Overfitting/Underfitting & Generalization

Consider the problem of polynomial curve fitting where we shall fit the data using a polynomial function of the form:

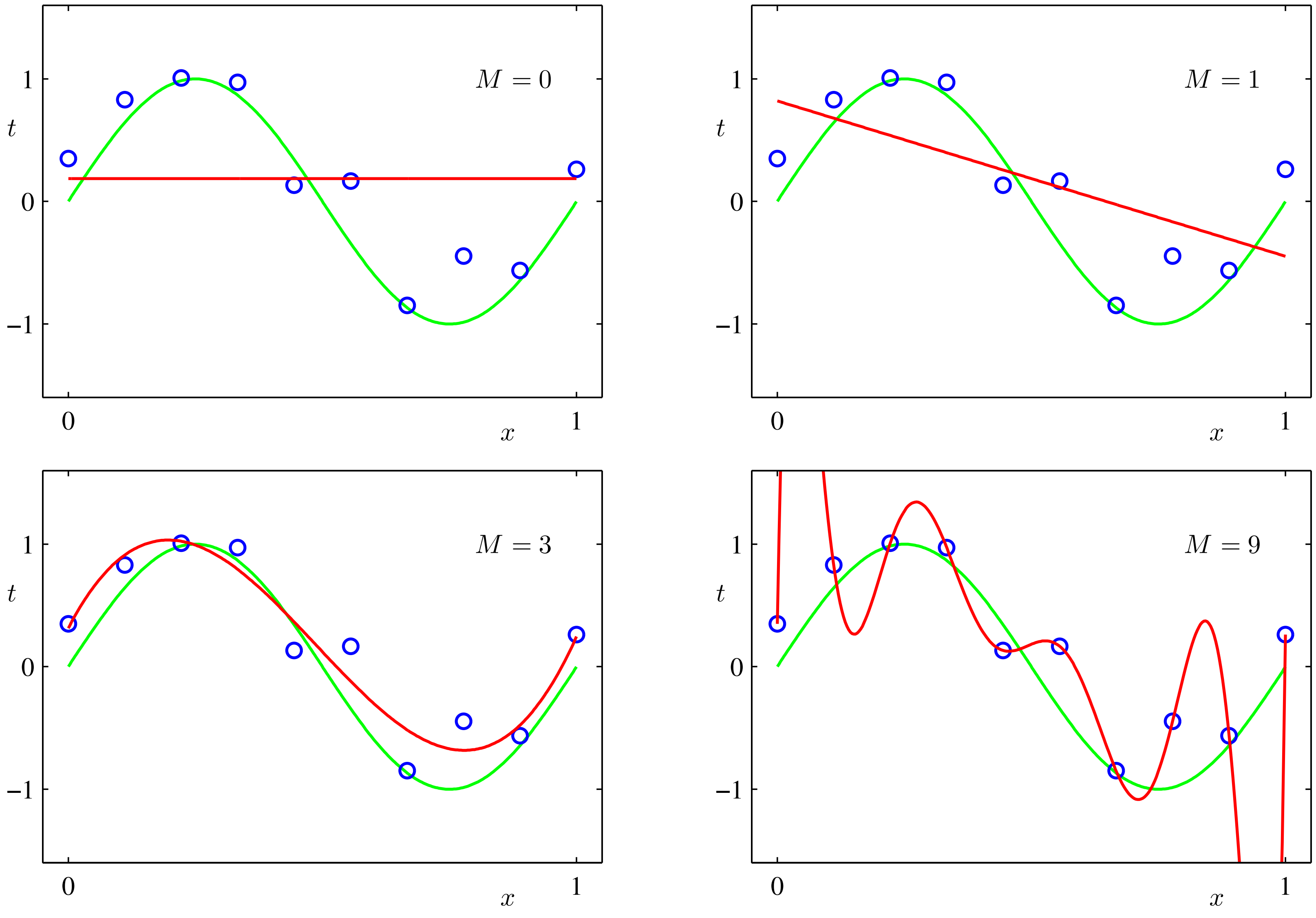
$$y(x, \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + \ldots + w_M x^M = \sum_{j=0}^{M} w_j x^j.$$

We measure the misfit of our predictive function $y(x, \mathbf{w})$ by means of error function which we like to minimize:

$$E(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^{N} \left(y(x_i, \mathbf{w}) - t_i\right)^2$$

where t_i is the corresponding target value in the given training data set.

Polynomial Curve Fitting



Polynomial Curve Fitting (cont.)

	$M = 0$	$M = 1$	$M = 3$	$M = 9$
$w_0^\star$	0.19	0.82	0.31	0.35
$w_1^\star$		-1.27	7.99	232.37
$w_2^\star$			-25.43	-5321.83
$w_3^\star$			17.37	48568.31
$w_4^\star$				-231639.30
$w_5^\star$				640042.26
$w_6^\star$				-1061800.52
$w_7^\star$				1042400.18
$w_8^\star$				-557682.99
$w_9^\star$				125201.43

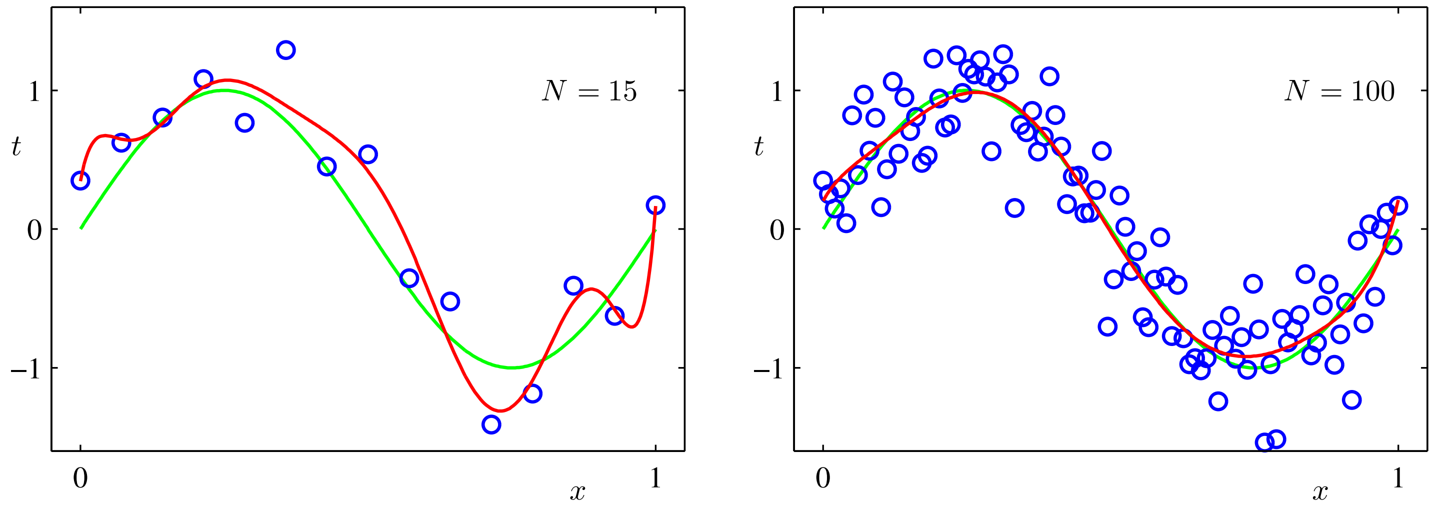
Table 1: Coefficients $\mathbf{w}^\star$ obtained from polynomials of various order. Observe the dramatically increase as the order of the polynomial increases (this table is taken from Bishop's book).

Polynomial Curve Fitting (cont.)

Observe:

- if M is too small then the model underfits the data
- if M is too large then the model overfits the data

If M is too large then the model is more flexible and is becoming increasingly tuned to random noise on the target values. It is interesting to note that the overfitting problem become less severe as the size of the data set increases.



Polynomial Curve Fitting (cont.)

One technique that can be used to control the overfitting phenomenon is the *regularization*.

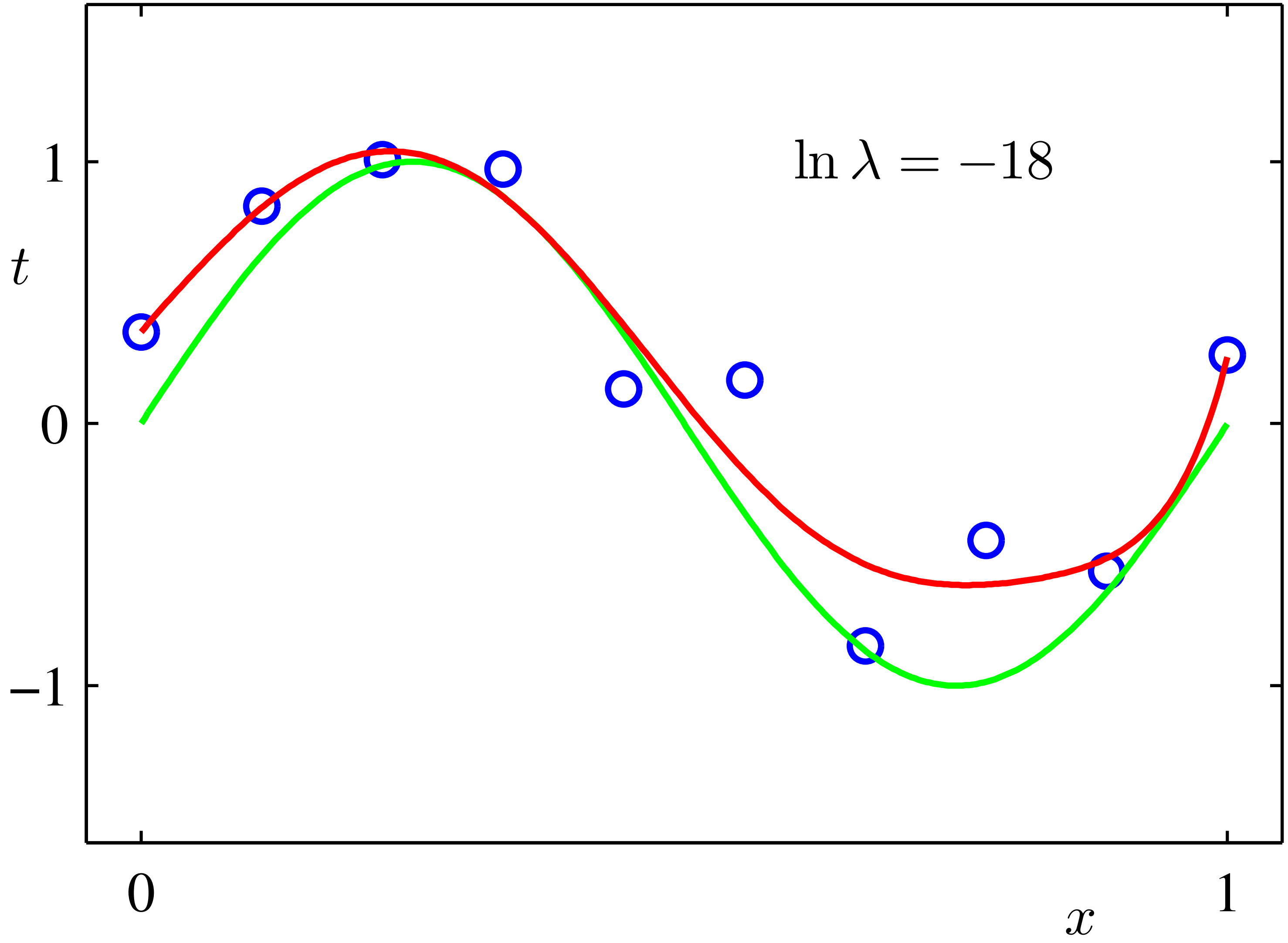
- Regularization involves adding a penalty term to the error function in order to discourage the coefficients from reaching large values.

The modified error function has the form:

$$\widehat{E}(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{N}(y(x_i,\mathbf{w}) - t_i)^2 + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w}.$$

By means of the penalty term one reduces the value of the coefficients (shrinkage method).

Regularized Polynomial Curve Fitting $M = 9$



Regularization in Neural Networks

- Number of input/output units is generally determined by the dimensionality of the data set.
- Number of hidden units M is free parameter that can be adjusted to obtain best predictive performance.
- Generalization error is not a simple function of M due to the presence of local minima in the error function.
- One straightforward way to deal with this problem is to increase stepwise the value of M and to choose the specific solution having the smallest test error.

Regularization in Neural Networks (cont.)

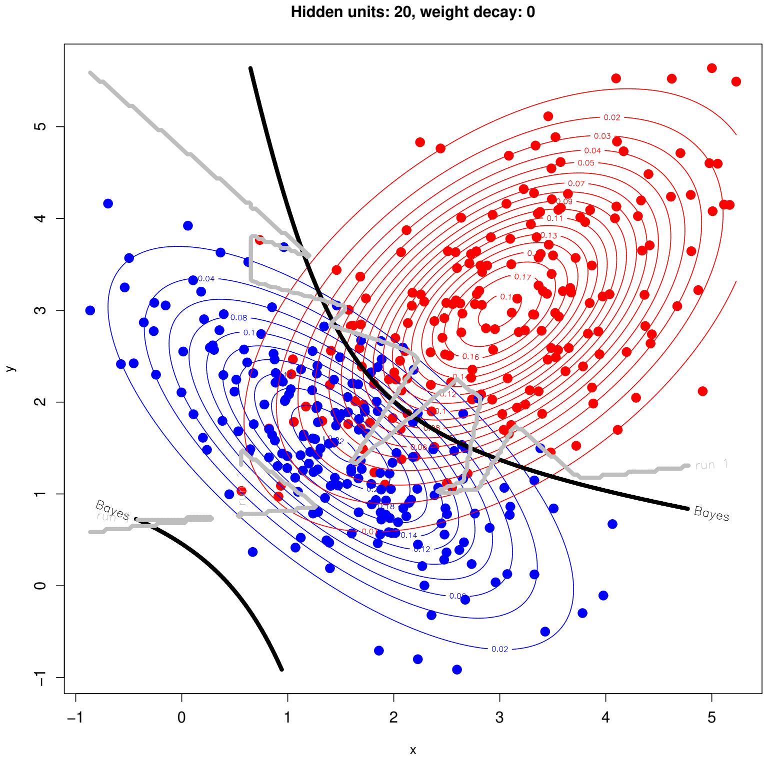
Equivalent to the regularized curve fitting approach, we can choose a relatively large value for M and control the complexity by the addition of a regularized term to the error function.

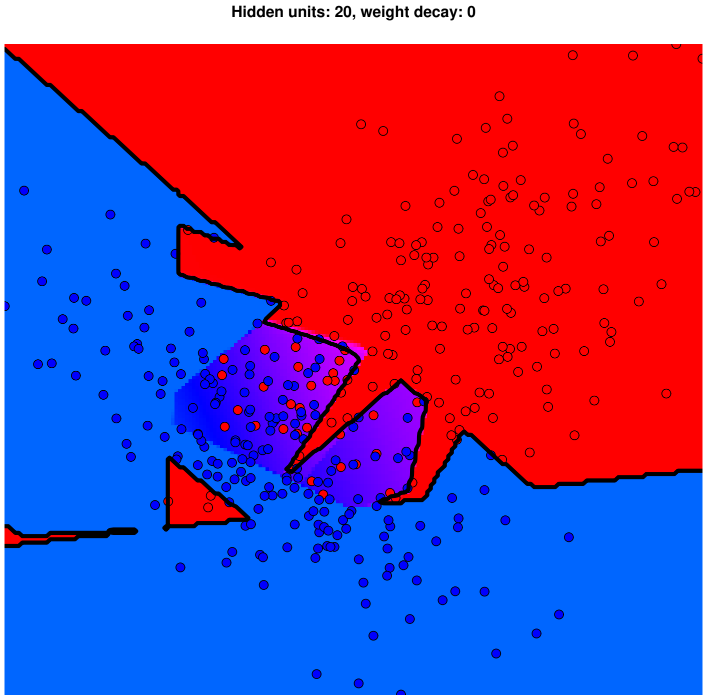
$$\widehat{E}(\mathbf{w}) = E(\mathbf{w}) + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w}$$

This form of regularization in neural networks is known as *weight decay*.

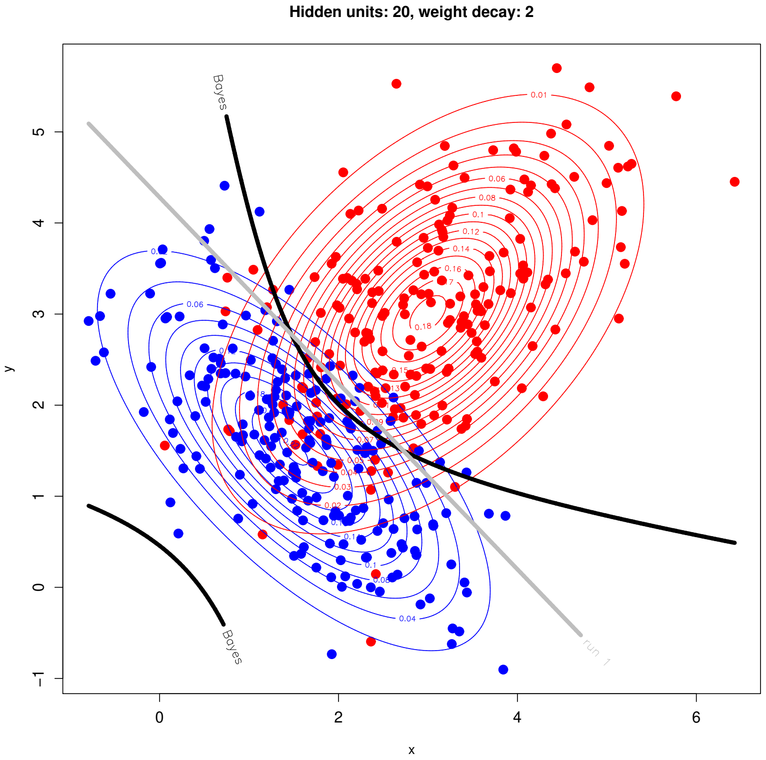
- Weight decay encourages weight values to decay towards zero, unless supported by the data.
- It can be considered as an example of a parameter shrinkage method because parameter values are shrunk towards zero.
- It can be also interpreted as the removal of non-useful connections during training.

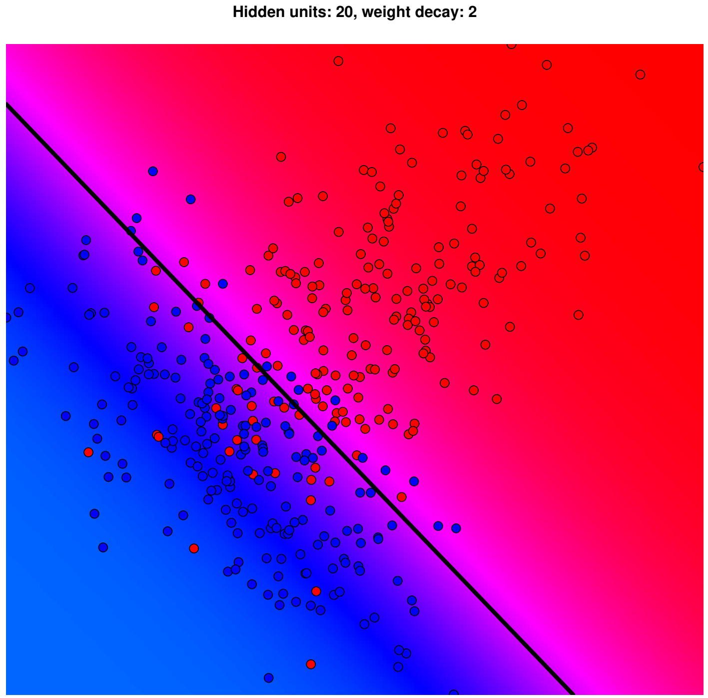
A too Overfitted Neural Network Model



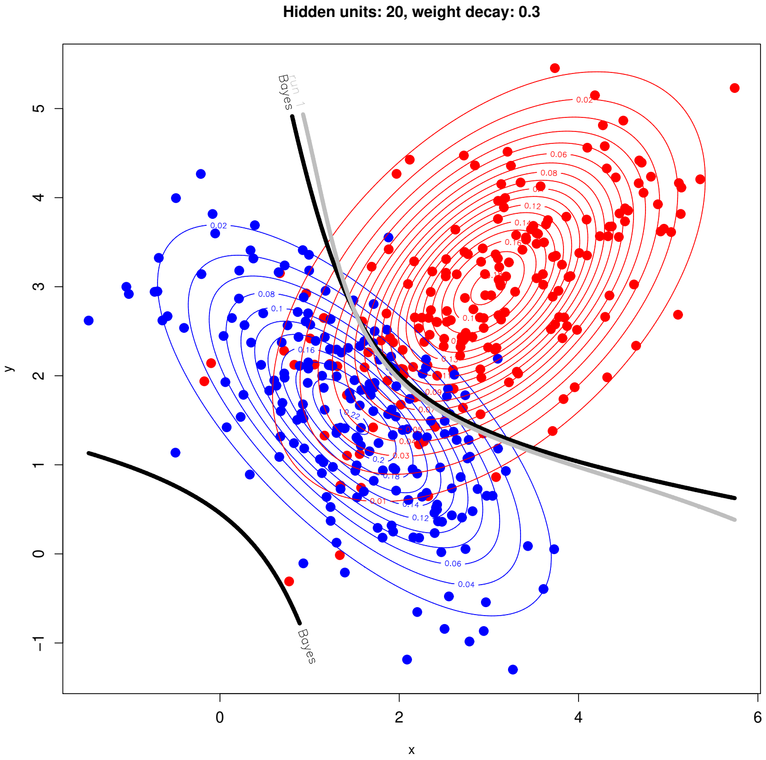


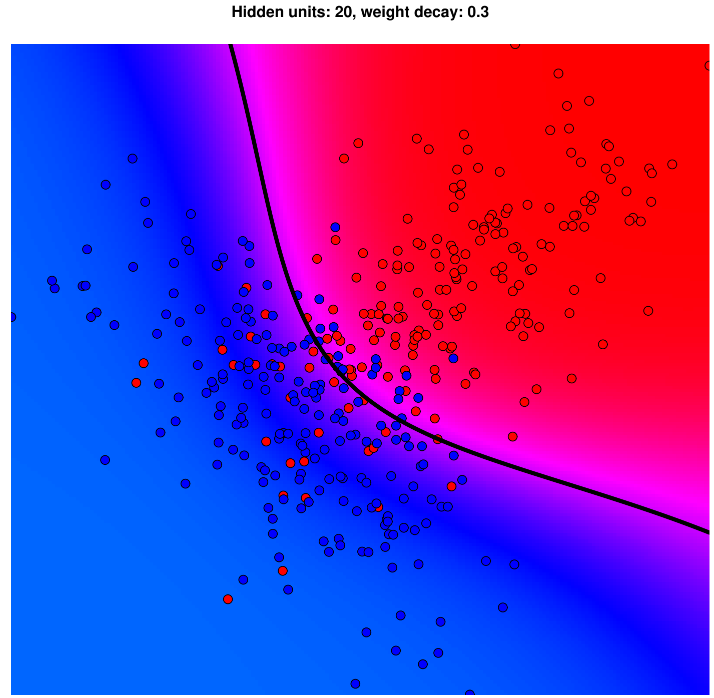
A too Underfitted Neural Network Model





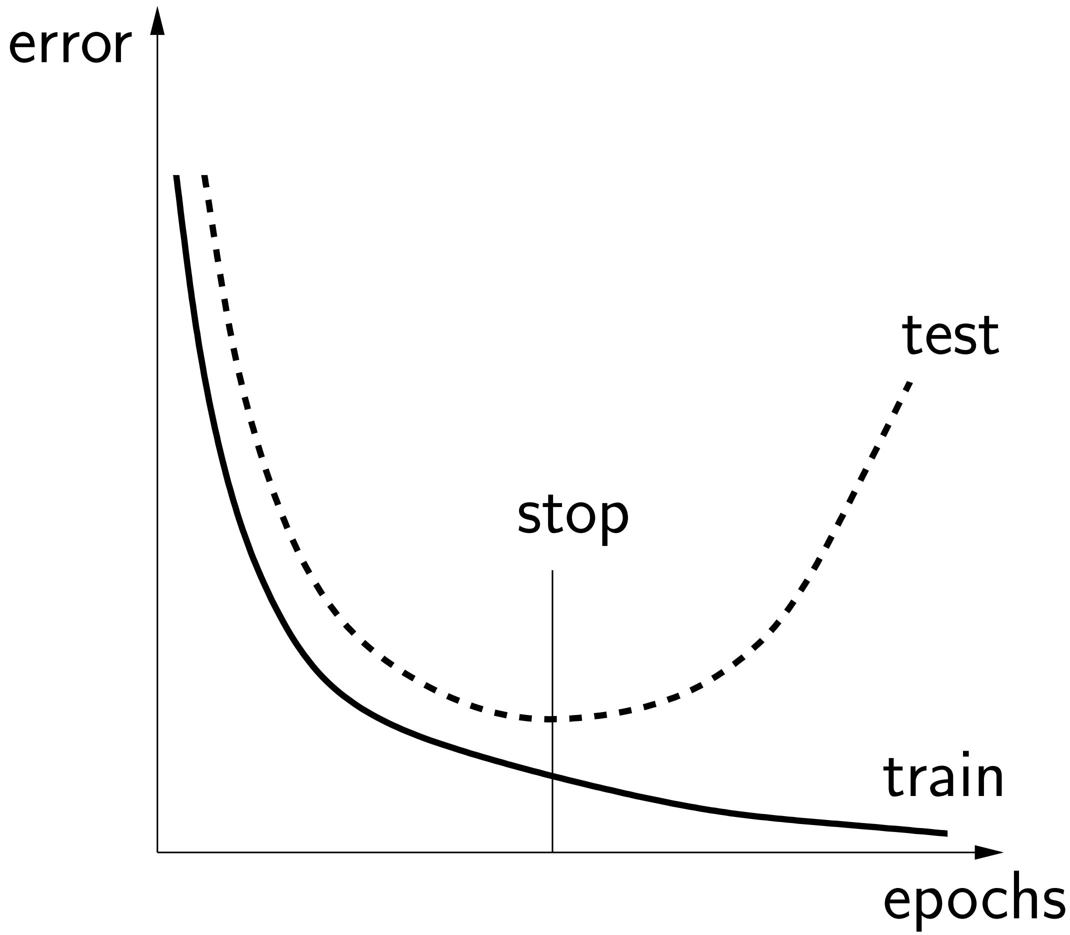
Model Complexity is Properly Penalized





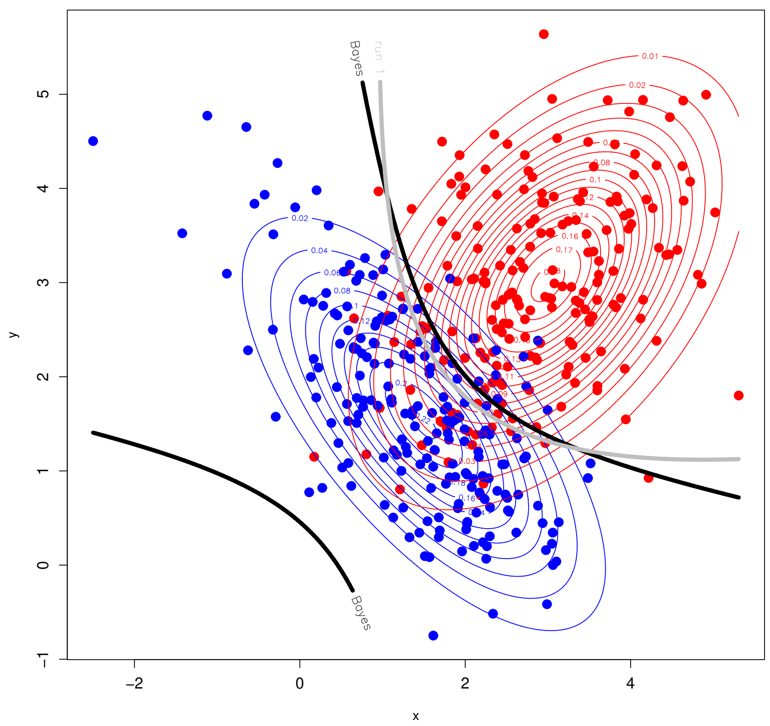
Regularization by Early Stopping

- Another alternative of regularization as a way of controlling the effective complexity of a network is the procedure of *early stopping*.

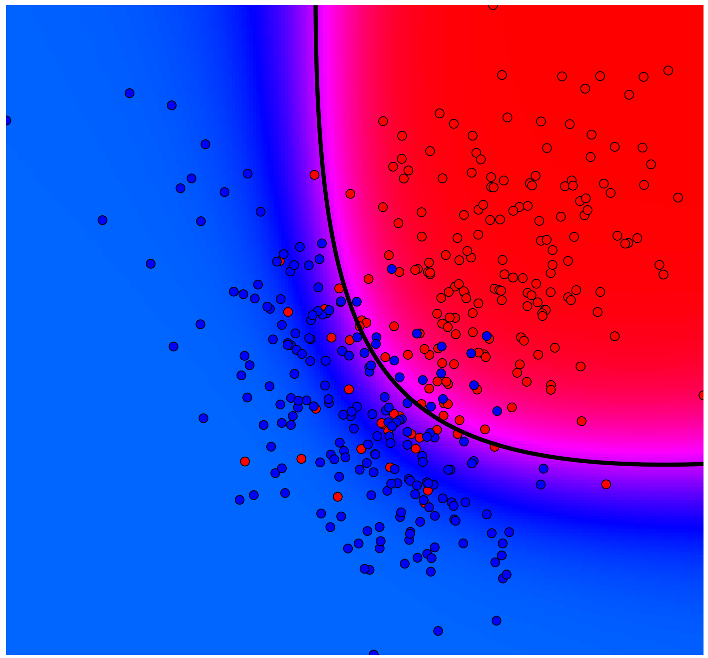


Example Early Stopping after 10 Epochs

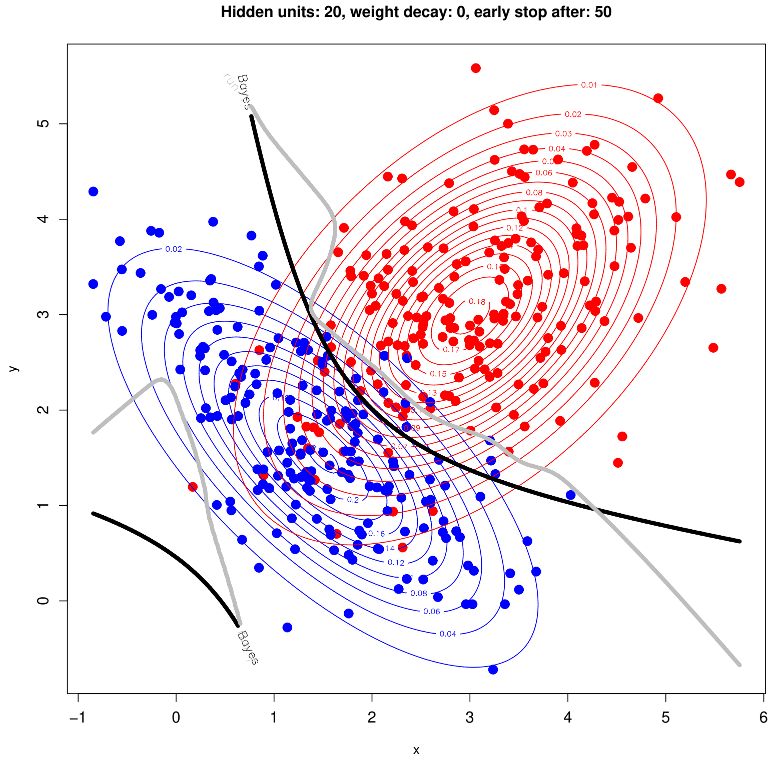
Hidden units: 20, weight decay: 0, early stop after: 10

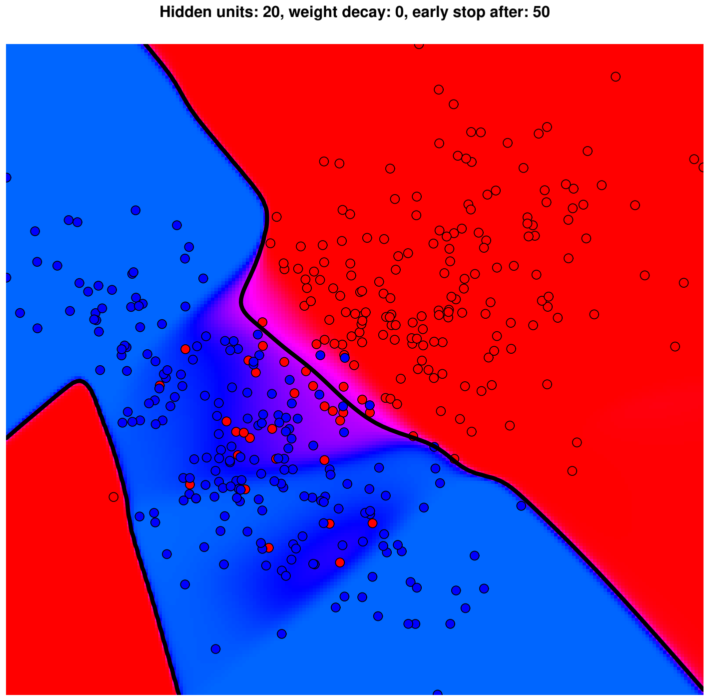


Hidden units: 20, weight decay: 0, early stop after: 10

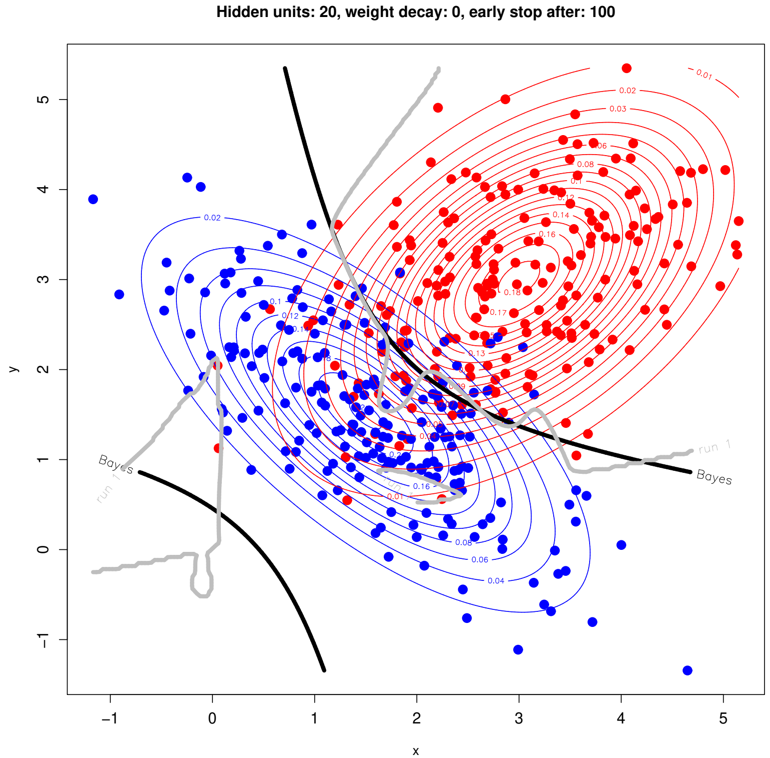


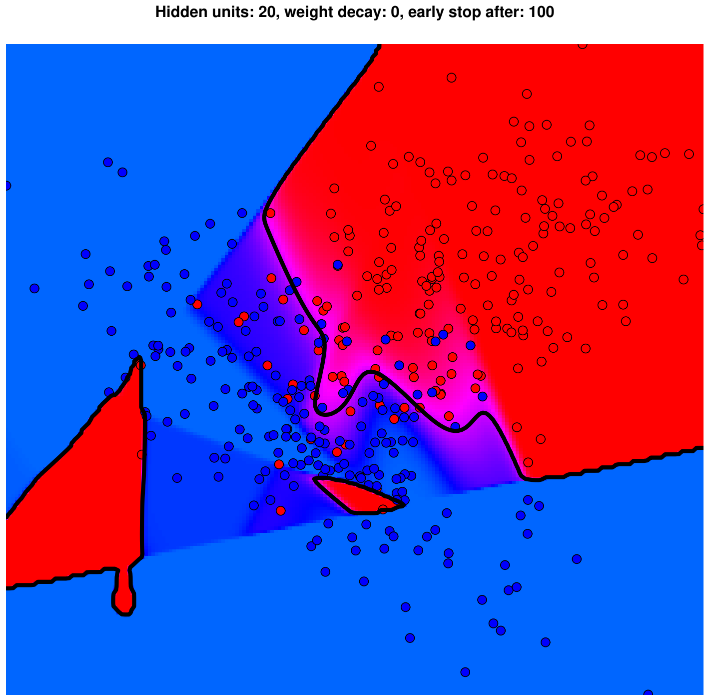
Example Early Stopping after 50 Epochs





Example Early Stopping after 100 Epochs





Cross-Validation & Prediction Error

Most commonly used method for estimating the prediction error (generalization error) on new data is *cross-validation*.

1	2	3	4	5
Train	Train	Train	Train	Test

1	2	3	4	5
Train	Train	Train	Test	Train

⋮

1	2	3	4	5
Test	Train	Train	Train	Train

Split training data into K roughly equal-sized parts (e.g. $K = 5$). Learn neural network on $K - 1$ training parts and predict the not seen testing part. Perform this for $k = 1, 2, \ldots, K$ and combine the K estimates of prediction error (test error).

Cross-Validation & Prediction Error (cont.)

Typical choices of K are 5 or 10. If training data set consists of N data points and $K = N$, then one obtains the *leave-one-out* cross-validation method.

```
# number of folds
folds <- 10
samps <- sample(rep(1:folds, length=n), n, replace=FALSE)

for (fold.iter in 1:folds) {

  train <- dataset[samps!=fold.iter,] # fit the model
  test  <- dataset[samps==fold.iter,] # predict

  # perform here neural network training on set train
  # perform here neural network prediction on set test
}
```

Bias-Variance Dilemma

- Idea: decompose generalization error into the two terms (bias,variance).
- Let us consider our curve fitting problem again. Let $h(\mathbf{x})$ be the true (but unknown) function with continuous valued output with noise.
- We want to estimate $h(\cdot)$ based on training data sets $\mathcal{D}$ each of size N.
- The natural way to measure the effectiveness of the estimator is to use the mean-square deviation from the desired optimal.
- Averaging over all training data sets $\mathcal{D}$ one gets the decomposition:

$$\mathbb{E}_{\mathcal{D}}\left[\{y(\mathbf{x},\mathcal{D}) - h(\mathbf{x})\}^2\right]$$
$$= \underbrace{\{\mathbb{E}_{\mathcal{D}}[y(\mathbf{x},\mathcal{D})] - h(\mathbf{x})\}^2}_{\text{bias}^2} + \underbrace{\mathbb{E}_{\mathcal{D}}\left[\{y(\mathbf{x},\mathcal{D}) - \mathbb{E}_{\mathcal{D}}[y(\mathbf{x},\mathcal{D})]\}^2\right]}_{\text{variance}}$$

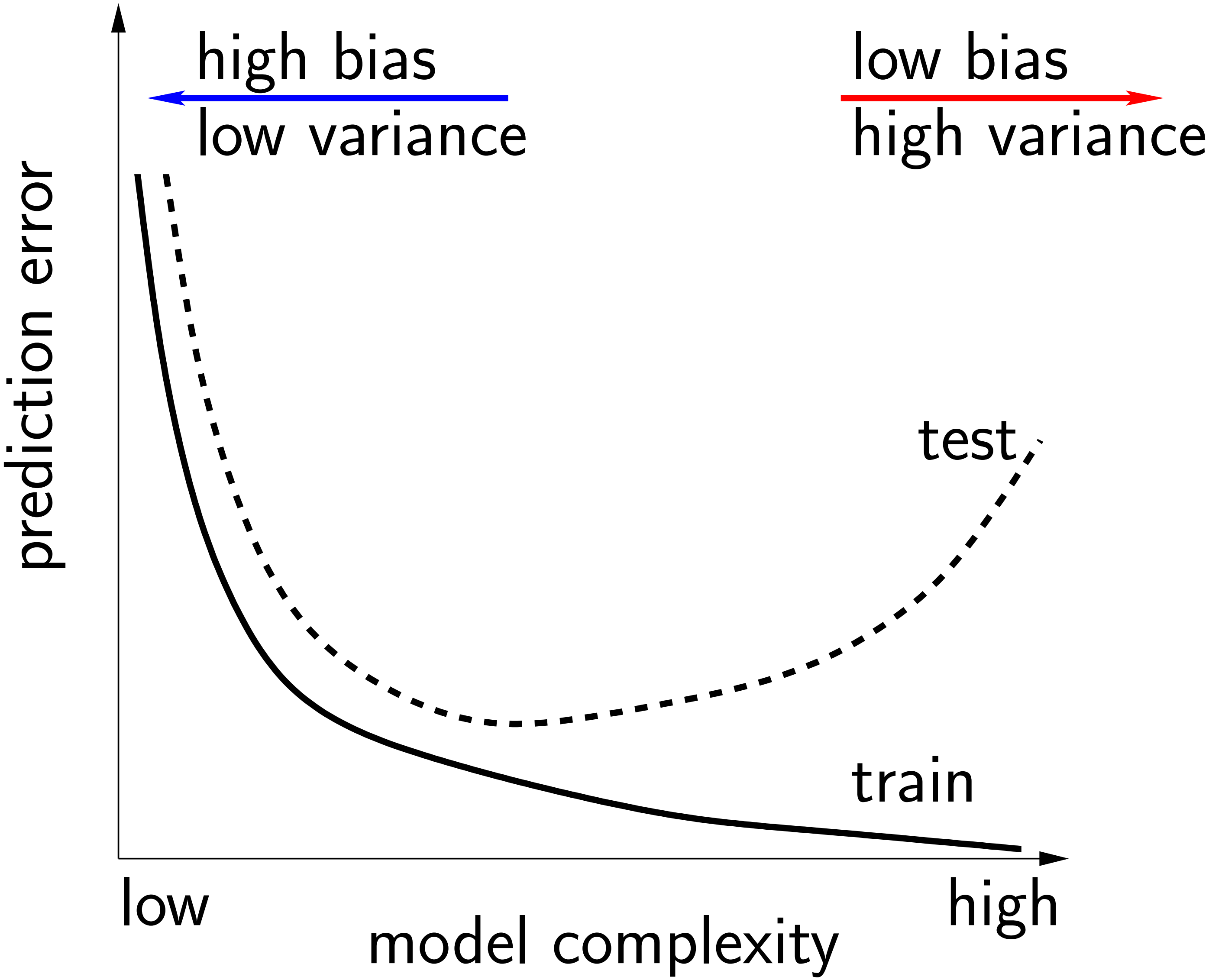
Bias-Variance Dilemma (cont.)

The bias-variance dilemma is general phenomenon and also occurs as the under/overfitting problem in neural networks. In the context of neural networks:

- Bias is a measure of how much the network output, averaged over all possible data sets differs from the desired function.
- Variance is a measure of how much the network output varies between data sets.

In early stage of training, the bias is large because the network is far from the desired function. If the network is trained too long, then the network will also have learned the noise specific in the data set.

Bias-Variance Dilemma (cont.)



Momentum

- Gradient descent can be very slow if η is too small, and can oscillate widely if η is too large.
- Idea: use fraction of the previous weight change and actual gradient term to control non-radical revisions in the updates.

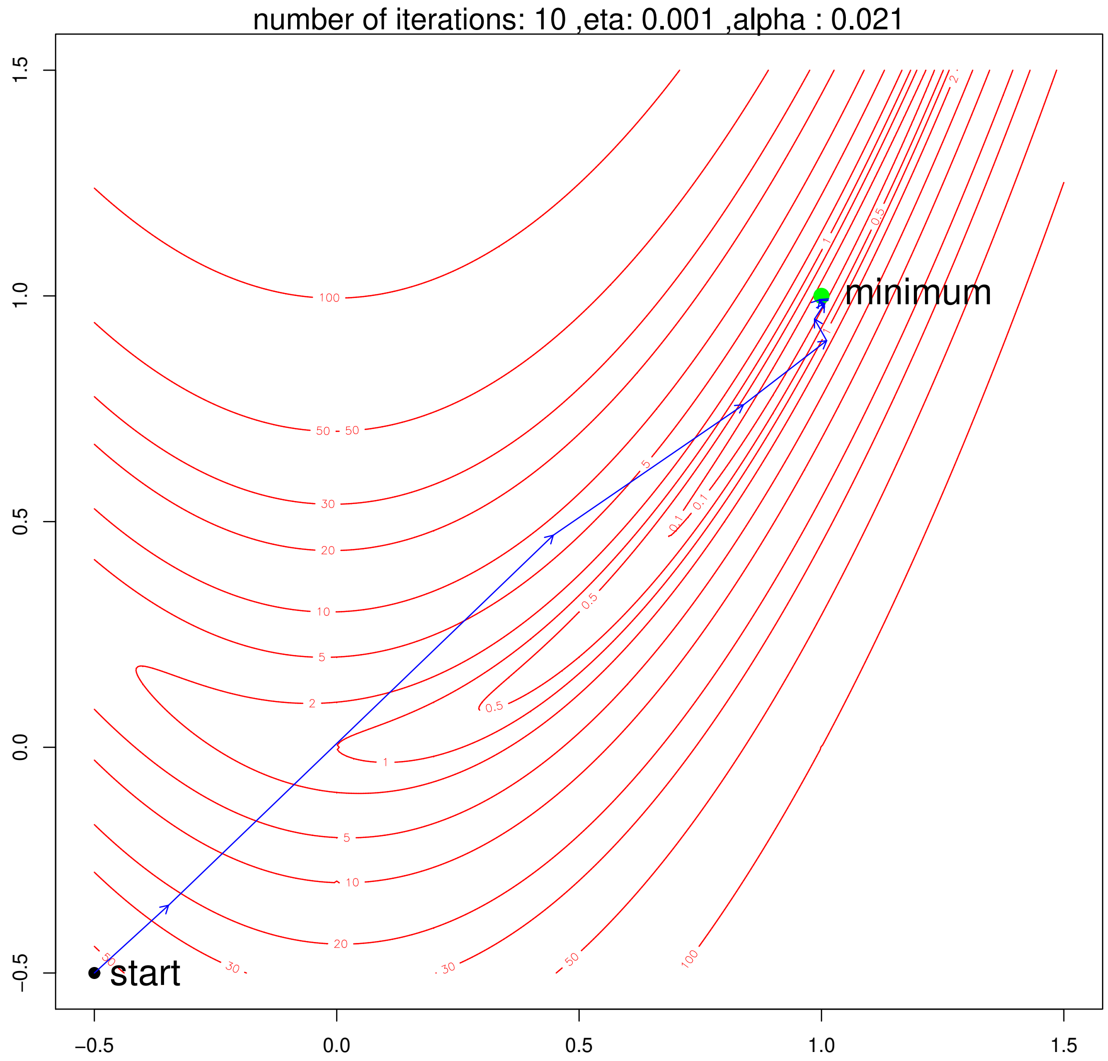
$$\Delta \mathbf{w}^{\text{new}} = -\eta \frac{\partial E}{\partial \mathbf{w}} + \alpha \Delta \mathbf{w}, \qquad 0 \leq \alpha \leq 1.$$

Momentum:

- can cancel side-to-side oscillations across the error valley,
- can cause a faster convergence when weight updates are all in the same direction because the learning rate is amplified.

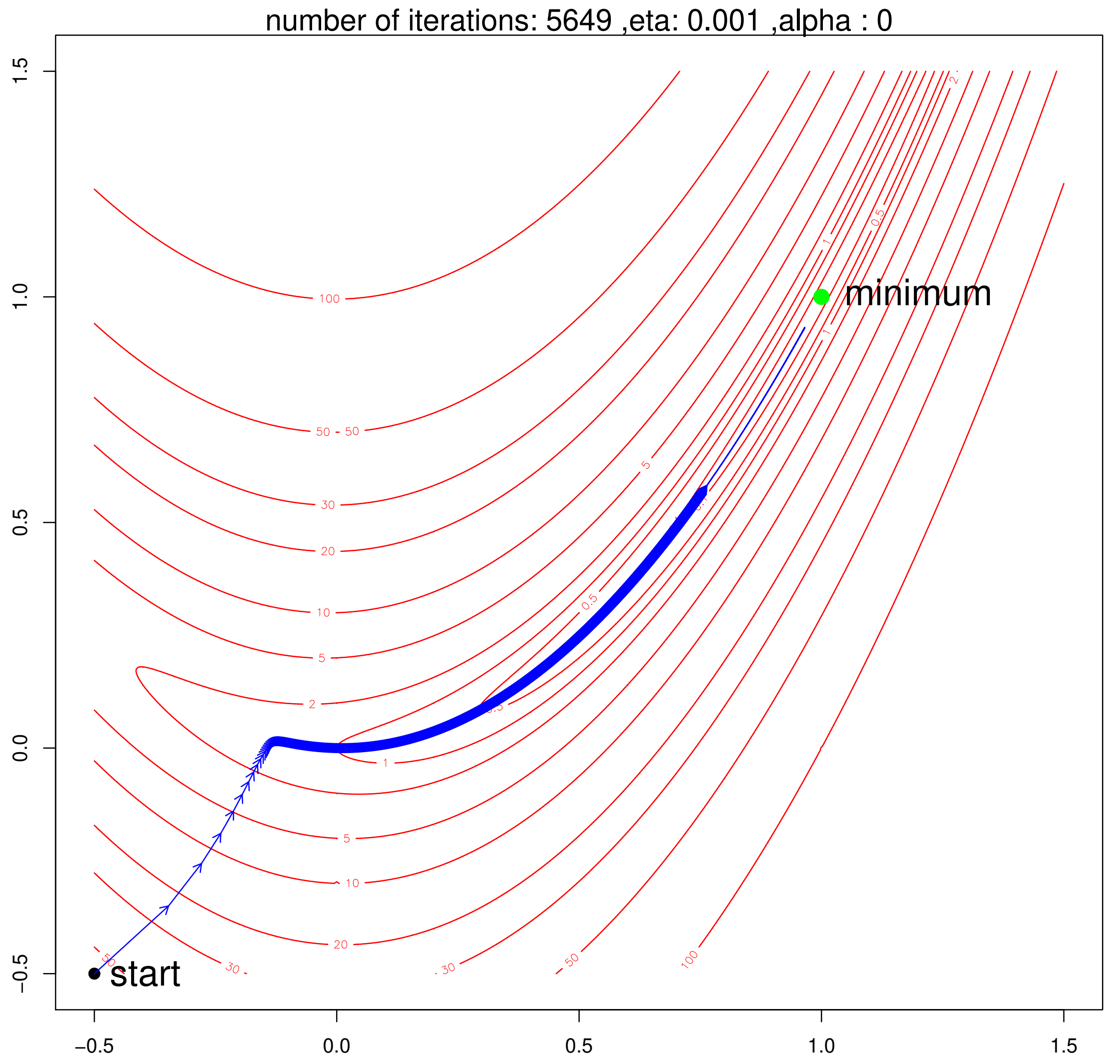
Momentum Example Rosenbrock Function

Rosenbrock function $f(x, y) = (1 - x)^2 + 100(y - x^2)^2$ has global minimum $f(x, y) = (0, 0)$ at $(1, 1)$. Momentum param. $\alpha = 0.021$, learning rate $\eta = 0.001$.



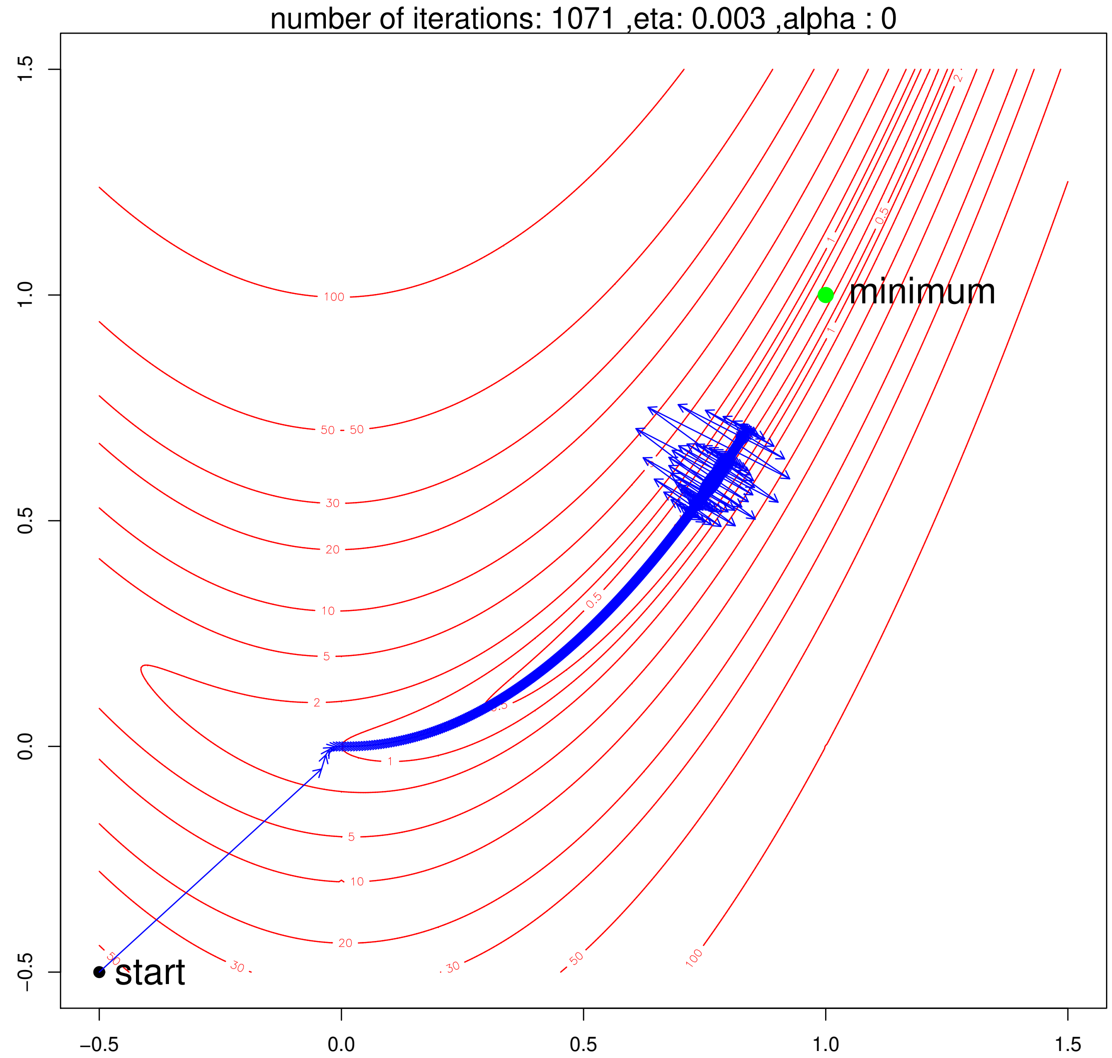
Momentum Example Rosenbrock Func. (cont.)

Setting $\alpha = 0$ (no momentum)



Momentum Example Rosenbrock Func. (cont.)

Setting $\alpha = 0$ (no momentum) and a larger learning rate η



Adaptive Parameter

- Adjust the parameters automatically as learning progress.
- Check if particular weight update decreased the error function.
  - If not we overleaped and η should be reduced.
  - If several steps in a row have decreased the error, we are too conservative and could try increasing η

$$\Delta\eta = \begin{cases} +a & \text{if } \Delta E < 0 \quad \text{consistently (e.g. last } K \text{ steps)} \\ -b\eta & \text{if } \Delta E > 0 \\ 0 & \text{otherwise} \end{cases}$$

Cross-Entropy Error Function

- Note, so far we used the sum-of-squares error function to express the error between output y and target value t.
- A commonly used alternative is the cross-entropy error function.

$$E = -\sum_{i=1}^{N} \Big(t^{(i)} \ln y^{(i)} + (1 - t^{(i)}) \ln(1 - y^{(i)}) \Big) \qquad (2)$$

- Let us consider a network with a single output y.
- We would like that $p(\mathcal{C}_1|\mathbf{x}) = y$ for class $\mathcal{C}_1$ and $p(\mathcal{C}_2|\mathbf{x}) = 1 - y$ for class $\mathcal{C}_2$, respectively.

Cross-Entropy Error Function (cont.)

This can be modeled as:

$$p(t|\mathbf{x}) = y^t(1-y)^{1-t}$$

when considering $t = 1$ if input vector $\mathbf{x}$ belongs to class $\mathcal{C}_1$ and $t = 0$ if it belongs to class $\mathcal{C}_2$.

- Given N input vectors which are drawn independently, one obtains:

$$\prod_{i=1}^{N} (y^{(i)})^{t^{(i)}} (1 - y^{(i)})^{1-t^{(i)}}. \tag{3}$$

Cross-Entropy Error Function (cont.)

- It is more convenient to minimize the negative logarithm of (3)

$$E = -\sum_{i=1}^{N}\left(t^{(i)}\ln y^{(i)} + (1-t^{(i)})\ln(1-y^{(i)})\right) \quad (4)$$