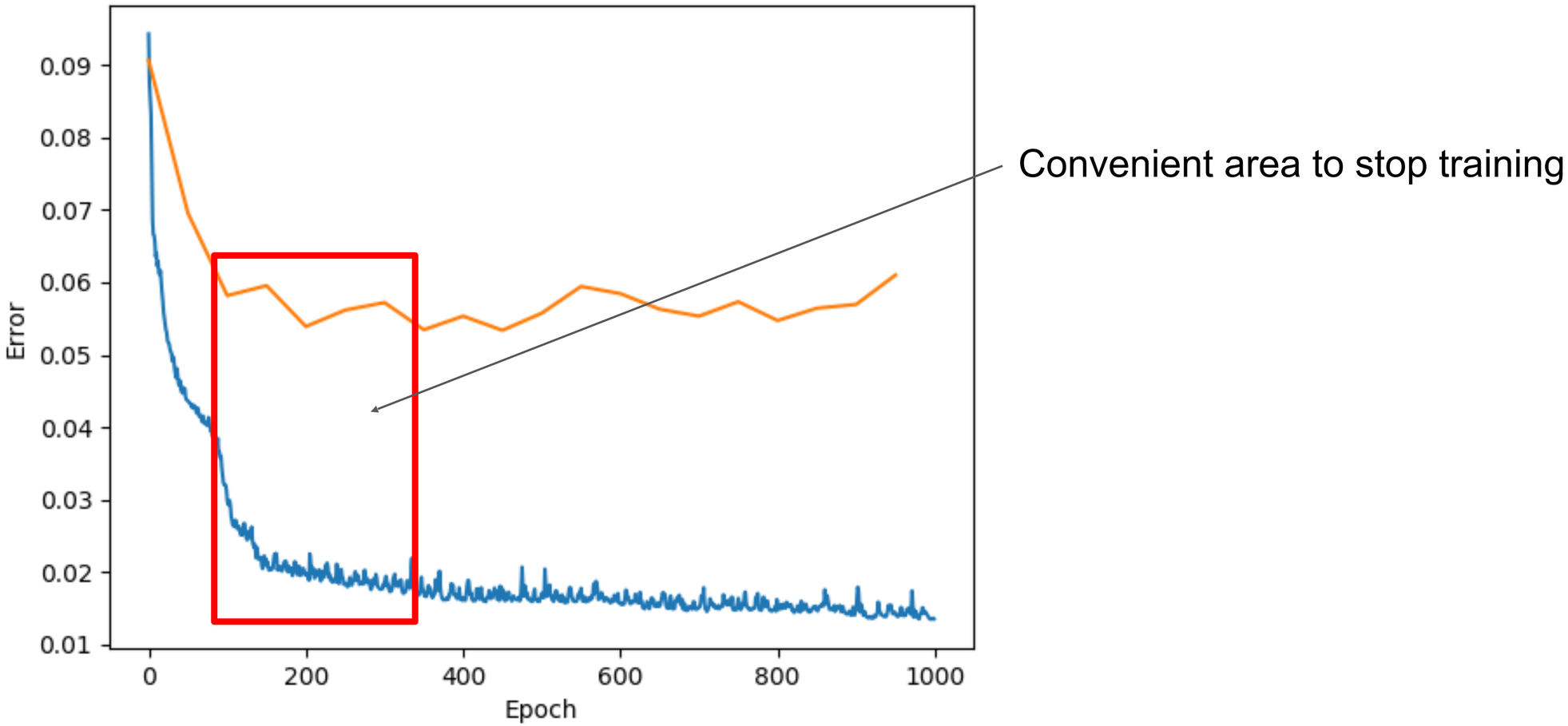
Early stopping

- **Validation set**: set of examples independent from training data
- Compare validation error with training error
- Stop training when accuracy of the validation set has saturated or starts increasing
- When the validation error is at is minimum the model is likely to generalize best





Early stopping



Regularization (Weight decay)

- Add a regularization term (weight restriction) to the cost function.
  - amount of regularization / regularization function
- Technique embedded in the model
- Smaller weights tend to introduce less complex model (large bias = more flexible). Small parameters often better to generalize. Lage parameters is sensitive to noisy data.
  - L2 regularization (for smaller parameters)
  - L1 regularization (leads to sparse parameter vector, can be good for selecting important features)

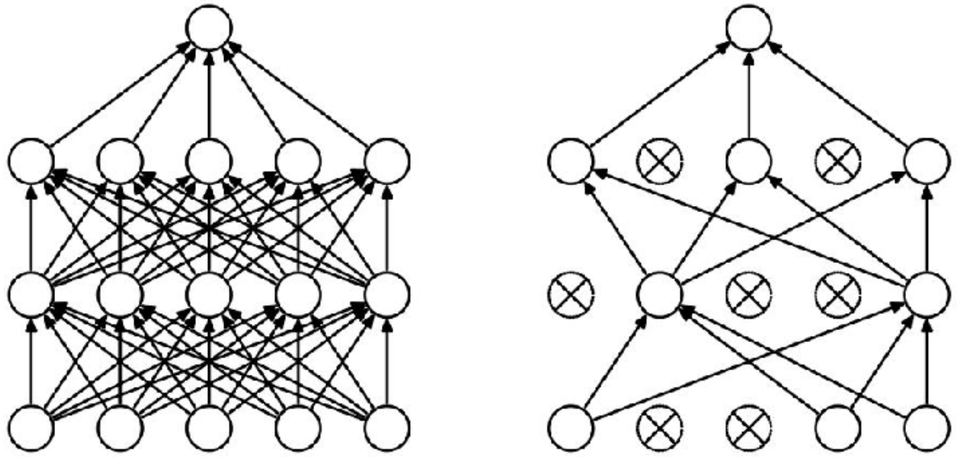
$$\tilde{C}(\theta) = C(\theta) + \alpha\Omega(\theta)$$

$$\Omega(\theta) = ||w||^2$$

$$\Omega(\theta) = ||w||$$

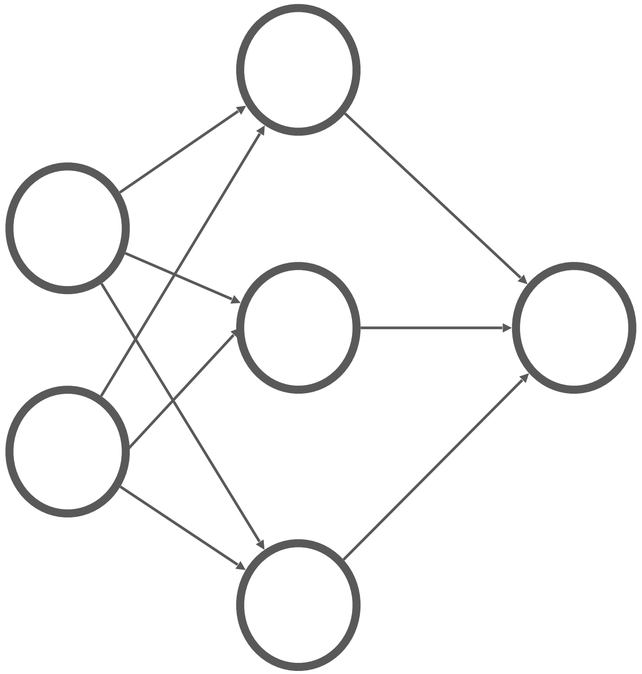
Dropout

- Dropout is to **randomly** select weights that we zero out (”block”) in a **training** step.
- This forces the network to explore other paths through the network to generate the correct output.
- This will slow down convergence
- It can be imagined as training many smaller **sub-networks** of the whole network. As it is trained you get the average of the different network architectures.
- This **generalizes** based on the concept on **averaging**. Instead of one network making the decision, you now have a network that is the combined evaluation of many sub-networks.

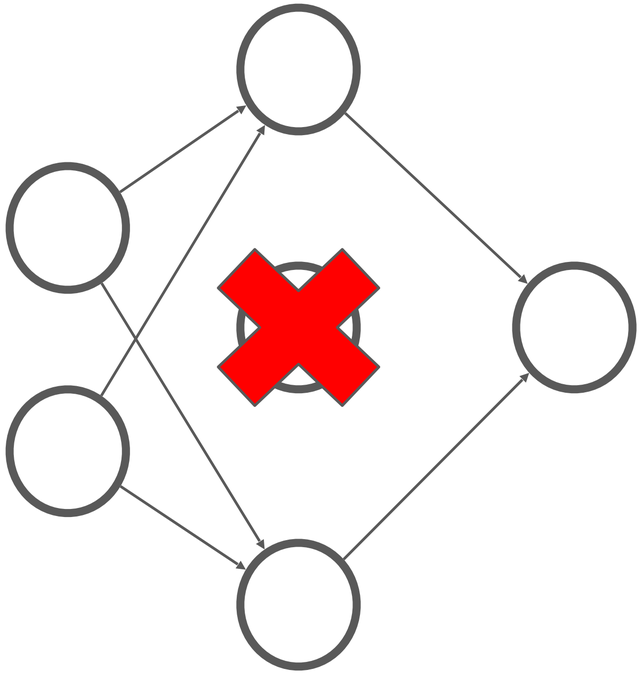


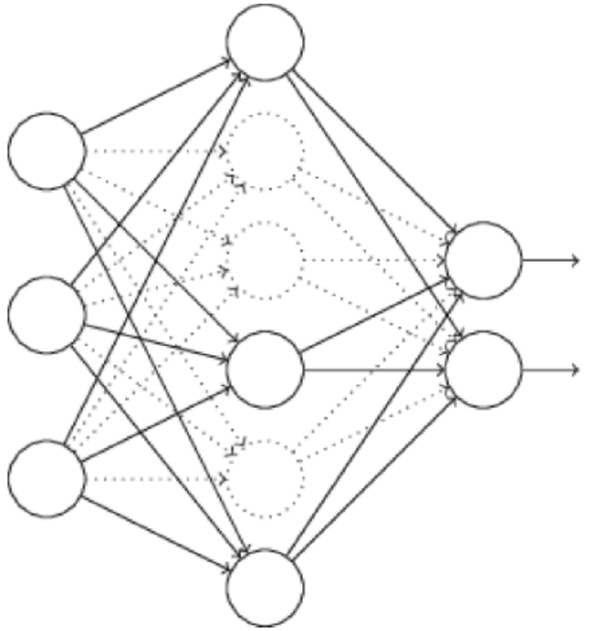
(a) Standard Neural Net (b) After applying dropout.

Dropout

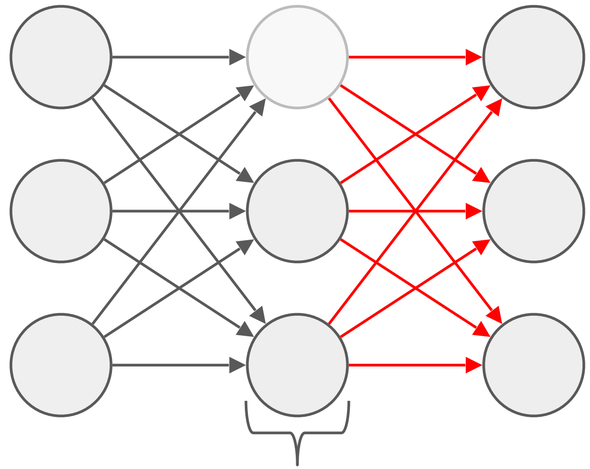


Dropout





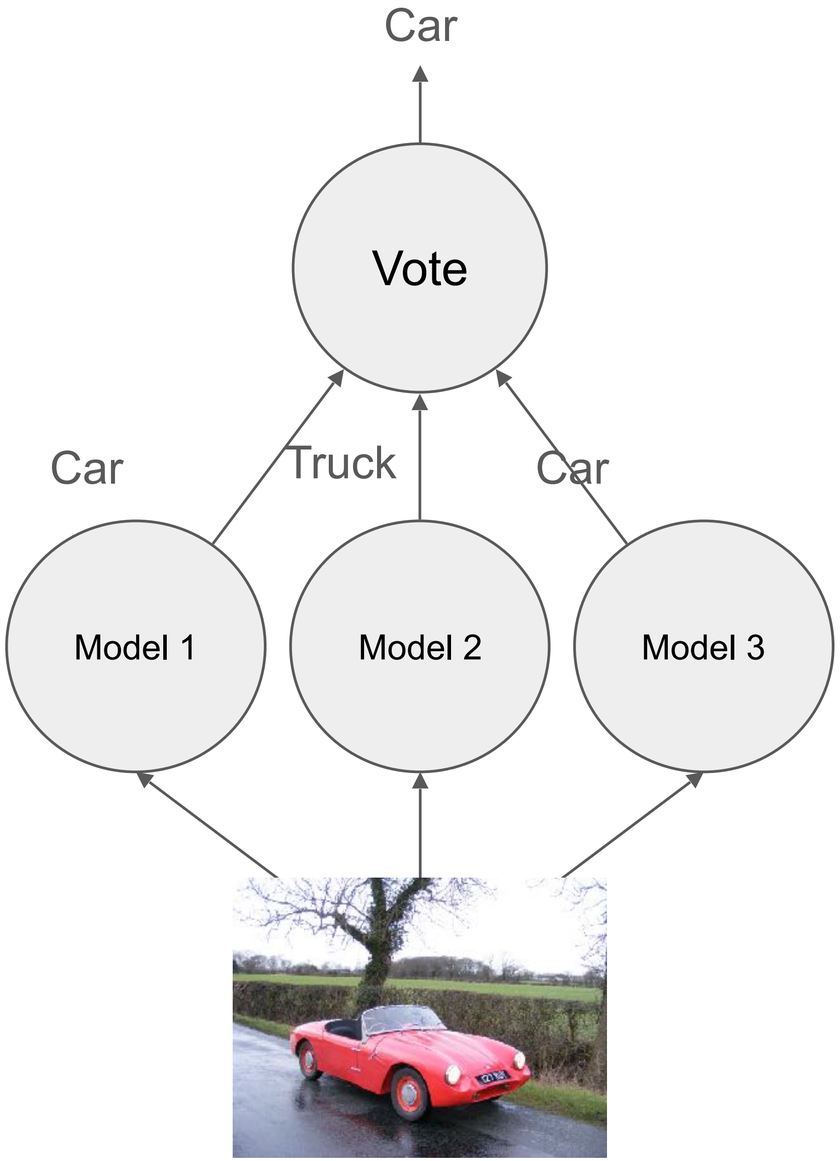
- Deactivate hidden neurons with a small probability p for each training iteration
- During inference, do not dropout neurons
- Makes network more robust (more general)



Dropout layer p=0.33

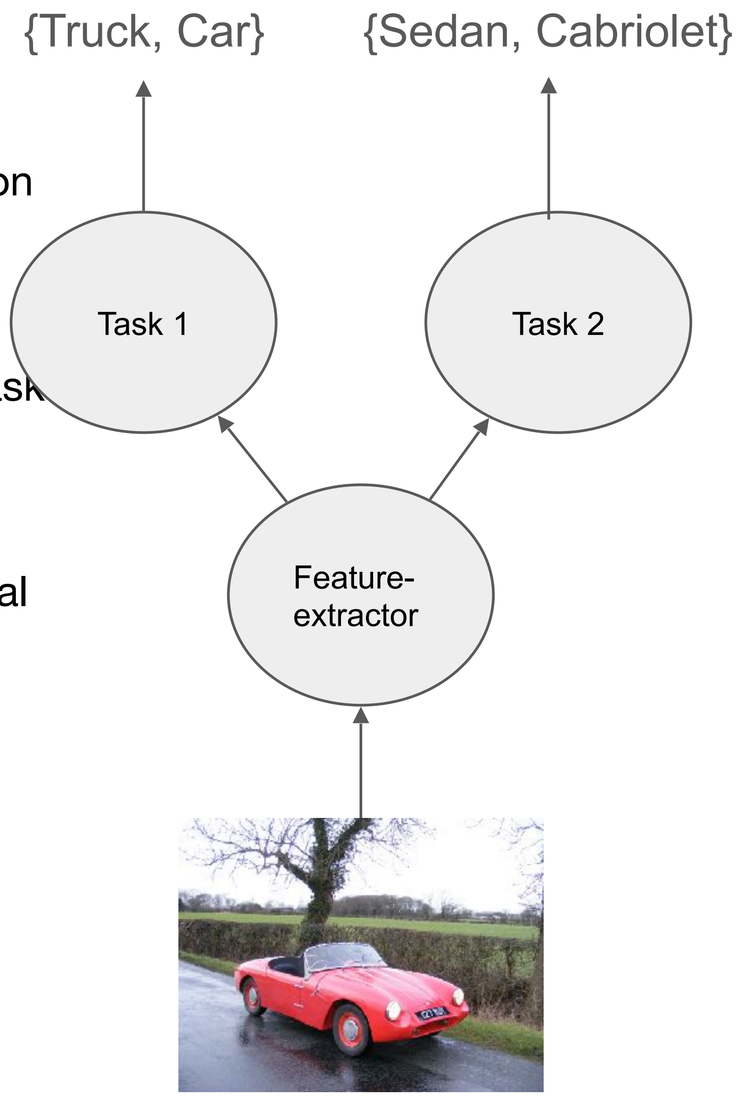
Ensemble learning

- Train several models
  - **Different network architecture**, same training set
  - Same network architecture, **different training sets**
- All models are fead the same test input and predicts a output independently from one another
- The outputs are voted on to produce the final output. Prevent overfitting by averaging output of the networks
- Many voting schemes are possible



Multitask learning

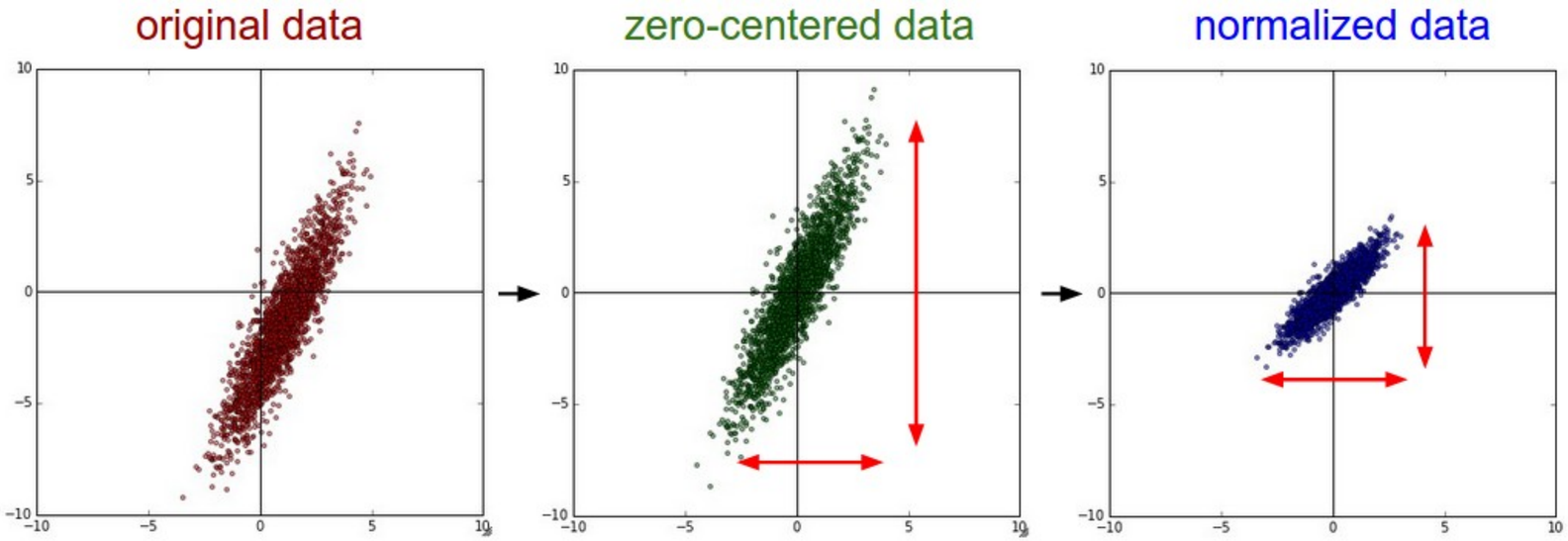
- **Multitask learning**
  - Eg. object detection features needs to be good for both bbox regression and classification
  - Learn more than one task at the same time
  - **Forces** the network to be **more general** since it has to perform well on more than one task
  - Task 1 and Task 2 has a **shared** feature extractor.
  - Forces the feature extractor to learn feature that are useful for both tasks, i.e more general features
- Too many parameters (high model capacity)
  - Reduce the number of parameters
  - CNN's share parameters across spatial locations
    - Forces the network to learn general filters across spatial locations



Preprocessing of Data

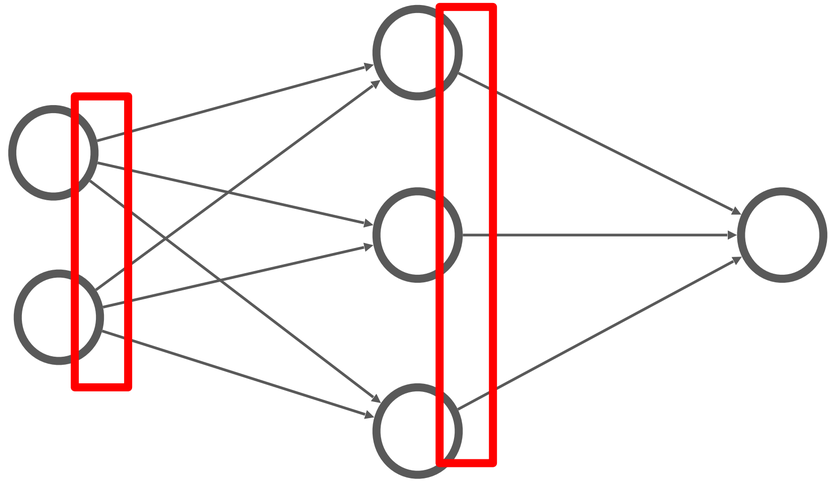
- (Cleaning and structuring)
- **Zero-Centering** of the data
  - Subtract mean image
  - Subtract mean of each channel
- **Normalization** of the data - divide by the standard deviation

Garbage in = garbage out



Batch normalization

- Normalizes the output of a previous activation layer by subtracting the batch mean and dividing by the batch standard deviation.
- Adds two trainable parameters to each layer, a “standard deviation” parameter and add a “mean” parameter
- Batch normalization allows each layer of a network to learn by itself a little bit more independently of other layers.
- Reduces overfitting
- Improves stability, and allows for higher learning rates



Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

Batch normalization

- Normalize the $z^{(j)}$ across the batch, j node in a layer l
  1. Normalize $z^{(j)}_{norm} = \frac{z^{(j)} - \mathrm{E}[z^{(j)}]}{\sqrt{\mathrm{Var}(z^{(j)})}}$, this is differentiable
  2. Scale $\hat{z}^{(j)} = \gamma^{(j)} z^{(j)}_{norm} + \beta^{(j)}$, undo norm: $\gamma^{(j)} = \sqrt{\mathrm{Var}(z^{(j)})}$ and $\beta^{(j)} = \mathrm{E}[z^{(j)}] \Rightarrow \hat{z}^{(j)} = z^{(j)}$
     - $\gamma^{(j)}$ and $\beta^{(j)}$ are learned with backpropagation
  3. Activate $a^{(j)} = f(\hat{z}^{(j)})$
- At test time the calculate the mean and the variance over all batches
- Can also normalize $a^{(j)}$, but it is more common to normalize $z^{(j)}$
- Makes the distribution of activations across batches more similar
- Regularization effect through noisy mean and variance estimates

(Parameter) Initialization

- Initialize to different values such that neurons do not learn the same pattern
- Random **initialization**
  - Not choose too large weights: makes neurons saturate and learning slow
  - Not choose too small weights: results in small values and learning slow
  - Weights ideally quite small and both positive and negative
    - Values of output weights of a neuron might be determined by number of input neurons

(Parameter) Initialization

- The initialization of the weight parameters affects the speed of training and may lead to early termination by saturating activation functions early
- If weights are too small, the input signal will shrink as it passes through the layers, until it is too weak to be useful for activation
- If weights are too large, the input signal will grow as it passes through the layers, until it is too strong to be useful (activates everything)
- Empirical results show that drawing weights from a normal distribution with mean 0 and low variance (< 1.0) creates good results

Xavier initialization

- Glorot and Bengio suggested in 2010 to draw initial weights in a layer from a normal distribution with mean 0 and variance equal to

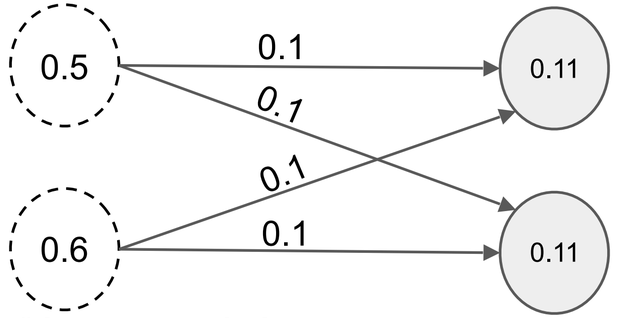
$$Var[W_i] = \frac{1}{n_{in} + n_{out}}$$

where Wi are the initial weights of a layer and nin and nout are the number of neurons connected from the previous and next layer respectively

- Strong empirical results

Initialization (more detail)

- If the input is normalized the trained weights is expected $\mathbf{E}(w_{ji}^l) = 0$
- $w_{ji}^l = c^l \Rightarrow a_j^l = a_k^l \Rightarrow \delta_j^l = \delta_k^l$
- To break this symmetry we use random initialization
  - E.g $w_{ji}^l \sim \mathcal{N}(0, 1^2)$
- If we initialize the weights as above, and $a_i^l = 1$



  - $\mathrm{Var}(z_j^l) = \sum_i^n \mathrm{Var}(w_{ji}^l) = n_{in}\,\mathrm{Var}(w_{ji}^l)$, so $\mathrm{Var}(z_j^l)$ grows with the number of input weights
  - This means that the probability of saturation increases with the number of input weights
  - To account for this we divide each weight with $\sqrt{n_{in}}$, this is equivalent to setting $\mathrm{Var}(w_{ij}) = \frac{1}{n_{in}}$
  - $\mathrm{Var}(z_j^l) = \sum_i^n \mathrm{Var}\left(\frac{1}{\sqrt{n}} w_{ji}^l\right) = \frac{1}{n}\sum_i^n \mathrm{Var}\left(w_{ji}^l\right) = \mathrm{Var}(w_{ji}^l) = 1 \Rightarrow z_j^l \sim \mathcal{N}(0, 1^2)$, this is Xavier initialization
- Other initializations
  - $\mathrm{Var}(w_{ij}^l) = \frac{2}{n_{in}+n_{out}}$, tries to account for the same effect for backprop - Glorot initialization
  - $\mathrm{Var}(w_{ij}^l) = \frac{2}{n_{in}}$, used with ReLU since it is 0 in half of the cases
- Biases is normally initialized to 0

Hyper-parameters

- Parameters of the model (set before training)
- Proper learning depends on many hyper-parameters (control the behavior of the training and has a big influence on the model's performance)
  - Learning rate
  - Number of hidden layers
  - Number of neurons in each hidden layers
  - Mini-batch size
  - Number of training epochs
  - Regularization parameter
  - Hyper-parameters related to CNNs
  - +++
- **Initialization and optimization**

Hyper-parameters (2)

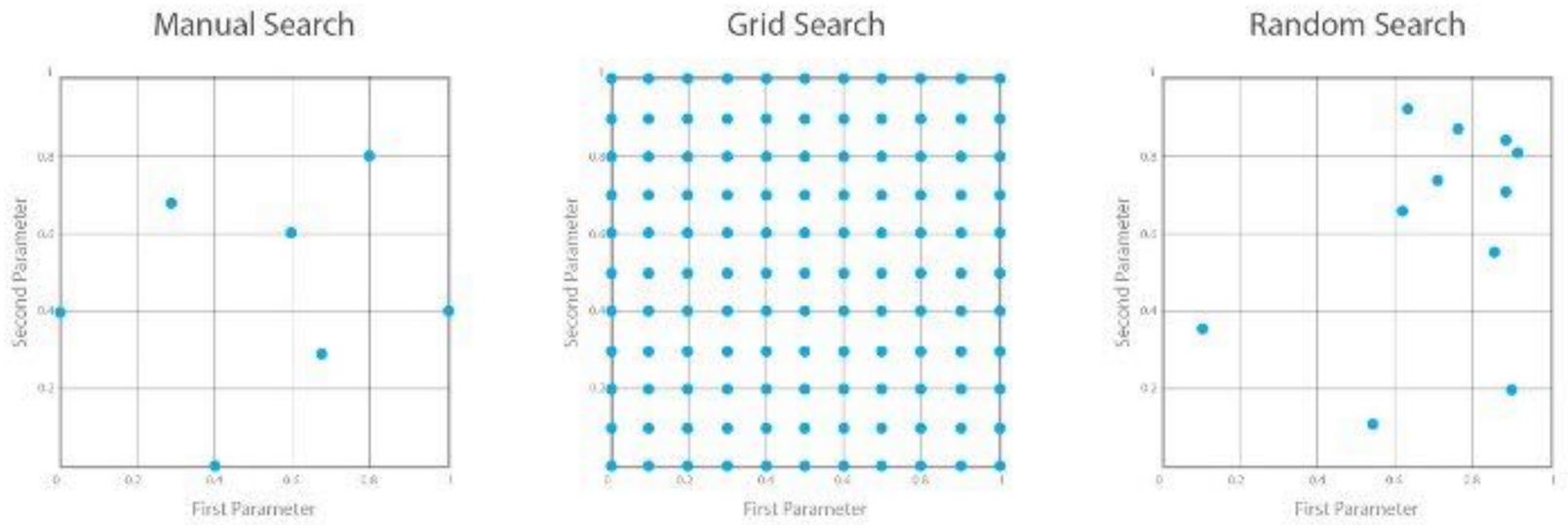
- Obtaining values for these might be challenging
- How to **set** these to appropriate values?
  - More rapid **experimentation**
    - Reduce the complexity of the task/model
    - Monitor performance more frequently
    - Small validation set
    - Incremental improvements

Hyper-parameters (3)

- How to set these to appropriate values?
  - Learning rate
    - Experiment with different learning rates
    - Estimate order of magnitude (largest learning rate that decreases cost)
    - Overshoot vs slow training
    - Learning rate schedule
  - Number of epochs
    - Early stopping (stop training when no more improvement)
  - Regularization parameter
    - Start with no regularization, obtain learning rate
    - Introduce regularization, improve learning rate
  - Mini-batch size
    - Less dependent of other hyper-parameters
    - Hardware utilization vs frequent updates

Hyper-parameters (4)

- How to set these to appropriate values?
  - Automated procedures for finding hyperparameters
    - Grid search / Random search



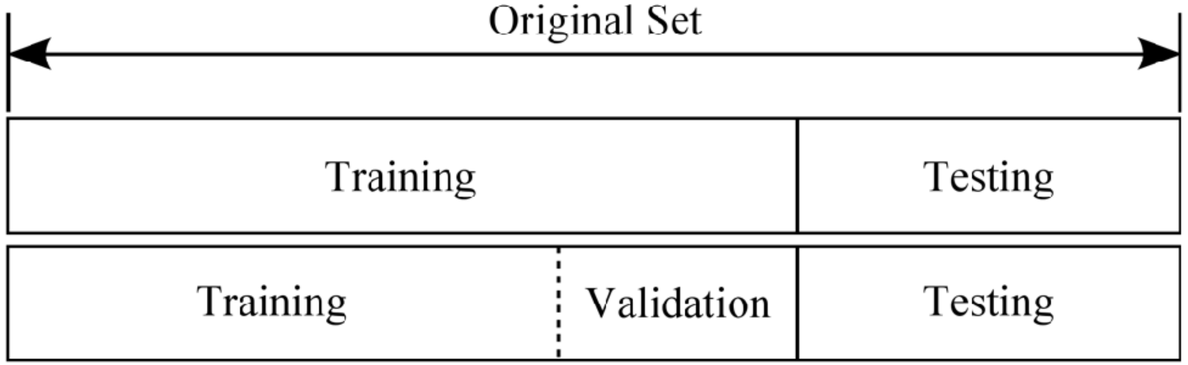
Hyperparameter optimization is the process of finding the best values for the hyperparameters:

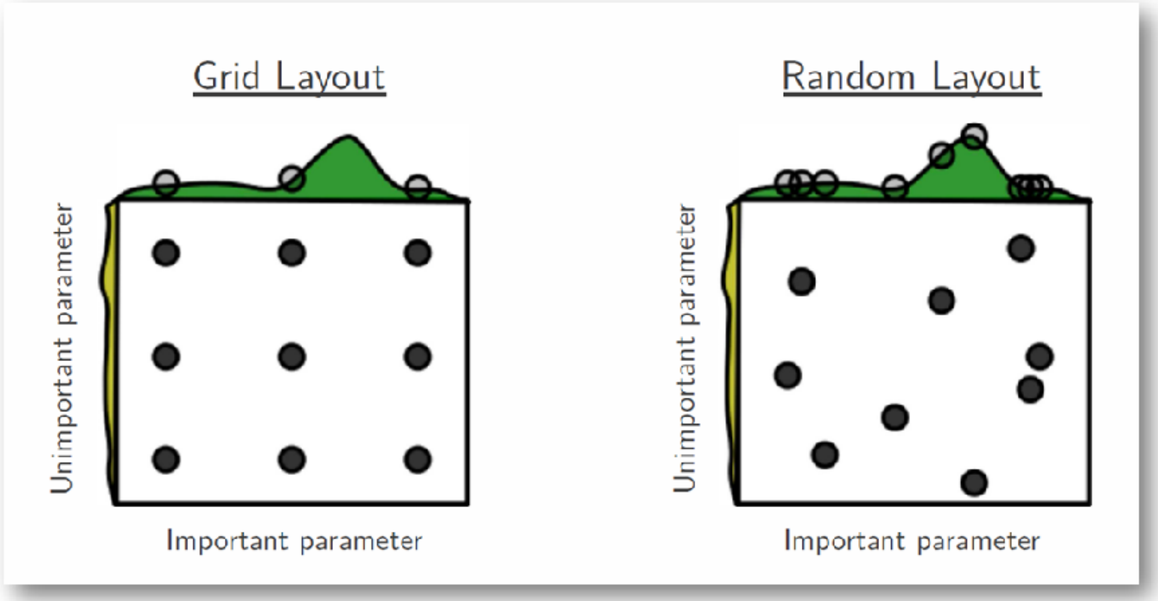
Some optimization algorithms:

- **Grid search:** Brute force approach that trains the algorithm for all combinations of learning rate and number of layers. It then uses cross validation to calculate the run with the best hyperparameter values.
- **Random Search:** Randomly samples the search space. Instead of testing *all* combinations it randomly picks out a sub set.
- **Automatic hyperparameter tuning:** Instead of brute forcing or randomly guessing it computes relationships between the hyperparameter values and model performance, to make smarter choices for the next parameter settings.

Hyper-parameters (5)

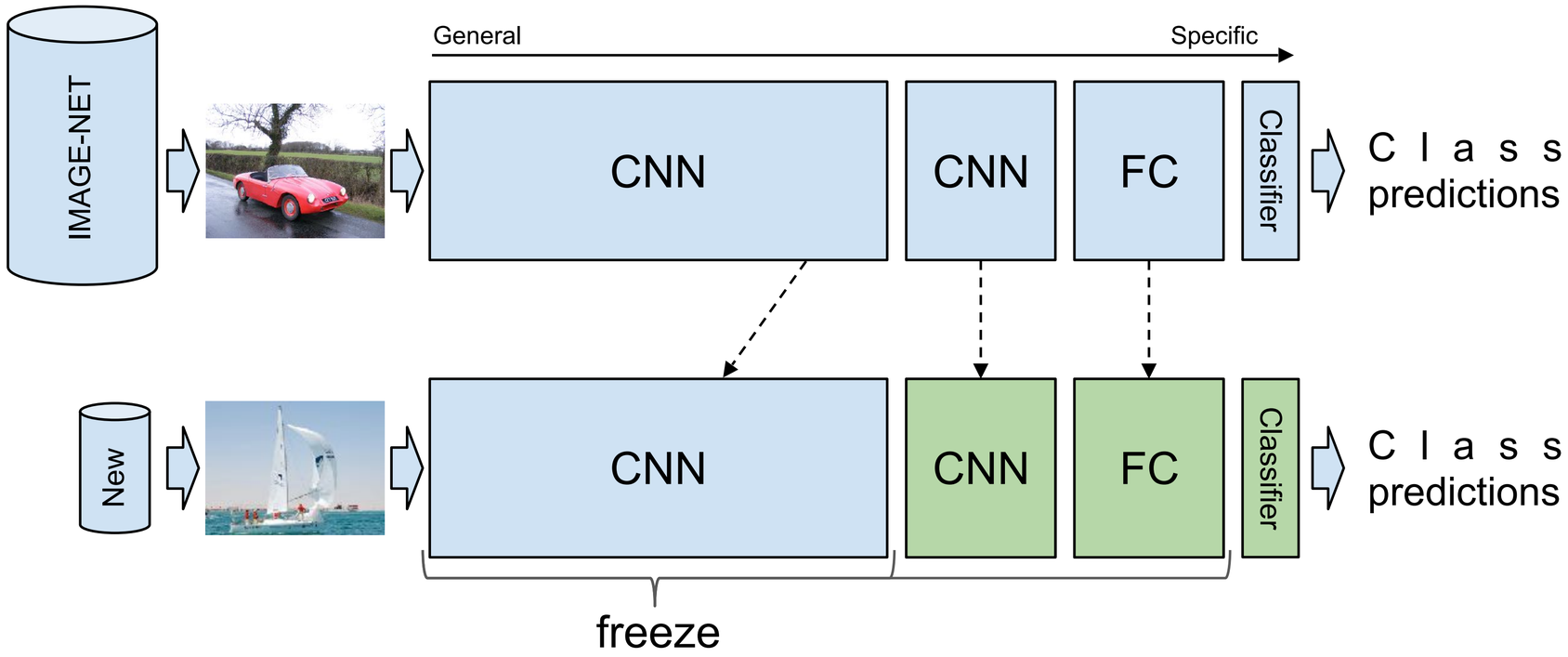
- Many Hyper-Parameters
  - Learning rate
  - Dropout rate
  - Amount of augmentation
  - etc
- To choose the best one based on validation set
- To ways of automate the search
  - Grid search
  - Random search





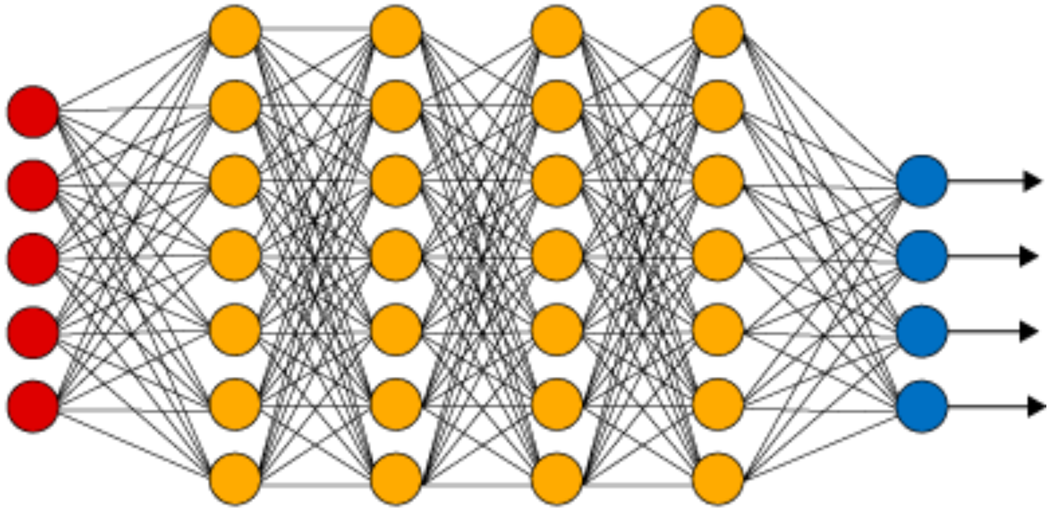
Transfer learning

- Pre-train on large available dataset, e.g. image net
- Using the pre-trained network we can
  - train a classifier, e.g. a SVM- or softmax-classifier, on the features it extracts
  - fine-tune the network with the task specific data



Challenges with Deep Neural Networks

- Why is training of deep neural networks hard?
  - **Vanishing** gradient descent problem
    - Gradients tend to become **smaller** as we move backward
      - Earlier layers learn slower
  - Exploding gradients
  - Unstable gradients



Next time..

CNN & Image Classification