7 - Optimizing Networks

Ludwig Krippahl

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Optimization

Summary

- Optimizers
- Learning rate
- Initializing weights
- Overfitting and model selection
  - Bias/Variance tradeoff
- Regularization methods in ANN

Optimizers

Optimizers

Minimizing the loss function

- We want to minimize the loss function (e.g. cross-entropy for ML) to obtain θ from some data
- Numerical optimization is outside the scope of this course
  - But it's useful to have some knowledge of the optimizers

Optimizers

Minimizing the loss function

- So far we saw `tf.optimizers.SGD`
  - Basic gradient descent algorithm, single learning rate.
  - Stochastic gradient descent: use gradient computed at each example, selected at random
  - Mini-batch gradient descent: updates after computing the total gradient from a batch of randomly selected examples.
  - Can include momentum (and you should use momentum, in general)
- This is just an alias for the `tf.keras.optimizers.SGD` class
  - We'll be using Keras explicitely from now on

Optimizers

Minimizing the loss function:

- Different parameters may best be changed at different rates

- `tf.keras.optimizers.Adagrad`
- Keeps sum of past (squared) gradients for all parameters
- Divides learning rate of each parameter by this sum
- Parameters with small gradients will have larger learning rates, and vice-versa
- Since Adagrad sums previous gradients, learning rates will shrink
- (good for convex problems)

Optimizers

Minimizing the loss function:

- Some parameters may be left with too large or too small gradients

- `tf.keras.optimizers.RMSProp`
- Keeps moving root of the mean of the squared gradients (RMS)
- Divides gradient by this moving RMS
- Updates will tend to be similar for all parameters.
- Since it uses a moving average, learning rates don't shrink
- Good for non-convex problems, and often used in recurrent neural networks
- Most famous unpublished optimizer

Optimizers

Minimizing the loss function

- `tf.keras.optimizers.Adam`
  - Adaptive Moment Estimation (Adam)
  - Momentum and different learning rates using an exponentially decaying average over the previous gradients
  - Fast to learn but may have convergence problems

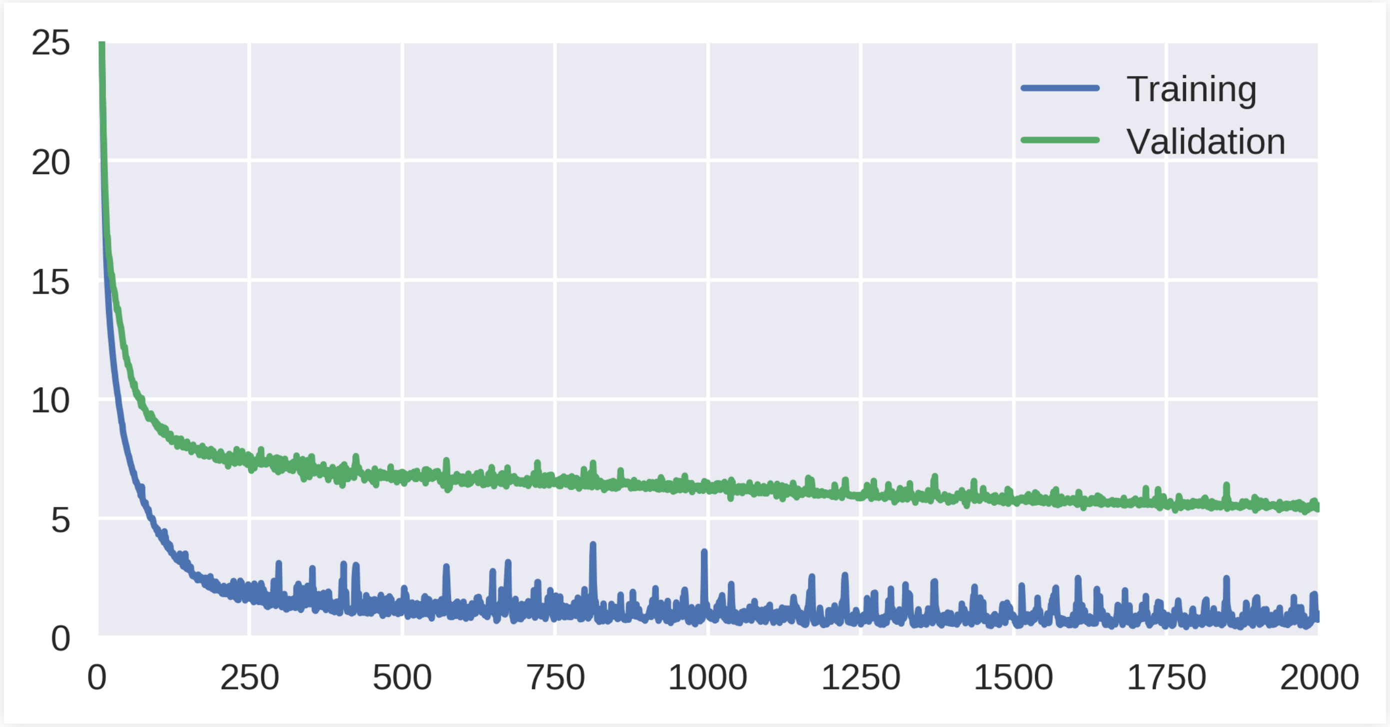
How to choose?

- There is no solid theoretical foundation for this
- So you must choose empirically
  - Which is just a fancy way of saying try and see what works...

Learning Rate

Choosing the best learning rate

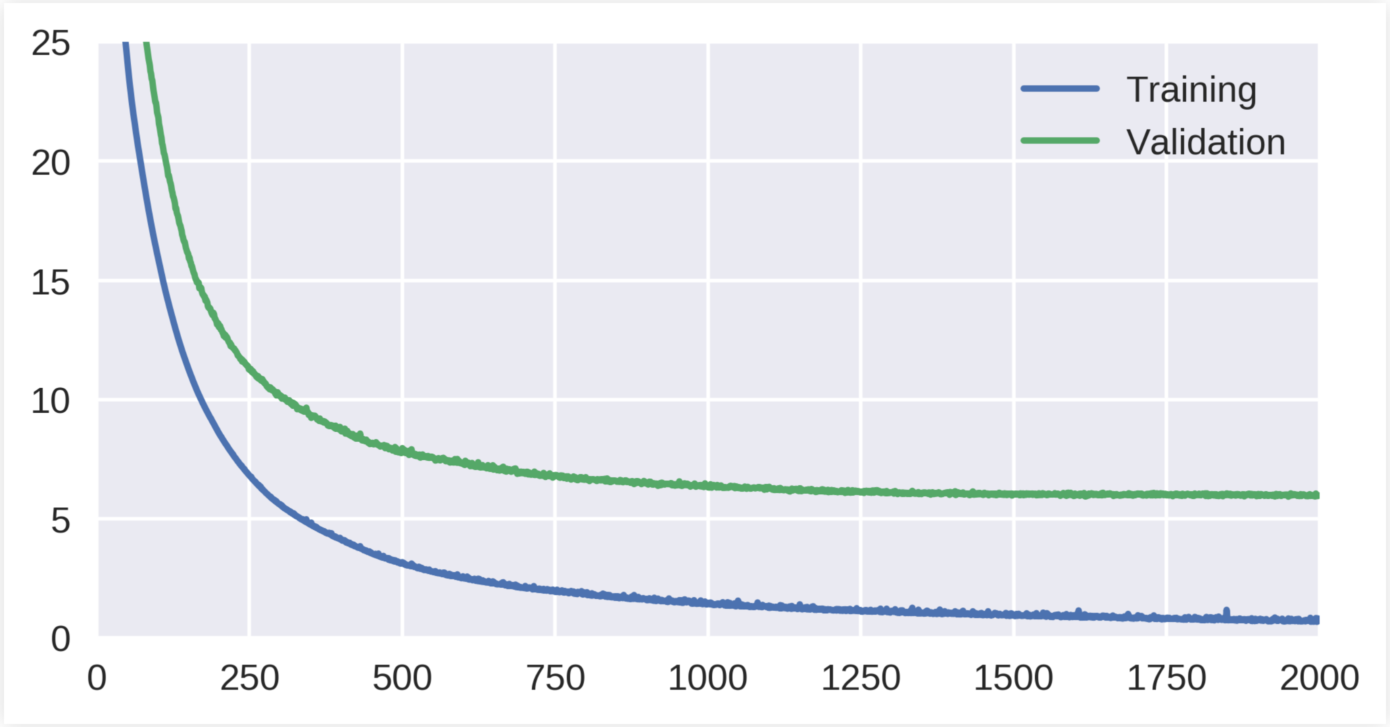
- Optimizers can have other parameters, but all have a learning rate
- Too high a learning rate can lead to convergence problems



FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Learning Rate

- However, if learning rate is too small training can take too long
- Try to make it as high as you can while still converging to low error
  - (you can experiment with a subset of your training set, even if overfitting)



Weight Initialization

Selecting the initial values

- Bias: these can start at 0
  - (or use some heuristic)
- Weights: must break symmetry, cannot all have same value
- But if they add up to large values it may cause problems
  - LeCun normal initializer:

$$W_{i,j} \sim N\left(0, \sqrt{\frac{1}{fan_{in}}}\right)$$

  - Normalized initialization (`glorot_uniform` in Keras):

$$W_{i,j} \sim U\left(-\sqrt{\frac{6}{fan_{in}+fan_{out}}}, \sqrt{\frac{6}{fan_{in}+fan_{out}}}\right)$$

- See: https://keras.io/initializers/

Batch Normalization

Normalizing (standardizing) activations

- Compute running averages and standard deviations during training
  - And standardize the inputs to each layer
- Just like we do for the inputs to the network, do for hidden layers too
  - Makes learning easier by preventing extreme values
  - Eliminates shifts in mean and variance during training
  - Reduces the need for each layer to adapt to the changes in the previous one
- This can be done easily in Keras
  - The mean, standard deviation and rescaling can all be part of backpropation
  - AutoDiff takes care of the derivatives
  - So we can add batch normalization as an additional layer

Overfitting and Validation

Overfitting and Validation

The goal of (supervised) learning is prediction

- And we want to predict outside of what we know

Overfitting

- The problem of adjusting too much to training data
  - and losing generalization
- Two ways of solving this:
  - Select the right model: model selection
  - Adjust training: regularization

Overfitting and Validation

How to check for overfitting

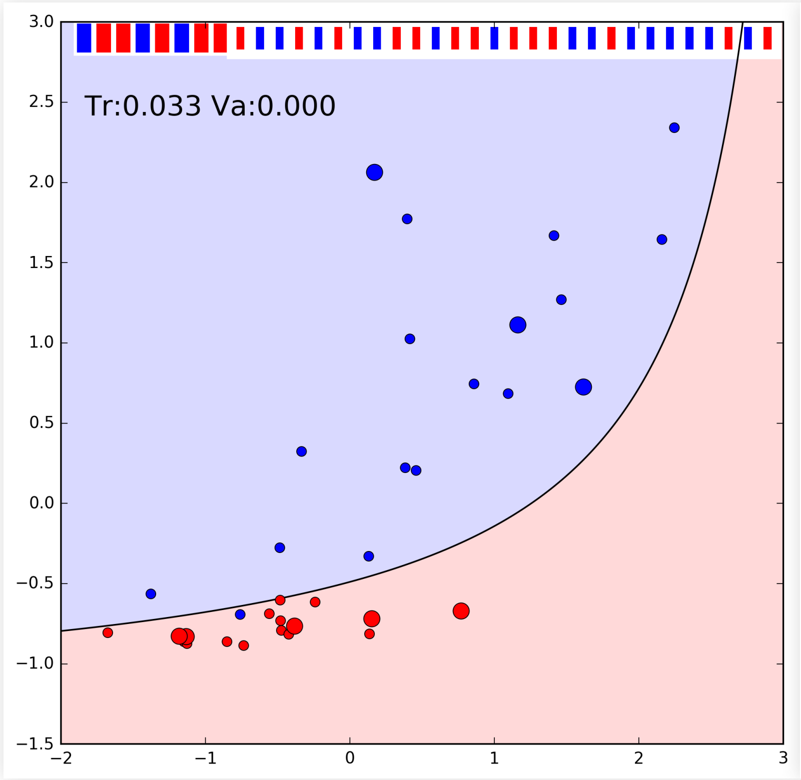
- We need to evaluate performance outside the training set
- Test set: we need to keep this for final evaluation of error rate
- We can use a validation set
- Or we can use cross-validation

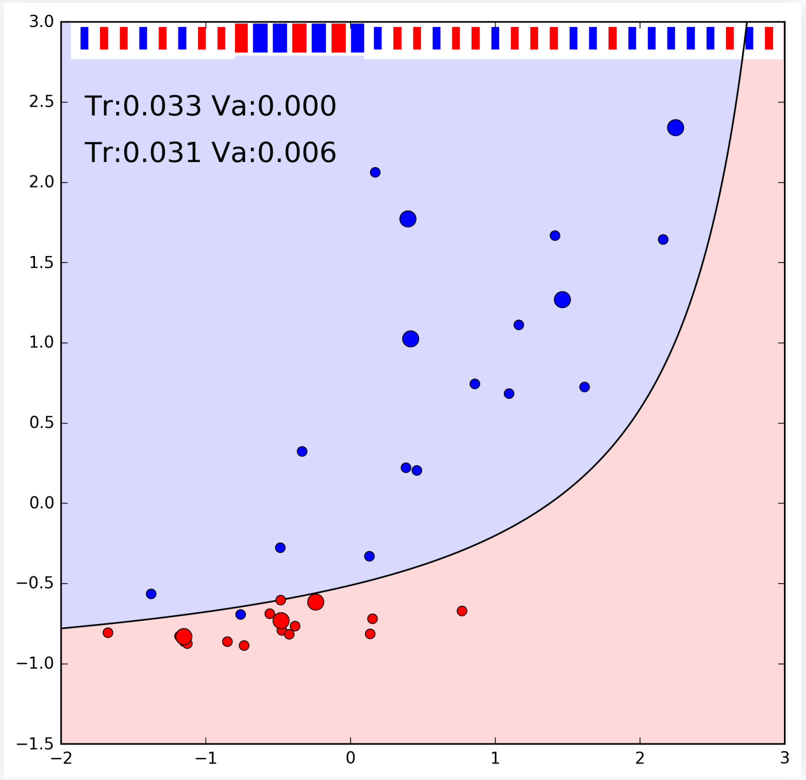
Overfitting and Validation

How to check for overfitting

- Cross-Validation:

• Split training set into K folds, average validations training on the k-1





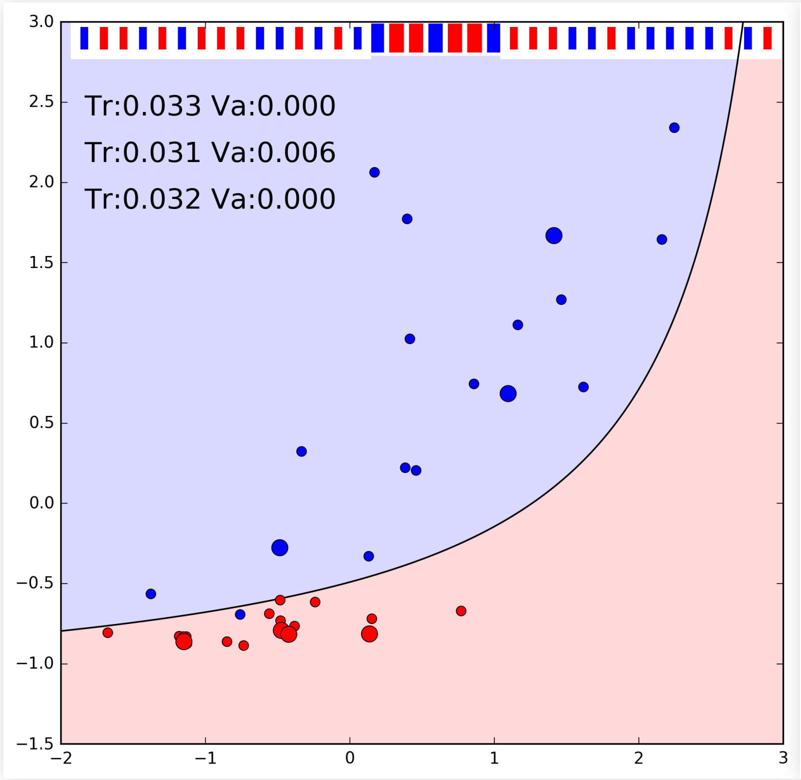
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

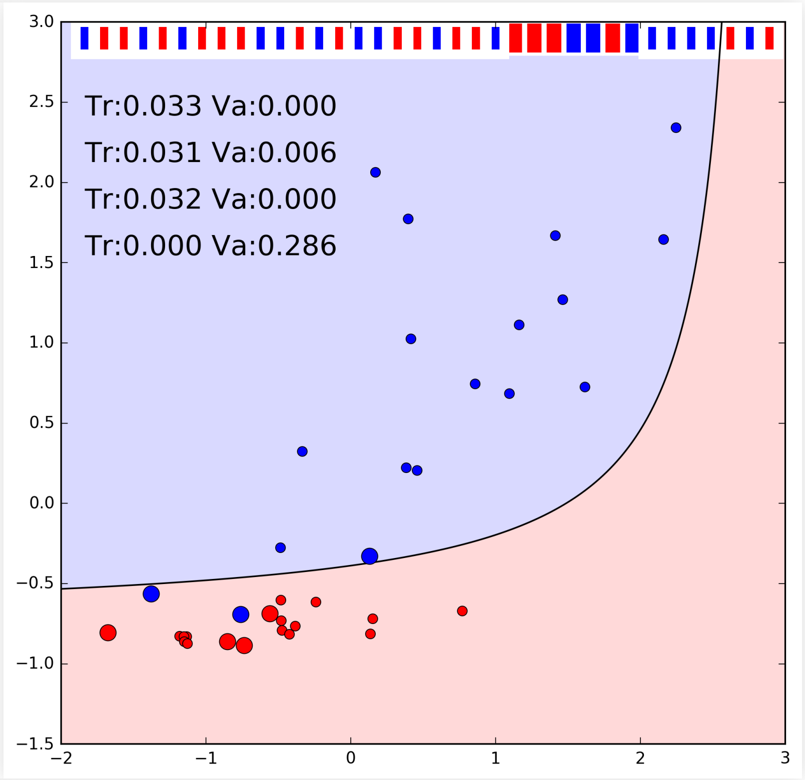
Overfitting and Validation

How to check for overfitting

- Cross-Validation:

• Split training set into K folds, average validations training on the k-1





FACULDADE DE CIÊNCIAS E TECNOLOGIA UNIVERSIDADE NOVA DE LISBOA

Overfitting and Validation

How to check for overfitting

- Option 1: train, validation for preventing overfitting, test
  - Good when we have lots of data (which is generally the case for DL)
- Option 2: Cross-validation on training set, test
  - Good when data is scarcer
  - Better estimate of true error
  - More computationally demanding
- Cross-validation is widely used outside deep learning
- With deep learning training and validation is more common
  - Deep networks take some time to train

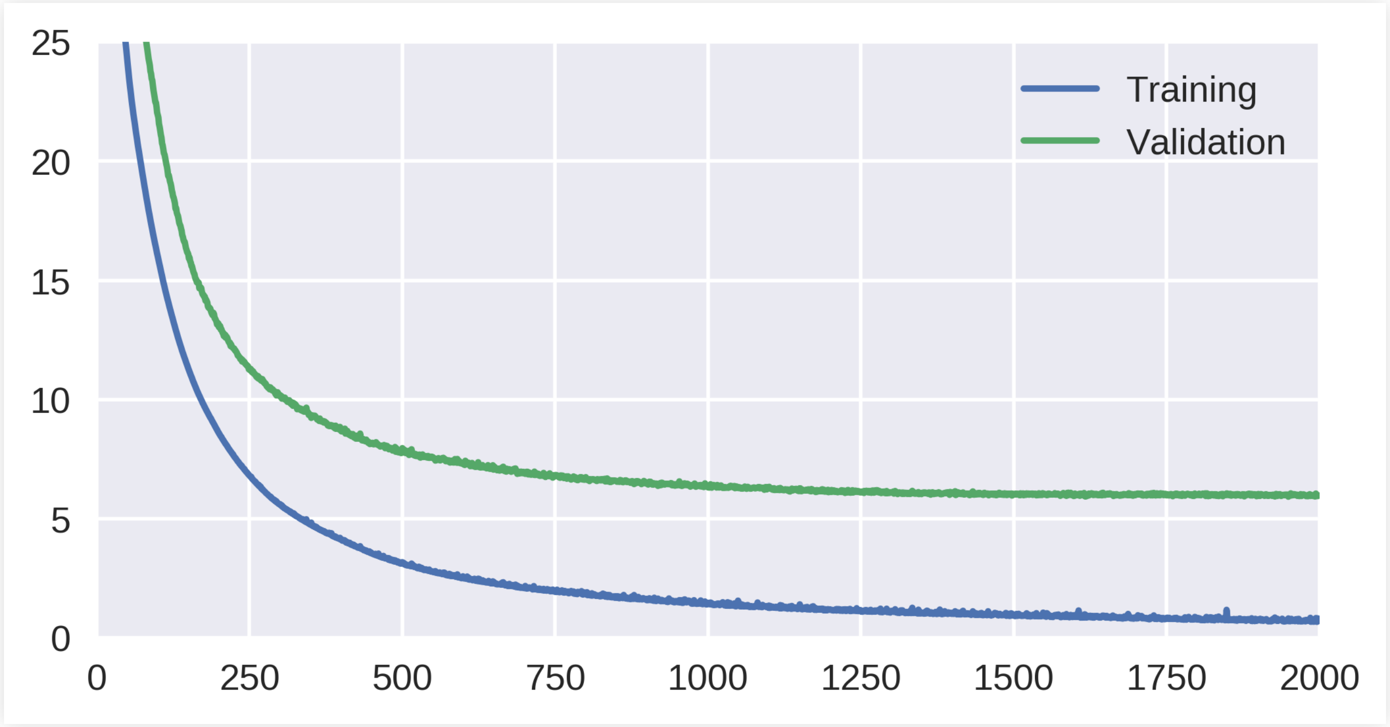
Estimating the true error

- True error: the expected error over all possible data
  - We cannot measure this, since we would need all possible data
- Must be estimated with a test set, outside the training set
- This cannot be the validation set if the validation set was used to optimize hyperparameters
  - We choose the combination with the smallest validation error, this makes the estimate biased.
- Solution: reserve a test set for final estimate of true error
  - This set should not be used for any choice or optimization

Overfitting

Model Selection

- If the model adapts too much to the data, the training error may be low but the true error high
  - Example: Auto MPG problem, 100-50-10-1 network.

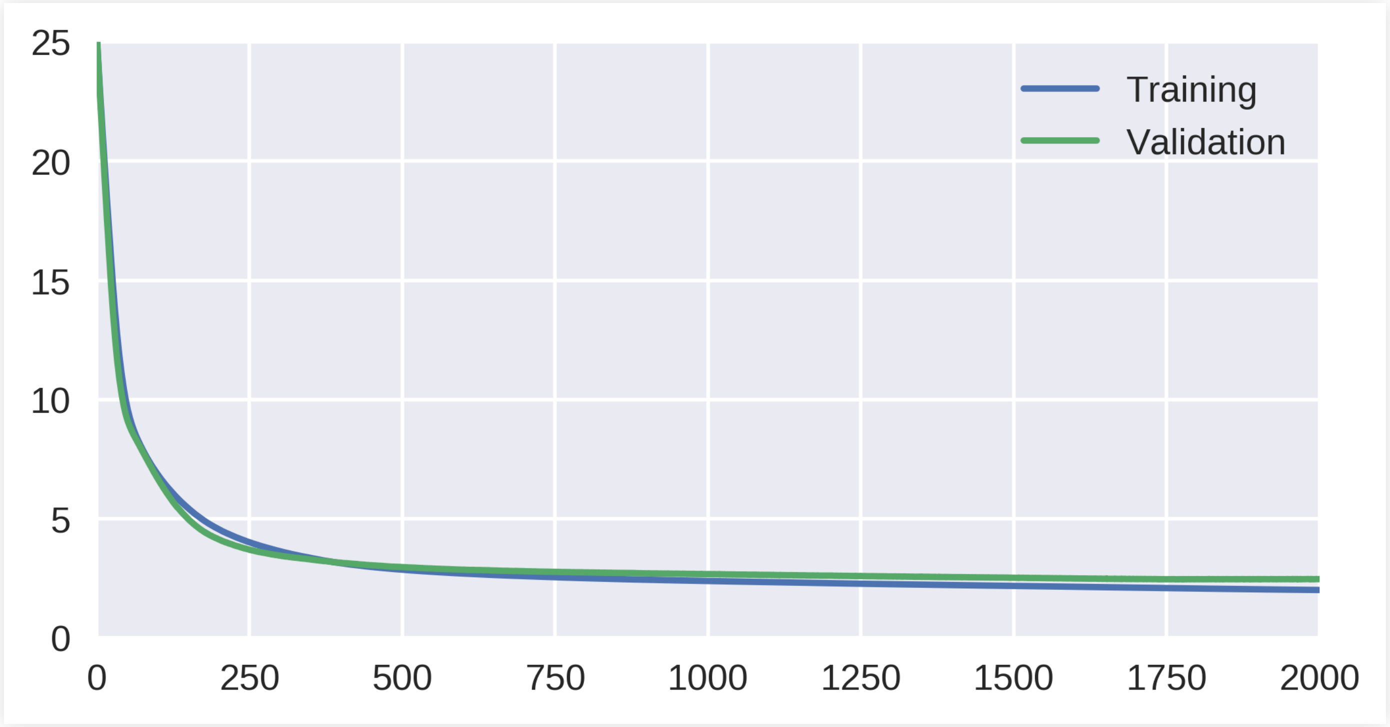


Overfitting

Model Selection

- One way of solving this problem is to use a simpler model (assuming it can fit the data)

• Example: Auto MPG problem, 30-10-1 network.



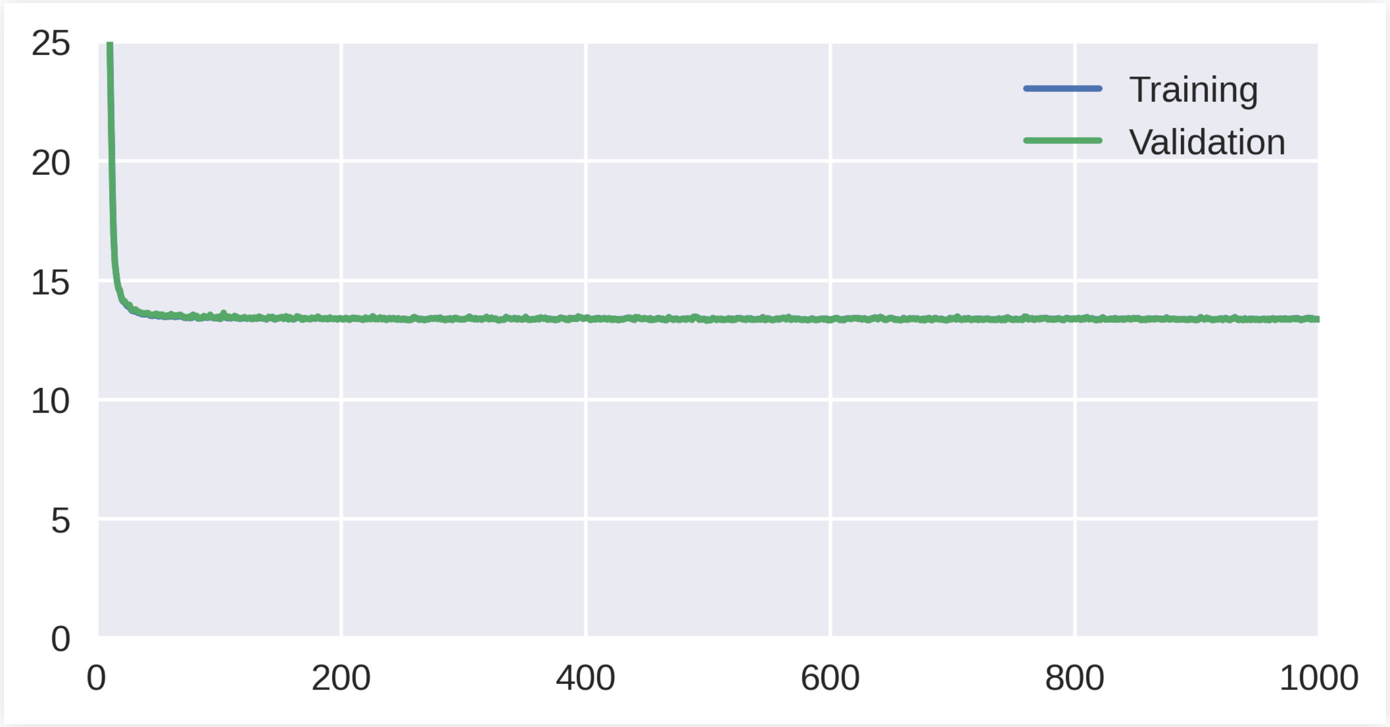
Overfitting

Model Selection

- If the model is too simple, then error may become high

• (Underfitting)

• Example: Auto MPG problem, 3-2-1 network.



Practical Issues

Bias and Variance

Bias and Variance

- Suppose we could train our model on many data sets
  - For each hypothesis, predict target value for one example
- We can decompose the error in two components:

Bias

- Deviation of the average estimate from the target value

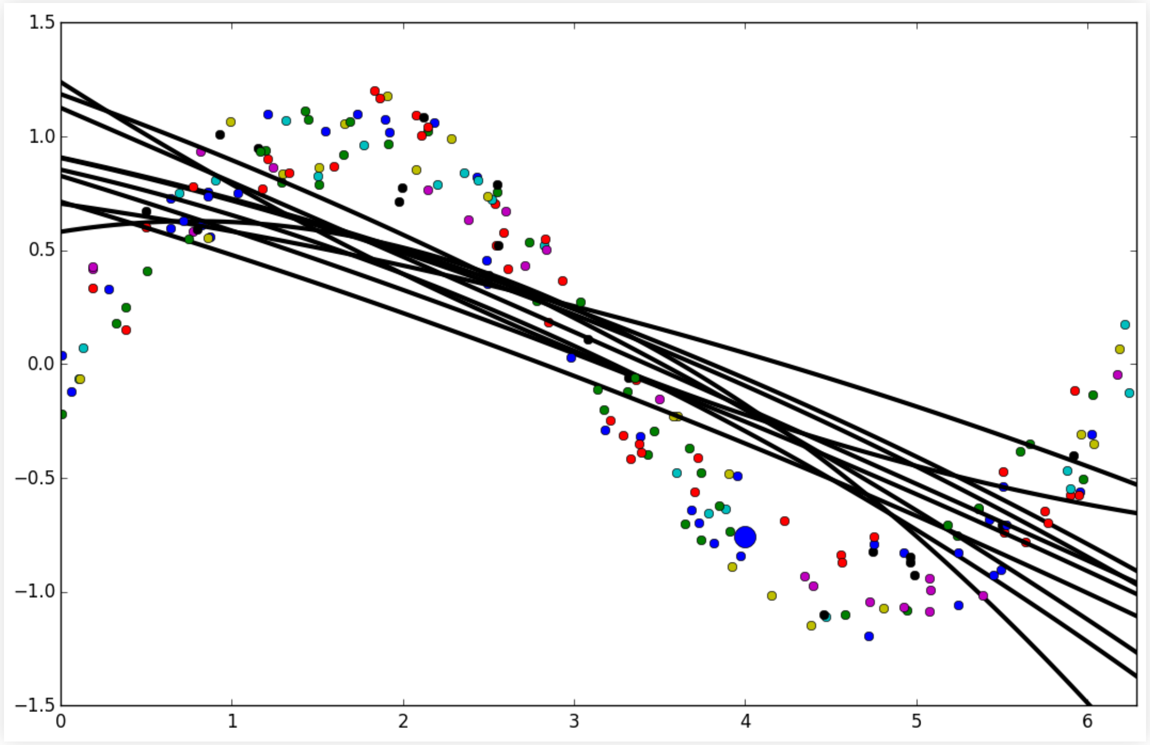
Variance

- Dispersion of predictions around their average

Bias and Variance

- Bias: deviation of the average estimate from the target value

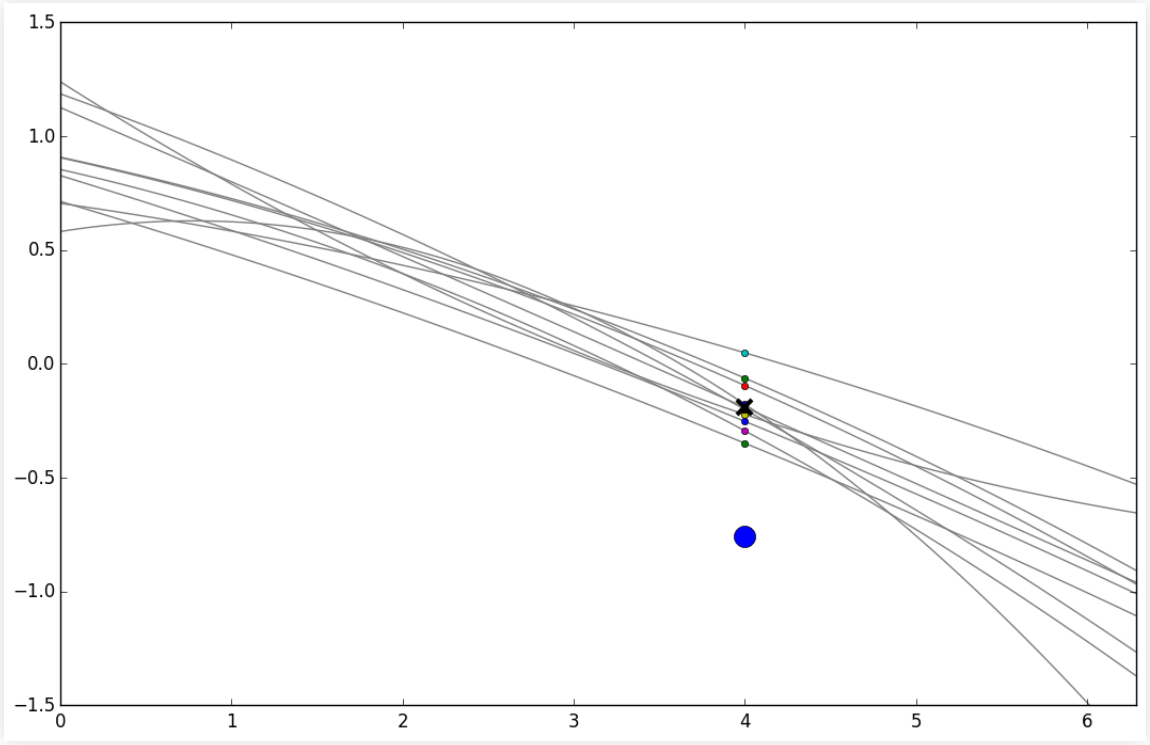
$$bias_n = (\bar{y}(x_n) - t_n)^2 \qquad bias = \frac{1}{N}\sum_{n=1}^{N}(\bar{y}(x_n) - t_n)^2$$



Bias and Variance

- Bias: deviation of the average estimate from the target value

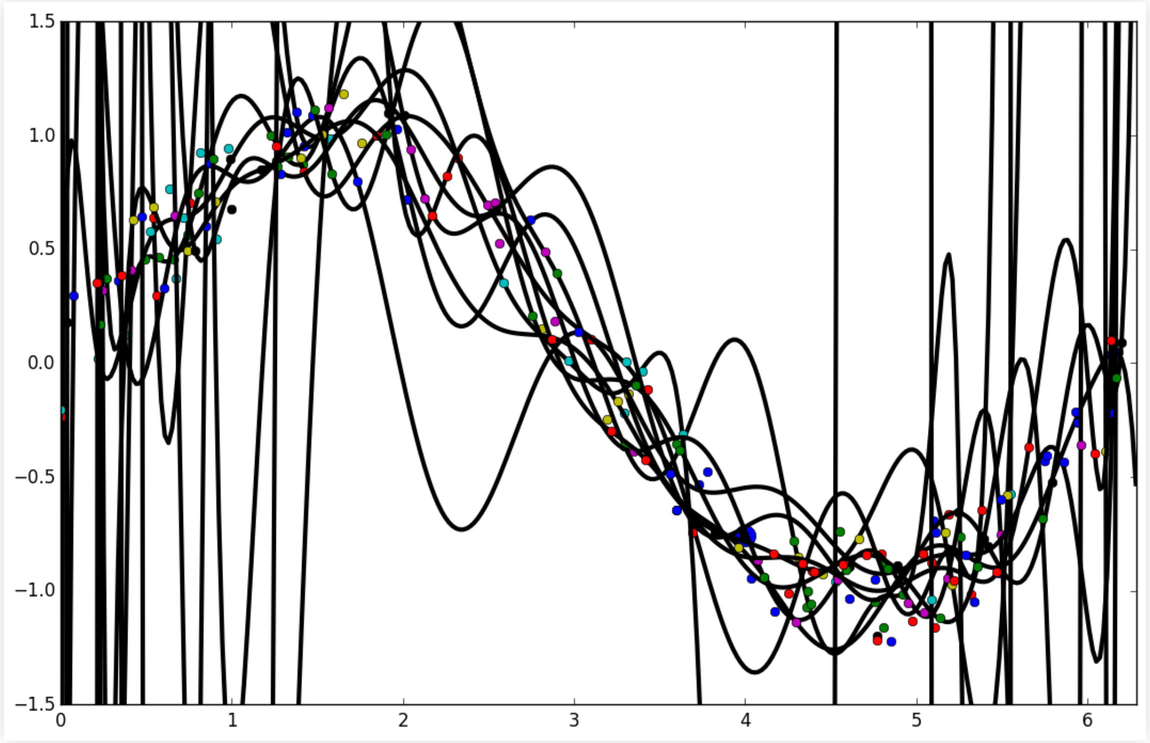
$$bias_n = (\bar{y}(x_n) - t_n)^2 \qquad bias = \frac{1}{N}\sum_{n=1}^{N}(\bar{y}(x_n) - t_n)^2$$



Bias and Variance

- Variance: dispersion of predictions around their average

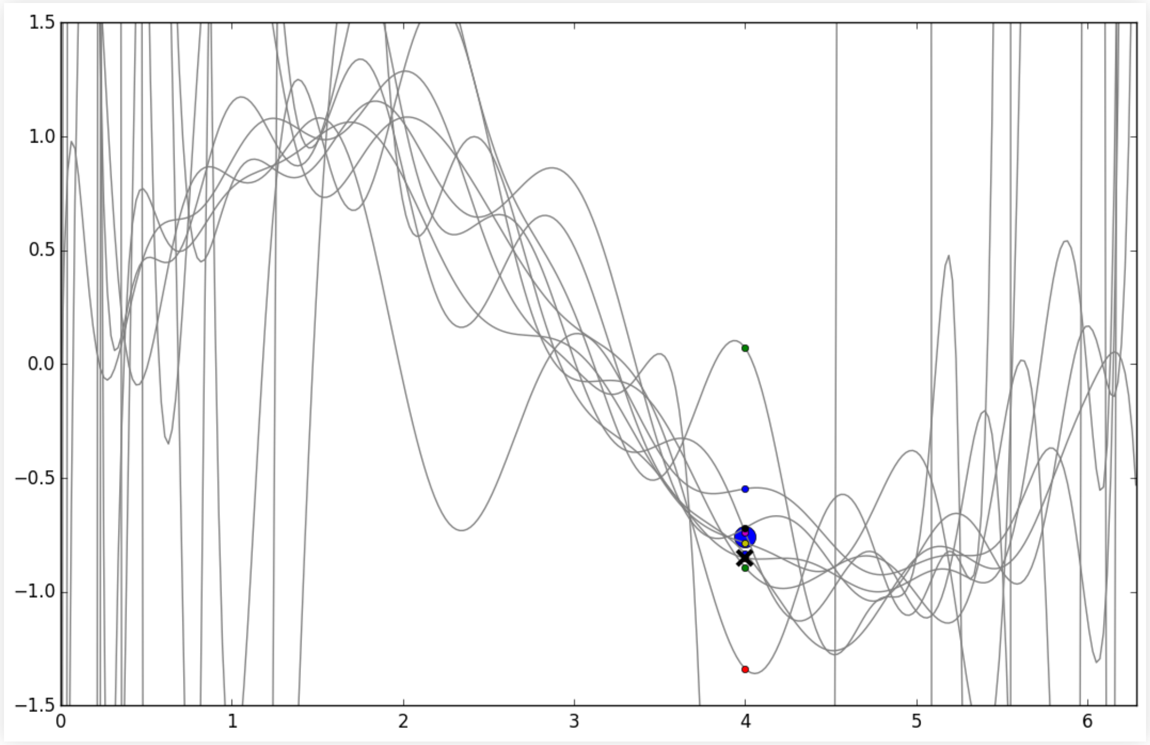
$$var_n = \frac{1}{M}\sum_{m=1}^{M}\left(\bar{y}(x_n) - y_m(x_n)\right)^2 \qquad var = \frac{1}{NM}\sum_{n=1}^{N}\sum_{m=1}^{M}\left(\bar{y}(x_n) - y_m(x_n)\right)^2$$



Bias and Variance

- Variance: dispersion of predictions around their average

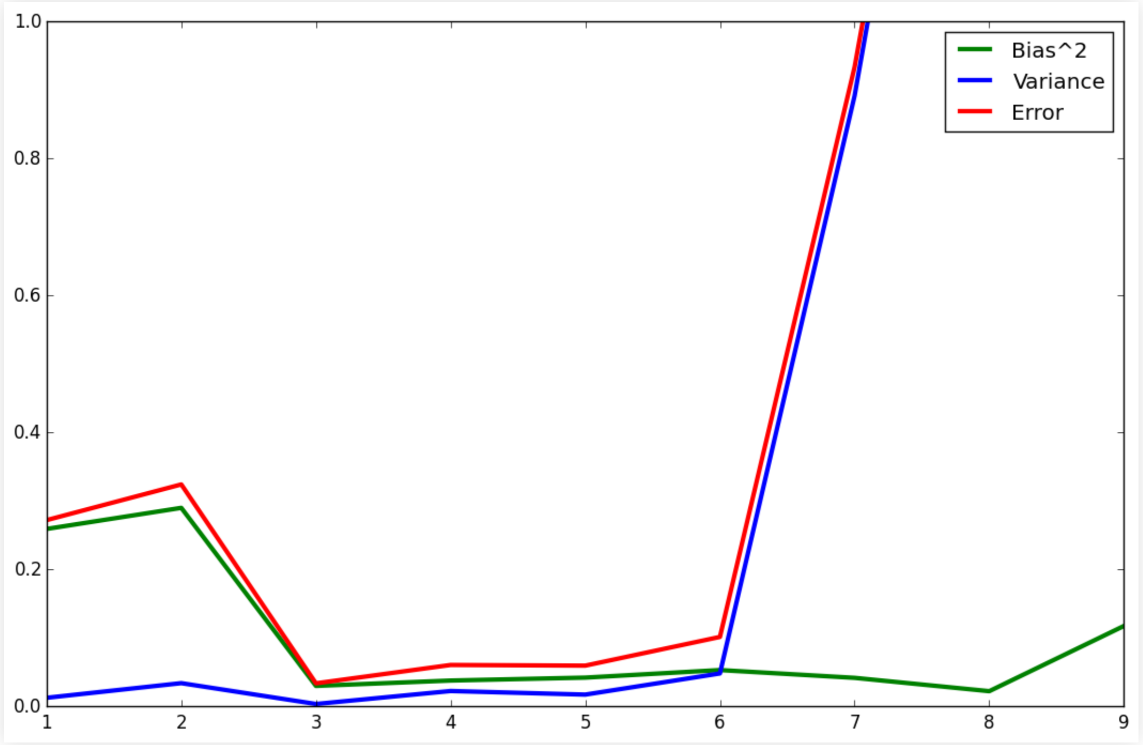
$$var_n = \frac{1}{M}\sum_{m=1}^{M}(\bar{y}(x_n) - y_m(x_n))^2 \qquad var = \frac{1}{NM}\sum_{n=1}^{N}\sum_{m=1}^{M}(\bar{y}(x_n) - y_m(x_n))^2$$



Bias and Variance

Bias-variance tradeoff

- Reducing $bias$ increases $variance$



Bias and Variance

High Bias

- Model cannot adjust the data (underfitting)
- Variance is low, true error is close to training error
- But error is high because the model is too "stiff"
- Need another model, or some combination of models
  - e.g. boosting
- This is not a problem with deep learning

High Variance

- Model adjusts too much to irrelevant details (overfitting)
- Training error is low but true error is high
- This is common in DL but may be fixed with regularization

Practical Issues

Regularization in ANN

Regularization

Regularization:

- Changes to how the model is trained to reduce overfitting
- Reduces variance (may increase bias, but still pay off)

Regularization

Penalizing parameter size

- To reduce variance, we can force parameters to remain small by adding a penalty to the objective (cost) function:

$$\tilde{J}(\theta; X, y) = J(\theta; X, y) + \alpha\Omega(\theta)$$

- Where α is the weight of the regularization

- Note: in ANN, generally only the input weights at each neuron are penalized and not the bias weights.

- The norm function $\Omega(\theta)$ usually takes these forms:

- L^2 Regularization (ridge regression): penalize $||\theta||^2$
- L^1 Regularization: penalize $\sum_i |\theta_i|$

Regularization

L^2 Regularization is weight decay

- If we penalize w^2, the gradient becomes:

$$\nabla\tilde{J}(\theta; X, y) = \nabla J(\theta; X, y) + 2\alpha w$$

- This means the update rule for the weight becomes

$$w \leftarrow w - \epsilon 2\alpha w - \epsilon\nabla J(\theta; X, y)$$

- We decrease the magnitude of w to $(1 - \epsilon 2\alpha)$ per update
- This causes weights that do not contribute to reducing the cost function to shrink

Regularization

L^1 Regularization

- If we penalize $|w|$, the gradient becomes:

$$\nabla \tilde{J}(\theta; X, y) = \nabla J(\theta; X, y) + \alpha\, sign(w)$$

- This penalizes parameters by a constant value, leading to a sparse solution
  - Some weights will have an optimal value of 0

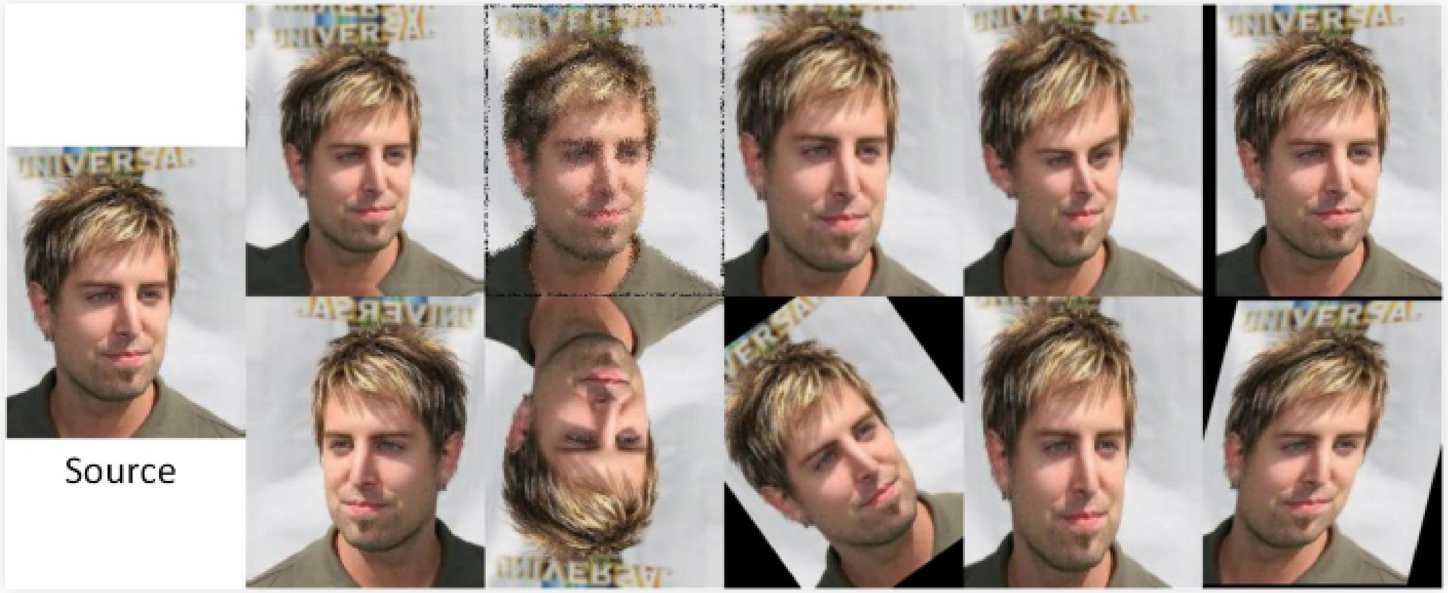
L^1 vs L^2 Regularization

- L^1 minimizes number of non-zero weights
- L^2 minimizes overall weight magnitude

Regularization

Dataset augmentation

- More data is generally better, although not always readily available
- But sometimes we cam make more data
- E.g. Image classification:
  - Translate images. Rotate or flip, if appropriate (not for character recognition)



Wang et al, 2019, "A survey of face data augmentation".

Regularization

Dataset augmentation by noise injection

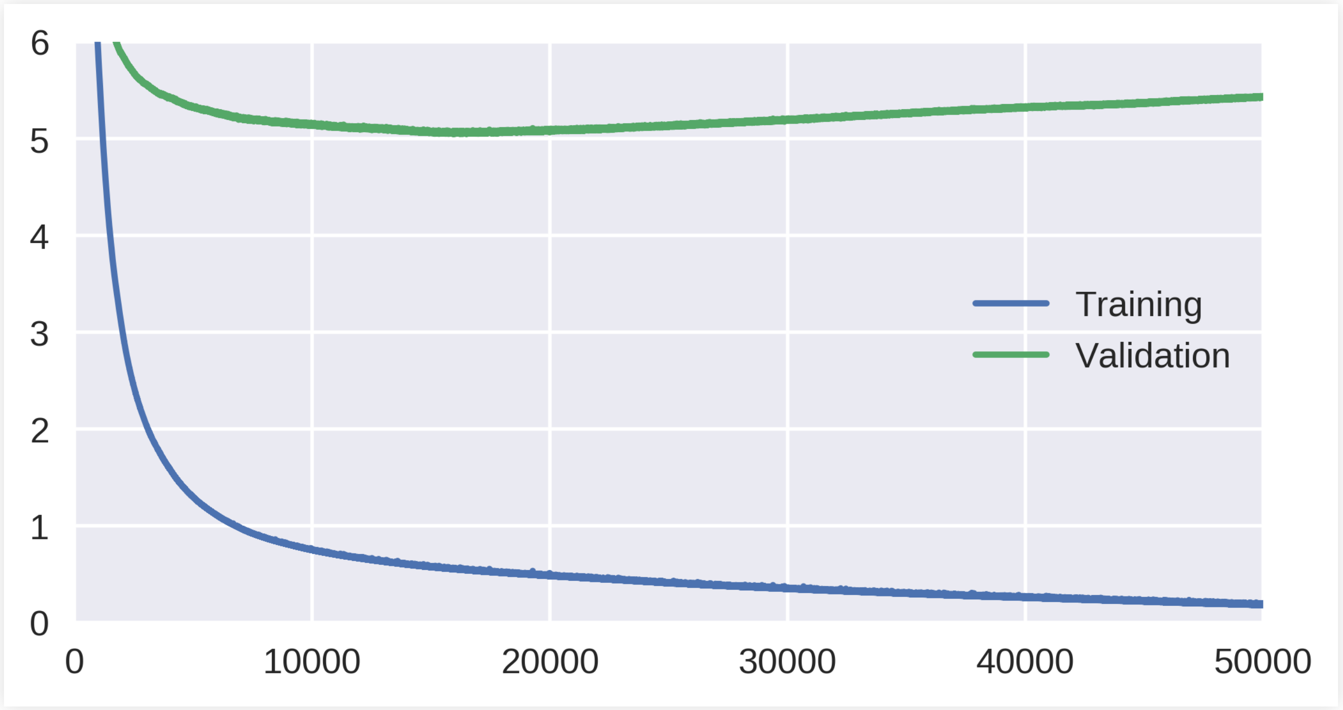
- Noise injection is an (implicit) form of dataset augmentation
  - Add (carefully) noise to inputs, or even to some hidden layers
- Noise can also be applied to the weights
- Or even the output
  - There may be errors in labelling
  - Or for label smoothing: use $\epsilon/(k-1)$ and $1-\epsilon$ instead of 0 and 1 for target
  - This prevents pushing softmax or sigmoid to infinity

Regularization

Early stopping

- Use validation to stop at best point

- Constraints weights to be closer to starting distribution



Regularization

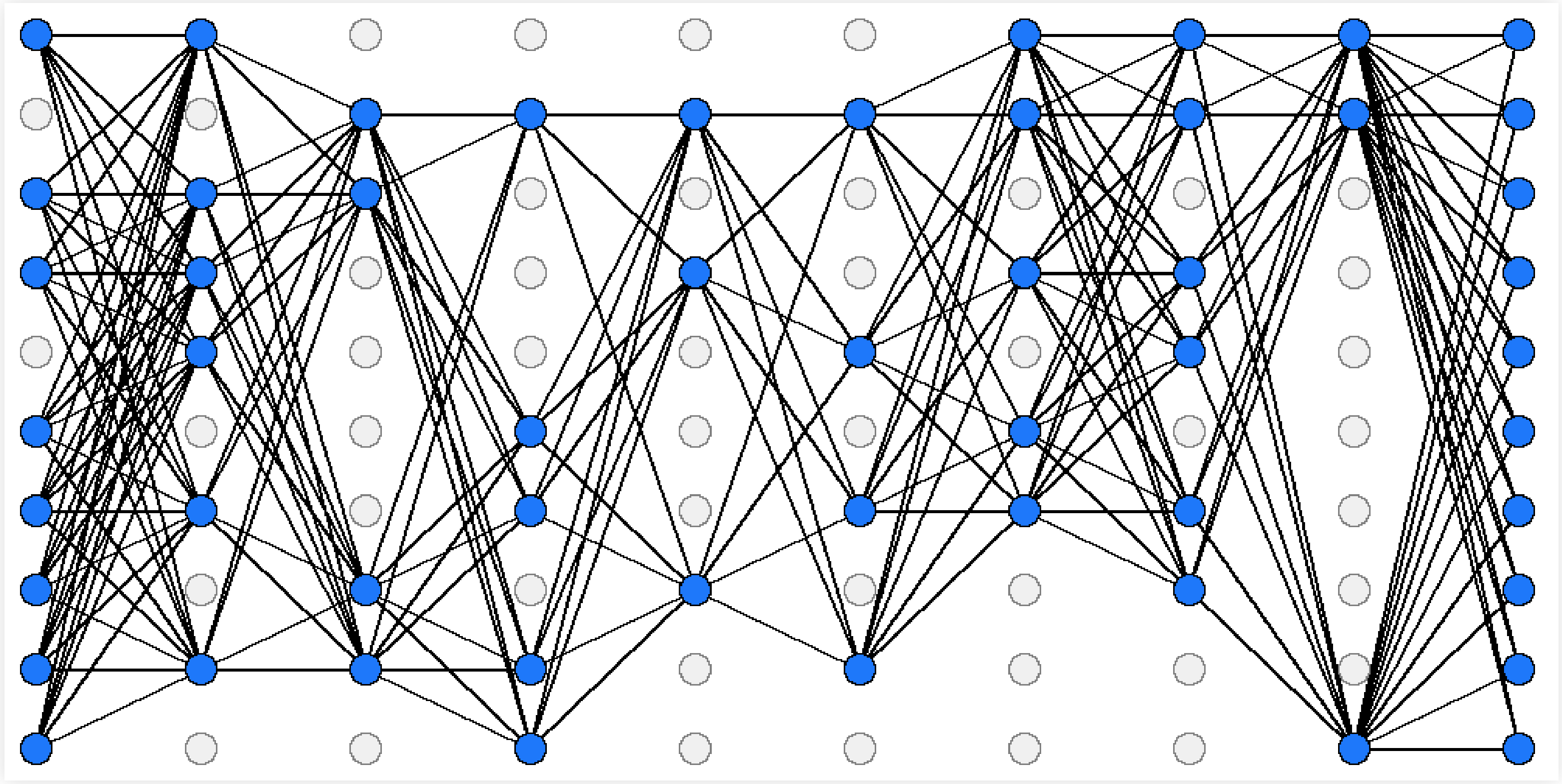
Bagging and dropout

- Bagging, or model averaging, consists in training a set of models and then using the average response (or majority vote)
  - This generally improves performance, as it reduces variance without affecting bias, and ANN can have high variance
  - However, it can be costly to train and use many deep models.

Regularization

Dropout

- "Turns off" random input and hidden neurons in each minibatch



Regularization

Dropout

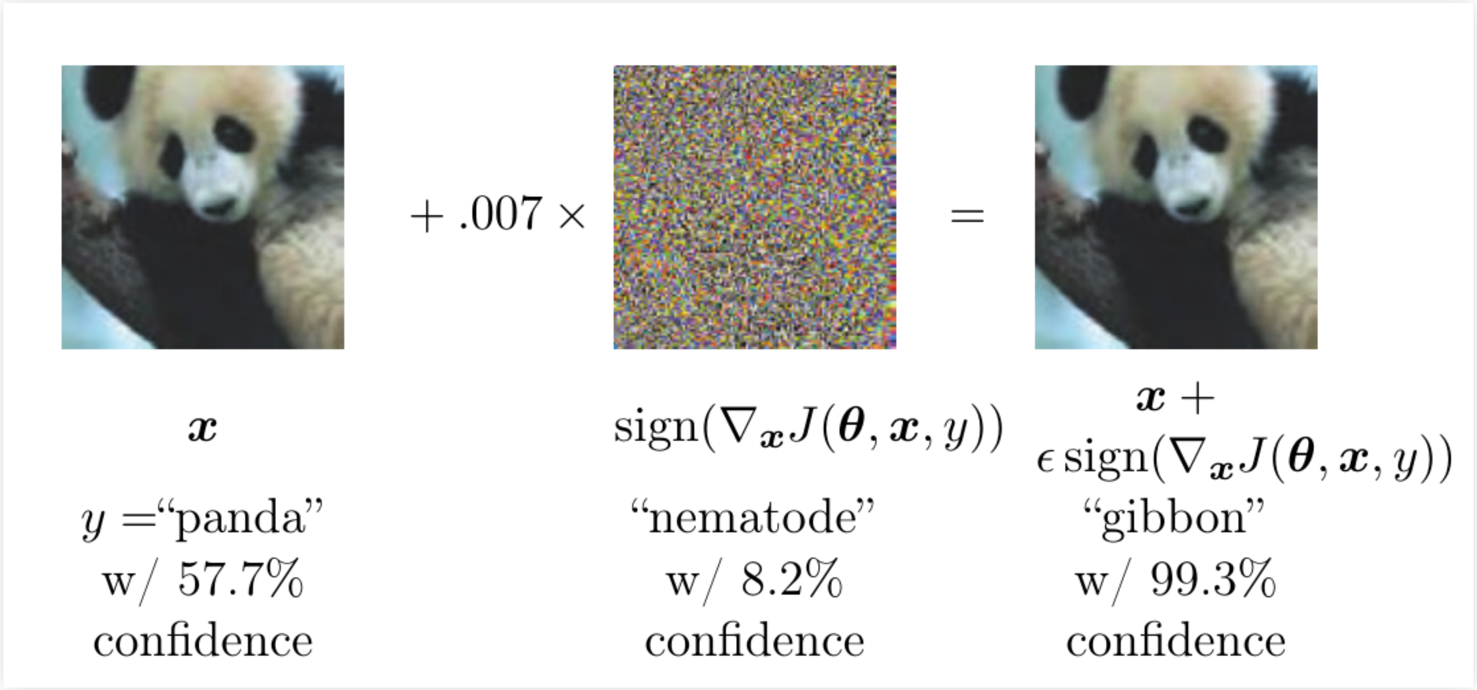
- Dropout does model averaging implicitely
- Turning off neurons at random trains an ensemble of many different networks
- After training, weights are scaled by the probability of being "on"
  - (same expected activation value)

Inverted dropout

- Instead of adjusting weights after training, increase activation of neurons left on during training
  - Automatically "scales" activations

Adversarial Training

- ANN can make strange mistakes with the "right" inputs


$+ .007 \times$
$=$
$\boldsymbol{x}$
$y =$"panda"
w/ 57.7% confidence
$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"nematode"
w/ 8.2% confidence
$\boldsymbol{x} + \epsilon\,\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"gibbon"
w/ 99.3% confidence

Goodfellow, Bengio, Courville, Deep Learning 2016

Regularization

Adversarial Training

- ANN can make strange mistakes with the "right" inputs
- But this can be prevented by using these adversarially perturbed examples during training
  - Forces the network to be locally constant in the neighbourhood of the training data

Selecting Hyperparameters

Many parameters to select

- Regularization, network shape and size, optimizers, etc

Several strategies:

- Manual Optimization
- Grid search
- Random Search
- Bayesian search

Summary

Optimization

Summary

- Optimizing training
  - Optimizers, initialization, learning rate, batch normalization
- Optimizing the predictions
  - Model selection
  - Bias and Variance
  - Regularization
  - Penalizing weights, Augmenting data
  - Noise, Early stop, dropout, Adversarial training

Further reading:

- Goodfellow et.al, Deep learning, Chaps 7 and 11, Sects 8.4; 8.7.1