Chapter 6

Model selection and overfitting

Neural networks and deep learning

Model Selection

In most pattern recognition techniques, there exist one or more free parameters. For example in MLPs, learning parameters, no of layers, number of neurons, etc.

Every set of parameters of the network leads to a specific model.

How do we determine the "optimum" parameter(s) or the model for a given approximation or classification problem?

Performance Estimation

How do we measure the performance of the network?

Some metrics:

1. Mean-square error/Root-mean square error for approximation — the mean-squared error or its square root. A measure of the deviation from actual.

$$MSE = \frac{1}{P}\sum_{p=1}^{P}\sum_{k=1}^{K}\left(d_{pk} - y_{pk}\right)^2 \text{ and } RMSE = \sqrt{\frac{1}{P}\sum_{p=1}^{P}\sum_{k=1}^{K}\left(d_{pk} - y_{pk}\right)^2}$$

2. Misclassification error for a classifier.

$$misclassification\ error = \sum_{p=1}^{P} 1\left(d_p \neq y_p\right)$$

where d_p is the target and y_p is the predicted output of pattern p. $1(\cdot)$ is the indicator function.

Terminology

Apparent error: (training error) the error on the training data. What the learning algorithm tries to optimize.

True error: the error that will be obtained in use (over the whole sample space). What we want to optimize *but* unknown.

However, the **apparent error** is not always a good estimate of the **true error**. It is just an optimistic measure of it.

Test error: (out-of-sample error) an estimate of the true error obtained by testing the network on some independent data.

Generally, a larger test set helps provide a greater confidence on the accuracy of the estimate.
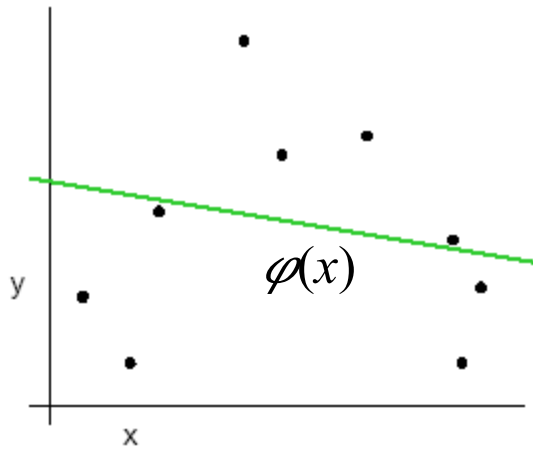
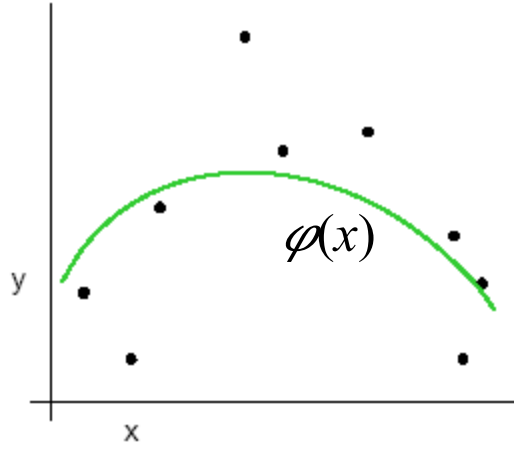# Example: Approximation

Given the following sample data:
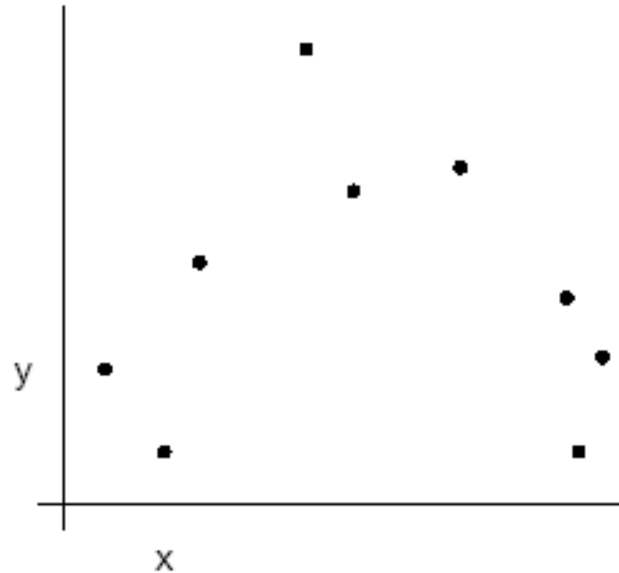
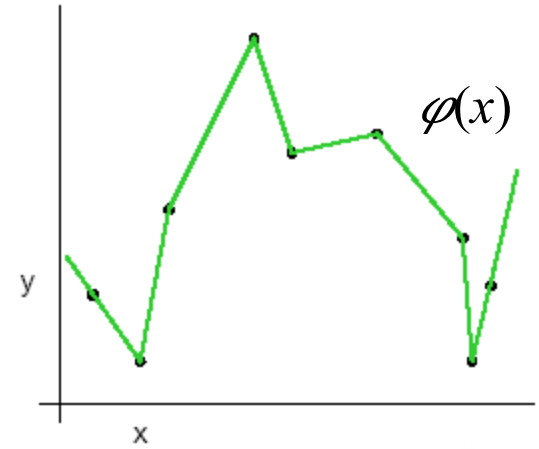Approximate $y = \varphi(x) + \varepsilon$ using an ANN

Example: Approximation

Which one is the best approximation model?

Linear Approximation,
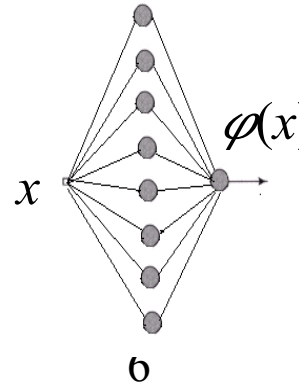Linear Activation fn

$x \longrightarrow \varphi(x)$

Quadratic (non-linear)
Approximation,
Sigmoid Activation fn

$x \longrightarrow \varphi(x)$

Piecewise
Linear
Approximation,
Linear
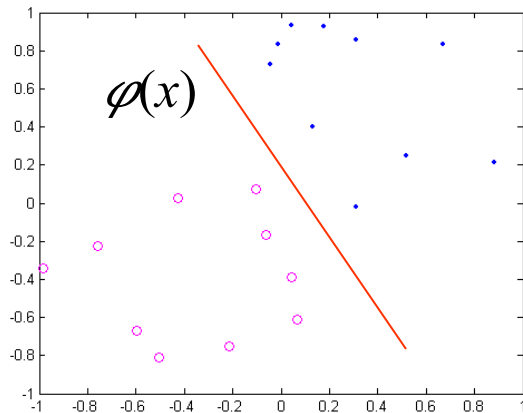Activation fn

$x \longrightarrow \varphi(x)$

6
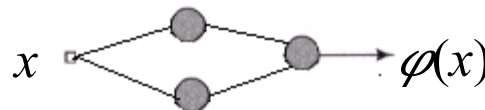
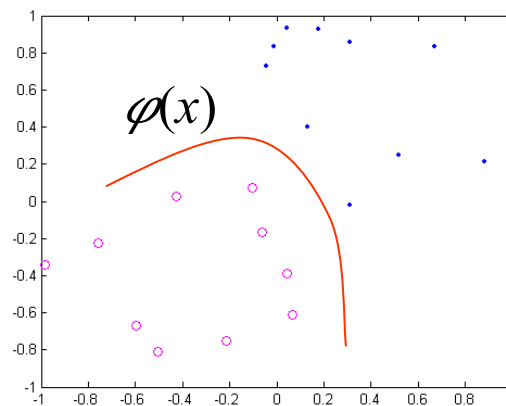## Given the following sample data:

Classify the following data using an ANN.
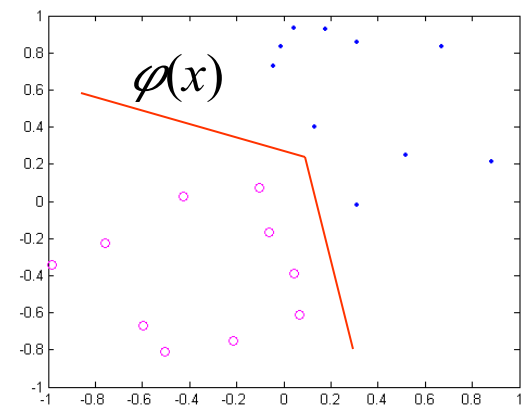
Example: Classification

Which one is the best classification model?

Linear Classification, Threshold Activation fn

Quadratic Classification, *Sigmoid Activation fn*
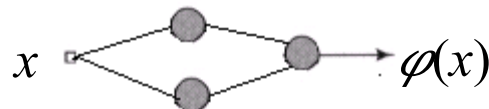
Piecewise Linear Classification, *Linear Activation fn*
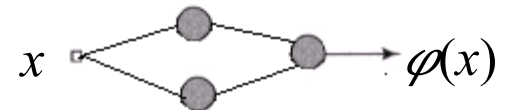
True Error Estimation

Intuition: Choose the model with the best fit to the data?

Meaning: Choose the model that provides the lowest error rate on the entire sample population. Of course, that error rate is the *true error rate*.

"However, to choose a model, we must first know how to estimate the error of a model."

The entire sample population is often unavailable.

Validation

In real applications, we only have access to a finite set of examples, usually smaller than we wanted.

Validation is the approach to use the entire example data available to select the model and estimate the error rate. The validation uses a part of the data to select the model, which is known as the *validation test.*

Validation attempts to solve fundamental problems:

• The final model tends to *overfit* the training data. It is not uncommon to have 100% correct classification on training data.

• There is no way of knowing how well the model performs on unseen data

• The error rate estimate will be overly optimistic (usually lower than the true error rate) .

Validation Methods

An effective approach is to split the entire data into subsets, i.e., Training/Validation/Testing datasets.

Some Validation Methods:

- The Holdout (1/3 - 2/3 Rule)

- Re-sampling techniques

    - Random Subsampling

    - K-fold Cross-Validation

    - Leave one out Cross-Validation

- Three-way data splits

Holdout Method

Split entire dataset into two sets:

- *Training set* (blue): used to train the classifier

- *Testing set* (red): used to estimate the error rate of the trained classifier on unseen data samples

Entire Sample Data

1/3 Testing Set	2/3 Training Set
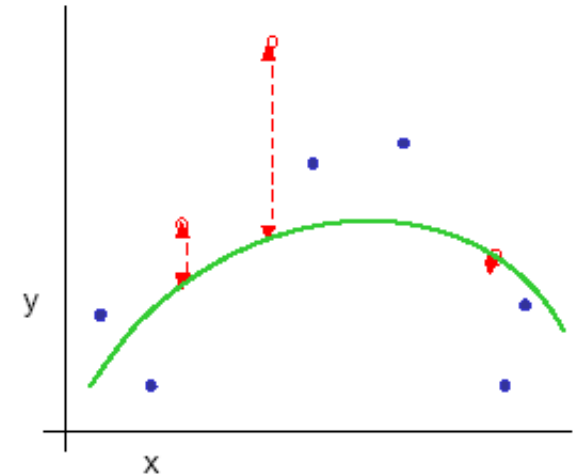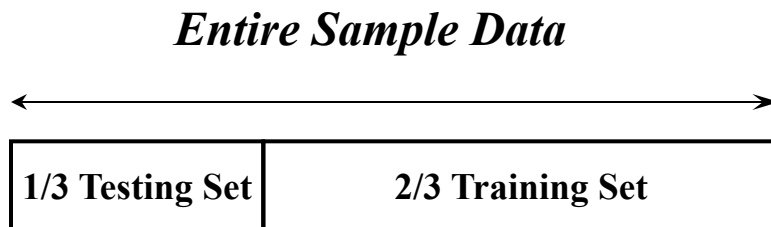
Holdout Method

The holdout method has two basic drawbacks:

• In problems where we have a sparse dataset we may not be able to afford the "luxury" of setting aside a portion of the dataset for testing

• Since it is a single train-and-test experiment, the holdout estimate of error rate will be misleading if we happen to get an "unfortunate" split.

Training Data

$f(x)$

Testing Data

y

x

An "unfortunate" split for an approximation example. Results in large test error.

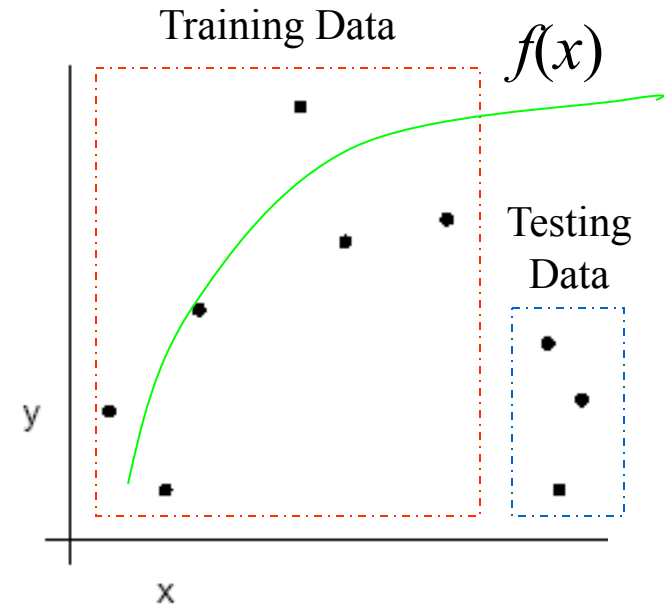# Random Sampling Methods

Limitations of the holdout can be overcome with a family of resampling methods at the expense of more computations:
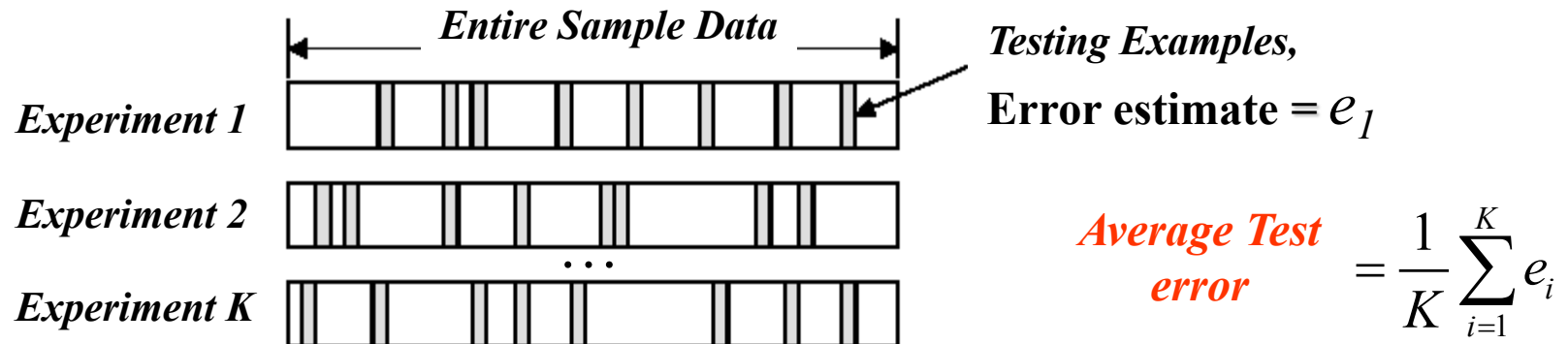
- Random Subsampling

- K-Fold Cross-Validation

- Leave-one-out Cross-Validation

14

K Data Splits
Random SubSampling

Random Subsampling performs *K* data splits of the dataset

• Each split randomly selects a (fixed) no. of examples

• For each data split we retrain the classifier from scratch with the training data

• Examples and estimate e_i with the test examples

Entire Sample Data

Experiment 1

Testing Examples,
Error estimate $= e_1$

Experiment 2

Experiment K

. . .

Average Test error $= \dfrac{1}{K}\sum_{i=1}^{K} e_i$

15

Example: K=2 Data Splits
Random SubSampling

Experiment 1

$x \rightarrow \varphi(x)$ $x \rightarrow \varphi(x)$ $x \rightarrow \varphi(x)$

$e_{1\,(\text{Model A})}$ $e_{1\,(\text{Model B})}$ $e_{1\,(\text{Model C})}$

Experiment 2

$x \rightarrow \varphi(x)$ $x \rightarrow \varphi(x)$ $x \rightarrow \varphi(x)$

$e_{2\,(\text{Model A})}$ $e_{2\,(\text{Model B})}$ $e_{2\,(\text{Model C})}$
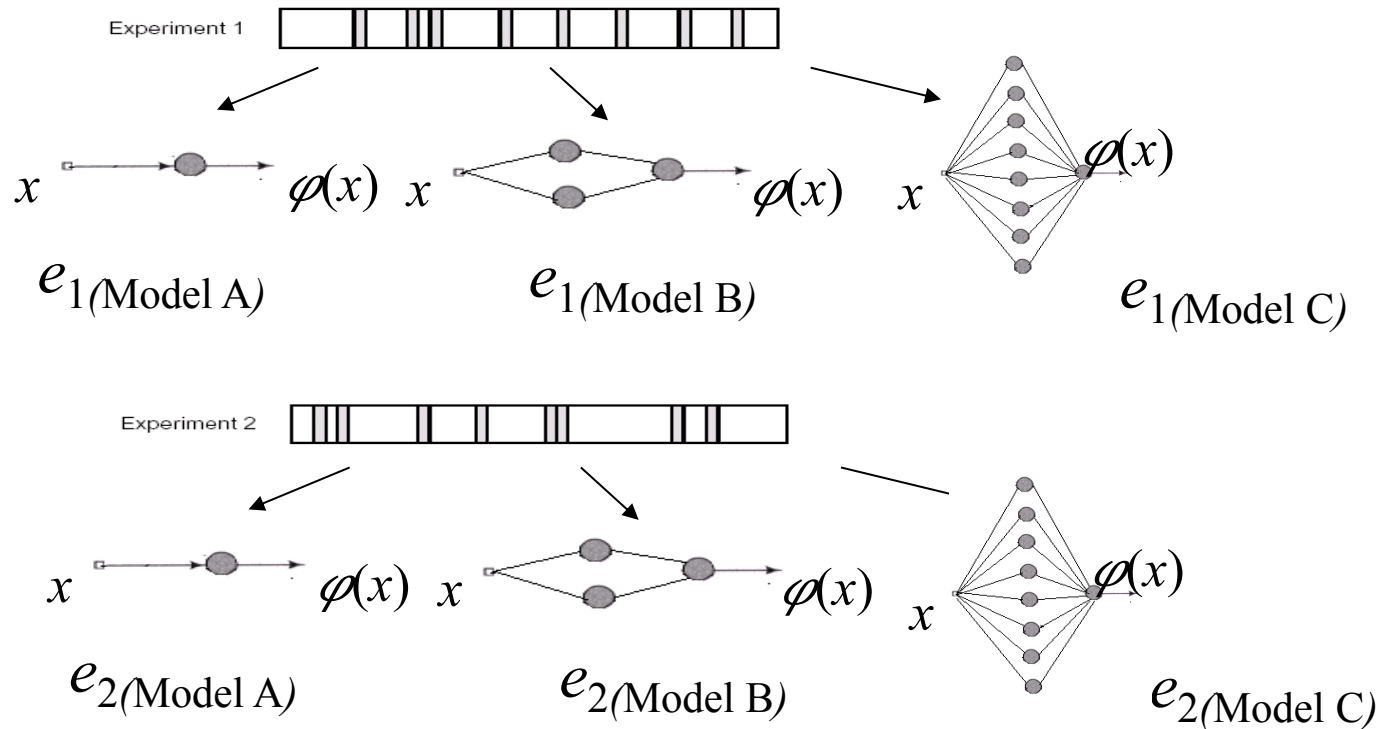
Average Test error $= \dfrac{1}{2}\left(e_1 + e_2\right)$

Choose the model with the best test error, i.e., lowest Average test error!

16

K-fold Cross Validation

### Create a *K*-fold partition of the the dataset:

• For each of *K* experiments, use *K*-1 folds for training and the remaining

• one fold for testing

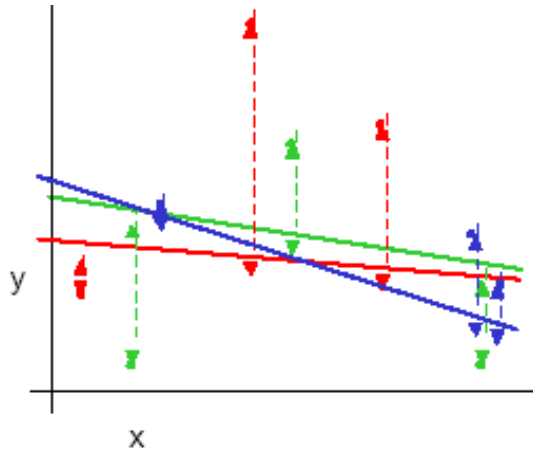$$\text{Average test error} = \frac{1}{K}\sum_{i=1}^{K} e_i$$

This error is also known as the cross-validation error.

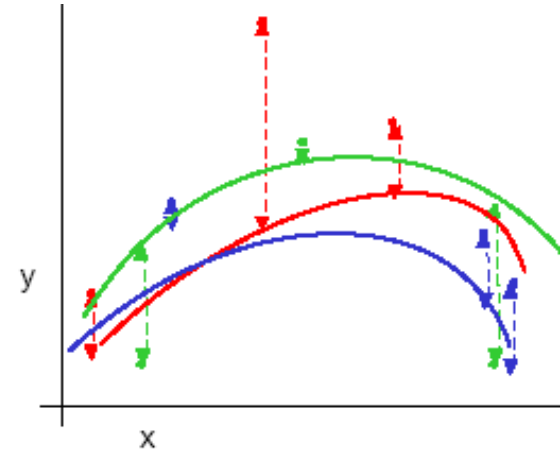### *K*-Fold Cross validation is similar to Random Subsampling

• The *advantage* of *K*-Fold Cross validation is that all examples in the dataset are eventually used for both training and testing
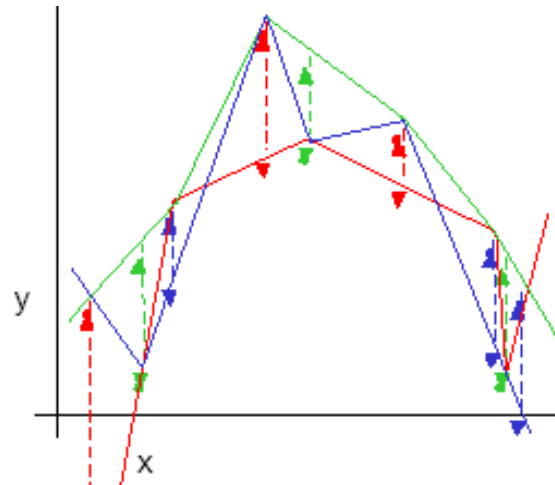
Entire Sample Data

Exp. 1

Exp. 2

Exp. 3

. . .

Exp. K

Testing Example, Error estimate $= e_4$
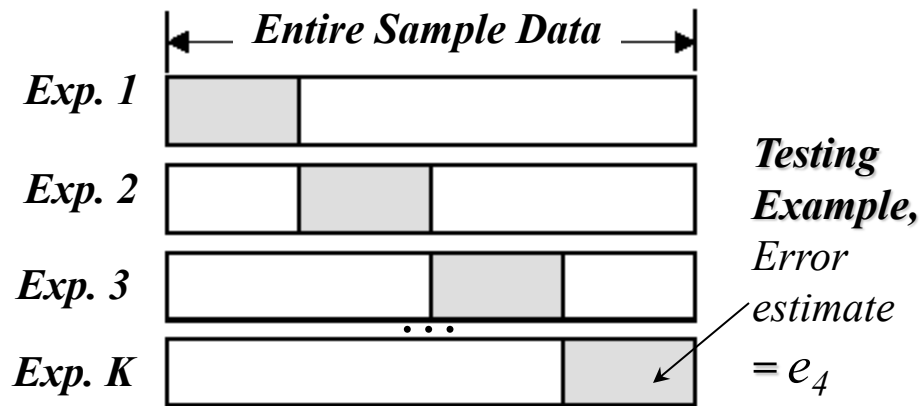
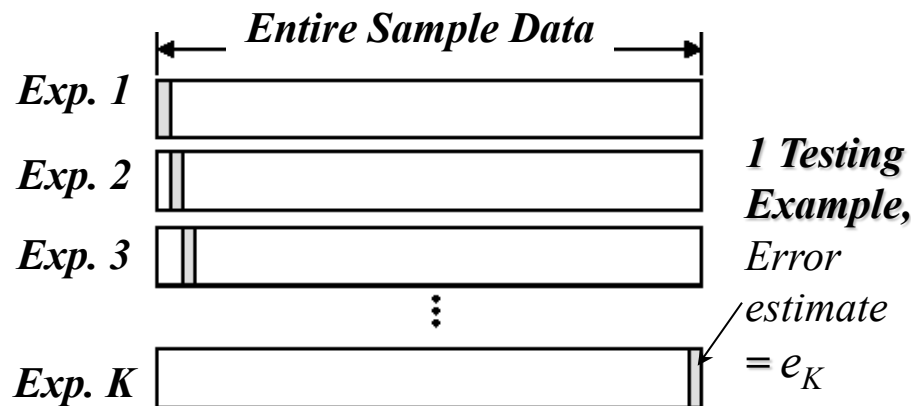# 3-fold Cross Validation

$e_{1(Model\ A)}$

$e_{1(Model\ B)}$

$e_{1(Model\ C)}$

Leave-One-Out (LOO) Cross Validation

Leave-One-Out is the degenerate case of *K*-Fold Cross Validation, where *K* is chosen as the total number of examples:

• For a dataset with N examples, perform N experiments, i.e., $N = K$.

• For each experiment use N-1 examples for training and the remaining one example for testing.

Entire Sample Data

Exp. 1

Exp. 2

Exp. 3

Exp. K

1 Testing Example,
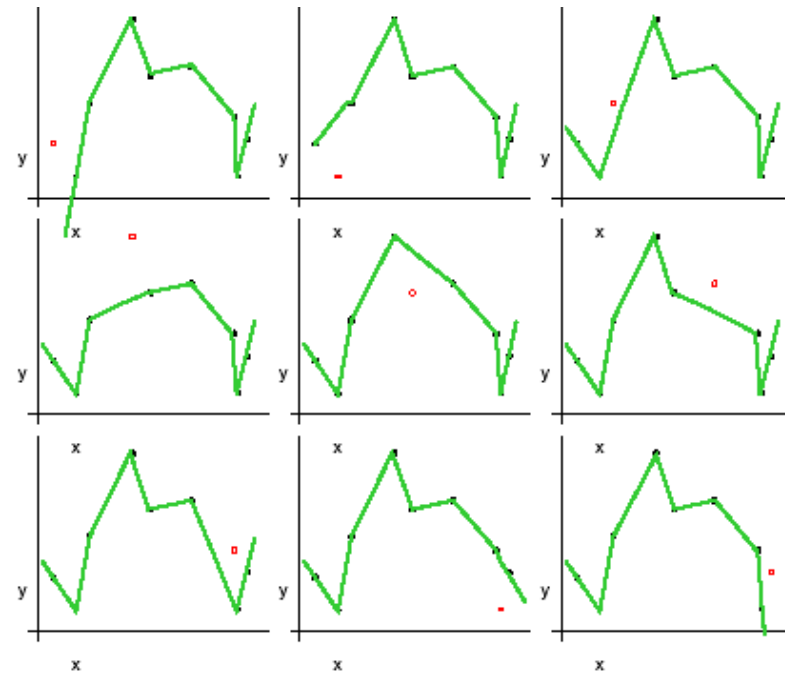*Error estimate* $= e_K$

$$\text{True error estimate} = \text{Average error} = \frac{1}{K}\sum_{i=1}^{K} e_i$$

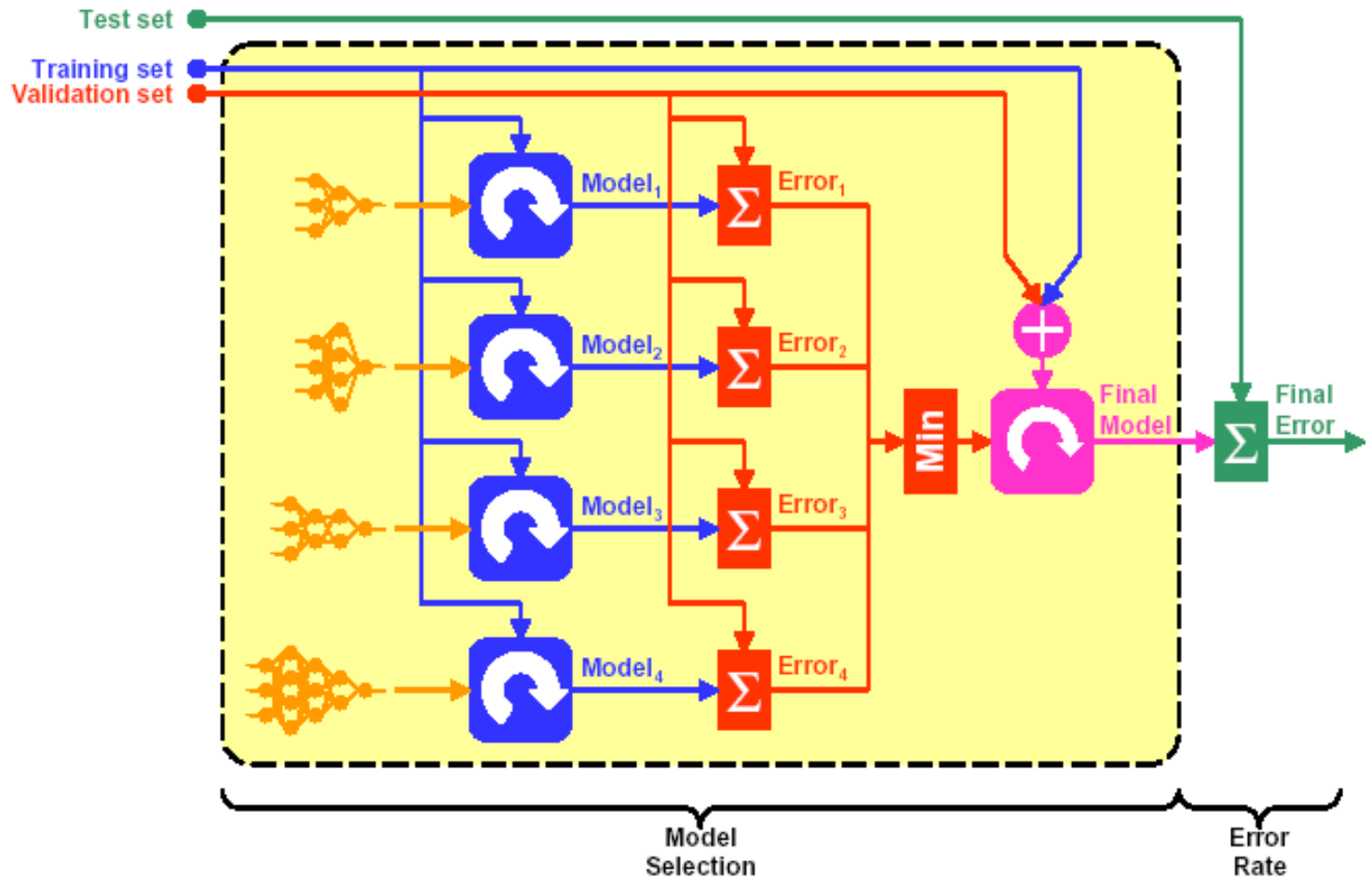# Leave-One-Out Cross Validation
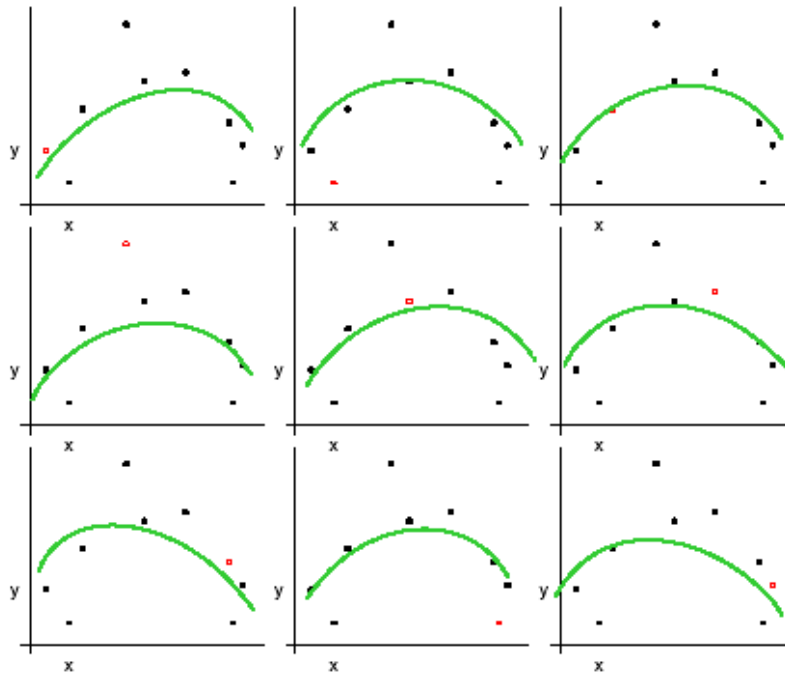
LOOCV Quadratic
Approximation

LOOCV Piecewise
Linear Approximation

Three-Way Data Splits Method

Three-Way Data Splits Method

If model selection and true error estimates are to be computed simultaneously, the data needs to be divided into three disjoint sets:

• **Training set**: examples for *learning* to fit the parameters of several possible classifiers. In the case of DNN, we would use the training set to find the "optimal" weights with the gradient descent rule.

$$\boldsymbol{W}^*, \boldsymbol{b}^* = \underset{\boldsymbol{W}, \boldsymbol{b}}{\arg min} \, J(\boldsymbol{W}, \boldsymbol{b})$$

• **Validation set**: examples to *determine* the error J_m of different models m, using the validation set. The optimal model m^* is given by

$$m^* = \underset{m}{\arg min} \, J_m$$

• **Training + Validation set:** combine examples used to re-train/redesign $model_{m^*}$, and find new "optimal" weights.

• **Test set**: examples used only to *assess* the performance of a *trained model* m^*. We will use the test data to estimate the error rate after we have trained the final model with train + validation data.

22

Three-Way Data Splits Method

Why separate test and validation sets?

• The error rate estimate of the final model on validation data will be biased (smaller than the true error rate) since the validation set is also used to select in the process of final model selection.

• After assessing the final model, an independent test set is required to estimate the performance of the final model.

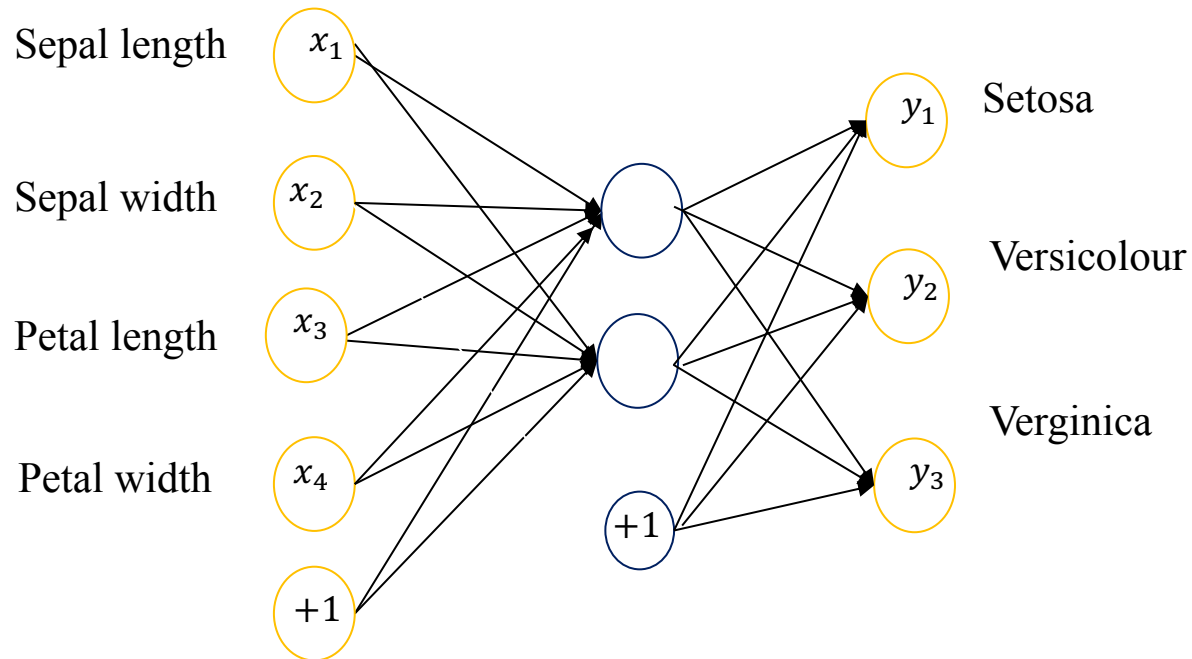**"NO FURTHERING TUNNING OF THE MODEL IS ALLOWED!"**

Iris dataset

Three classes of iris flower: Setosa, versicolour, and virginica
Four features: Sepal length, sepal width, petal length, petal width
150 data points

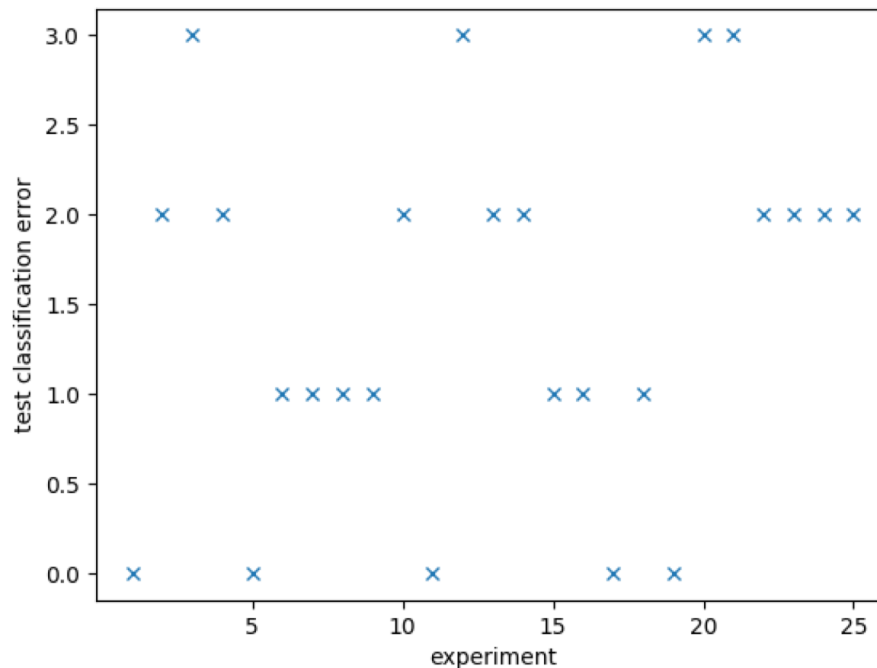FFN with one hidden layer. Determine number of hidden neurons?

Example 1: Random subsampling

150 data points
In each experiment, 50 points for testing and 100 for training

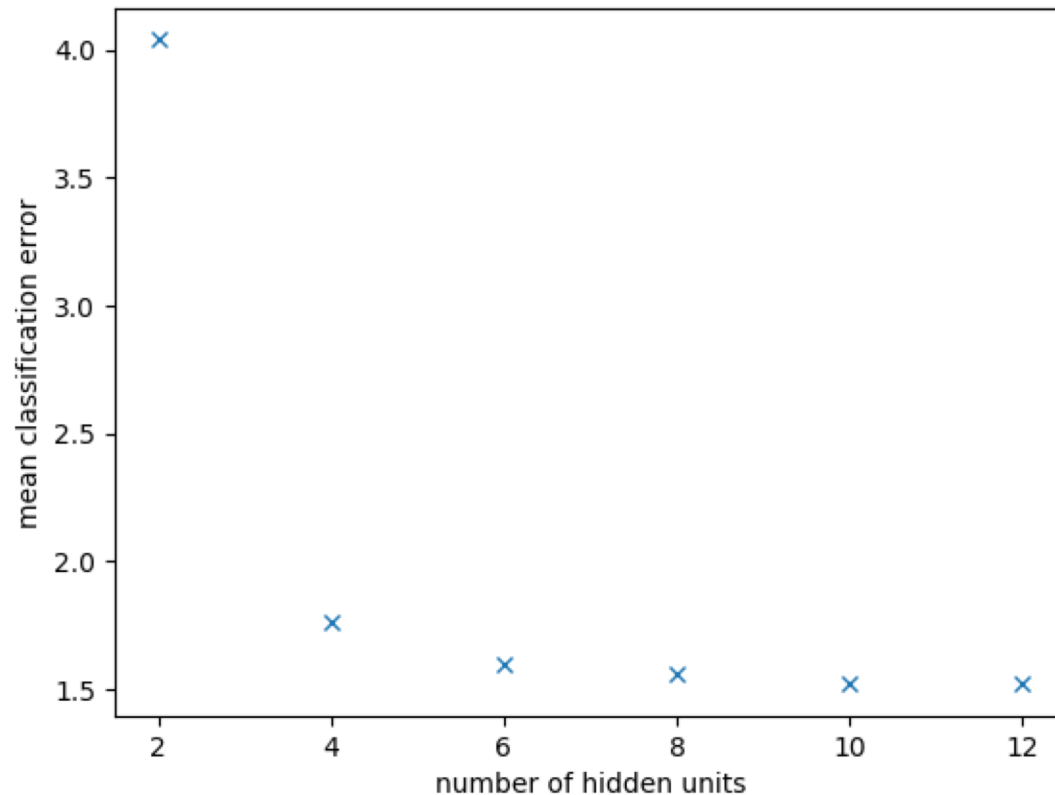Example: 5 hidden neurons, 25 experiments
Misclassification error:

Mean error = 1.48

Example 1

For different number of hidden units, misclassification errors in 25 experiments

Optimum number of hidden units = 10
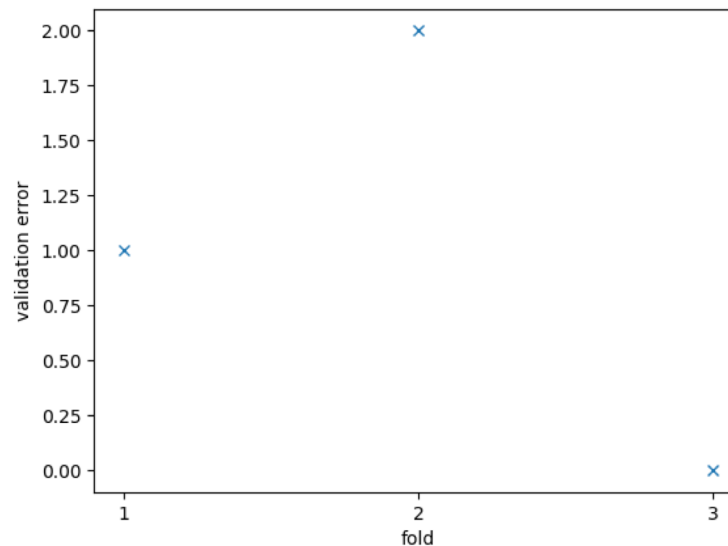
Example 2: Cross validation

150 data points
3-fold cross validation: 50 data points in one fold
Two-folds are used for training and one fold for testing

Example: hidden number of units = 5

3-fold cross-validation (CV) error = 1.0

Example 2

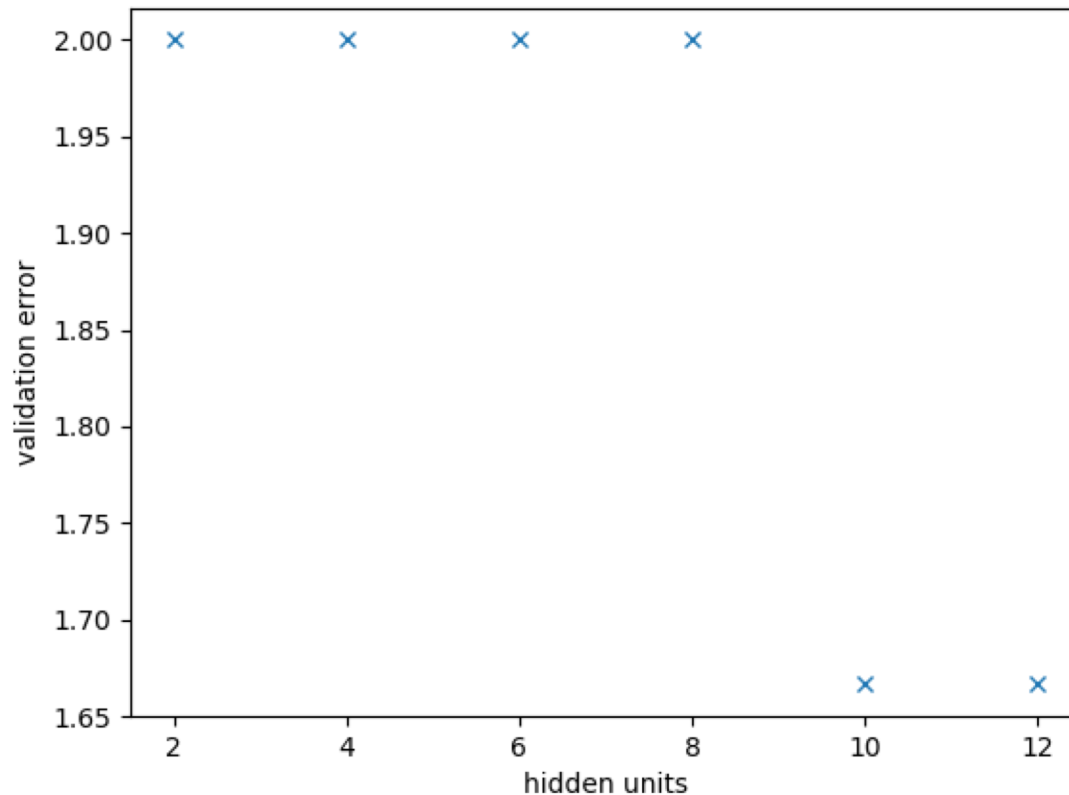Mean CV error for 25 experiments for different number of hidden units:
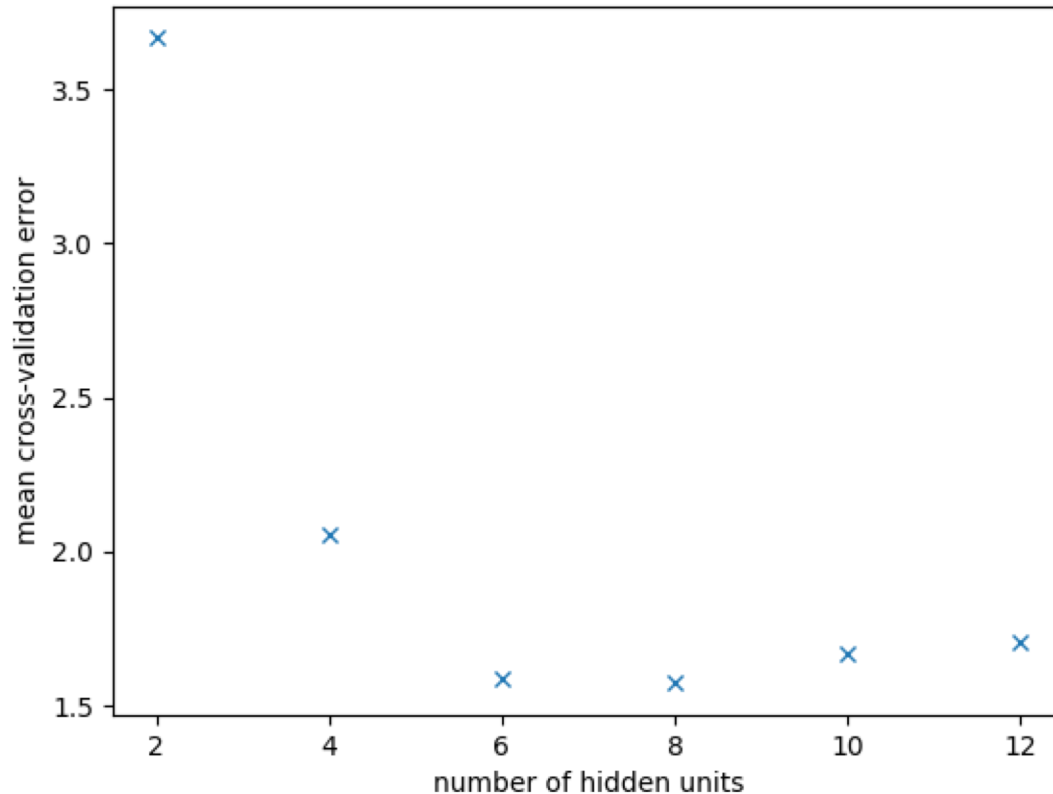
Optimum number of hidden neurons = 8
Cross validation error = 1.57

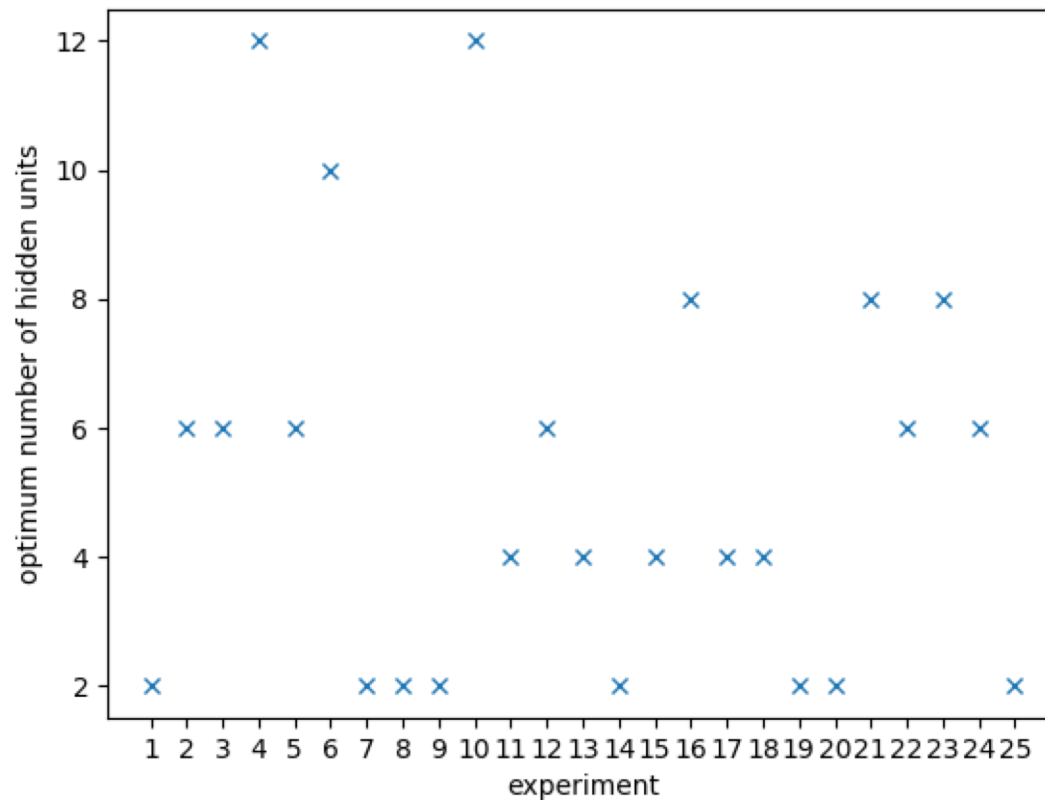# Example 3: Three-way data splits

150 data points
30 data points for testing: 120 data points for cross validation

Example 3

For 25 experiments:

Optimum number of hidden neurons = 2

Example 3
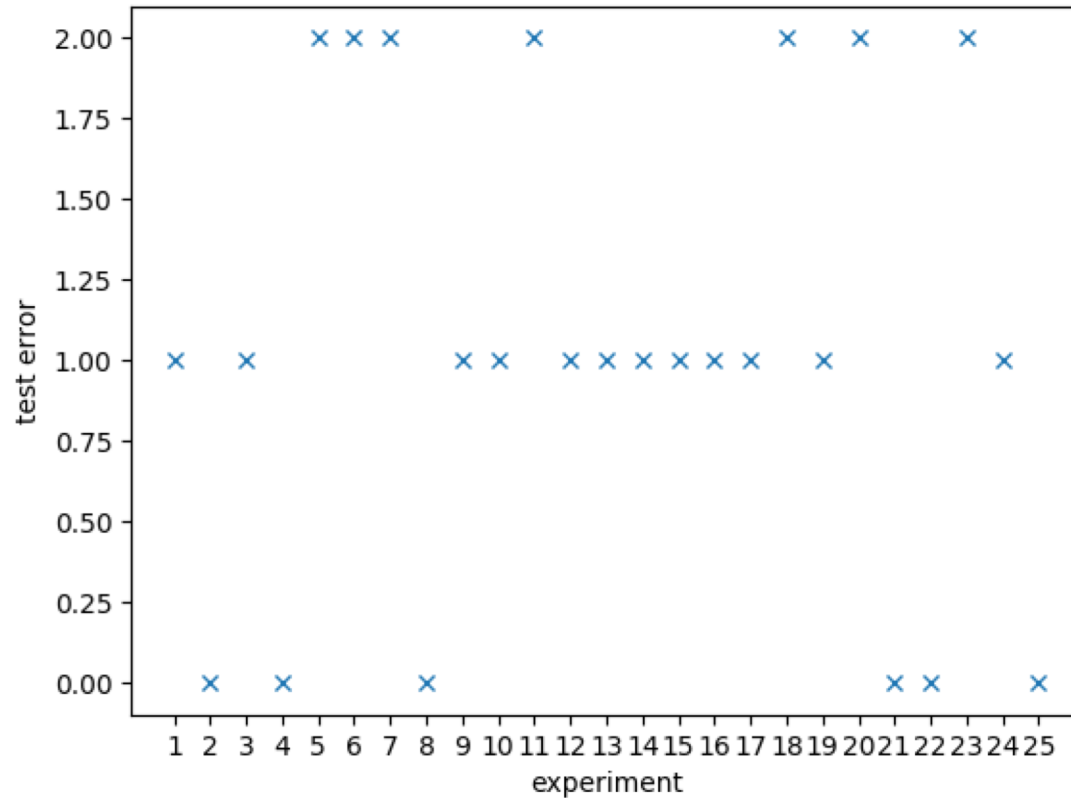
Mean test error = 1.04

Model Complexity

Complex models: models with many adjustable weights and biases will

- more likely to be able to solve the required task,

- more likely to memorize the training data without solving the task.

Simple models: The simpler the model that can learn from the training data is more likely to <u>generalize</u> over the entire sample space.

This is the fundamental trade-off:

• Too simple — cannot do the task, e.g. 5 hidden neurons. (*underfitting*)

• Too complex — cannot generalize from small and noisy datasets well, e.g. 20 hidden neurons. (*overfitting*)

Overfitting

Overfitting is one of the problems that occur during training of neural networks.

The training error of the network is driven to a very small value at the expense of the test error. The network learns to respond correctly to the training inputs but is unable to generalize to produce correct outputs to novel inputs.

Overfitting

Overfitting occurs when the number weights is too large and the network learn training patterns too much.
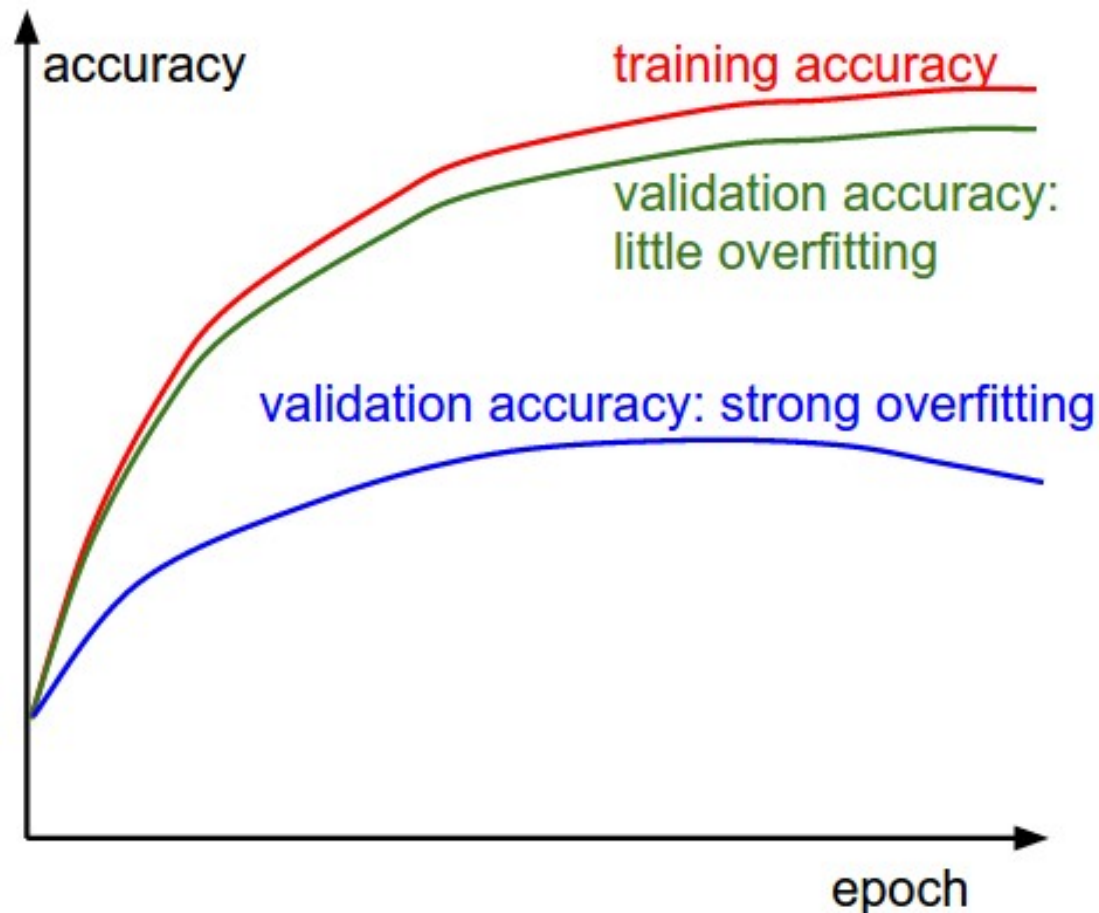
Methods to overcome overfitting

1. Early stopping

2. Regularization of weights

3. Dropouts

Early stopping

Training of the network is to be stopped when the test error starts increasing.

Early stopping can be used in validation by stopping when the validation error is minimum.

36

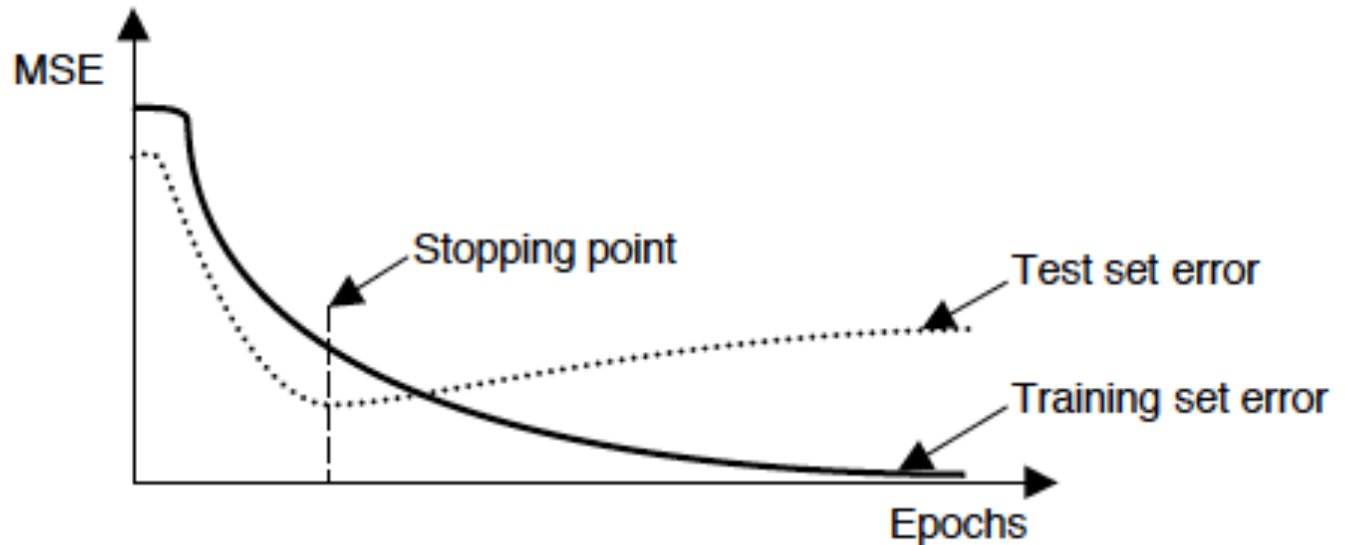# Regularization of weights

During overfitting, some weights attain large values to reduce training error, jeopardizing its ability to generalizing. In order to avoid this, a *penalty* term (*regularization* term) is added to the cost function.

For a network with weights $\boldsymbol{W} = \{w_{ij}\}$ and bias $\boldsymbol{b}$, the penalized (or regularized) cost function $J_1(\boldsymbol{W}, \boldsymbol{b})$ is defined as

$$J_1(\boldsymbol{W}, \boldsymbol{b}) = J(\boldsymbol{W}, \boldsymbol{b}) + \beta_1 \sum_{i,j} |w_{ij}| + \beta_2 \sum_{ij} (w_{ij})^2$$

where $J(\boldsymbol{W}, \boldsymbol{b})$ is the standard cost function (i.e., m.s.e. or cross-entropy),

$$L^1 - norm = \sum_{i,j} |w_{ij}|$$

$$L^2 - norm = \sum_{ij} (w_{ij})^2$$

And β_1 and β_2 are known as L^1 and L^2 regularization (penalty) constants, respectively. These penalties discourage weights from attaining large values.

Regularization of weights

For neural networks, regularization on bias terms is usually not considered and L^2 regularization is most popular.

$$J_1(\boldsymbol{W}, \boldsymbol{b}) = J(\boldsymbol{W}, \boldsymbol{b}) + \beta_2 \sum_{ij} (w_{ij})^2$$

Gradient of the regularized cost wrt weights:

$$\nabla_{\boldsymbol{W}} J_1 = \nabla_{\boldsymbol{W}} J + \beta_2 \frac{\partial \left(\sum_{ij} w_{ij}^2 \right)}{\partial \boldsymbol{W}} \qquad \text{(A)}$$

Regularization of weights

$$L^2 - norm = \sum_{ij} (w_{ij})^2 = w_{11}^2 + w_{12}^2 + \cdots w_{Kn}^2$$

$$\frac{\partial \left(\sum_{ij} (w_{ij})^2 \right)}{\partial \boldsymbol{W}} = \begin{pmatrix} \dfrac{\partial \sum (w_{ij})^2}{\partial w_{11}} & \dfrac{\partial \sum (w_{ij})^2}{\partial w_{12}} & \cdots & \dfrac{\partial \sum (w_{ij})^2}{\partial w_{1k}} \\ \dfrac{\partial \sum (w_{ij})^2}{\partial w_{21}} & \dfrac{\partial \sum (w_{ij})^2}{\partial w_{22}} & \cdots & \dfrac{\partial \sum (w_{ij})^2}{\partial w_{2k}} \\ \vdots & \vdots & \vdots & \vdots \\ \dfrac{\partial \sum (w_{ij})^2}{\partial w_{n1}} & \dfrac{\partial \sum (w_{ij})^2}{\partial w_{n2}} & \cdots & \dfrac{\partial \sum (w_{ij})^2}{\partial w_{nK}} \end{pmatrix} = 2\boldsymbol{W}$$

Regularization of weights

Substituting in (A):

$$\nabla_{\boldsymbol{W}} J_1 = \nabla_{\boldsymbol{W}} J + \beta_2 \frac{\partial\left(\sum_{ij} w_{ij}{}^2\right)}{\partial \boldsymbol{W}} = \nabla_{\boldsymbol{W}} J + 2\beta_2 \boldsymbol{W}$$

For gradient decent learning that uses regularized cost function:
$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \alpha \nabla_{\boldsymbol{W}} J_1$$

Substituting above:

$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \alpha(\nabla_{\boldsymbol{W}} J + \beta \boldsymbol{W})$$

where $\beta = 2\beta_2$
β is known as the *weight decay parameter*.

Classification of MNIST images

http://yann.lecun.com/exdb/mnist/
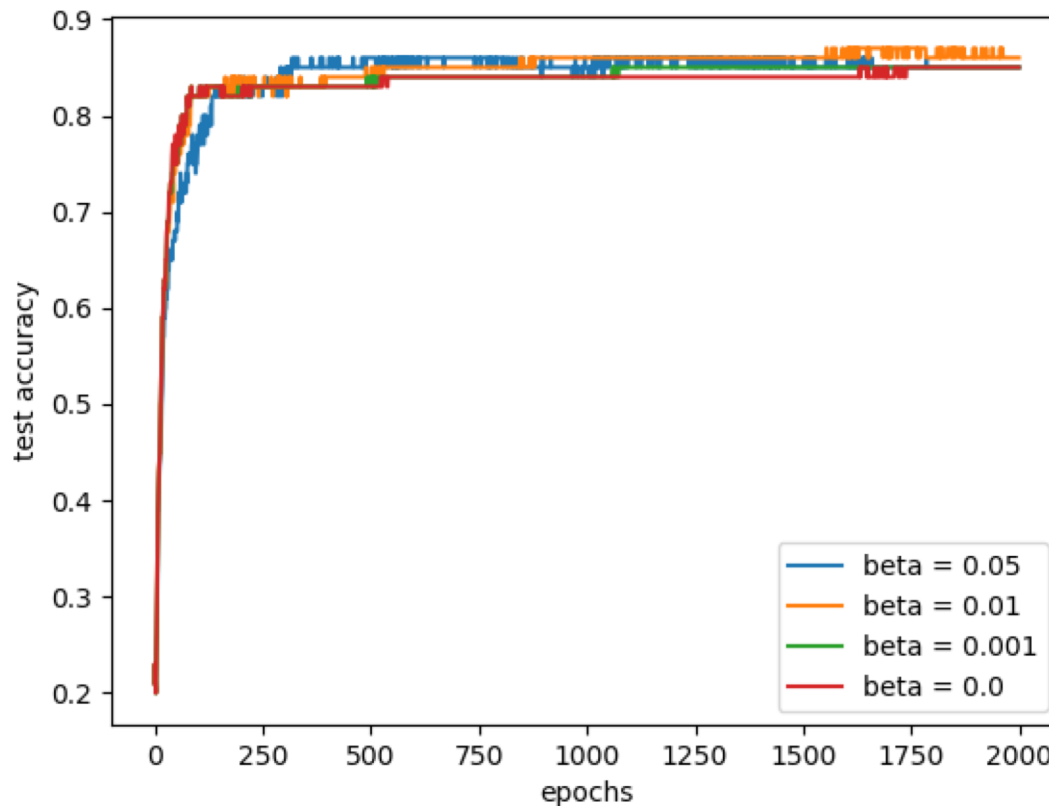

Each image is 28x28 size.
Intensities are in the range [0, 255].
Training set: 60,000
Test set: 10,000

# Example 4: Training with different weight regularization

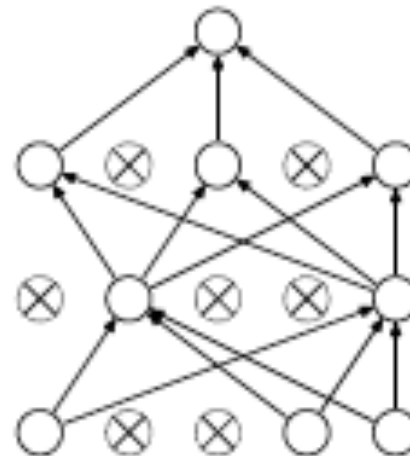600 training patterns and 100 test patterns

Dropouts

Deep neural networks with a large number of parameters (weights, biases, etc.) are powerful learning machines. Overfitting can be avoided by training only a fraction of weights in each iteration.

The key idea of 'dropouts' is to randomly drop neurons (along with their connections) from the networks during training. This prevents neurons from co-adapting and thereby reduces overfitting.
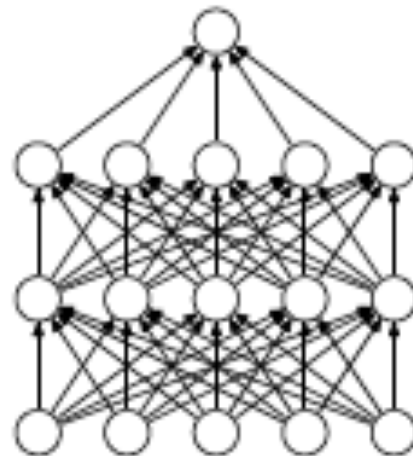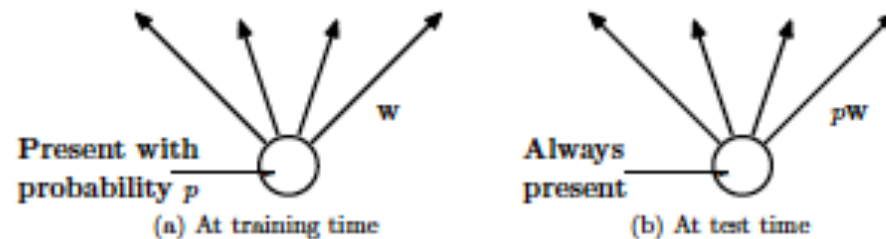

(a) Standard Neural Net   (b) After applying dropout.

43

Dropouts

Present with probability p W   (a) At training time

Always present pW (b) At test time

The units (neurons) are presented with a probability p and presented to the next layer with weight W to the next layer at the training time.

At test time, the weights are always present and presented to the network with weights multiplied by p. The output at the test time is same as the expected output at the training time.

Applying dropouts result in a 'thinned network' that consists of only neurons that survived.

44

Dropouts

Dropout neural networks can be trained in a similar manner to standard networks. The only difference is that in each mini-batch training, we train on a thinned network dropping out units. The gradients of each parameter are averaged over the training cases in each mini-batch

Example 5: Training with dropouts