Artificial Intelligence

Lecturer 8 – Machine Learning

Brigitte Jaumard
Dept of Computer Science and Software Engineering
Concordia University
Montreal (Quebec) Canada

Introduction of Machine learning

- Definitions of Machine learning…
  - → A process by which a system improves its performance `[Simon, 1983]`
  - → Any computer program that improves its performance at some task through experience `[Mitchell, 1997]`
  - → Programming computers to optimize a performance criterion using example data or past experience `[Alpaydin, 2004]`
- Representation of the learning problem `[Mitchell, 1997]`

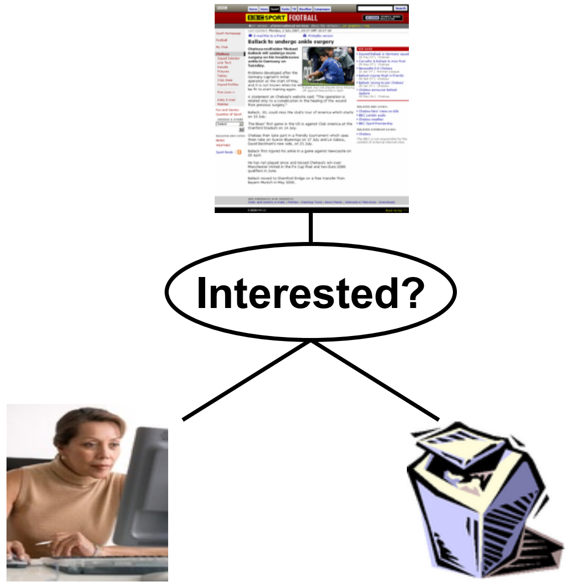
  Learning = Improving with experience at some task

  - Improve over task **T**
  - With respect to performance measure **P**
  - Based on experience **E**

Application examples of ML (1)

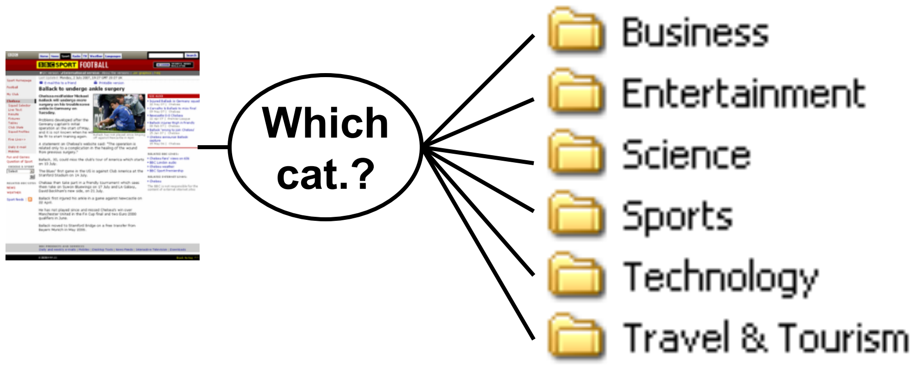
Web pages filtering problem

- **T**: to predict which Web pages a given user is interested in
- **P**: % of Web pages correctly predicted
- **E**: a set of Web pages identified as interested/uninterested for the user



Web pages categorization problem

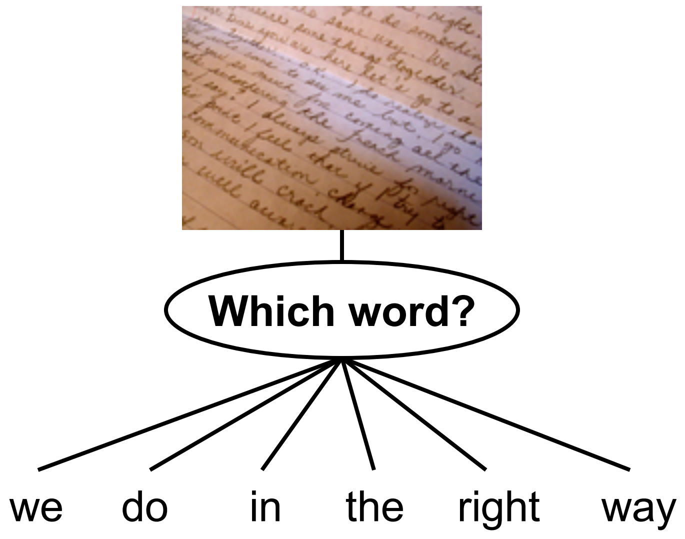
- **T**: to categorize Web pages in predefined categories
- **P**: % of Web pages correctly categorized
- **E**: a set of Web pages with specified categories



Application examples of ML (2)

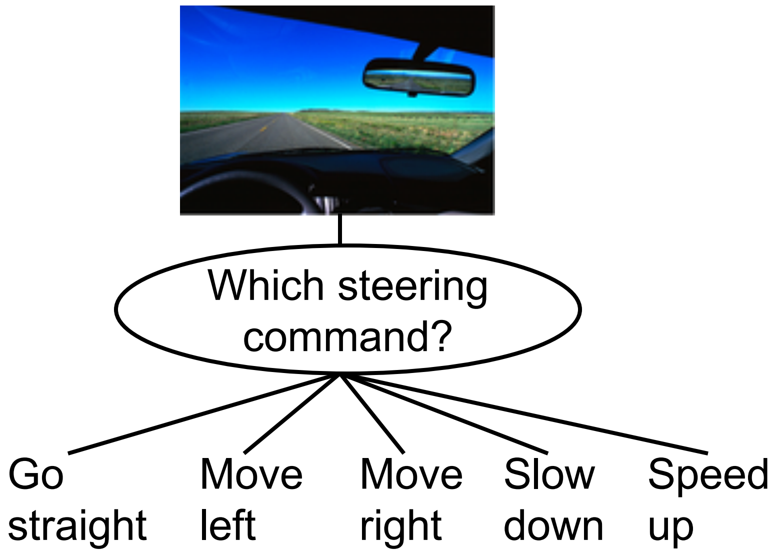
Handwriting recognition problem

- **T**: to recognize and classify handwritten words within images
- **P**: % of words correctly classified
- **E**: a database of handwritten words with given classifications (i.e., labels)



Robot driving problem

- **T**: to drive on public highways using vision sensors
- **P**: average distance traveled before an error (as judged by human overseer)
- **E**: a sequence of images and steering commands recorded while observing a human driver



Key elements of a ML problem (1)

- Selection of the training examples
  - Direct or indirect training feedback
  - With teacher (i.e., with labels) or without
  - The training examples set should be representative of the future test examples
- Choosing the target function (a.k.a. hypothesis, concept, etc.)
  - F: $X \rightarrow \{0,1\}$
  - F: $X \rightarrow$ a set of labels
  - F: $X \rightarrow R^+$ (i.e., the positive real numbers domain)
  - …

Key elements of a ML problem (2)

- Choosing a representation of the target function
  - A polynomial function
  - A set of rules
  - A decision tree
  - A neural network
  - …
- Choosing a learning algorithm that learns (approximately) the target function
  - Regression-based
  - Rule induction
  - ID3 or C4.5
  - Back-propagation
  - …

Issues in Machine Learning (1)

- Learning algorithm
  - What algorithms can approximate the target function?
  - Under which conditions does a selected algorithm converge (approximately) to the target function?
  - For a certain problem domain and given a representation of examples which algorithm performs best?
- Training examples
  - How many training examples are sufficient?
  - How does the size of the training set influence the accuracy of the learned target function?
  - How does noise and/or missing-value data influence the accuracy?

Issues in Machine Learning (2)

- Learning process
  - What is the best strategy for selecting a next training example? How do selection strategies alter the complexity of the learning problem?
  - How can prior knowledge (held by the system) help?
- Learning capability
  - What target function should the system learn?

    Representation of the target function: expressiveness vs. complexity
  - What are the theoretical limits of learnability?
  - How can the system generalize from the training examples?

    To avoid the overfitting problem
  - How can the system automatically alter its representation?

    To improve its ability to represent and learn the target function

Types of learning problems

- A rough (and somewhat outdated) classification of learning problems:
  - **Supervised learning**, where we get a set of training inputs and outputs
    - classification, regression
  - **Unsupervised learning**, where we are interested in capturing inherent organization in the data
    - clustering, density estimation
  - **Reinforcement learning**, where we only get feedback in the form of how well we are doing (not what we should be doing)
    - Planning

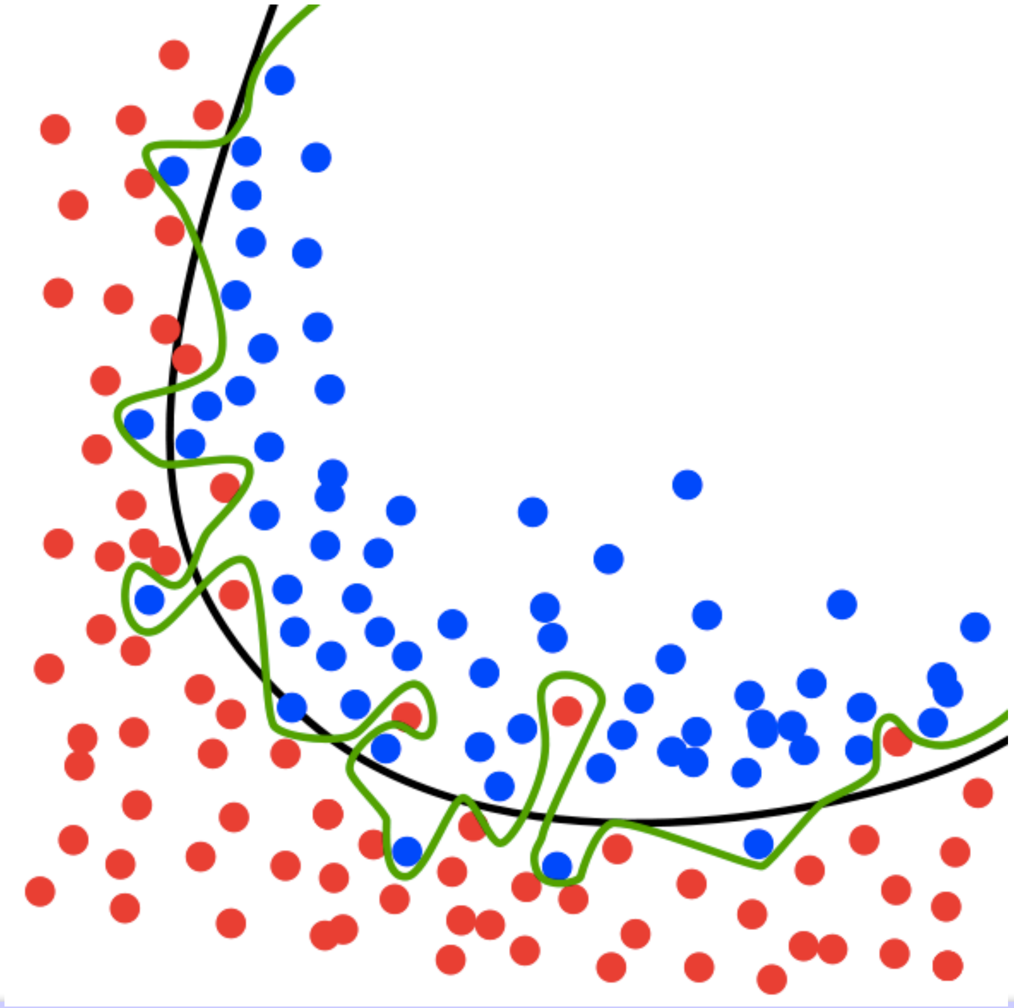
EVALUATION

Evaluating performance

- Different objectives:
  - Selecting the right model for a problem.
  - Testing performance of a new algorithm.
  - Evaluating impact on a new application.

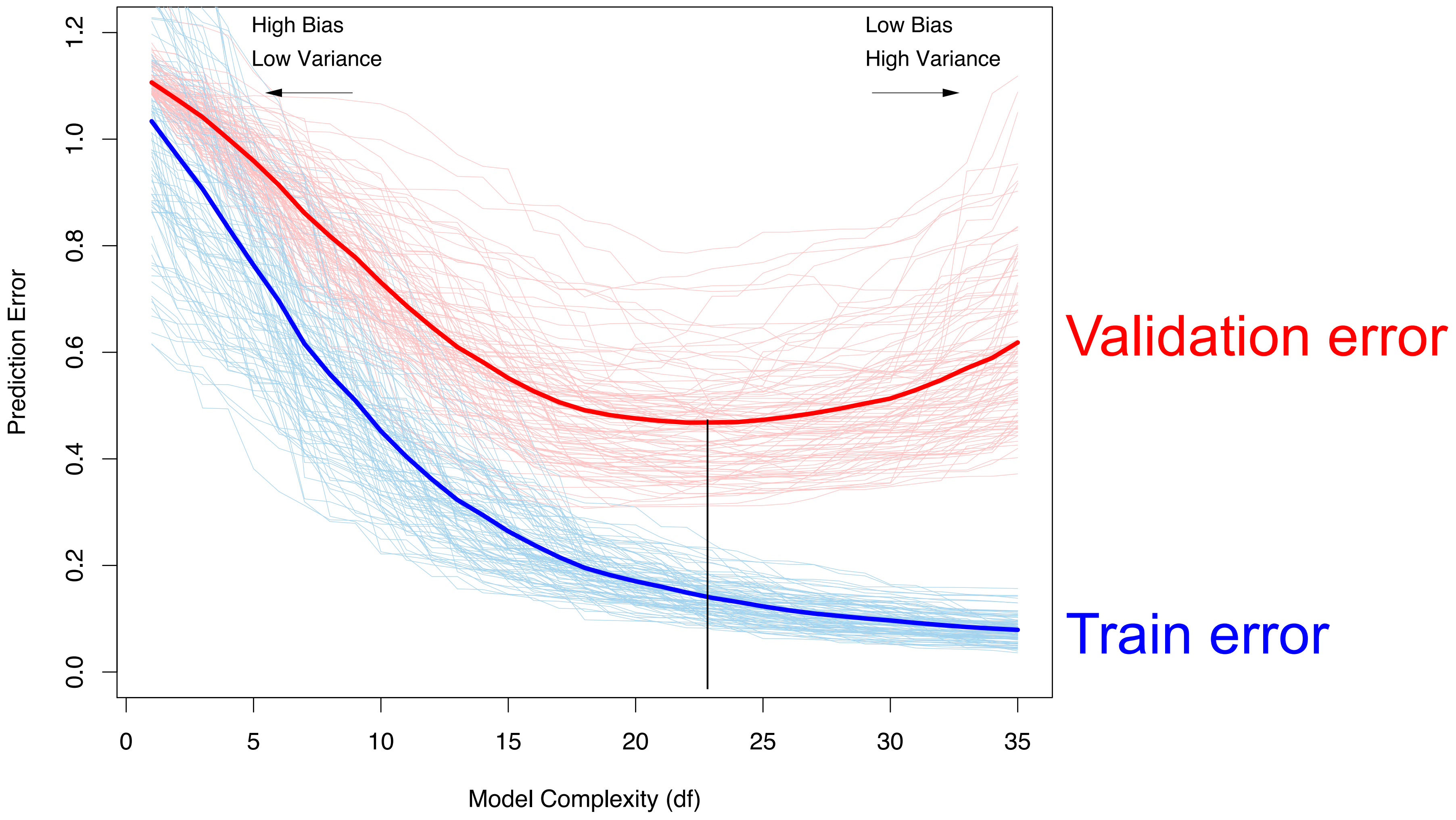
Overfitting

- Adding more degrees of freedom (more features) always seems to improve the solution!



Minimizing the error

- Find the low point in the validation error:



Performance metrics for classification

- Not all errors have equal impact!
- There are different types of mistakes, particularly in the classification setting.

Performance metrics for classification

- Not all errors have equal impact!

- There are different types of mistakes, particularly in the classification setting.
  - E.g. Consider the diagnostic of a disease. Two types of mis-diagnostics:
    - Patient does not have disease but received positive diagnostic (Type I error);
    - Patient has disease but it was not detected (Type II error).

Performance metrics for classification

- Not all errors have equal impact!
- There are different types of mistakes, particularly in the classification setting.
  - E.g. Consider the diagnostic of a disease. Two types of mis-diagnostics:
    - Patient does not have disease but received positive diagnostic (Type I error);
    - Patient has disease but it was not detected (Type II error).
  - E.g. Consider the problem of spam classification:
    - A message that is not spam is assigned to the spam folder (Type I error);
    - A message that is spam appears in the regular folder (Type II error).

Performance metrics for classification

- Not all errors have equal impact!

- There are different types of mistakes, particularly in the classification setting.
  - E.g. Consider the diagnostic of a disease. Two types of mis-diagnostics:
    - Patient does not have disease but received positive diagnostic (Type I error);
    - Patient has disease but it was not detected (Type II error).
  - E.g. Consider the problem of spam classification:
    - A message that is not spam is assigned to the spam folder (Type I error);
    - A message that is spam appears in the regular folder (Type II error).

- How many Type I errors are you willing to tolerate, for a reasonable rate of Type II errors ?

Terminology

- Type of classification outputs:
  - True positive (m11): Example of class 1 predicted as class 1.
  - False positive (m01): Example of class 0 predicted as class 1. Type 1 error.
  - True negative (m00): Example of class 0 predicted as class 0.
  - False negative (m10): Example of class 1 predicted as class 0. Type II error.
- Total number of instances: m = m00 + m01 + m10 + m11

Terminology

- Type of classification outputs:
  - True positive (m11): Example of class 1 predicted as class 1.
  - False positive (m01): Example of class 0 predicted as class 1. Type 1 error.
  - True negative (m00): Example of class 0 predicted as class 0.
  - False negative (m10): Example of class 1 predicted as class 0. Type II error.
- Total number of instances: m = m00 + m01 + m10 + m11
- Error rate: (m01 + m10) / m
  - If the classes are imbalanced (e.g. 10% from class 1, 90% from class 0), one can achieve low error (e.g. 10%) by classifying everything as coming from class 0!

Confusion matrix

- Many software packages output this matrix.

$$\begin{bmatrix} m_{00} & m_{01} \\ m_{10} & m_{11} \end{bmatrix}$$

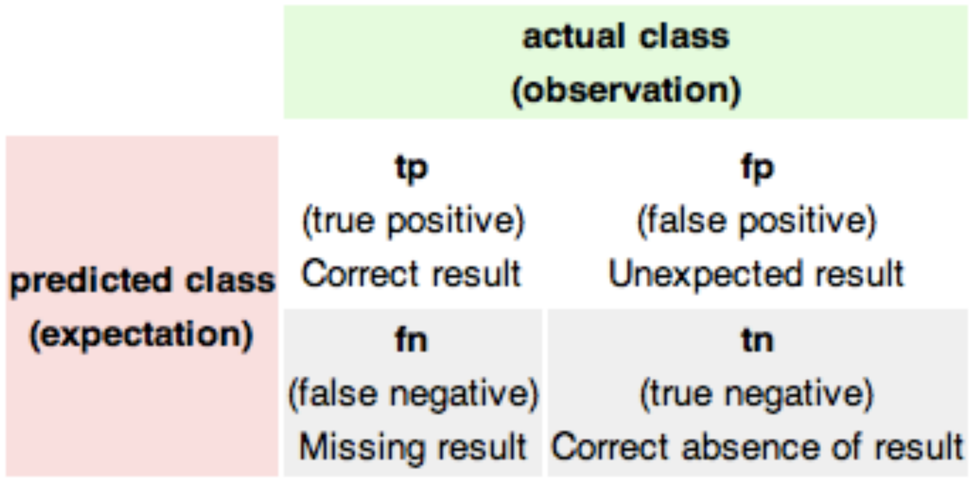
Confusion matrix

- Many software packages output this matrix.

$$\begin{bmatrix} m_{00} & m_{01} \\ m_{10} & m_{11} \end{bmatrix}$$

- Be careful! Sometimes the format is slightly different

(E.g. *http://en.wikipedia.org/wiki/Precision_and_recall#Definition_.28classification_context.29*)



	actual class (observation)	
predicted class (expectation)	**tp** (true positive) Correct result	**fp** (false positive) Unexpected result
	fn (false negative) Missing result	**tn** (true negative) Correct absence of result

Common measures

Accuracy	= (TP+ TN) / (TP + FP + FN + TN)
Precision	= True positives / Total number of declared positives
	= TP / (TP+ FP)
Recall	= True positives / Total number of actual positives
	= TP / (TP + FN)

Common measures

- Accuracy = (TP+ TN) / (TP + FP + FN + TN)
- Precision = True positives / Total number of declared positives

  = TP / (TP+ FP)
- Recall = True positives / Total number of actual positives

  = TP / (TP + FN)

Text classification

- Sensitivity is the same as recall.
- Specificity = True negatives / Total number of actual negatives

  = TN / (FP + TN)

Medicine

Common measures

- Accuracy = (TP+ TN) / (TP + FP + FN + TN)
- Precision = True positives / Total number of declared positives

  = TP / (TP+ FP)

Text classification

- Recall = True positives / Total number of actual positives

  = TP / (TP + FN)

Medicine

- Sensitivity is the same as recall.
- Specificity = True negatives / Total number of actual negatives

  = TN / (FP + TN)

- False positive rate = FP / (FP + TN)

Common measures

- Accuracy = (TP+ TN) / (TP + FP + FN + TN)
- Precision = True positives / Total number of declared positives

  = TP / (TP+ FP)

Text classification

- Recall = True positives / Total number of actual positives

  = TP / (TP + FN)

- Sensitivity is the same as recall.

Medicine

- Specificity = True negatives / Total number of actual negatives

  = TN / (FP + TN)

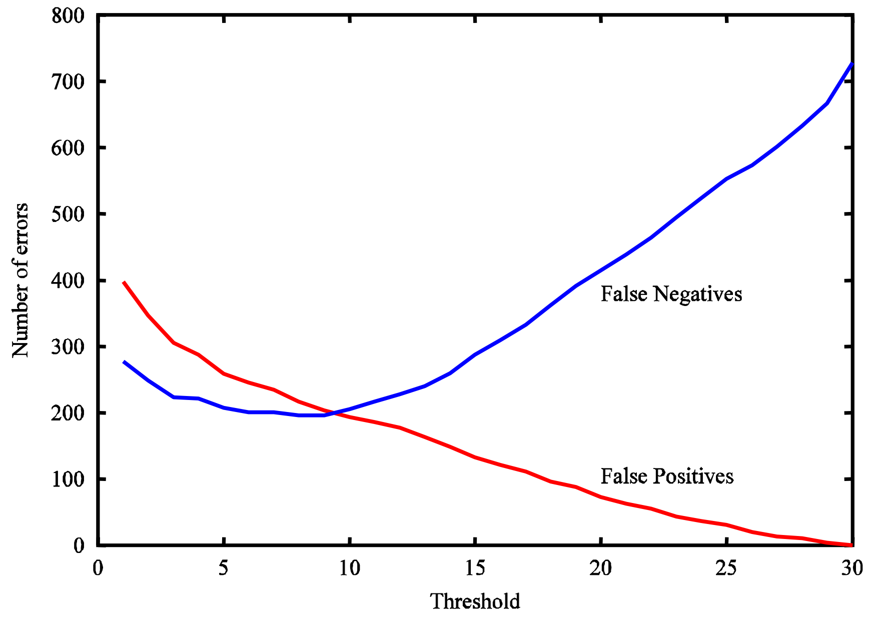
- False positive rate = FP / (FP + TN)
- F1 measure

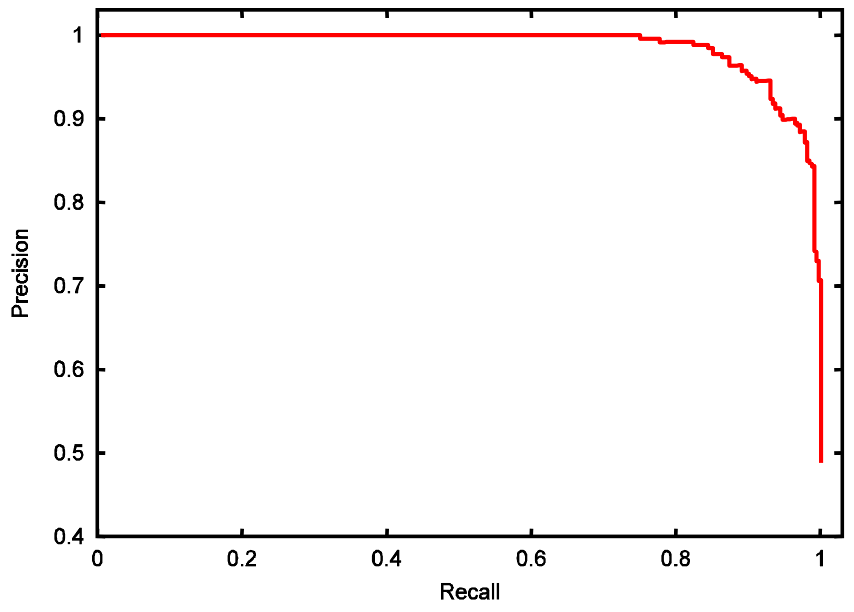
$$F = 2 \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}}$$

Trade-off

- Often have a trade-off between false positives and false negatives.

E.g. Consider 30 different classifiers trained on a class. Classify a new sample as positive if K classifiers output positive. Vary K between 0 and 30.



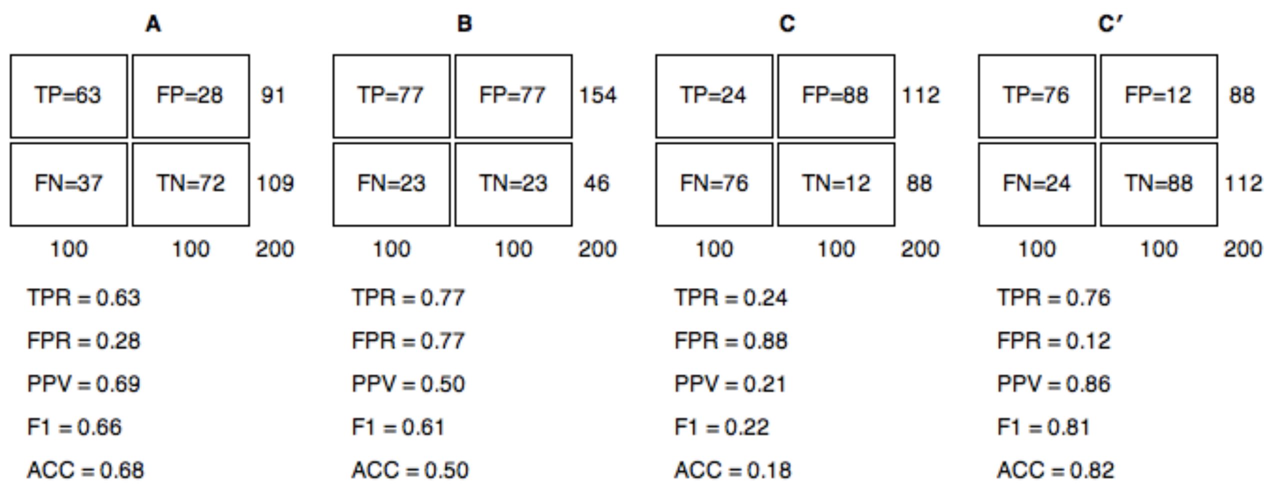


Receiver-operator characteristic (ROC) curve

- Characterizes the performance of a binary classifier over a range of classification thresholds

Data from 4 prediction results:

ROC curve:



A

TP=63	FP=28	91
FN=37	TN=72	109
100	100	200

TPR = 0.63
FPR = 0.28
PPV = 0.69
F1 = 0.66
ACC = 0.68

B

TP=77	FP=77	154
FN=23	TN=23	46
100	100	200

TPR = 0.77
FPR = 0.77
PPV = 0.50
F1 = 0.61
ACC = 0.50

C

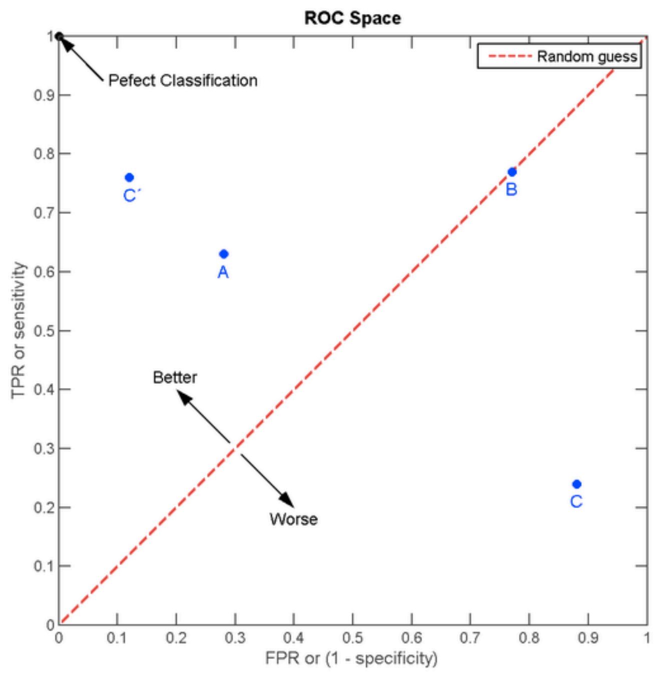
TP=24	FP=88	112
FN=76	TN=12	88
100	100	200

TPR = 0.24
FPR = 0.88
PPV = 0.21
F1 = 0.22
ACC = 0.18

C′

TP=76	FP=12	88
FN=24	TN=88	112
100	100	200

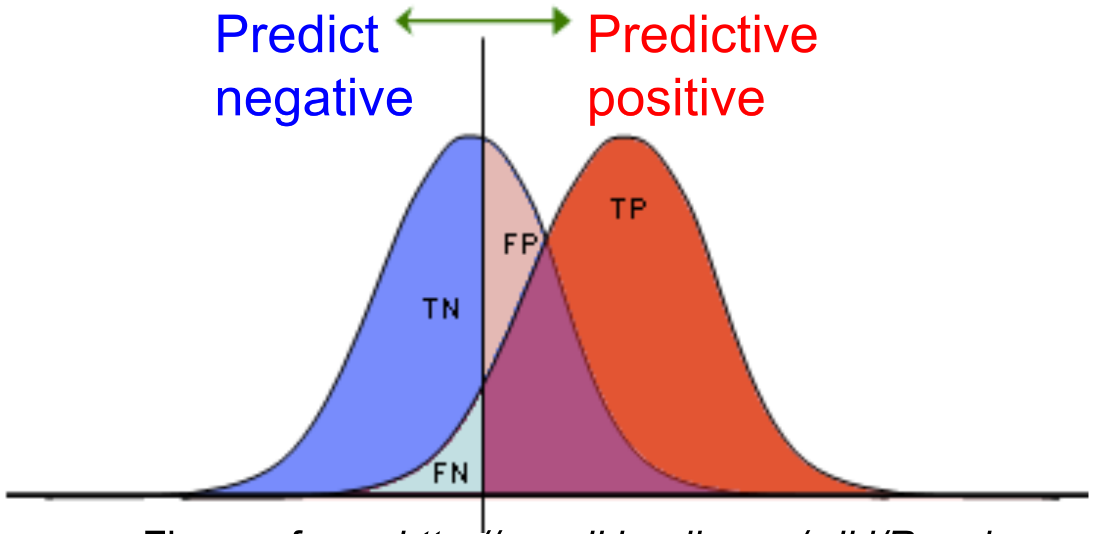
TPR = 0.76
FPR = 0.12
PPV = 0.86
F1 = 0.81
ACC = 0.82

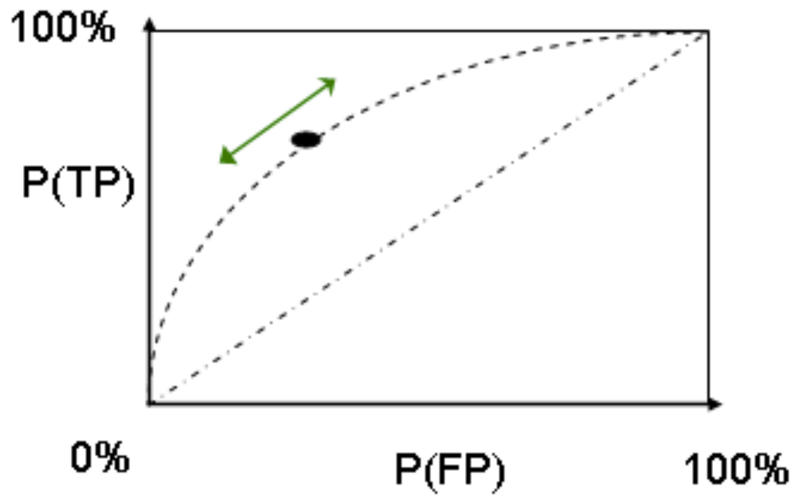


Example from: *http://en.wikipedia.org/wiki/Receiver_operating_characteristic*

Understanding the ROC curve

- Consider a classification problem where data is generated by 2 Gaussians (blue = negative class; red = positive class).
- Consider the decision boundary (shown as a vertical line on the left figure), where you predict Negative on the left of the boundary and predict Positive on the right of the boundary.
- Changing that boundary defines the ROC curve on the right.

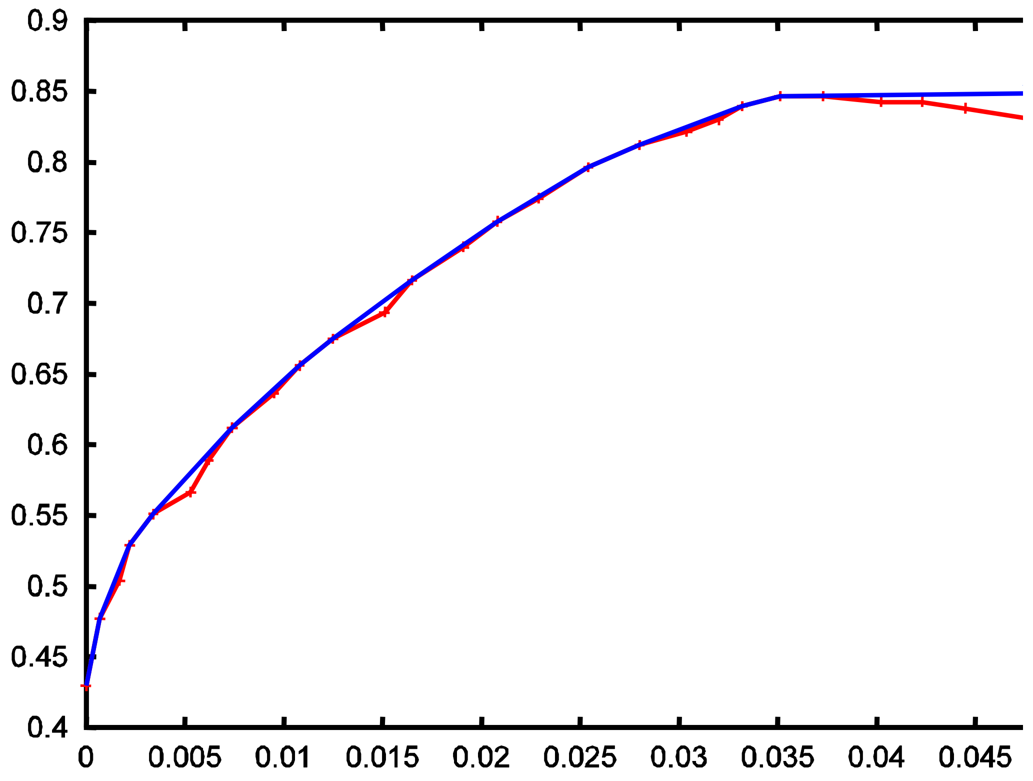




Figures from: *http://en.wikipedia.org/wiki/Receiver_operating_characteristic*

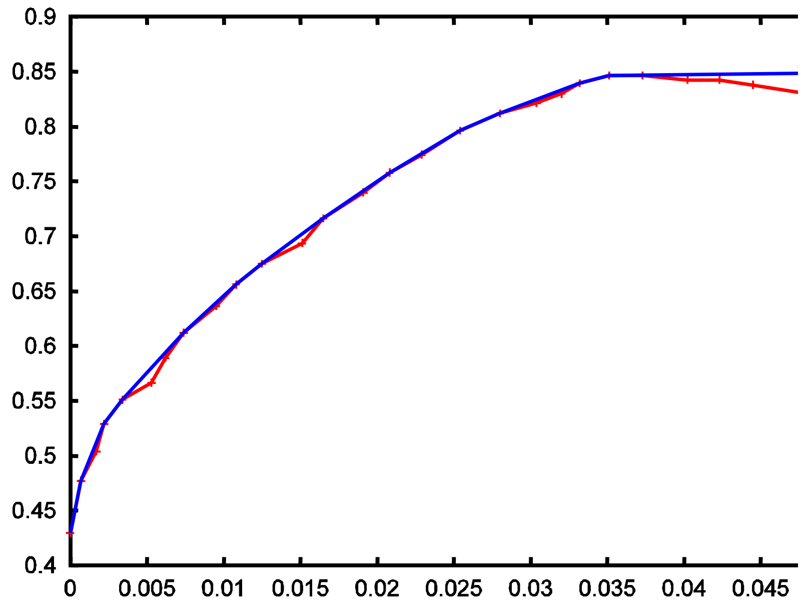
Building the ROC curve

- In many domains, the empirical ROC curve will be non-convex (red line). Take the convex hull of the points (blue line).



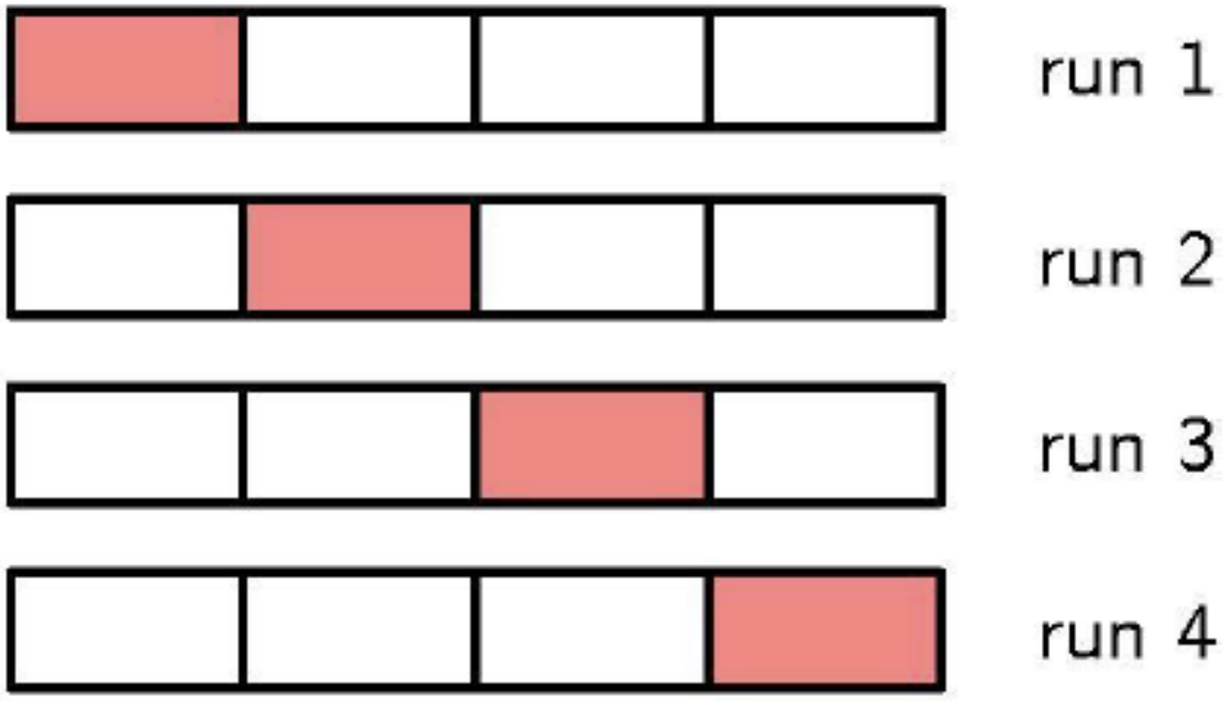
Using the ROC curve

- To compare 2 algorithms over a range of classification thresholds, consider the Area Under the Curve (AUC).
  - A perfect algorithm has AUC=1.
  - A random algorithm has AUC=0.5.
  - Higher AUC doesn't mean all performance measures are better.



K-fold cross-validation

- **Single test-train split**: Estimation test error with high variance.
- **4-fold test-train splits**: Better estimation of the test error, because it is averaged over four different test-train splits.



K-fold cross-validation

- K=1: High variance estimate of Err().

  Fast to compute.

- K>1: Improved estimate of Err(); wastes 1/K of the data.

  K times more expensive to compute.

K-fold cross-validation

- K=1: High variance estimate of Err().

  Fast to compute.

- K>1: Improved estimate of Err(); wastes 1/K of the data.

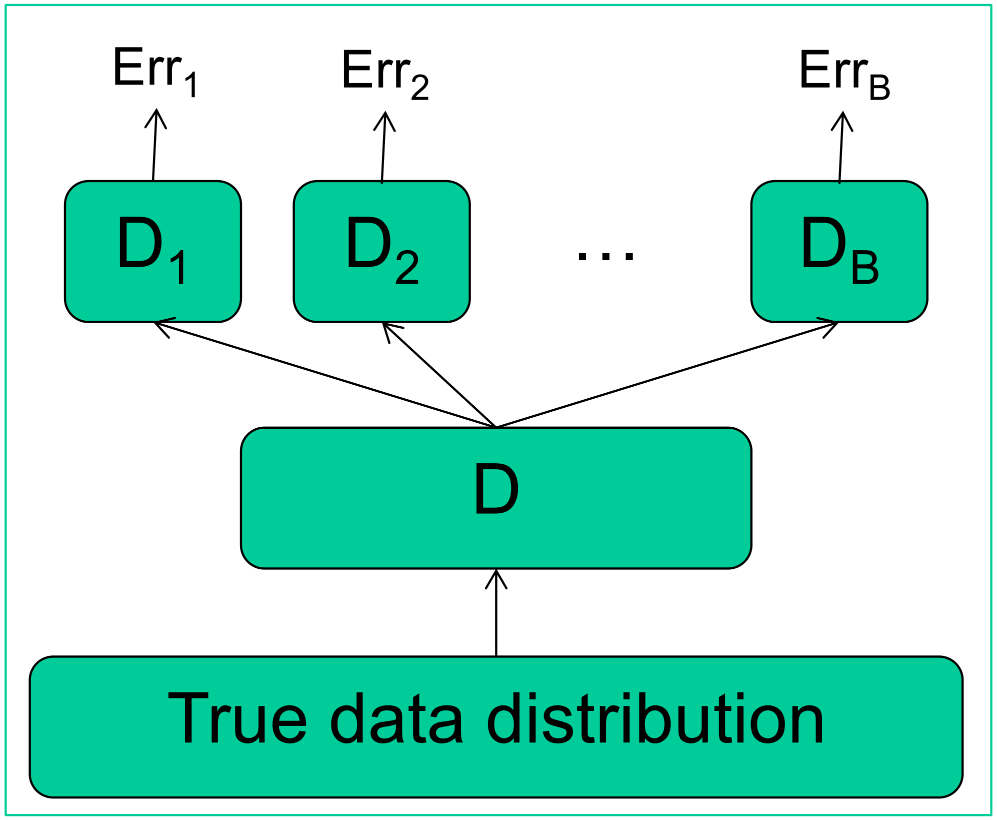
  K times more expensive to compute.

- K=N: Lowest variance estimate of Err(). Doesn't waste data.

  N times slower to compute than single train/validate split.

Brief aside: Bootstrapping

- **Basic idea**: Given a dataset *D* with *N* examples.
  - Randomly draw (with replacement) *B* datasets of size *N* from *D*.
  - Estimate the measure of interest on each of the *B* datasets.
  - Take the mean of the estimates.



Is this a good measure for estimating the error?

Bootstrapping the error

- Use a dataset *b* to fit a hypothesis *f^b*. Use the original dataset *D* to evaluate the error. Average over all bootstrap sets *b* in *B*.

$$\widehat{\mathrm{Err}}_{\mathrm{boot}} = \frac{1}{B}\frac{1}{N}\sum_{b=1}^{B}\sum_{i=1}^{N} L(y_i, \hat{f}^{*b}(x_i)).$$

- **Problem**: Some of the same samples are used for training the learning and validation.

Bootstrapping the error

- Use a dataset *b* to fit a hypothesis *f^b*. Use the original dataset *D* to evaluate the error. Average over all bootstrap sets *b* in *B*.

$$\widehat{\mathrm{Err}}_{\mathrm{boot}} = \frac{1}{B}\frac{1}{N}\sum_{b=1}^{B}\sum_{i=1}^{N} L(y_i, \hat{f}^{*b}(x_i)).$$

- **Problem**: Some of the same samples are used for training the learning and validation.

- **Better idea**: Include the error of a data sample *i* only over classifiers trained with those bootstrap sets *b* in which *i* isn't included (denoted C^{-i}).

$$\widehat{\mathrm{Err}}^{(1)} = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{|C^{-i}|}\sum_{b\in C^{-i}} L(y_i, \hat{f}^{*b}(x_i)).$$

(Note: Bootstrapping is a very general ideal, which can be applied for empirically estimating many different quantities.)

Strategy #1

Consider a classification problem with a large number of features, greater than the number of examples (m>>n). Consider the following strategies to avoid over-fitting in such a problem.

Strategy 1:

1. Check for correlation between each feature (individually) and the output. Keep a small set of features showing strong correlation.
2. Divide the examples into k groups at random.
3. Using the features from step 1 and the examples from k-1 groups from step 2, build a classifier.
4. Use this classifier to predict the output for the examples in group k and measure the error.
5. Repeat steps 3-4 for each group to produce the cross-validation estimate of the error.

Strategy #2

Consider a classification problem with a large number of features, greater than the number of examples (m>>n). Consider the following strategies to avoid over-fitting in such a problem.

Strategy 2:

1. Divide the examples into k groups at random.
2. For each group, find a small set of features showing strong correlation with the output.
3. Using the features and examples from k-1 groups from step 1, build a classifier.
4. Use this classifier to predict the output for the examples in group k and measure the error.
5. Repeat 2-4 for each group to produce the cross-validation estimate of the error.

Strategy #3

Consider a classification problem with a large number of features, greater than the number of examples (m>>n). Consider the following strategies to avoid over-fitting in such a problem.

Strategy 3:

1. Randomly sample n' examples.
2. For the sampled data, find a small set of features showing strong correlation with the outptut
3. Using the examples from step 1 and features from step 2, build a classifier.
4. Use this classifier to predict the output for those examples in the dataset that are not in n' and measure the error.
5. Repeat steps 1-4 *k* times to produce the cross-validation estimate of the error.

Summary of 3 strategies

Strategy 1:

1. Check for correlation between each feature (individually) and the output. Keep a small set of features showing strong correlation.
2. Divide the examples into k groups at random.
3. Using the features from step 1 and the examples from k-1 groups from step 2, build a classifier.
4. Use this classifier to predict the output for the examples in group k and measure the error.
5. Repeat steps 3-4 for each group to produce the cross-validation estimate of the error.

Strategy 2:

1. Divide the examples into k groups at random.
2. For each group, find a small set of features showing strong correlation with the output.
3. Using the features and examples from k-1 groups from step 1, build a classifier.
4. Use this classifier to predict the output for the examples in group k and measure the error.
5. Repeat 2-4 for each group to produce the cross-validation estimate of the error.

Strategy 3:

1. Randomly sample n' examples.
2. For the sampled data, find a small set of features showing strong correlation with the ouptut
3. Using the examples from step 1 and features from step 2, build a classifier.
4. Use this classifier to predict the output for those examples in the dataset that are not in n' and measure the error.
5. Repeat steps 1-4 *k* times to produce the cross-validation estimate of the error.

Discussion

- **Strategy 1** is prone to overfitting, because the full dataset is considered in step 1, to select the features. Thus we do not get an unbiased estimate of the generalization error in step 5.
- **Strategy 2** is closest to standard k-fold cross-validation. One can view the joint procedure of selecting the features and building the classifier as the training step, to be applied (separately) on each training fold.
- **Strategy 3** is closer to a **bootstrap** estimate. It can give a good estimate of the generalization error, but the estimate will possibly have higher variance than the one obtained using Strategy 2.

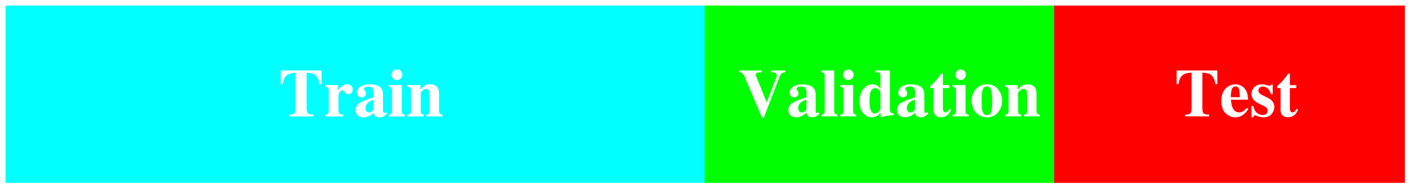
A word of caution

- Intensive use of cross-validation can overfit!
- E.g. Given a dataset with 50 examples and 1000 features.
  - Consider 1000 linear regression models, each built with a single feature.
  - The best of those 1000 will look very good!
  - But it would have looked good even if the output was random!

What should we do about this?

To avoid overfitting to the validation set

- When you need to optimize **many** parameters of your model or learning algorithm.
- Use three datasets:
  - The training set is used to estimate the parameters of the model.
  - The validation set is used to estimate the prediction error for the given model.
  - The test set is used to estimate the generalization error once the model is fixed.



Lessons for evaluating ML algorithms

- Always compare to a simple baseline:
  - In classification:
    - Classify all samples as the majority class.
    - Classify with a threshold on a single variable.
  - In regression:
    - Predict the average of the output for all samples.
    - Compare to a simple linear regression.
- Use K-fold cross validation to properly estimate the error. If necessary, use a validation set to estimate hyper-parameters.
- Consider appropriate measures for fully characterizing the performance: Accuracy, Precision, Recall, F1, AUC.

ARTIFICIAL NEURAL NETWORKS

Artificial neural networks

- Artificial neural network (ANN)
  - Inspired by biological neural systems, i.e., human brains
  - ANN is a network composed of a number of artificial neurons
- Neuron
  - Has an input/output (I/O) characteristic
  - Implements a local computation
- The output of a unit is determined by
  - Its I/O characteristic
  - Its interconnections to other units
  - Possibly external inputs

Artificial neural networks

- ANN can be seen as a parallel distributed information processing structure
- ANN has the ability to learn, recall, and generalize from training data by assigning and adjusting the interconnection weights
- The overall function is determined by
  - The network topology
  - The individual neuron characteristic
  - The learning/training strategy
  - The training data

Applications of ANNs

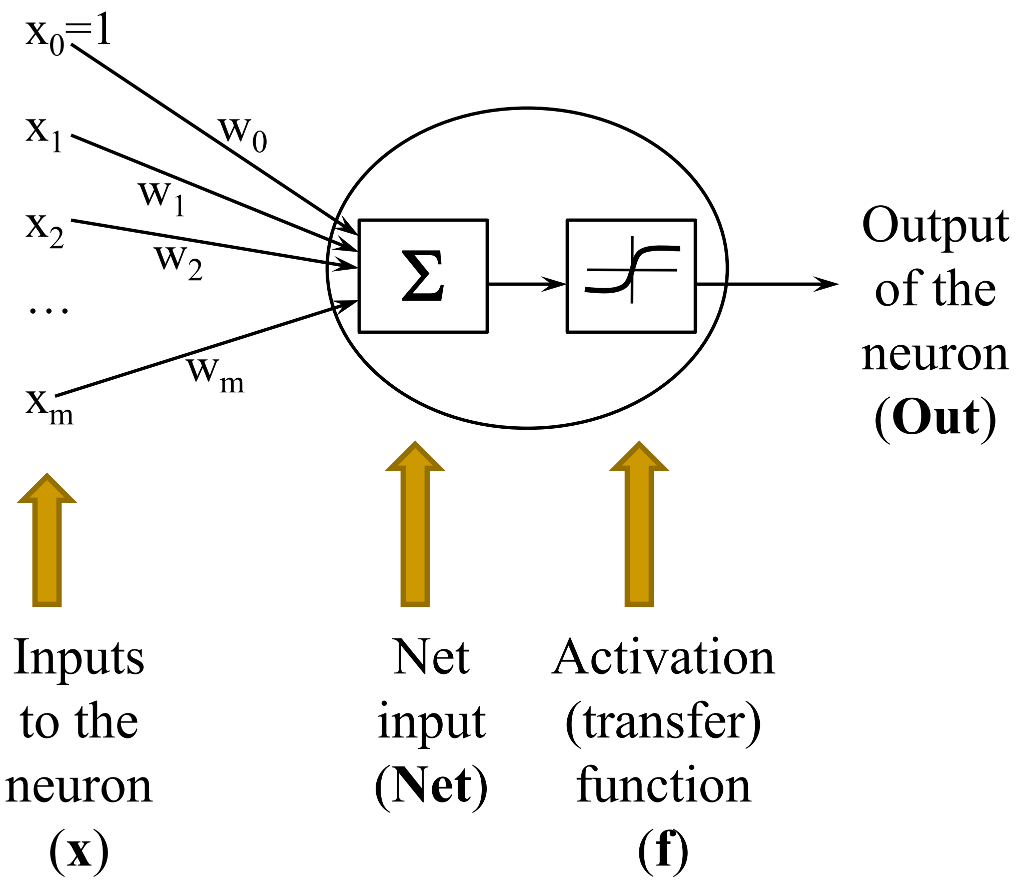
- Image processing and computer vision
  - E.g., image matching, preprocessing, segmentation and analysis, computer vision, image compression, stereo vision, and processing and understanding of time-varying images
- Signal processing
  - E.g., seismic signal analysis and morphology
- Pattern recognition
  - E.g., feature extraction, radar signal classification and analysis, speech recognition and understanding, fingerprint identification, character recognition, face recognition, and handwriting analysis
- Medicine
  - E.g., electrocardiographic signal analysis and understanding, diagnosis of various diseases, and medical image processing

Applications of ANNs

- Military systems
  - E.g., undersea mine detection, radar clutter classification, and tactical speaker recognition
- Financial systems
  - E.g., stock market analysis, real estate appraisal, credit card authorization, and securities trading
- Planning, control, and search
  - E.g., parallel implementation of constraint satisfaction problems, solutions to Traveling Salesman, and control and robotics
- Power systems
  - E.g., system state estimation, transient detection and classification, fault detection and recovery, load forecasting, and security assessment
- ...

Structure and operation of a neuron

- The **input signals** to the neuron (x_i, i = 1..m)
  - Each input x_i is associated to a weight w_i
- The **bias** w_0 (with the input x_0 = 1)
- **Net input** is an integration function of the inputs – `Net(`**`w,x`**`)`
- **Activation (transfer) function** computes the output of the neuron – `f(Net(`**`w,x`**`))`
- **Output** of the neuron: `Out=f(Net(`**`w,x`**`))`



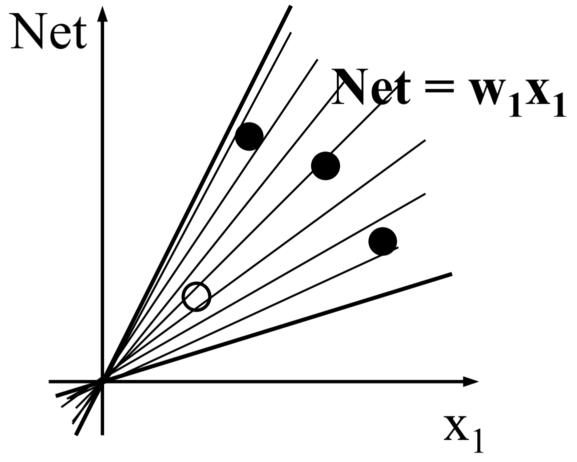
x_0=1
x_1
x_2
…
x_m
w_0
w_1
w_2
w_m
Σ
Output of the neuron (**Out**)
Inputs to the neuron (**x**)
Net input (**Net**)
Activation (transfer) function (**f**)

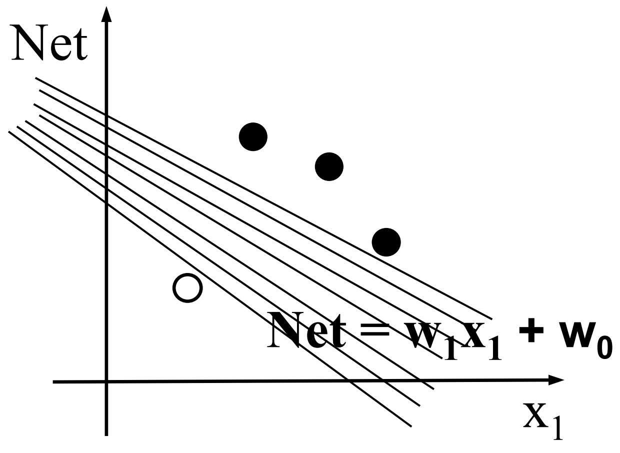
Net input and the bias

- The net input is typically computed using a linear function

$$Net = w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_m x_m = w_0.1 + \sum_{i=1}^{m} w_i x_i = \sum_{i=0}^{m} w_i x_i$$

- The importance of the bias (w_0)
  - → The family of separation functions $\mathtt{Net=w_1x_1}$ cannot separate the instances into two classes
  - → The family of functions $\mathtt{Net=w_1x_1+w_0}$ can
  - → Model has a low bias when it predicts the training data well





Artificial Intelligence

Lecturer 8 – Machine Learning

Brigitte Jaumard
Dept of Computer Science and Software Engineering
Concordia University
Montreal (Quebec) Canada

Introduction of Machine learning

- Definitions of Machine learning…
  - → A process by which a system improves its performance [Simon, 1983]
  - → Any computer program that improves its performance at some task through experience [Mitchell, 1997]
  - → Programming computers to optimize a performance criterion using example data or past experience [Alpaydin, 2004]
- Representation of the learning problem [Mitchell, 1997]

  Learning = Improving with experience at some task

  - Improve over task **T**
  - With respect to performance measure **P**
  - Based on experience **E**

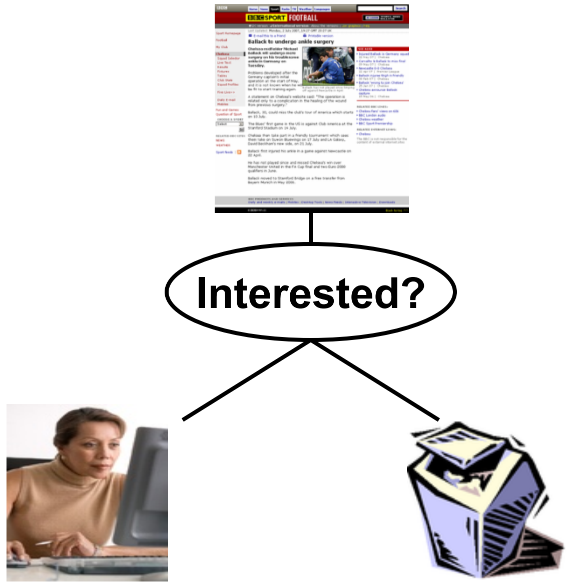
Application examples of ML (1)

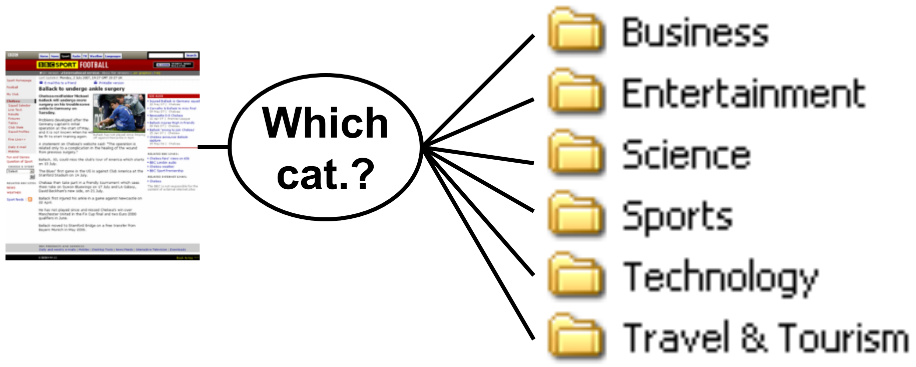
Web pages filtering problem

- **T**: to predict which Web pages a given user is interested in
- **P**: % of Web pages correctly predicted
- **E**: a set of Web pages identified as interested/uninterested for the user

Web pages categorization problem

- **T**: to categorize Web pages in predefined categories
- **P**: % of Web pages correctly categorized
- **E**: a set of Web pages with specified categories

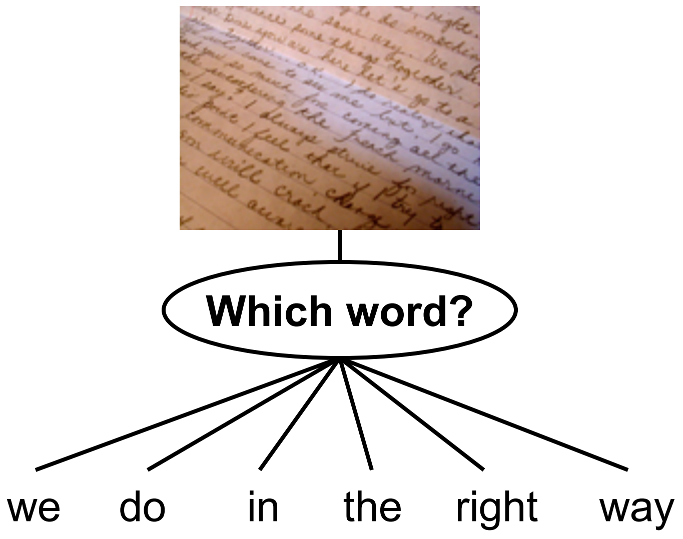




Application examples of ML (2)

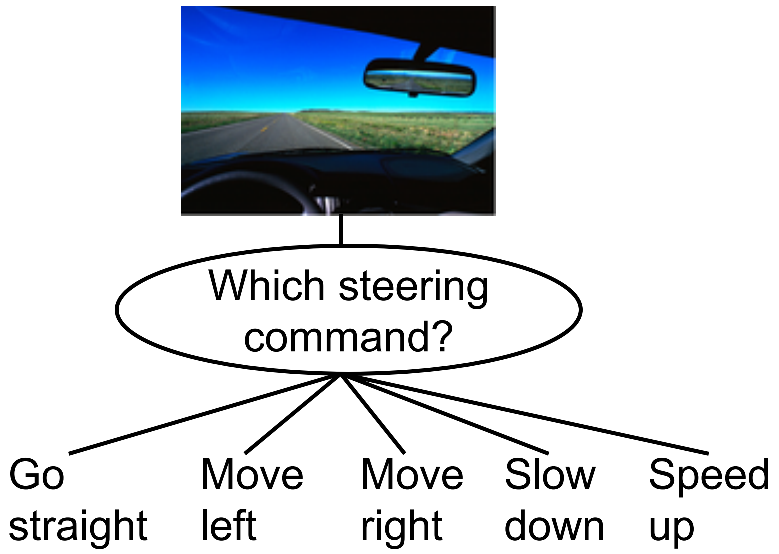
Handwriting recognition problem

- **T**: to recognize and classify handwritten words within images
- **P**: % of words correctly classified
- **E**: a database of handwritten words with given classifications (i.e., labels)



Robot driving problem

- **T**: to drive on public highways using vision sensors
- **P**: average distance traveled before an error (as judged by human overseer)
- **E**: a sequence of images and steering commands recorded while observing a human driver



Key elements of a ML problem (1)

- Selection of the training examples
  - Direct or indirect training feedback
  - With teacher (i.e., with labels) or without
  - The training examples set should be representative of the future test examples
- Choosing the target function (a.k.a. hypothesis, concept, etc.)
  - F: $X \rightarrow \{0,1\}$
  - F: $X \rightarrow$ a set of labels
  - F: $X \rightarrow R^+$ (i.e., the positive real numbers domain)
  - …

Key elements of a ML problem (2)

- Choosing a representation of the target function
  - A polynomial function
  - A set of rules
  - A decision tree
  - A neural network
  - …
- Choosing a learning algorithm that learns (approximately) the target function
  - Regression-based
  - Rule induction
  - ID3 or C4.5
  - Back-propagation
  - …

Issues in Machine Learning (1)

- Learning algorithm
  - What algorithms can approximate the target function?
  - Under which conditions does a selected algorithm converge (approximately) to the target function?
  - For a certain problem domain and given a representation of examples which algorithm performs best?
- Training examples
  - How many training examples are sufficient?
  - How does the size of the training set influence the accuracy of the learned target function?
  - How does noise and/or missing-value data influence the accuracy?

Issues in Machine Learning (2)

- Learning process
  - What is the best strategy for selecting a next training example? How do selection strategies alter the complexity of the learning problem?
  - How can prior knowledge (held by the system) help?
- Learning capability
  - What target function should the system learn?

    Representation of the target function: expressiveness vs. complexity
  - What are the theoretical limits of learnability?
  - How can the system generalize from the training examples?

    To avoid the overfitting problem
  - How can the system automatically alter its representation?

    To improve its ability to represent and learn the target function

Types of learning problems

- A rough (and somewhat outdated) classification of learning problems:
  - **Supervised learning**, where we get a set of training inputs and outputs
    - classification, regression
  - **Unsupervised learning**, where we are interested in capturing inherent organization in the data
    - clustering, density estimation
  - **Reinforcement learning**, where we only get feedback in the form of how well we are doing (not what we should be doing)
    - Planning

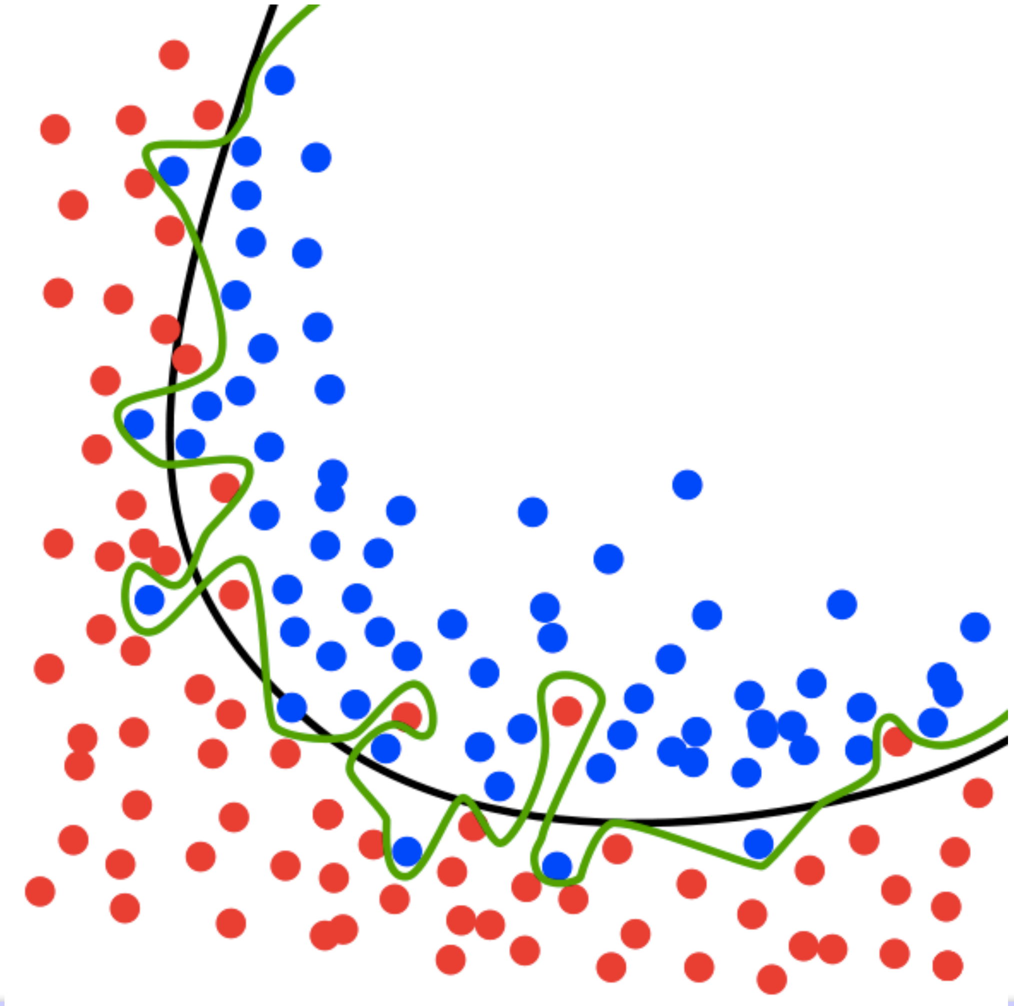
EVALUATION

Evaluating performance

- Different objectives:
  - Selecting the right model for a problem.
  - Testing performance of a new algorithm.
  - Evaluating impact on a new application.

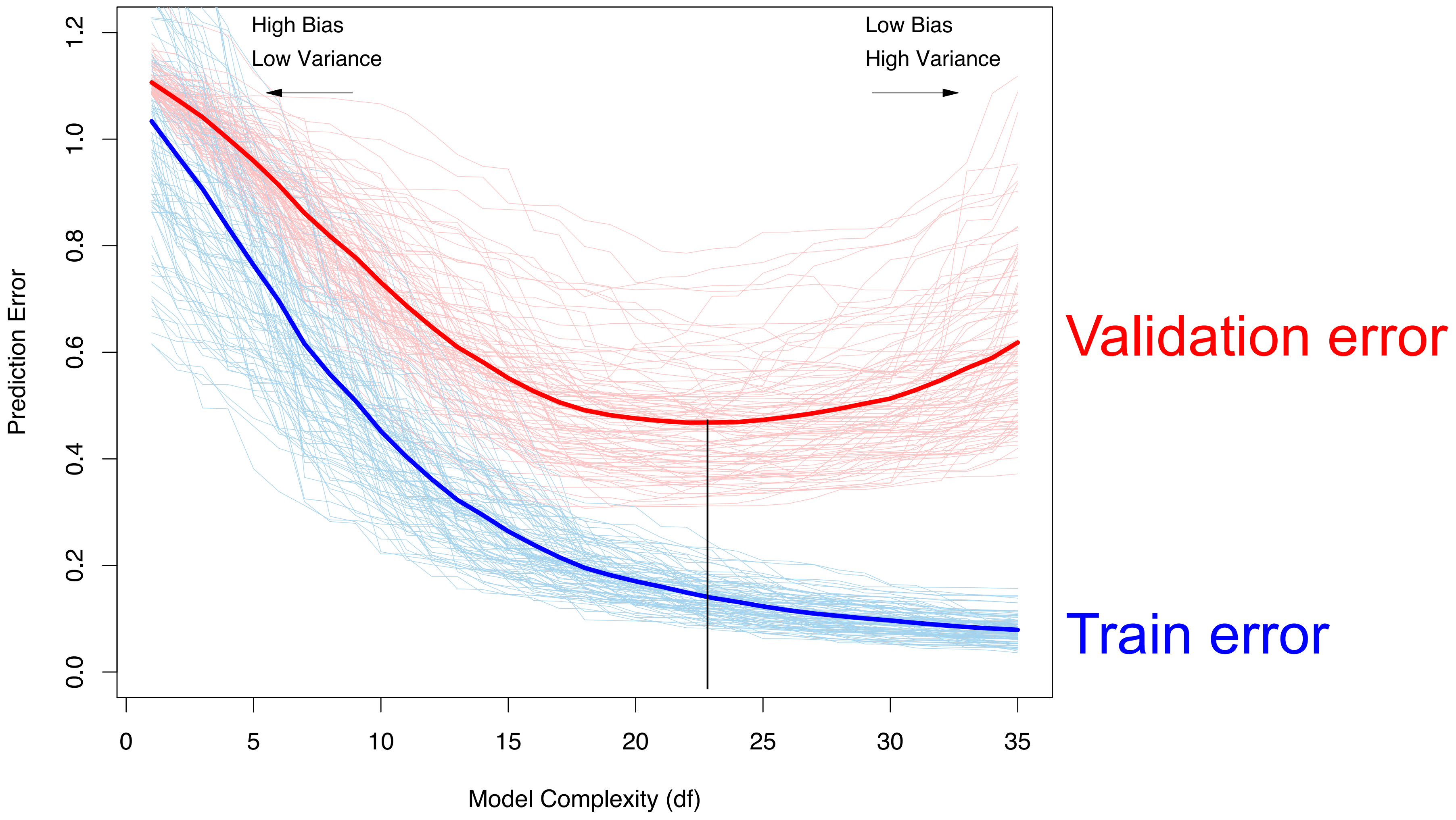
Overfitting

- Adding more degrees of freedom (more features) always seems to improve the solution!



Minimizing the error

- Find the low point in the validation error:



Performance metrics for classification

- Not all errors have equal impact!

- There are different types of mistakes, particularly in the classification setting.

Performance metrics for classification

- Not all errors have equal impact!
- There are different types of mistakes, particularly in the classification setting.
  - E.g. Consider the diagnostic of a disease. Two types of mis-diagnostics:
    - Patient does not have disease but received positive diagnostic (Type I error);
    - Patient has disease but it was not detected (Type II error).

Performance metrics for classification

- Not all errors have equal impact!
- There are different types of mistakes, particularly in the classification setting.
  - E.g. Consider the diagnostic of a disease. Two types of mis-diagnostics:
    - Patient does not have disease but received positive diagnostic (Type I error);
    - Patient has disease but it was not detected (Type II error).
  - E.g. Consider the problem of spam classification:
    - A message that is not spam is assigned to the spam folder (Type I error);
    - A message that is spam appears in the regular folder (Type II error).

Performance metrics for classification

- Not all errors have equal impact!
- There are different types of mistakes, particularly in the classification setting.
  - E.g. Consider the diagnostic of a disease. Two types of mis-diagnostics:
    - Patient does not have disease but received positive diagnostic (Type I error);
    - Patient has disease but it was not detected (Type II error).
  - E.g. Consider the problem of spam classification:
    - A message that is not spam is assigned to the spam folder (Type I error);
    - A message that is spam appears in the regular folder (Type II error).
- How many Type I errors are you willing to tolerate, for a reasonable rate of Type II errors ?

Terminology

- Type of classification outputs:
  - True positive (m11): Example of class 1 predicted as class 1.
  - False positive (m01): Example of class 0 predicted as class 1. Type 1 error.
  - True negative (m00): Example of class 0 predicted as class 0.
  - False negative (m10): Example of class 1 predicted as class 0. Type II error.
- Total number of instances: m = m00 + m01 + m10 + m11

Terminology

- Type of classification outputs:
  - True positive (m11): Example of class 1 predicted as class 1.
  - False positive (m01): Example of class 0 predicted as class 1. Type 1 error.
  - True negative (m00): Example of class 0 predicted as class 0.
  - False negative (m10): Example of class 1 predicted as class 0. Type II error.
- Total number of instances: m = m00 + m01 + m10 + m11

- Error rate: (m01 + m10) / m
  - If the classes are imbalanced (e.g. 10% from class 1, 90% from class 0), one can achieve low error (e.g. 10%) by classifying everything as coming from class 0!

Confusion matrix

- Many software packages output this matrix.

$$\begin{bmatrix} m_{00} & m_{01} \\ m_{10} & m_{11} \end{bmatrix}$$

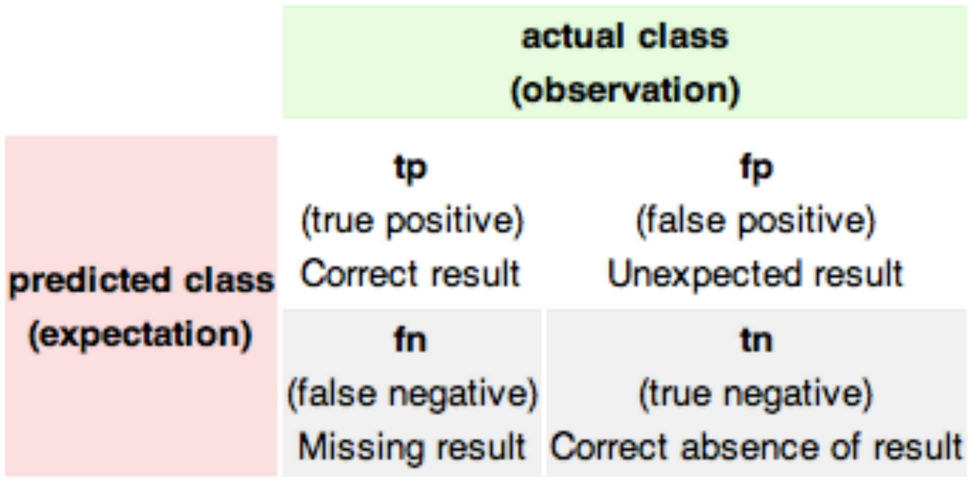
Confusion matrix

- Many software packages output this matrix.

$$\begin{bmatrix} m_{00} & m_{01} \\ m_{10} & m_{11} \end{bmatrix}$$

- Be careful! Sometimes the format is slightly different

(E.g. *http://en.wikipedia.org/wiki/Precision_and_recall#Definition_.28classification_context.29*)



	actual class (observation)	
predicted class (expectation)	**tp** (true positive) Correct result	**fp** (false positive) Unexpected result
	fn (false negative) Missing result	**tn** (true negative) Correct absence of result

Common measures

Accuracy	= (TP+ TN) / (TP + FP + FN + TN)
Precision	= True positives / Total number of declared positives
	= TP / (TP+ FP)
Recall	= True positives / Total number of actual positives
	= TP / (TP + FN)

Common measures

- Accuracy = (TP+ TN) / (TP + FP + FN + TN)
- Precision = True positives / Total number of declared positives

  = TP / (TP+ FP)

Text classification

- Recall = True positives / Total number of actual positives

  = TP / (TP + FN)

Medicine

- Sensitivity is the same as recall.
- Specificity = True negatives / Total number of actual negatives

  = TN / (FP + TN)

Common measures

- Accuracy = (TP+ TN) / (TP + FP + FN + TN)
- Precision = True positives / Total number of declared positives
  = TP / (TP+ FP)
- Recall = True positives / Total number of actual positives
  = TP / (TP + FN)

Text classification

- Sensitivity is the same as recall.
- Specificity = True negatives / Total number of actual negatives
  = TN / (FP + TN)

Medicine

- False positive rate = FP / (FP + TN)

Common measures

- Accuracy = (TP+ TN) / (TP + FP + FN + TN)
- Precision = True positives / Total number of declared positives
  = TP / (TP+ FP)
- Recall = True positives / Total number of actual positives
  = TP / (TP + FN)

Text classification

- Sensitivity is the same as recall.
- Specificity = True negatives / Total number of actual negatives
  = TN / (FP + TN)

Medicine

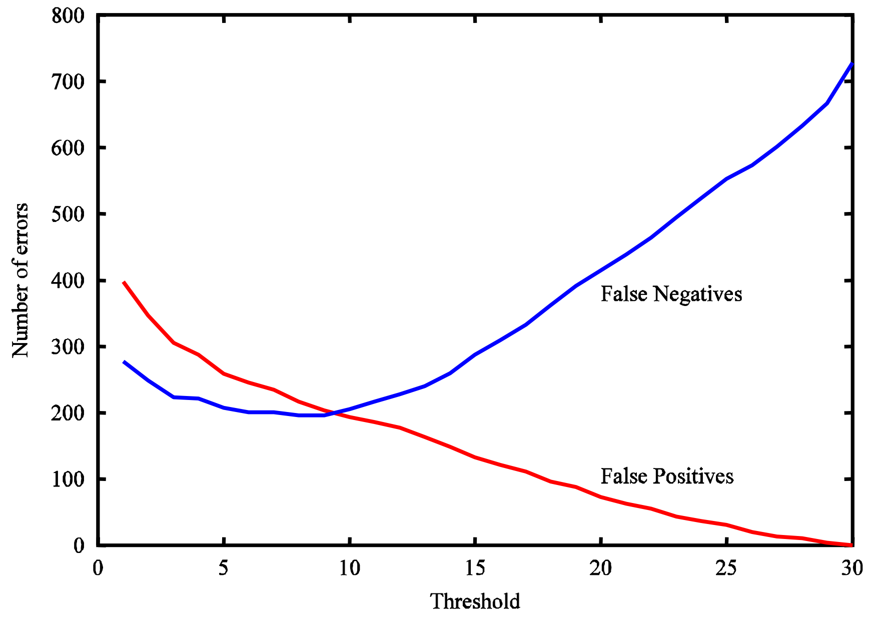
- False positive rate = FP / (FP + TN)
- F1 measure

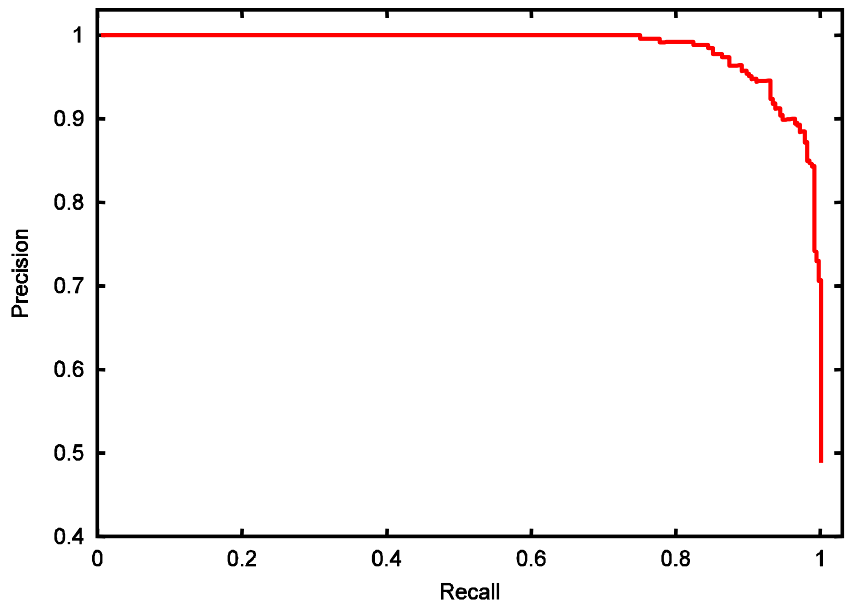
$$F = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

Trade-off

- Often have a trade-off between false positives and false negatives.

E.g. Consider 30 different classifiers trained on a class. Classify a new sample as positive if K classifiers output positive. Vary K between 0 and 30.

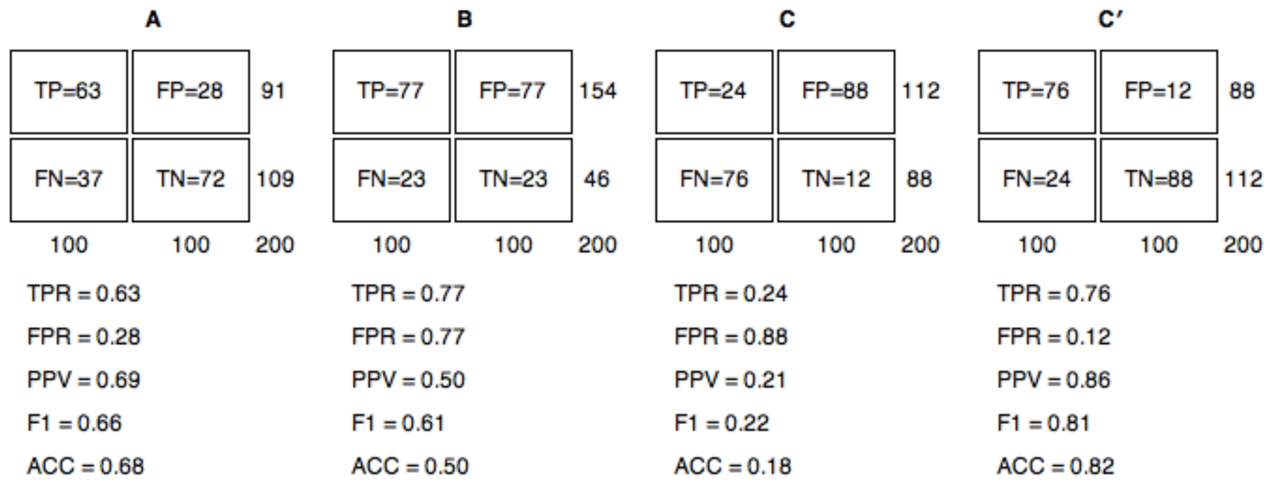




Receiver-operator characteristic (ROC) curve

- Characterizes the performance of a binary classifier over a range of classification thresholds

Data from 4 prediction results:



A

TP=63	FP=28	91
FN=37	TN=72	109
100	100	200

TPR = 0.63
FPR = 0.28
PPV = 0.69
F1 = 0.66
ACC = 0.68

B

TP=77	FP=77	154
FN=23	TN=23	46
100	100	200

TPR = 0.77
FPR = 0.77
PPV = 0.50
F1 = 0.61
ACC = 0.50

C

TP=24	FP=88	112
FN=76	TN=12	88
100	100	200

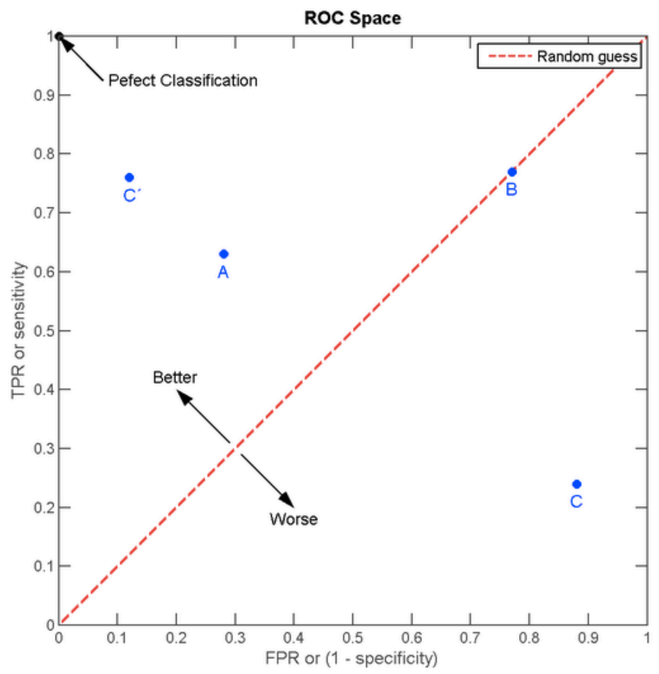
TPR = 0.24
FPR = 0.88
PPV = 0.21
F1 = 0.22
ACC = 0.18

C′

TP=76	FP=12	88
FN=24	TN=88	112
100	100	200

TPR = 0.76
FPR = 0.12
PPV = 0.86
F1 = 0.81
ACC = 0.82

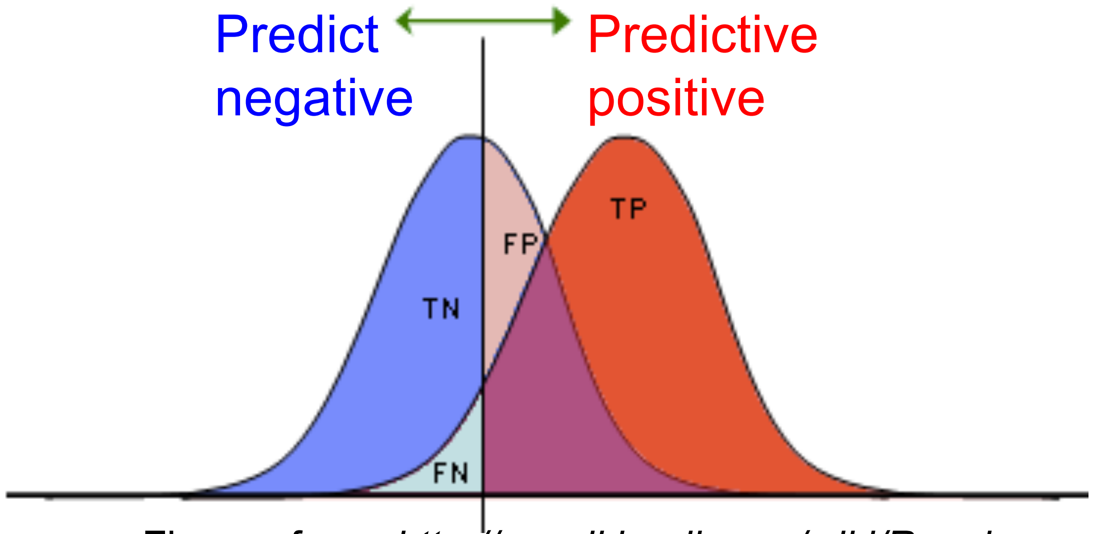
ROC curve:

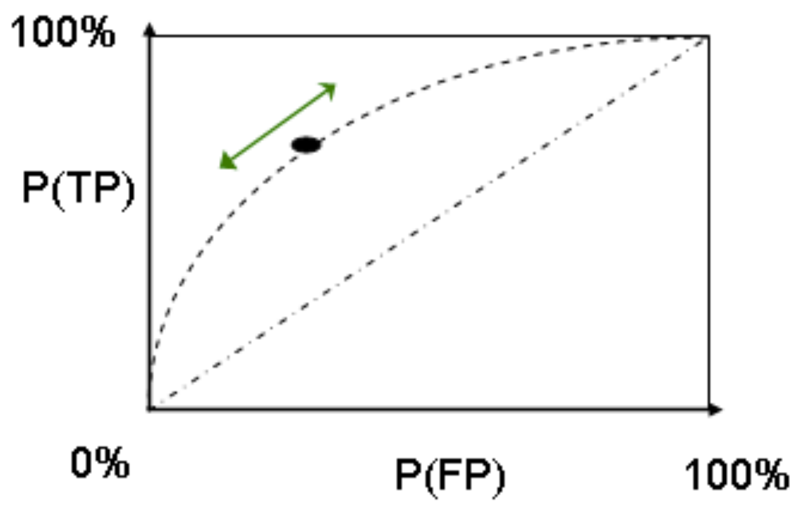


Example from: *http://en.wikipedia.org/wiki/Receiver_operating_characteristic*

Understanding the ROC curve

- Consider a classification problem where data is generated by 2 Gaussians (blue = negative class; red = positive class).
- Consider the decision boundary (shown as a vertical line on the left figure), where you predict Negative on the left of the boundary and predict Positive on the right of the boundary.
- Changing that boundary defines the ROC curve on the right.

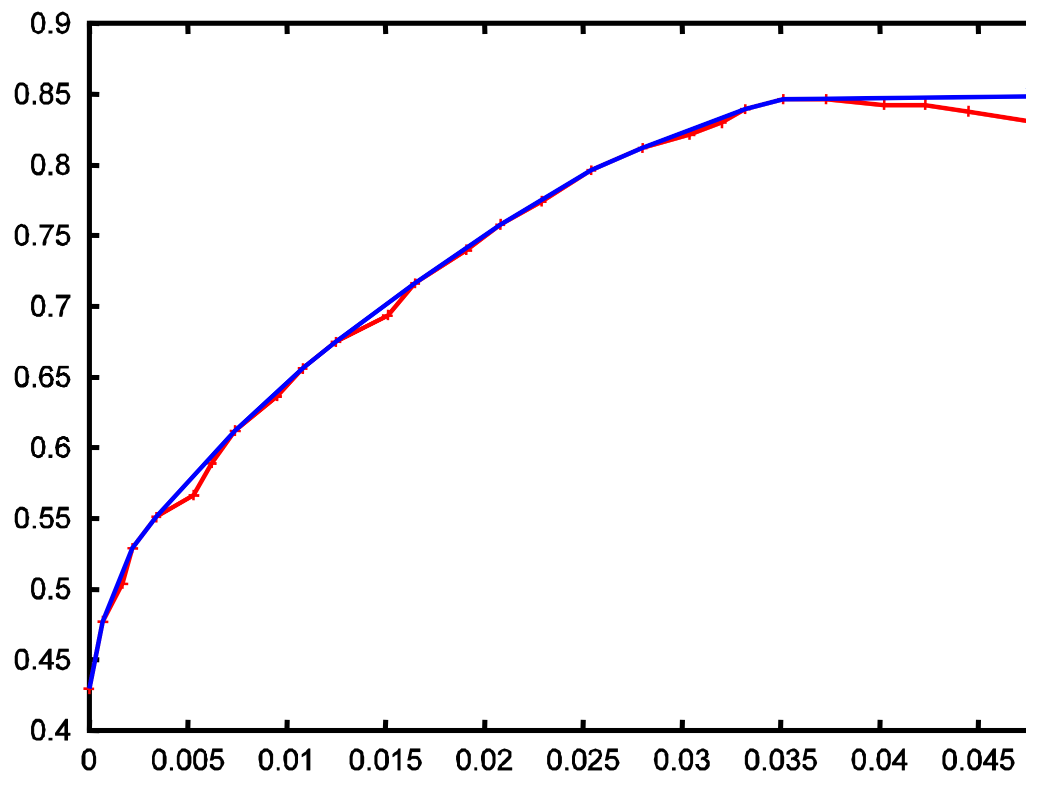




Figures from: *http://en.wikipedia.org/wiki/Receiver_operating_characteristic*

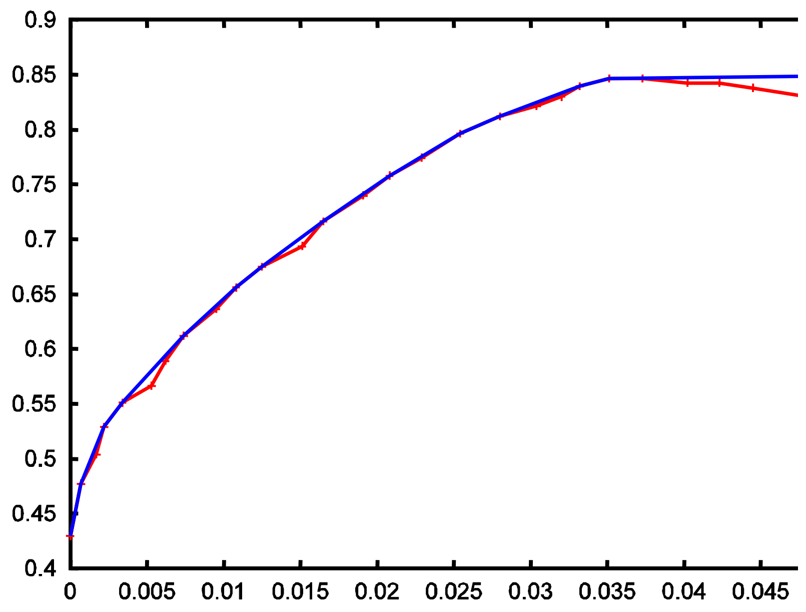
Building the ROC curve

- In many domains, the empirical ROC curve will be non-convex (red line). Take the convex hull of the points (blue line).



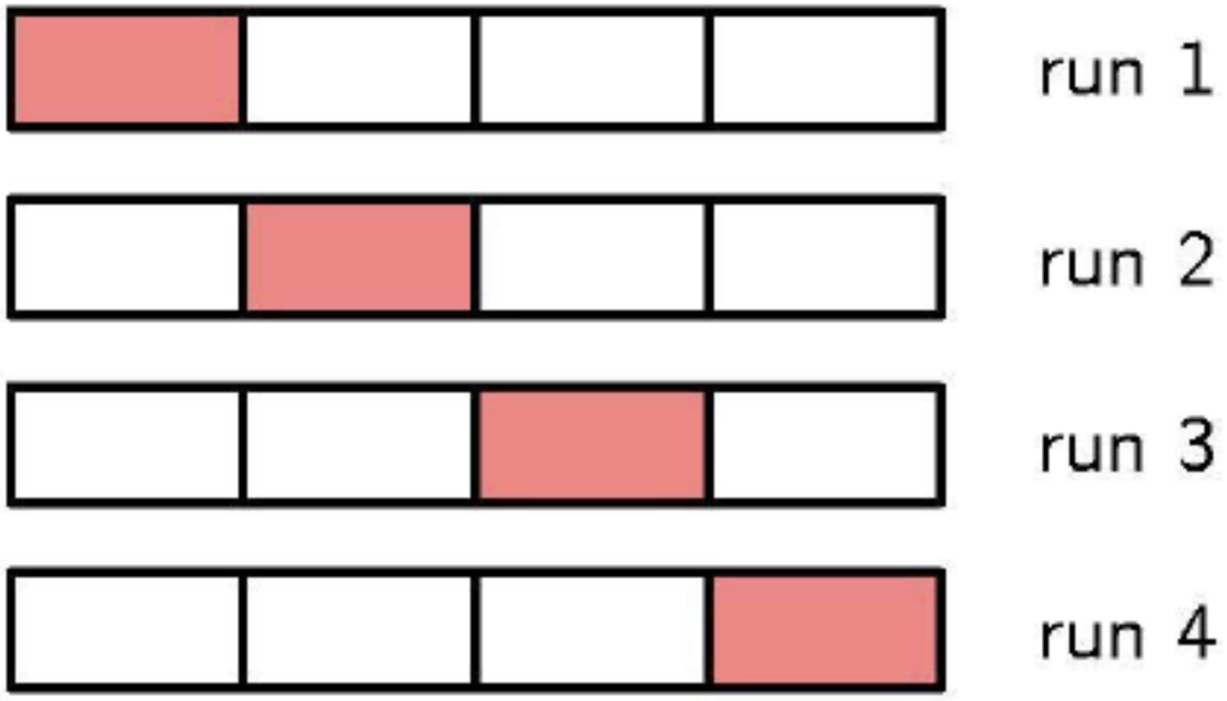
Using the ROC curve

- To compare 2 algorithms over a range of classification thresholds, consider the Area Under the Curve (AUC).
  - A perfect algorithm has AUC=1.
  - A random algorithm has AUC=0.5.
  - Higher AUC doesn't mean all performance measures are better.



K-fold cross-validation

- **Single test-train split**: Estimation test error with high variance.
- **4-fold test-train splits**: Better estimation of the test error, because it is averaged over four different test-train splits.



K-fold cross-validation

- K=1: High variance estimate of Err().

  Fast to compute.

- K>1: Improved estimate of Err(); wastes 1/K of the data.

  K times more expensive to compute.

K-fold cross-validation

- K=1: High variance estimate of Err().

  Fast to compute.

- K>1: Improved estimate of Err(); wastes 1/K of the data.

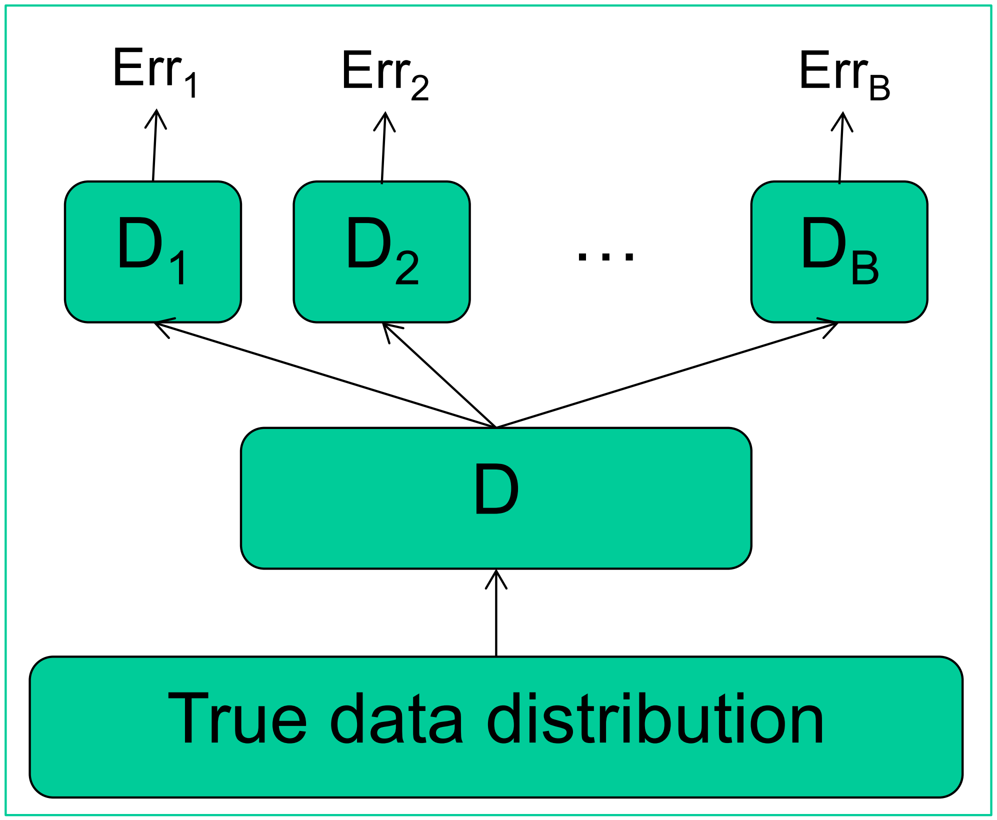
  K times more expensive to compute.

- K=N: Lowest variance estimate of Err(). Doesn’t waste data.

  N times slower to compute than single train/validate split.

Brief aside: Bootstrapping

- **Basic idea**: Given a dataset *D* with *N* examples.
  - Randomly draw (with replacement) *B* datasets of size *N* from *D*.
  - Estimate the measure of interest on each of the *B* datasets.
  - Take the mean of the estimates.



Is this a good measure

for estimating the error?

Bootstrapping the error

- Use a dataset *b* to fit a hypothesis f^b. Use the original dataset *D* to evaluate the error. Average over all bootstrap sets *b* in *B*.

$$\widehat{\mathrm{Err}}_{\mathrm{boot}} = \frac{1}{B}\frac{1}{N}\sum_{b=1}^{B}\sum_{i=1}^{N} L(y_i, \hat{f}^{*b}(x_i)).$$

- **Problem**: Some of the same samples are used for training the learning and validation.

Bootstrapping the error

- Use a dataset *b* to fit a hypothesis f^b. Use the original dataset *D* to evaluate the error. Average over all bootstrap sets *b* in *B*.

$$\widehat{\mathrm{Err}}_{\mathrm{boot}} = \frac{1}{B}\frac{1}{N}\sum_{b=1}^{B}\sum_{i=1}^{N} L(y_i, \hat{f}^{*b}(x_i)).$$

- **Problem**: Some of the same samples are used for training the learning and validation.

- **Better idea**: Include the error of a data sample *i* only over classifiers trained with those bootstrap sets *b* in which *i* isn't included (denoted C^{-i}).

$$\widehat{\mathrm{Err}}^{(1)} = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{|C^{-i}|}\sum_{b\in C^{-i}} L(y_i, \hat{f}^{*b}(x_i)).$$

(Note: Bootstrapping is a very general ideal, which can be applied for empirically estimating many different quantities.)

Strategy #1

Consider a classification problem with a large number of features, greater than the number of examples (m>>n). Consider the following strategies to avoid over-fitting in such a problem.

Strategy 1:

1. Check for correlation between each feature (individually) and the output. Keep a small set of features showing strong correlation.
2. Divide the examples into k groups at random.
3. Using the features from step 1 and the examples from k-1 groups from step 2, build a classifier.
4. Use this classifier to predict the output for the examples in group k and measure the error.
5. Repeat steps 3-4 for each group to produce the cross-validation estimate of the error.

Strategy #2

Consider a classification problem with a large number of features, greater than the number of examples (m>>n). Consider the following strategies to avoid over-fitting in such a problem.

Strategy 2:

1. Divide the examples into k groups at random.
2. For each group, find a small set of features showing strong correlation with the output.
3. Using the features and examples from k-1 groups from step 1, build a classifier.
4. Use this classifier to predict the output for the examples in group k and measure the error.
5. Repeat 2-4 for each group to produce the cross-validation estimate of the error.

Strategy #3

Consider a classification problem with a large number of features, greater than the number of examples (m>>n). Consider the following strategies to avoid over-fitting in such a problem.

Strategy 3:

1. Randomly sample n' examples.
2. For the sampled data, find a small set of features showing strong correlation with the outptut
3. Using the examples from step 1 and features from step 2, build a classifier.
4. Use this classifier to predict the output for those examples in the dataset that are not in n' and measure the error.
5. Repeat steps 1-4 *k* times to produce the cross-validation estimate of the error.

Summary of 3 strategies

Strategy 1:

1. Check for correlation between each feature (individually) and the output. Keep a small set of features showing strong correlation.
2. Divide the examples into k groups at random.
3. Using the features from step 1 and the examples from k-1 groups from step 2, build a classifier.
4. Use this classifier to predict the output for the examples in group k and measure the error.
5. Repeat steps 3-4 for each group to produce the cross-validation estimate of the error.

Strategy 2:

1. Divide the examples into k groups at random.
2. For each group, find a small set of features showing strong correlation with the output.
3. Using the features and examples from k-1 groups from step 1, build a classifier.
4. Use this classifier to predict the output for the examples in group k and measure the error.
5. Repeat 2-4 for each group to produce the cross-validation estimate of the error.

Strategy 3:

1. Randomly sample n' examples.
2. For the sampled data, find a small set of features showing strong correlation with the ouptut
3. Using the examples from step 1 and features from step 2, build a classifier.
4. Use this classifier to predict the output for those examples in the dataset that are not in n' and measure the error.
5. Repeat steps 1-4 *k* times to produce the cross-validation estimate of the error.

Discussion

- **Strategy 1** is prone to overfitting, because the full dataset is considered in step 1, to select the features. Thus we do not get an unbiased estimate of the generalization error in step 5.
- **Strategy 2** is closest to standard k-fold cross-validation. One can view the joint procedure of selecting the features and building the classifier as the training step, to be applied (separately) on each training fold.
- **Strategy 3** is closer to a **bootstrap** estimate. It can give a good estimate of the generalization error, but the estimate will possibly have higher variance than the one obtained using Strategy 2.

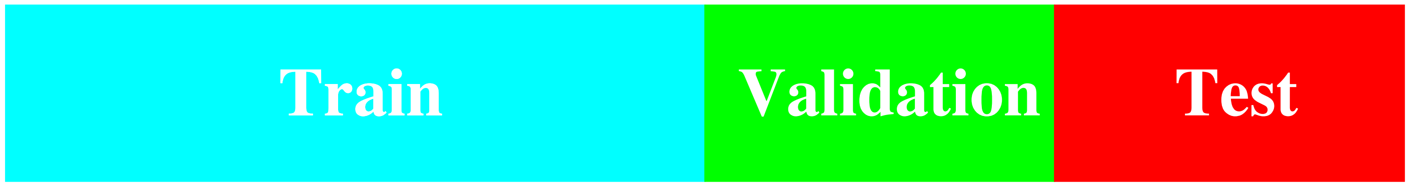
A word of caution

- Intensive use of cross-validation can overfit!
- E.g. Given a dataset with 50 examples and 1000 features.
  - Consider 1000 linear regression models, each built with a single feature.
  - The best of those 1000 will look very good!
  - But it would have looked good even if the output was random!

What should we do about this?

To avoid overfitting to the validation set

- When you need to optimize **many** parameters of your model or learning algorithm.
- Use three datasets:
  - The training set is used to estimate the parameters of the model.
  - The validation set is used to estimate the prediction error for the given model.
  - The test set is used to estimate the generalization error once the model is fixed.



Lessons for evaluating ML algorithms

- Always compare to a simple baseline:
  - In classification:
    - Classify all samples as the majority class.
    - Classify with a threshold on a single variable.
  - In regression:
    - Predict the average of the output for all samples.
    - Compare to a simple linear regression.
- Use K-fold cross validation to properly estimate the error. If necessary, use a validation set to estimate hyper-parameters.
- Consider appropriate measures for fully characterizing the performance: Accuracy, Precision, Recall, F1, AUC.

ARTIFICIAL NEURAL NETWORKS

Artificial neural networks

- Artificial neural network (ANN)
  - Inspired by biological neural systems, i.e., human brains
  - ANN is a network composed of a number of artificial neurons
- Neuron
  - Has an input/output (I/O) characteristic
  - Implements a local computation
- The output of a unit is determined by
  - Its I/O characteristic
  - Its interconnections to other units
  - Possibly external inputs

Artificial neural networks

- ANN can be seen as a parallel distributed information processing structure
- ANN has the ability to learn, recall, and generalize from training data by assigning and adjusting the interconnection weights
- The overall function is determined by
    - The network topology
    - The individual neuron characteristic
    - The learning/training strategy
    - The training data

Applications of ANNs

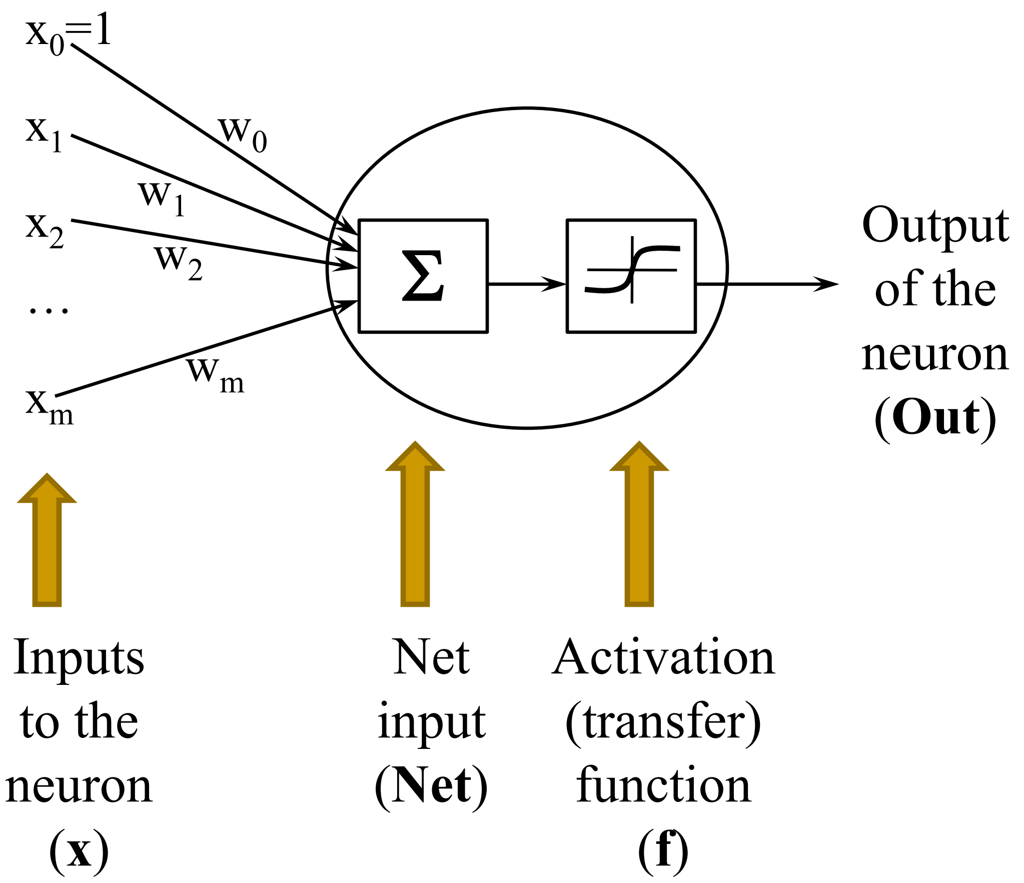
- Image processing and computer vision
  - E.g., image matching, preprocessing, segmentation and analysis, computer vision, image compression, stereo vision, and processing and understanding of time-varying images
- Signal processing
  - E.g., seismic signal analysis and morphology
- Pattern recognition
  - E.g., feature extraction, radar signal classification and analysis, speech recognition and understanding, fingerprint identification, character recognition, face recognition, and handwriting analysis
- Medicine
  - E.g., electrocardiographic signal analysis and understanding, diagnosis of various diseases, and medical image processing

Applications of ANNs

- Military systems
  - E.g., undersea mine detection, radar clutter classification, and tactical speaker recognition
- Financial systems
  - E.g., stock market analysis, real estate appraisal, credit card authorization, and securities trading
- Planning, control, and search
  - E.g., parallel implementation of constraint satisfaction problems, solutions to Traveling Salesman, and control and robotics
- Power systems
  - E.g., system state estimation, transient detection and classification, fault detection and recovery, load forecasting, and security assessment
- ...

Structure and operation of a neuron

- The **input signals** to the neuron (x_i, i = 1..m)
  - Each input x_i is associated to a weight w_i
- The **bias** w_0 (with the input $x_0 = 1$)
- **Net input** is an integration function of the inputs – `Net(w,x)`
- **Activation (transfer) function** computes the output of the neuron – `f(Net(w,x))`
- **Output** of the neuron: `Out=f(Net(w,x))`



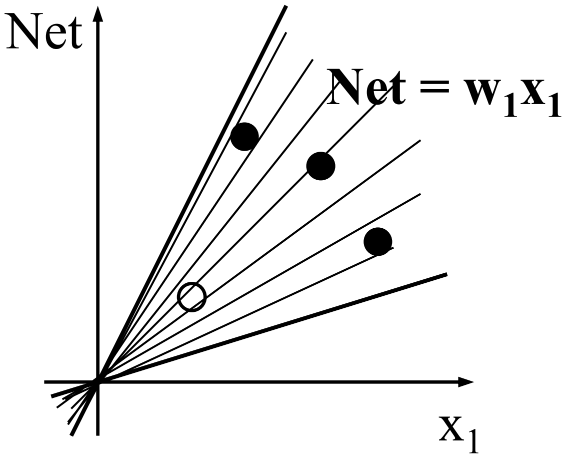
x_0=1
x_1
x_2
…
x_m
w_0
w_1
w_2
w_m
Σ
Output of the neuron (**Out**)
Inputs to the neuron (**x**)
Net input (**Net**)
Activation (transfer) function (**f**)

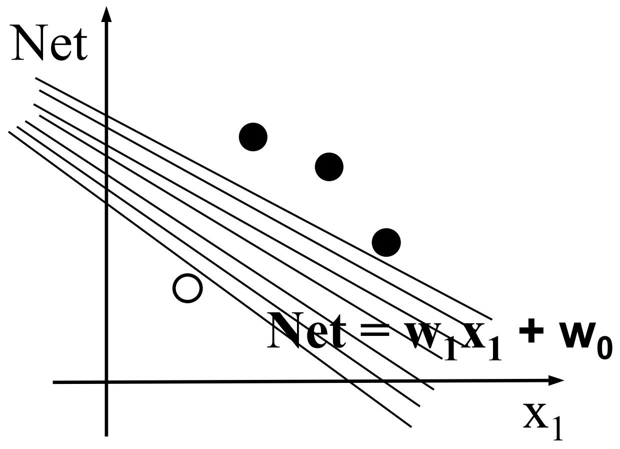
Net input and the bias

- The net input is typically computed using a linear function

$$Net = w_0 + w_1x_1 + w_2x_2 + \ldots + w_mx_m = w_0.1 + \sum_{i=1}^{m} w_ix_i = \sum_{i=0}^{m} w_ix_i$$

- The importance of the bias (w_0)
  - → The family of separation functions `Net=w₁x₁` cannot separate the instances into two classes
  - → The family of functions `Net=w₁x₁+w₀` can
  - → Model has a low bias when it predicts the training data well





Activation Functions

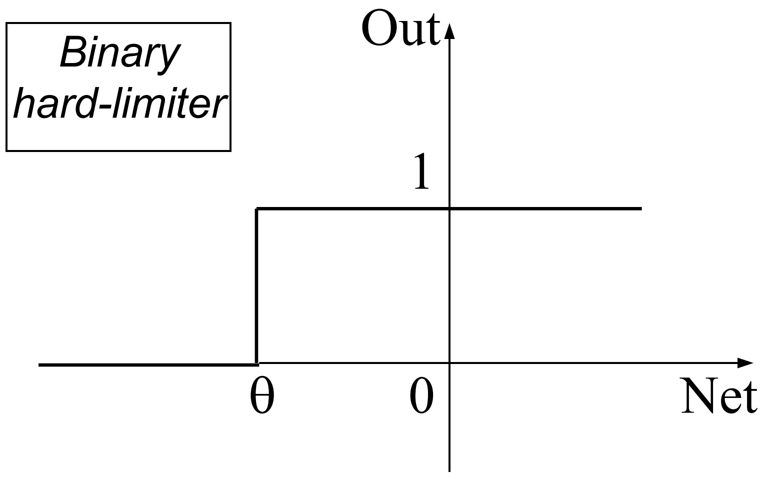
- Activation functions are an extremely important feature of the artificial neural networks.
- Decide whether a neuron should be activated or not, i.e.,
- Whether the information that the neuron is receiving is relevant for the given information or should be ignored
- Y = Activation (Σ(weights x input) + bias)
- Nonlinear transformation that we do over the input signal.
- Transform output is seen to next layer as input

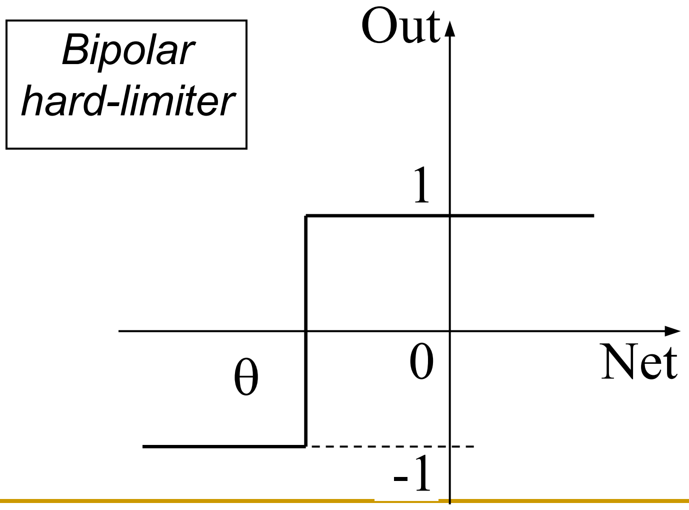
Activation function – Hard-limiter

- Also called the threshold function
- The output of the hard-limiter is either of the two values
- θ is the threshold value
- **Disadvantage**: neither continuous nor continuously differentiable

$$Out(Net) = hl1(Net, \theta) = \begin{cases} 1, \text{if} \quad Net \geq \theta \\ 0, \text{if otherwise} \end{cases}$$

$$Out(Net) = hl2(Net, \theta) = sign(Net, \theta)$$

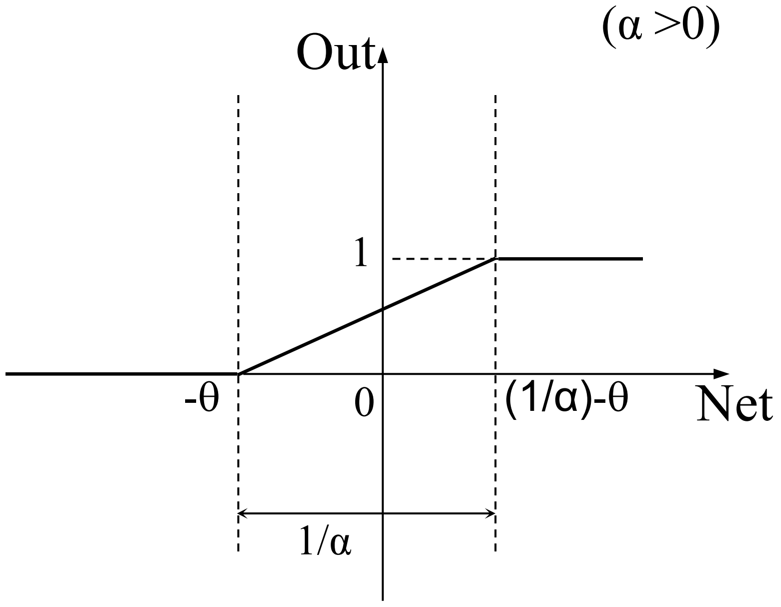




Activation function – Threshold logic

$$Out(Net) = tl(Net, \alpha, \theta) = \begin{cases} 0, \text{ if} & Net < -\theta \\ \alpha(Net+\theta), \text{if} & -\theta \le Net \le \frac{1}{\alpha} - \theta \\ 1, \text{ if} & Net > \frac{1}{\alpha} - \theta \end{cases}$$

$$= \max(0, \min(1, \alpha(Net+\theta)))$$

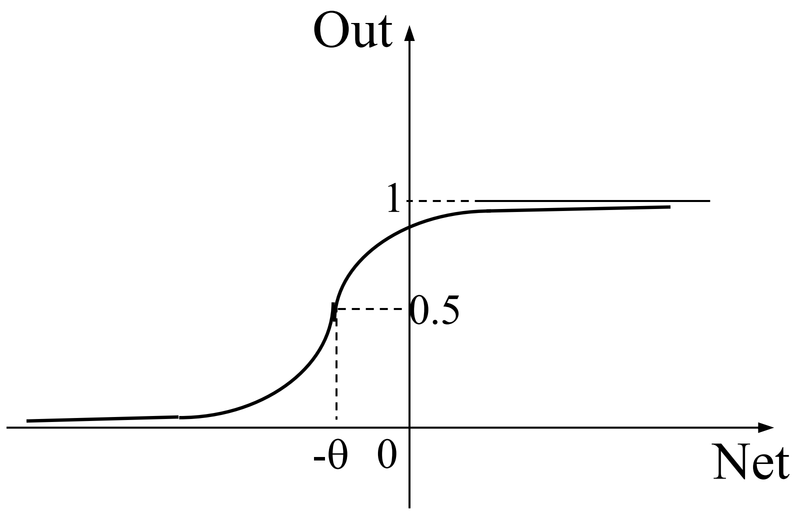


- It is called also saturating linear function
- A combination of linear and hard-limiter activation functions
- α decides the slope in the linear range
- **Disadvantage**: continuous – but not continuously differentiable

Activation function – Sigmoidal

$$Out(Net) = sf(Net, \alpha, \theta) = \frac{1}{1 + e^{-\alpha(Net+\theta)}}$$

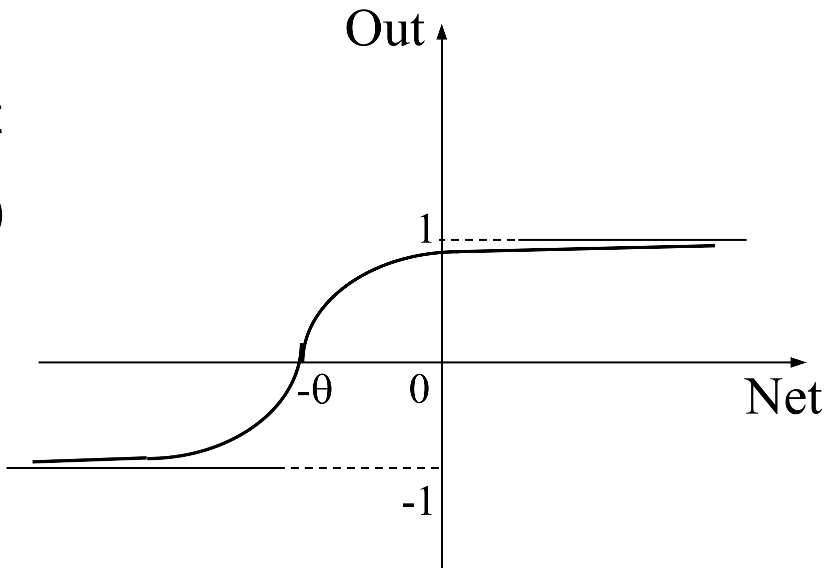
- Most often used in ANNs
- The slope parameter α is important
- The output value is always in (0,1)
- **Advantage**
  - Both continuous and continuously differentiable
  - The derivative of a sigmoidal function can be expressed in terms of the function itself



Activation function – Hyperbolic tangent

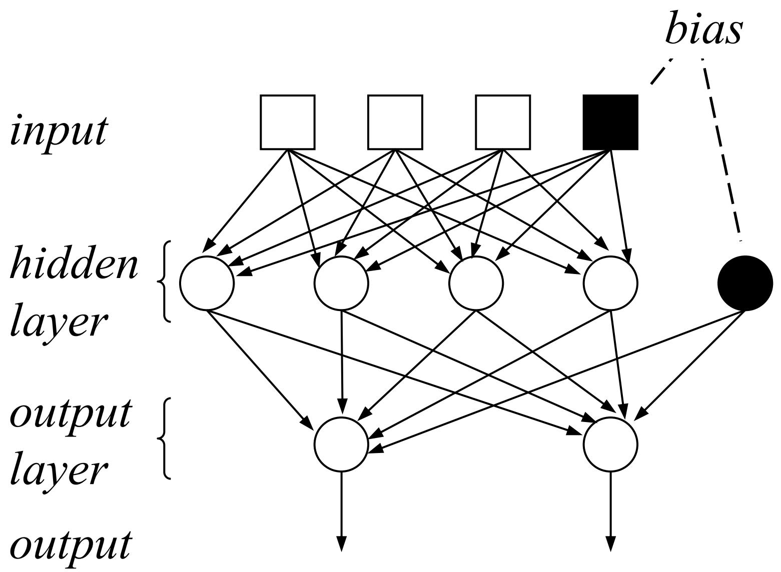
$$Out(Net) = \tanh(Net, \alpha, \theta) = \frac{1 - e^{-\alpha(Net+\theta)}}{1 + e^{-\alpha(Net+\theta)}} = \frac{2}{1 + e^{-\alpha(Net+\theta)}} - 1$$

- Also often used in ANNs
- The slope parameter α is important
- The output value is always in (-1,1)
- **Advantage**
  - Both continuous and continuously differentiable
  - The derivative of a `tanh` function can be expressed in terms of the function itself



Network structure

- Topology of an ANN is composed by:
  - The number of input signals and output signals
  - The number of layers
  - The number of neurons in each layer
  - The number of weights in each neuron
  - The way the weights are linked together within or between the layer(s)
  - Which neurons receive the (error) correction signals
- Every ANN must have
  - exactly one input layer
  - exactly one output layer
  - zero, one, or more than one hidden layer(s)



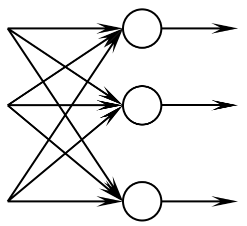
- An ANN with one hidden layer
- Input space: 3-dimensional
- Output space: 2-dimensional
- In total, there are 6 neurons
  - 4 in the hidden layer
  - 2 in the output layer

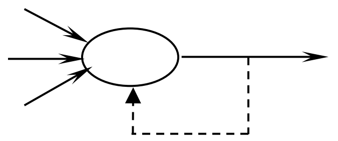
Network structure

- A layer is a group of neurons
- A hidden layer is any layer between the input and the output layers
- Hidden nodes do not directly interact with the external environment
- An ANN is said to be ***fully connected*** if every output from one layer is connected to every node in the next layer
- An ANN is called ***feed-forward network*** if no node output is an input to a node in the same layer or in a preceding layer
- When node outputs can be directed back as inputs to a node in the same (or a preceding) layer, it is a ***feedback network***
    - If the feedback is directed back as input to the nodes in the same layer, then it is called ***lateral feedback***
- Feedback networks that have closed loops are called ***recurrent networks***

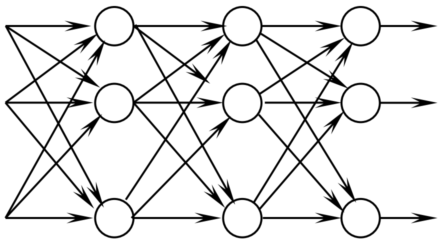
Network structure – Example

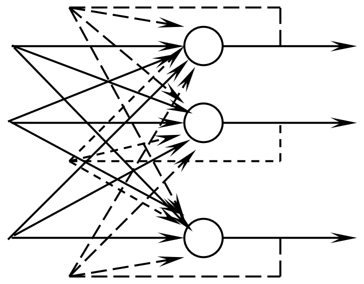
single layer
feed-forward
network





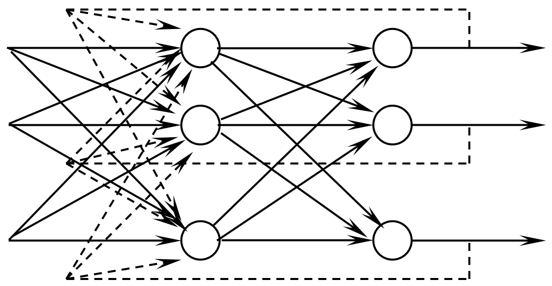
single node with
feedback to itself





single layer
recurrent
network

multilayer
feed-forward
network



multilayer
recurrent
network

Learning rules

- Two kinds of learning in neural networks
  - *Parameter learning*
    - → Focus on the update of the connecting weights in an ANN
  - *Structure learning*
    - → Focus on the change of the network structure, including the number of processing elements and their connection types
- These two kinds of learning can be performed simultaneously or separately
- Most of the existing learning rules are the type of parameter learning
- We focus the parameter learning

General weight learning rule

- At a learning step (*t*) the adjustment of the weight vector ***w*** is proportional to the product of the learning signal $r^{(t)}$ and the input $\boldsymbol{x}^{(t)}$

  $$\Delta \mathbf{w}^{(t)} \sim r^{(t)}.\mathbf{x}^{(t)}$$

  $$\Delta \mathbf{w}^{(t)} = \eta.r^{(t)}.\mathbf{x}^{(t)}$$

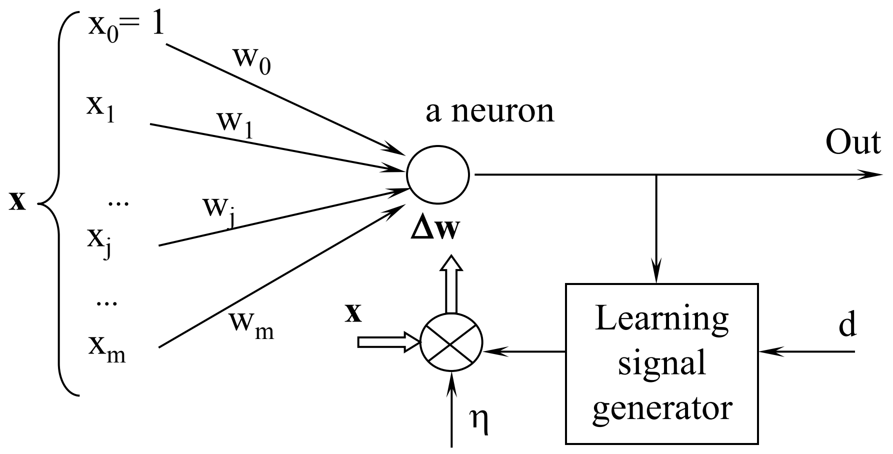
  where η (>0) is the learning rate

- The learning signal *r* is a function of ***w***, ***x***, and the desired output *d*

  $$r = g(\mathbf{w},\mathbf{x},d)$$

- The general weight learning rule

  $$\Delta \mathbf{w}^{(t)} = \eta.g(\mathbf{w}^{(t)},\mathbf{x}^{(t)},d^{(t)}).\mathbf{x}^{(t)}$$



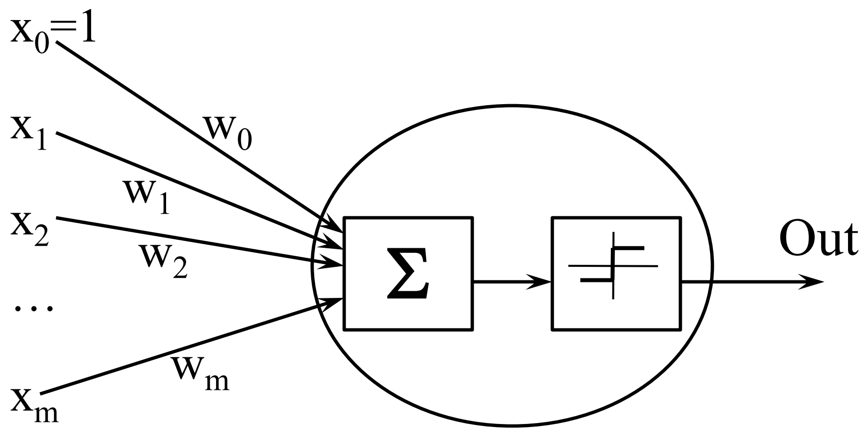
Note that x_j can be either:

- an (external) input signal, or
- an output from another neuron

Perceptron

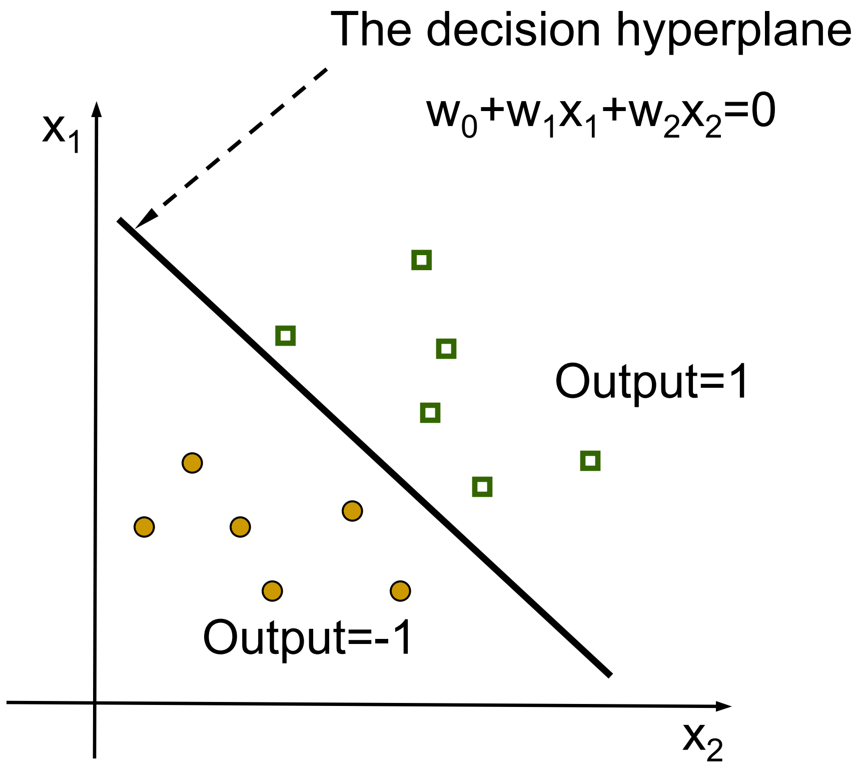
- A perceptron is the simplest type of ANNs
- Use the hard-limit activation function

$$Out = sign(Net(w,x)) = sign\left(\sum_{j=0}^{m} w_j x_j\right)$$



- For an instance ***x***, the perceptron output is
  - 1, if Net(**w**,**x**)>0
  - -1, otherwise

Perceptron – Illustration



Perceptron – Learning

- Given a training set D= {(**x**,d)}
    - ***x*** is the input vector
    - *d* is the desired output value (i.e., -1 or 1)
- The perceptron learning is to determine a weight vector that makes the perceptron produce the correct output (-1 or 1) for every training instance
- If a training instance ***x*** is correctly classified, then no update is needed
- If d=1 but the perceptron outputs -1, then the weight ***w*** should be updated so that Net(**w**,**x**) is increased
- If d=-1 but the perceptron outputs 1, then the weight ***w*** should be updated so that Net(**w**,**x**) is decreased

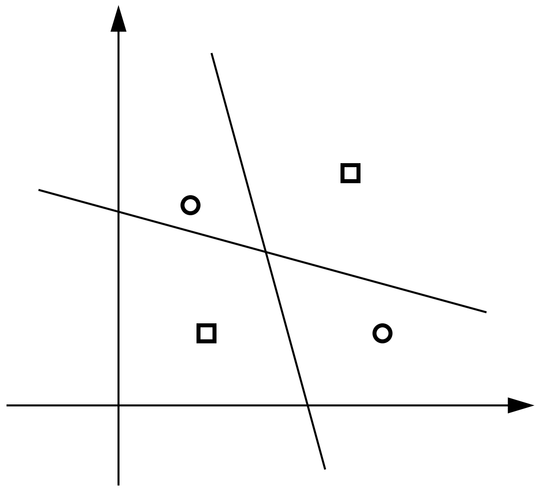
```
Perceptron_incremental(D, η)
Initialize w  ($w_i \leftarrow$ an initial (small) random value)
do
    for each training instance $(\mathbf{x}, d) \in D$
        Compute the real output value Out
        if (Out≠d)
            $\mathbf{w} \leftarrow \mathbf{w} + \eta(d - Out)\mathbf{x}$
    end for
until all the training instances in D are correctly classified
return w
```

```
Perceptron_batch(D, $\eta$)
Initialize w  ($w_i \leftarrow$ an initial (small) random value)
do
    $\Delta \mathbf{w} \leftarrow 0$
    for each training instance $(\mathbf{x}, d) \in D$
        Compute the real output value Out
        if ($Out \neq d$)
            $\Delta \mathbf{w} \leftarrow \Delta \mathbf{w} + \eta(d - Out)\mathbf{x}$
    end for
    $\mathbf{w} \leftarrow \mathbf{w} + \Delta \mathbf{w}$
until all the training instances in D are correctly classified
return w
```

Perceptron - Limitation

- The perceptron learning procedure is proven to converge if
  - The training instances are linearly separable
  - With a sufficiently small η used
- The perceptron may not converge if the training instances are not linearly separable
- We need to use the **delta rule**
  - Converges toward a best-fit approximation of the target function
  - The delta rule uses **gradient descent** to search the hypothesis space (of possible weight vectors) to find the weight vector that best fits the training instances

A perceptron cannot correctly classify this training set!



Error (cost) function

- Let's consider an ANN that has *n* output neurons
- Given a training instance (**x**,d), the **training error** made by the currently estimated weights vector **w**:

$$E_{\mathbf{x}}(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{n}\left(d_i - Out_i\right)^2$$

- The **training error** made by the currently estimated weights vector **w** over the entire training set *D*:

$$E_D(\mathbf{w}) = \frac{1}{|D|}\sum_{\mathbf{x}\in D} E_{\mathbf{x}}(\mathbf{w})$$

Gradient descent

- **Gradient** of *E* (denoted as ∇E) is a vector
  - The direction points most uphill
  - The length is proportional to steepness of hill
- The gradient of ∇E specifies the direction that produces the **steepest increase** in *E*

$$\nabla E(\mathbf{w}) = \left(\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \ldots, \frac{\partial E}{\partial w_N} \right)$$

where *N* is the number of the weights in the network (i.e., *N* is the length of **w**)

- Hence, the direction that produces the **steepest decrease** is the negative of the gradient of *E*

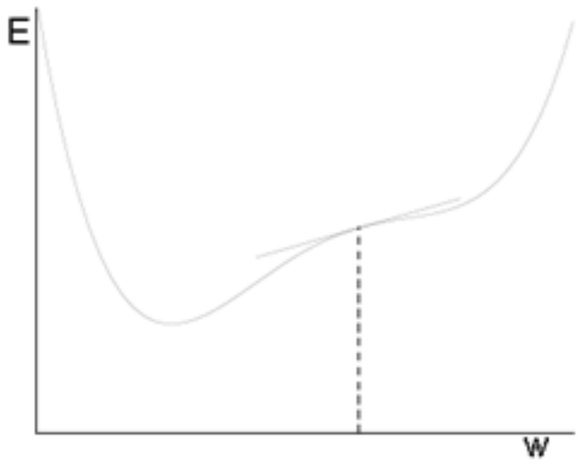
$$\Delta \mathbf{w} = -\eta \, . \, \nabla E(\mathbf{w}) \, ; \qquad \Delta w_i = -\eta \frac{\partial E}{\partial w_i}, \quad \forall i = 1..N$$

- Requirement: The activation functions used in the network must be continuous functions of the weights, differentiable everywhere

Gradient descent – Illustration

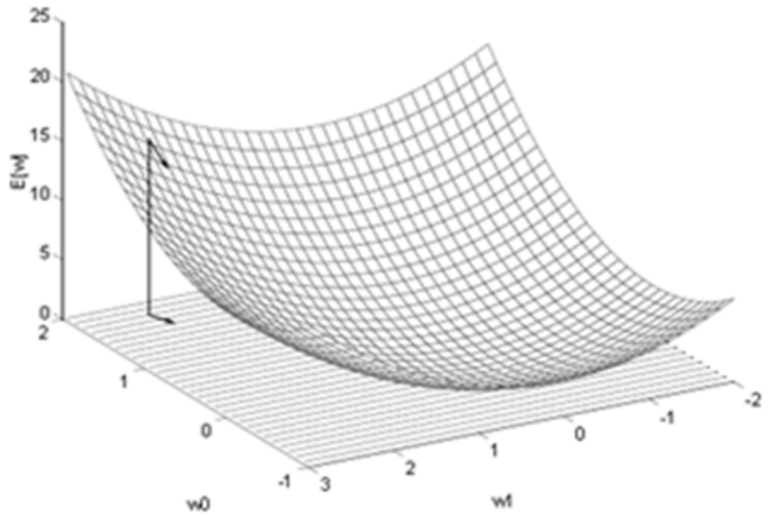
One-dimensional

E(w)



Two-dimensional

$E(w_1,w_2)$



```
Gradient_descent_incremental (D, η)
Initialize w  ($w_i \leftarrow$ an initial (small) random value)
do
    for each training instance $(\mathbf{x}, d) \in D$
        Compute the network output
        for each weight component $w_i$
            $w_i \leftarrow w_i - \eta(\partial E_{\mathbf{x}} / \partial w_i)$
        end for
    end for
until (stopping criterion satisfied)
return w
```

Stopping criterion: # of iterations (epochs), threshold error, etc.

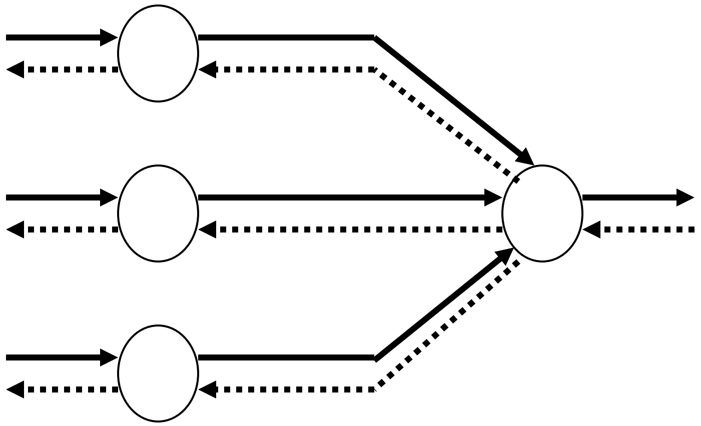
Multi-layer NNs and Back-propagation alg.

- As we have seen, a perceptron can only express a linear decision surface
- A multi-layer NN learned by the back-propagation (BP) algorithm can represent *highly non-linear decision surfaces*
- The BP learning algorithm is used to learn the weights of a multi-layer NN
  - *Fixed structure* (i.e., fixed set of neurons and interconnections)
  - For every neuron the activation function must be *continuously differentiable*
- The BP algorithm *employs gradient descent* in the weight update rule
  - To minimize the error between the actual output values and the desired output ones, given the training instances

Back-propagation algorithm (1)

- Back-propagation algorithm searches for the weights vector that **minimizes the total error** made over the training set
- Back-propagation consists of the two phases
  - **Signal forward** phase. The input signals (i.e., the input vector) are <u>propagated (forwards)</u> from the input layer to the output layer (through the hidden layers)
  - **Error backward** phase
    - Since the desired output value for the current input vector is known, the error is computed
    - Starting at the output layer, the error is <u>propagated backwards</u> through the network, layer by layer, to the input layer
    - The error back-propagation is performed by recursively computing the local gradient of each neuron

Back-propagation algorithm (2)

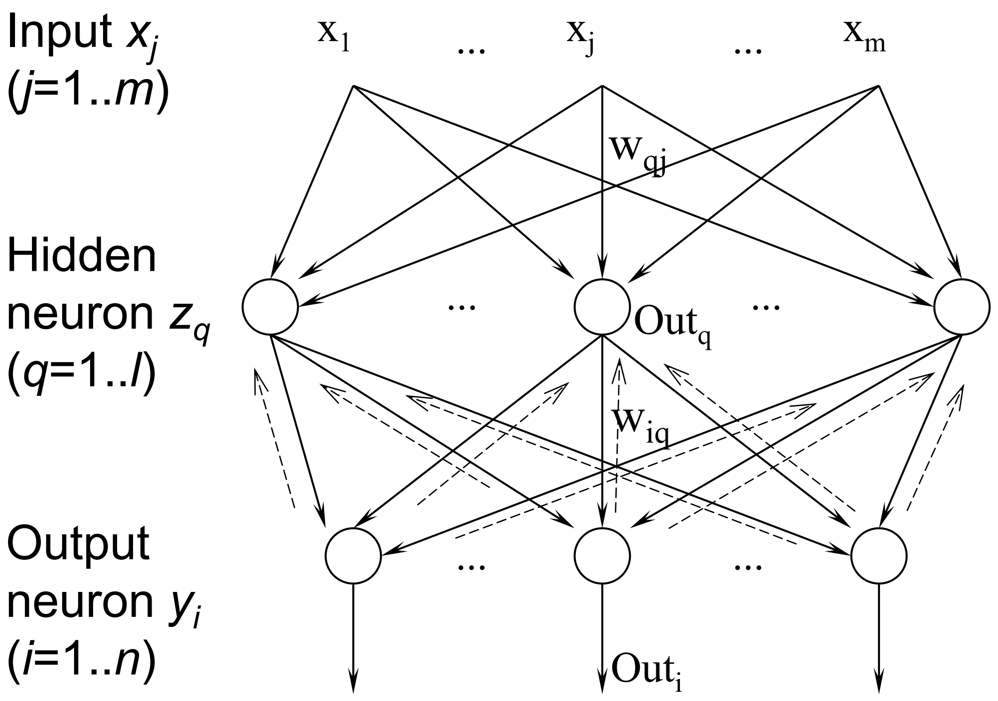


→ Signal forward phase
- Network activation

⇠ Error backward phase
- Output error computation
- Error propagation

Derivation of BP alg. – Network structure

- Let's use this 3-layer NN to illustrate the details of the BP learning algorithm
- m input signals x_j (j=1..m)
- l hidden neurons z_q (q=1..l)
- n output neurons y_i (i=1..n)
- w_{qj} is the weight of the interconnection from input signal x_j to hidden neuron z_q
- w_{iq} is the weight of the interconnection from hidden neuron z_q to output neuron y_i
- Out_q is the (local) output value of hidden neuron z_q
- Out_i is the network output w.r.t. the output neuron y_i



BP algorithm – Forward phase (1)

- For each training instance ***x***
  - The input vector ***x*** is *propagated* from the input layer to the output layer
  - The network produces an actual output ***Out*** (i.e., a vector of Out_i, i=1..n)

- Given an input vector ***x***, a neuron z_q in the hidden layer receives a net input of

$$Net_q = \sum_{j=1}^{m} w_{qj} x_j$$

…and produces a (local) output of

$$Out_q = f(Net_q) = f\left(\sum_{j=1}^{m} w_{qj} x_j\right)$$

where $f(.)$ is the activation (transfer) function of neuron z_q

BP algorithm – Forward phase (2)

- The net input for a neuron y_i in the output layer is

$$Net_i = \sum_{q=1}^{l} w_{iq} Out_q = \sum_{q=1}^{l} w_{iq} f\left(\sum_{j=1}^{m} w_{qj} x_j \right)$$

- Neuron y_i produces the output value (i.e., an output of the network)

$$Out_i = f(Net_i) = f\left(\sum_{q=1}^{l} w_{iq} Out_q \right) = f\left(\sum_{q=1}^{l} w_{iq} f\left(\sum_{j=1}^{m} w_{qj} x_j \right) \right)$$

- The vector of output values Out_i (i=1..n) is the actual network output, given the input vector $\boldsymbol{x}$

BP algorithm – Backward phase (1)

- For each training instance ***x***
  - The error signals resulting from the difference between the desired output ***d*** and the actual output ***Out*** are computed
  - The error signals are *back-propagated* from the output layer to the previous layers to update the weights
- Before discussing the error signals and their back propagation, we first define an error (cost) function

$$E(w) = \frac{1}{2}\sum_{i=1}^{n}\left(d_i - Out_i\right)^2 = \frac{1}{2}\sum_{i=1}^{n}\left[d_i - f\left(Net_i\right)\right]^2$$

$$= \frac{1}{2}\sum_{i=1}^{n}\left[d_i - f\left(\sum_{q=1}^{l} w_{iq} Out_q\right)\right]^2$$

BP algorithm – Backward phase (2)

- According to the gradient-descent method, the weights in the **hidden-to-output** connections are updated by

$$\Delta w_{iq} = -\eta \frac{\partial E}{\partial w_{iq}}$$

- Using the derivative chain rule for $\partial E/\partial w_{iq}$, we have

$$\Delta w_{iq} = -\eta\left[\frac{\partial E}{\partial Out_i}\right]\left[\frac{\partial Out_i}{\partial Net_i}\right]\left[\frac{\partial Net_i}{\partial w_{iq}}\right] = \eta\left[d_i - Out_i\right]\left[f'\left(Net_i\right)\right]\left[Out_q\right] = \eta\delta_i Out_q$$

(note that the negative sign is incorporated in $\partial E/\partial Out_i$)

- δ_i is the **error signal** of neuron y_i in the **output layer**

$$\delta_i = -\frac{\partial E}{\partial Net_i} = -\left[\frac{\partial E}{\partial Out_i}\right]\left[\frac{\partial Out_i}{\partial Net_i}\right] = \left[d_i - Out_i\right]\left[f'\left(Net_i\right)\right]$$

where Net_i is the net input to neuron y_i in the output layer, and $f'(\text{Net}_i)=\partial f(\text{Net}_i)/\partial \text{Net}_i$

BP algorithm – Backward phase (3)

- To update the weights of the **input-to-hidden** connections, we also follow gradient-descent method and the derivative chain rule

$$\Delta w_{qj} = -\eta \frac{\partial E}{\partial w_{qj}} = -\eta \left[\frac{\partial E}{\partial Out_q} \right] \left[\frac{\partial Out_q}{\partial Net_q} \right] \left[\frac{\partial Net_q}{\partial w_{qj}} \right]$$

- From the equation of the error function *E(**w**)*, it is clear that each error term $(d_i - y_i)$ (i=1..n) is a function of Out_q

$$E(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^{n} \left[d_i - f\left(\sum_{q=1}^{l} w_{iq} Out_q \right) \right]^2$$

BP algorithm – Backward phase (4)

- Evaluating the derivative chain rule, we have

$$\Delta w_{qj} = \eta \sum_{i=1}^{n} \left[\left(d_i - Out_i \right) f'\left(Net_i \right) w_{iq} \right] f'\left(Net_q \right) x_j$$

$$= \eta \sum_{i=1}^{n} \left[\delta_i w_{iq} \right] f'\left(Net_q \right) x_j = \eta \delta_q x_j$$

- $\boldsymbol{\delta_q}$ is the **error signal** of neuron z_q in the **hidden layer**

$$\delta_q = -\frac{\partial E}{\partial Net_q} = -\left[\frac{\partial E}{\partial Out_q} \right] \left[\frac{\partial Out_q}{\partial Net_q} \right] = f'\left(Net_q \right) \sum_{i=1}^{n} \delta_i w_{iq}$$

where Net_q is the net input to neuron z_q in the hidden layer, and $f'(\mathrm{Net_q}) = \partial f(\mathrm{Net_q}) / \partial \mathrm{Net_q}$

BP algorithm – Backward phase (5)

- According to the error equations δ_i and δ_q above, the **error signal** of a neuron in a **hidden** layer is different from the error signal of a neuron in the **output** layer
- Because of this difference, the derived weight update procedure is called the *generalized delta learning rule*
- The **error signal** δ_q of a **hidden** neuron z_q can be determined
  - in terms of the **error signals** δ_i of the neurons y_i (i.e., that z_q connects to) in the **output** layer
  - with the coefficients are just the weights w_{iq}
- The important feature of the BP algorithm: **the weights update rule is local**
  - To compute the weight change for a given connection, we need only the quantities available at both ends of that connection!

BP algorithm – Backward phase (6)

- The discussed derivation can be easily extended to the network with more than one hidden layer by using the chain rule continuously
- The general form of the BP update rule is

$$\Delta w_{ab} = \eta \delta_a x_b$$

  - *b* and *a* refer to the two ends of the ($b \rightarrow a$) connection (i.e., from neuron (or input signal) *b* to neuron *a*)
  - x_b is the output of the hidden neuron (or the input signal) *b*,
  - δ_a is the error signal of neuron *a*

Back_propagation_incremental(D, η)

A network with Q feed-forward layers, $q = 1,2,...,Q$

${}^qNet_i$ and ${}^qOut_i$ are the net input and output of the i^{th} neuron in the q^{th} layer

The network has m input signals and n output neurons

${}^qw_{ij}$ is the weight of the connection from the j^{th} neuron in the $(q\text{-}1)^{th}$ layer to the i^{th} neuron in the q^{th} layer

Step 0 (Initialization)

Choose $E_{threshold}$ (a tolerable error)

Initialize the weights to small random values

Set E=0

Step 1 (Training loop)

Apply the input vector of the k^{th} training instance to the input layer (q=1)

${}^qOut_i = {}^1Out_i = x_i^{(k)}, \forall I$

Step 2 (Forward propagation)

Propagate the signal forward through the network, until the network outputs (in the output layer) ${}^QOut_i$ have all been obtained

$${}^qOut_i = f\left({}^qNet_i\right) = f\left(\sum_j {}^qw_{ij}\, {}^{q-1}Out_j\right)$$

Step 3 (Output error measure)

Compute the error and error signals ${}^{Q}\delta_i$ for every neuron in the output layer

$$E = E + \frac{1}{2}\sum_{i=1}^{n}(d_i^{(k)} - {}^{Q}Out_i)^2$$

$${}^{Q}\delta_i = (d_i^{(k)} - {}^{Q}Out_i)f'({}^{Q}Net_i)$$

Step 4 (Error back-propagation)

Propagate the error backward to update the weights and compute the error signals ${}^{q-1}\delta_i$ for the preceding layers

$$\Delta{}^{q}w_{ij} = \eta.({}^{q}\delta_i).({}^{q-1}Out_j); \qquad {}^{q}w_{ij} = {}^{q}w_{ij} + \Delta{}^{q}w_{ij}$$

$${}^{q-1}\delta_i = f'({}^{q-1}Net_i)\sum_j {}^{q}w_{ji}\,{}^{q}\delta_j; \quad \text{for all } q = Q, Q-1, \ldots, 2$$

Step 5 (One epoch check)

Check whether the entire training set has been exploited (i.e., one epoch)

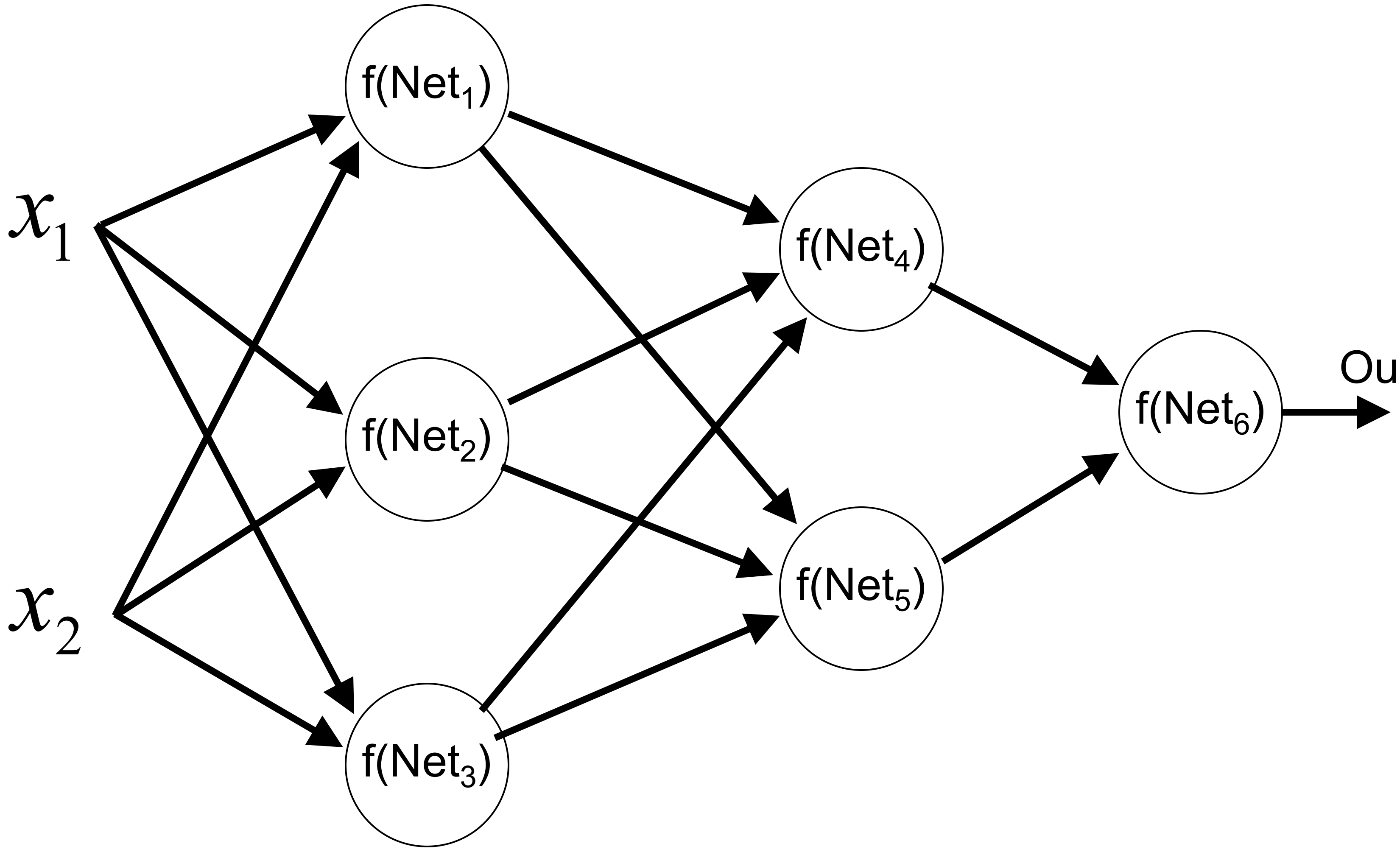
If the entire training set has been exploited, then go to step 6; otherwise, go to step 1

Step 6 (Total error check)

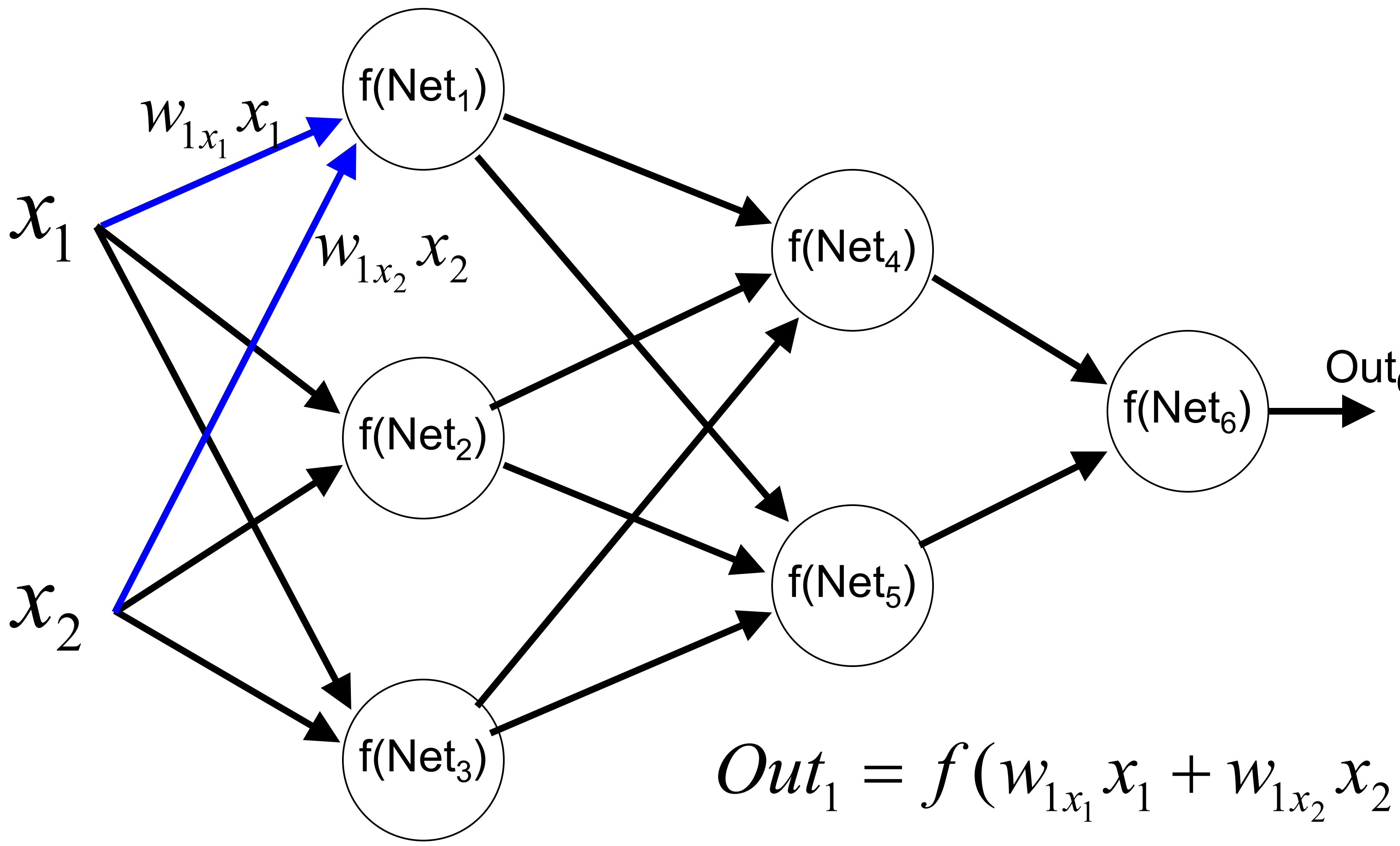
If the current total error is acceptable ($E<E_{threshold}$) then the training process terminates and output the final weights;

Otherwise, reset E=0, and initiate the new training epoch by going to step 1

BP illustration – Forward phase (1)

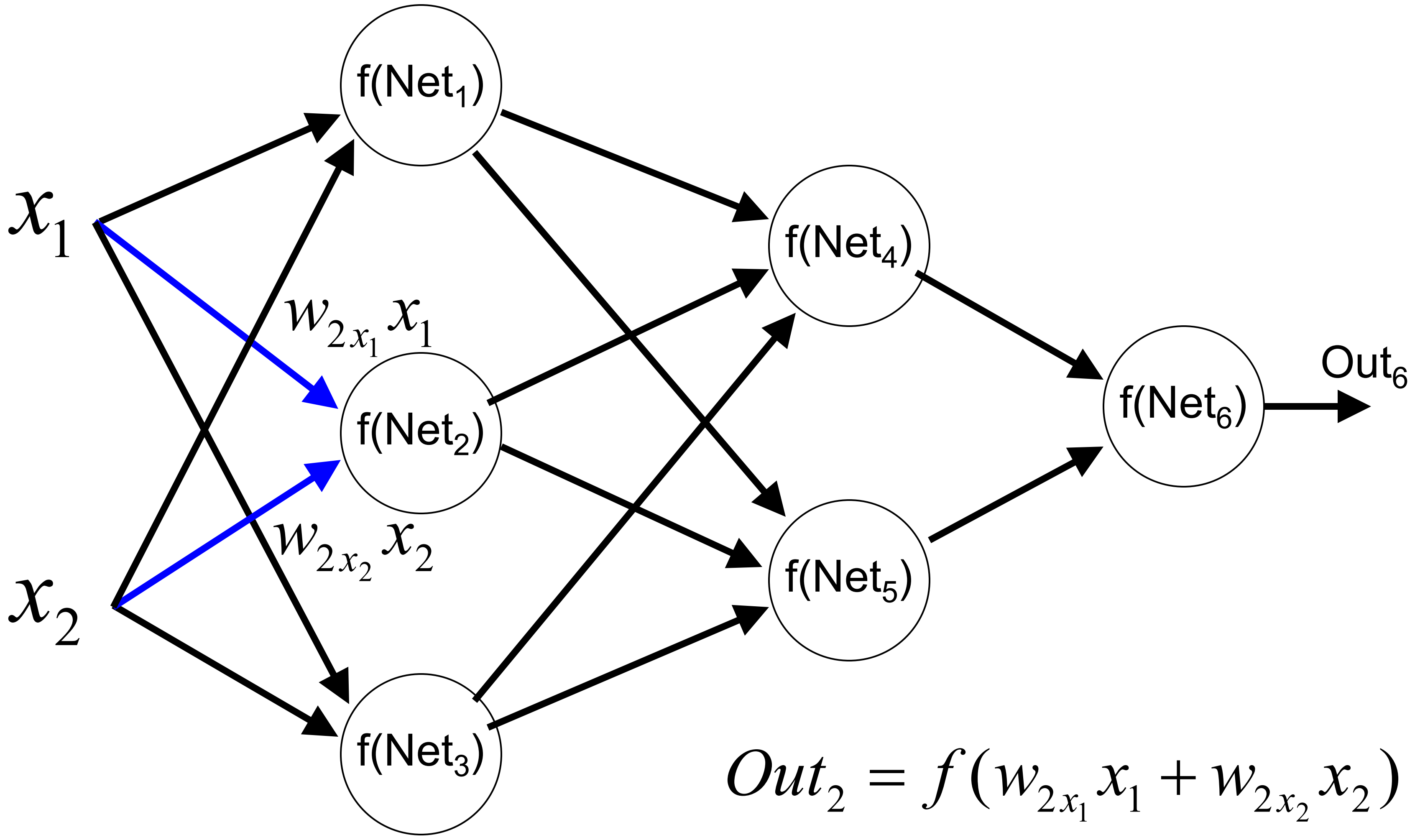


BP illustration – Forward phase (2)



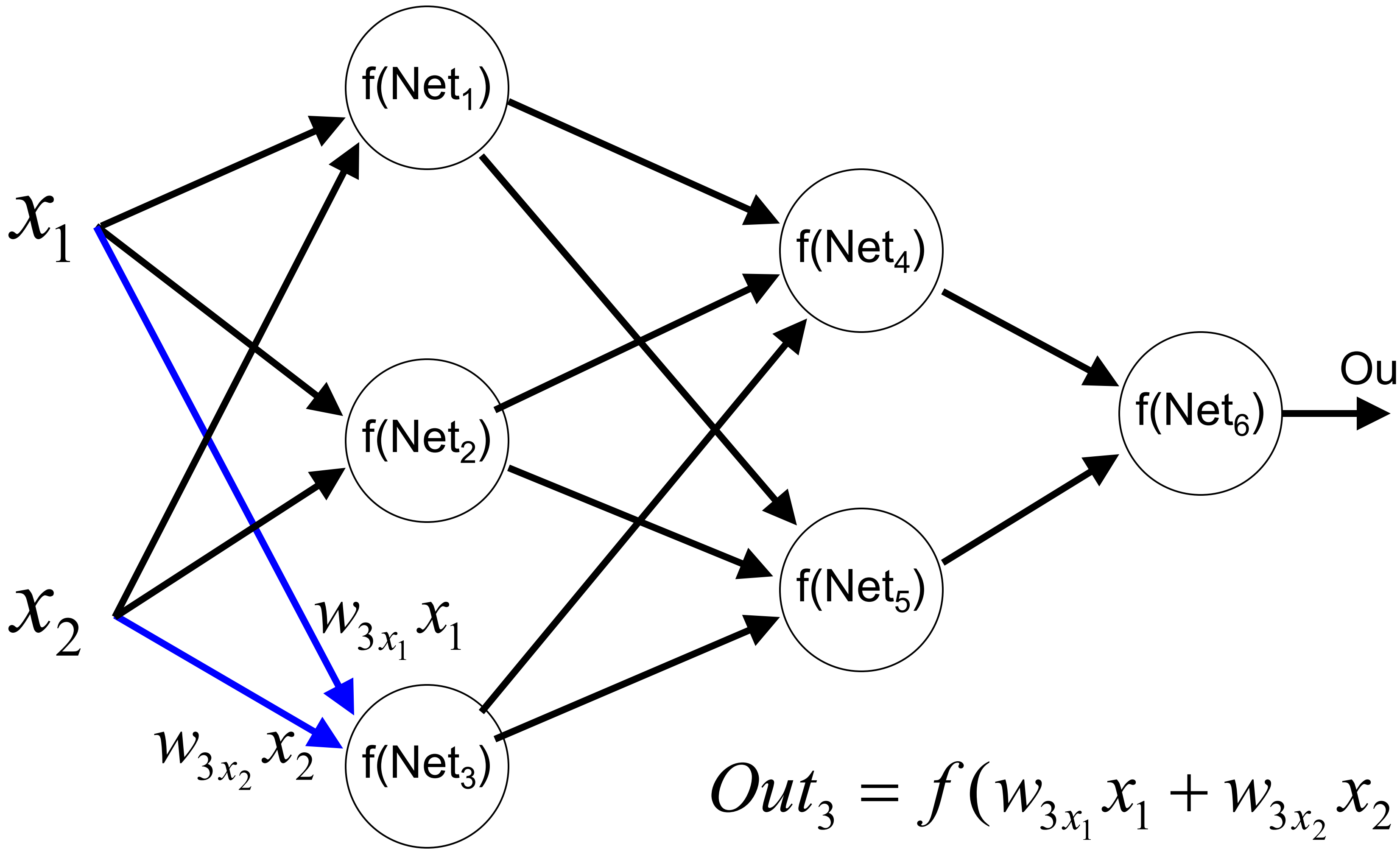
$$Out_1 = f(w_{1x_1}x_1 + w_{1x_2}x_2)$$

BP illustration – Forward phase (3)



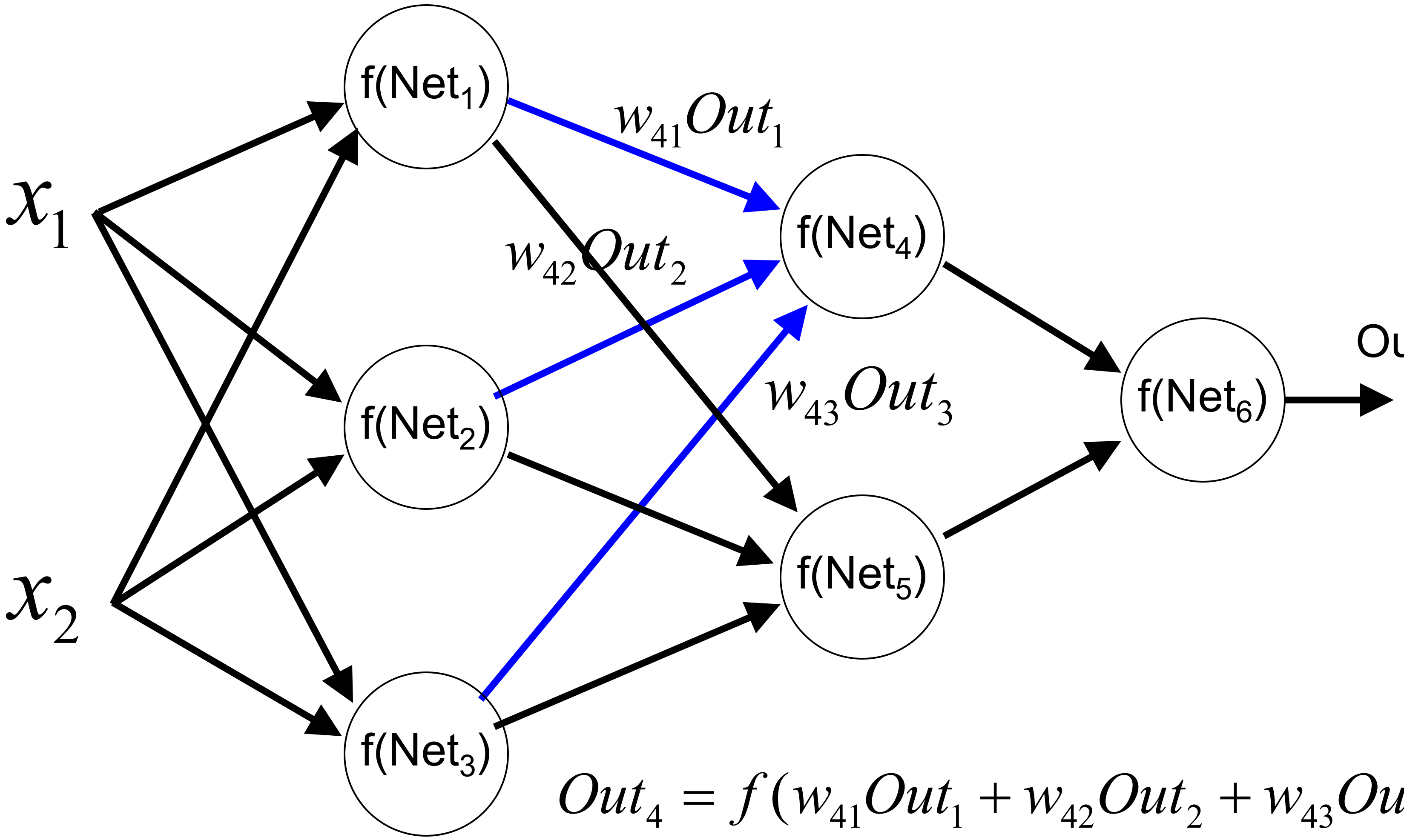
$$Out_2 = f(w_{2x_1}x_1 + w_{2x_2}x_2)$$

BP illustration – Forward phase (4)



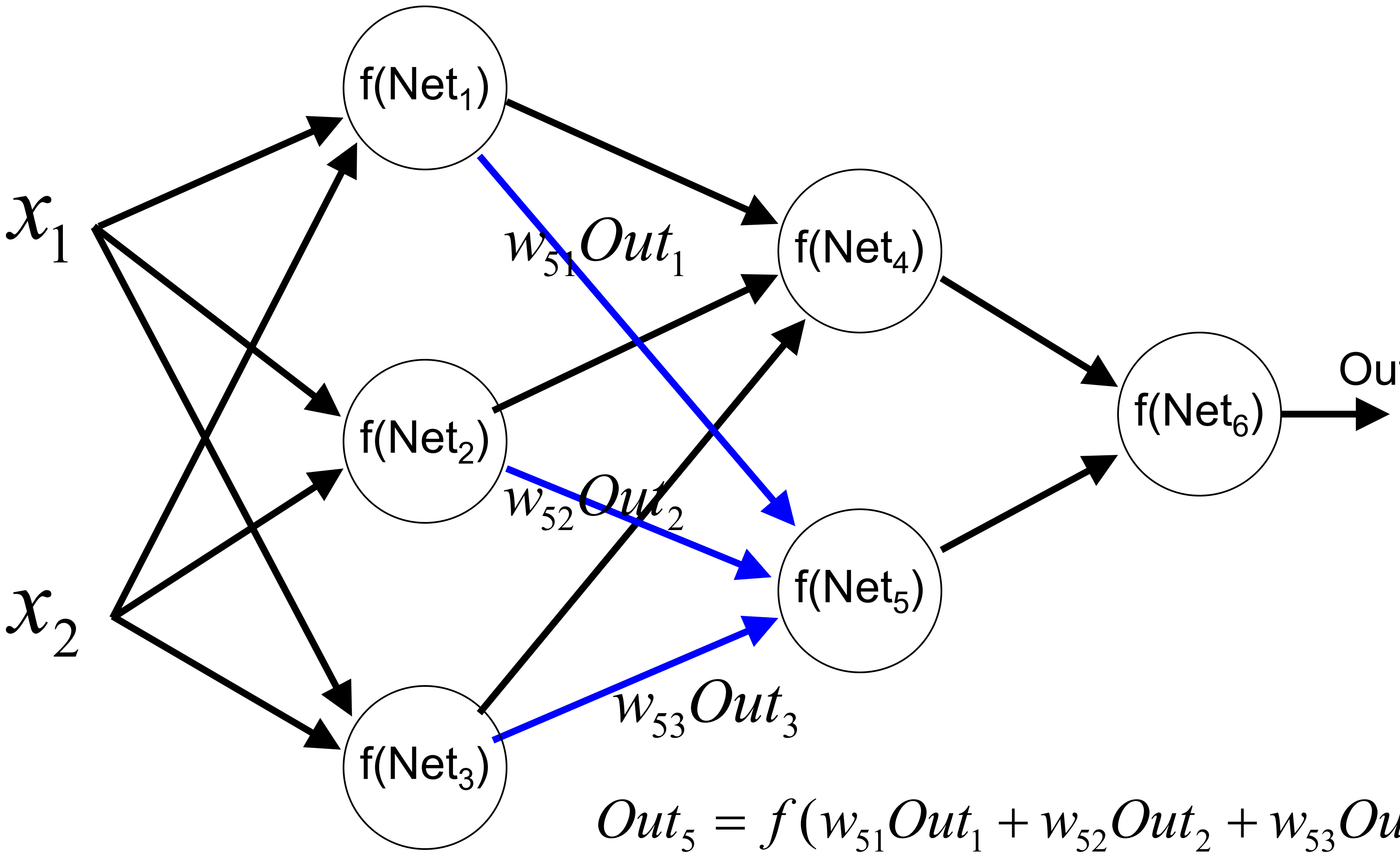
$$Out_3 = f(w_{3x_1}x_1 + w_{3x_2}x_2)$$

BP illustration – Forward phase (5)



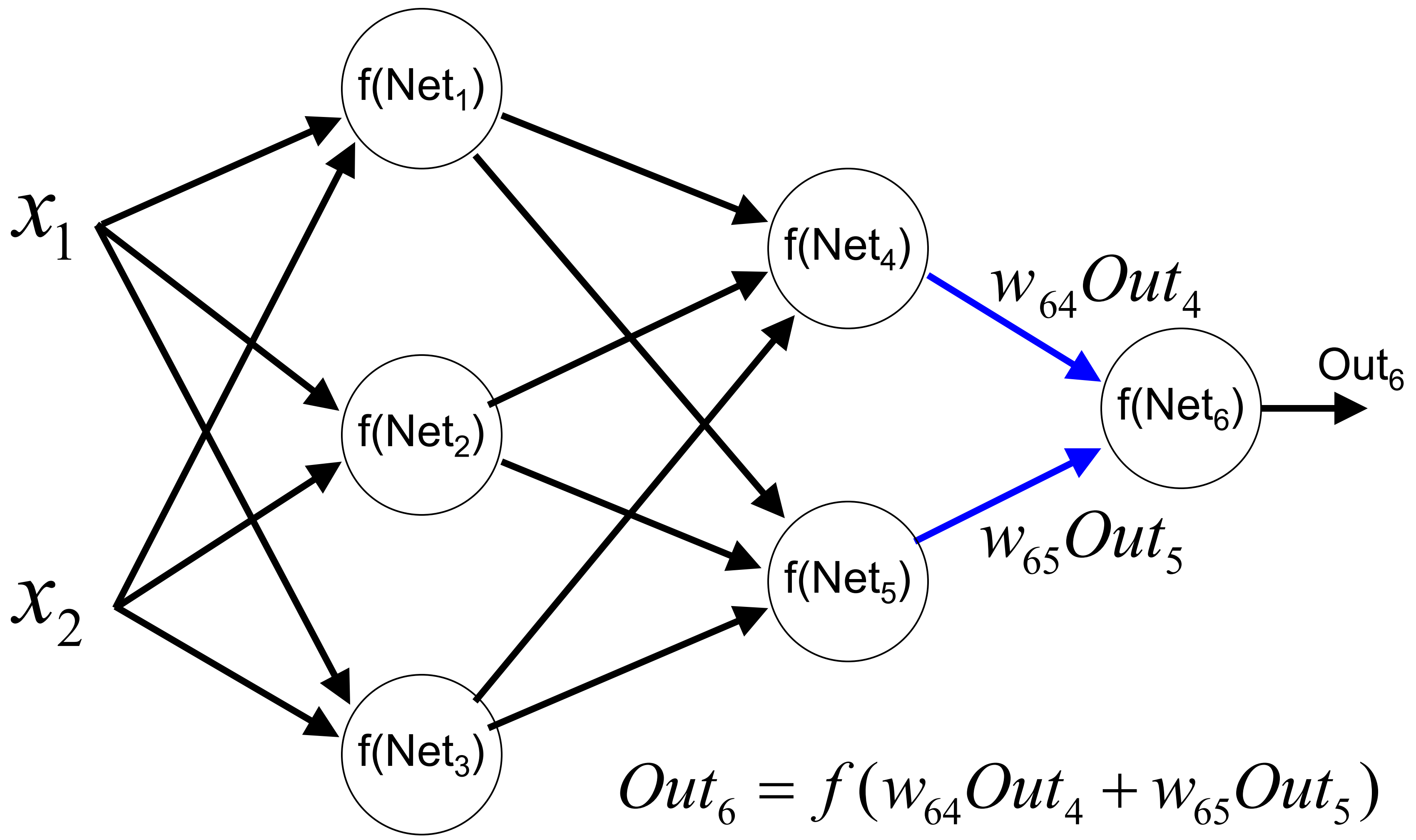
$$Out_4 = f(w_{41}Out_1 + w_{42}Out_2 + w_{43}Out_3)$$

BP illustration – Forward phase (6)



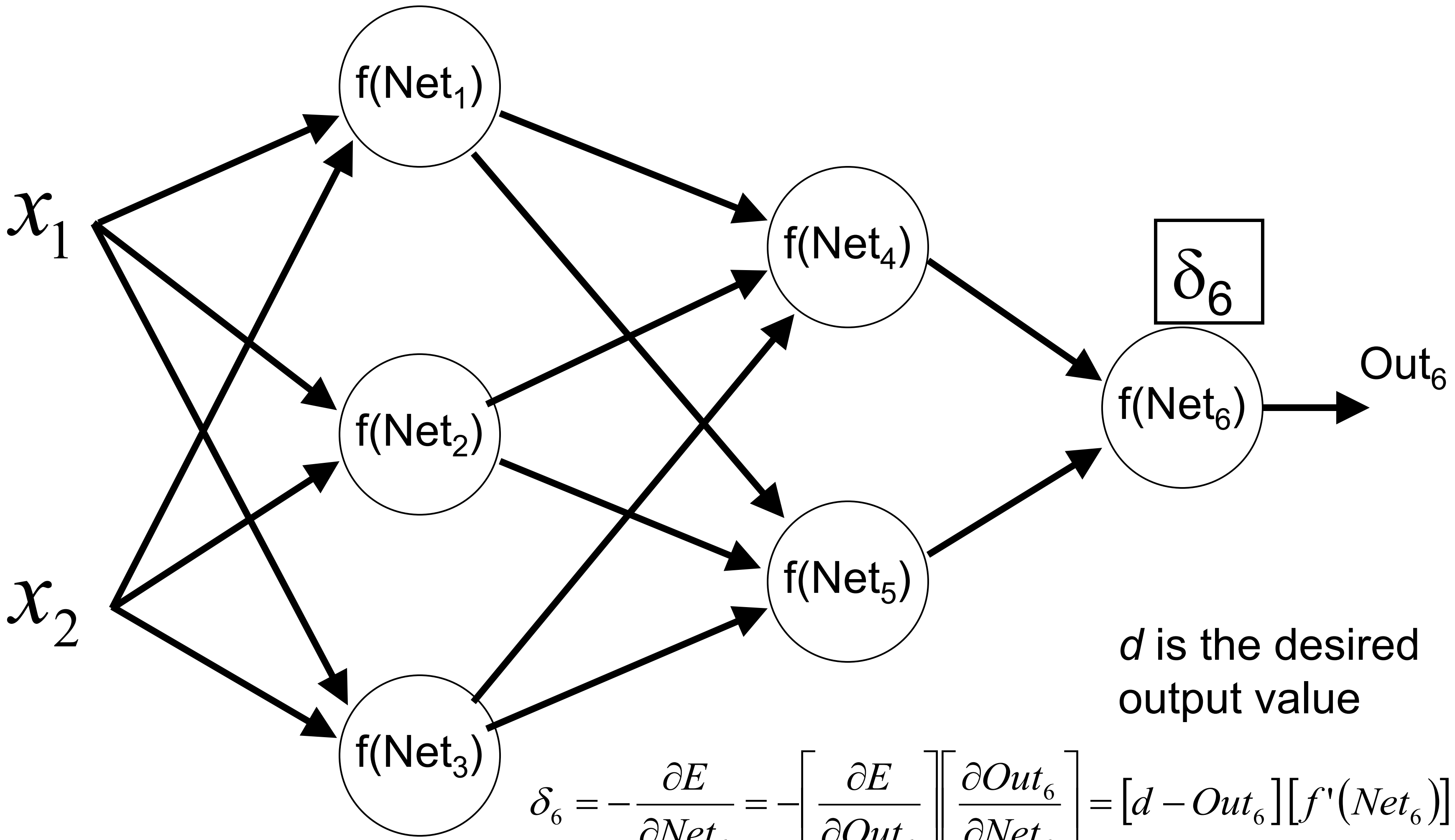
$$Out_5 = f(w_{51}Out_1 + w_{52}Out_2 + w_{53}Out_3)$$

BP illustration – Forward phase (7)



$$Out_6 = f(w_{64}Out_4 + w_{65}Out_5)$$

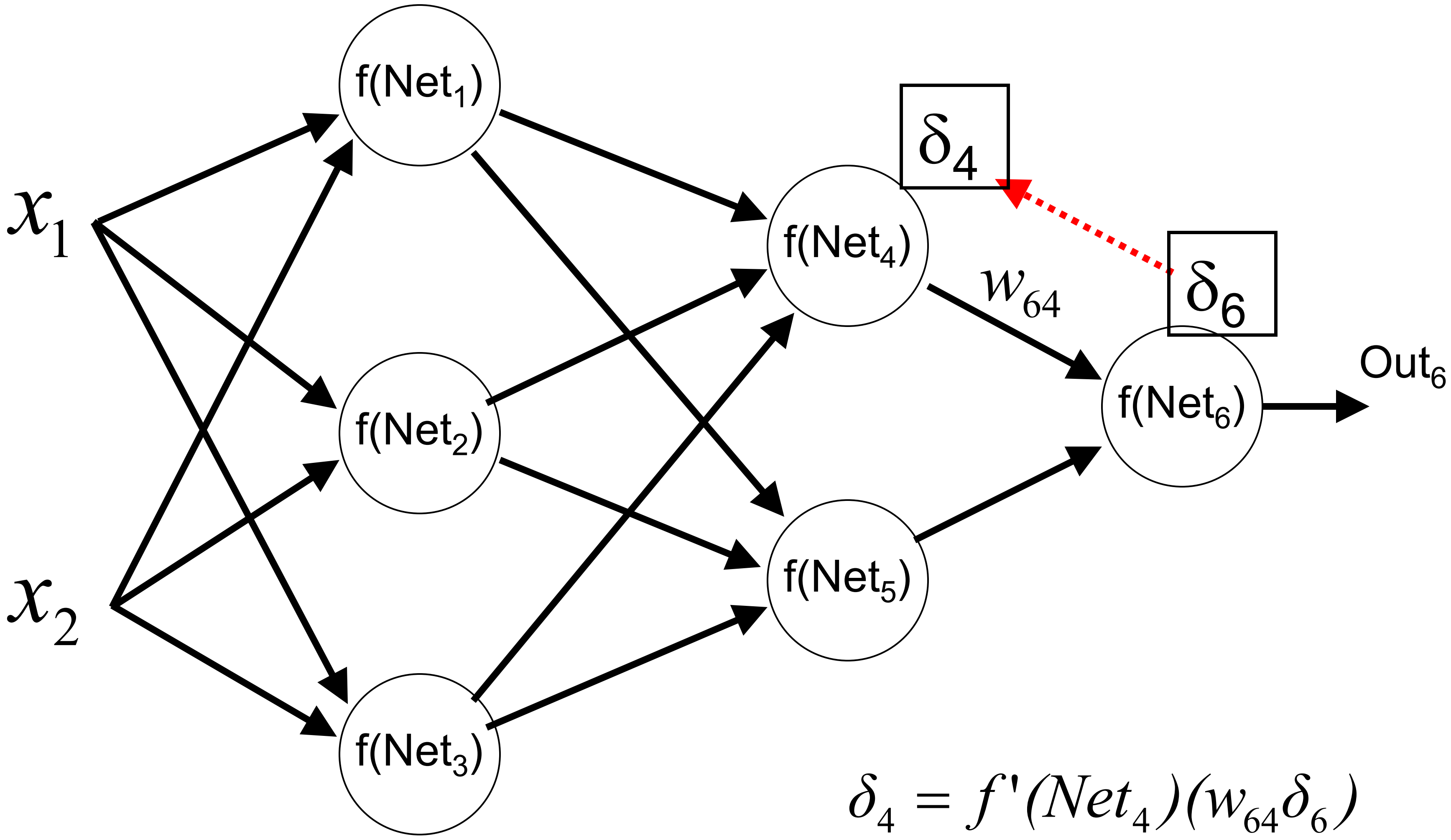
BP illustration – Compute the error



d is the desired output value

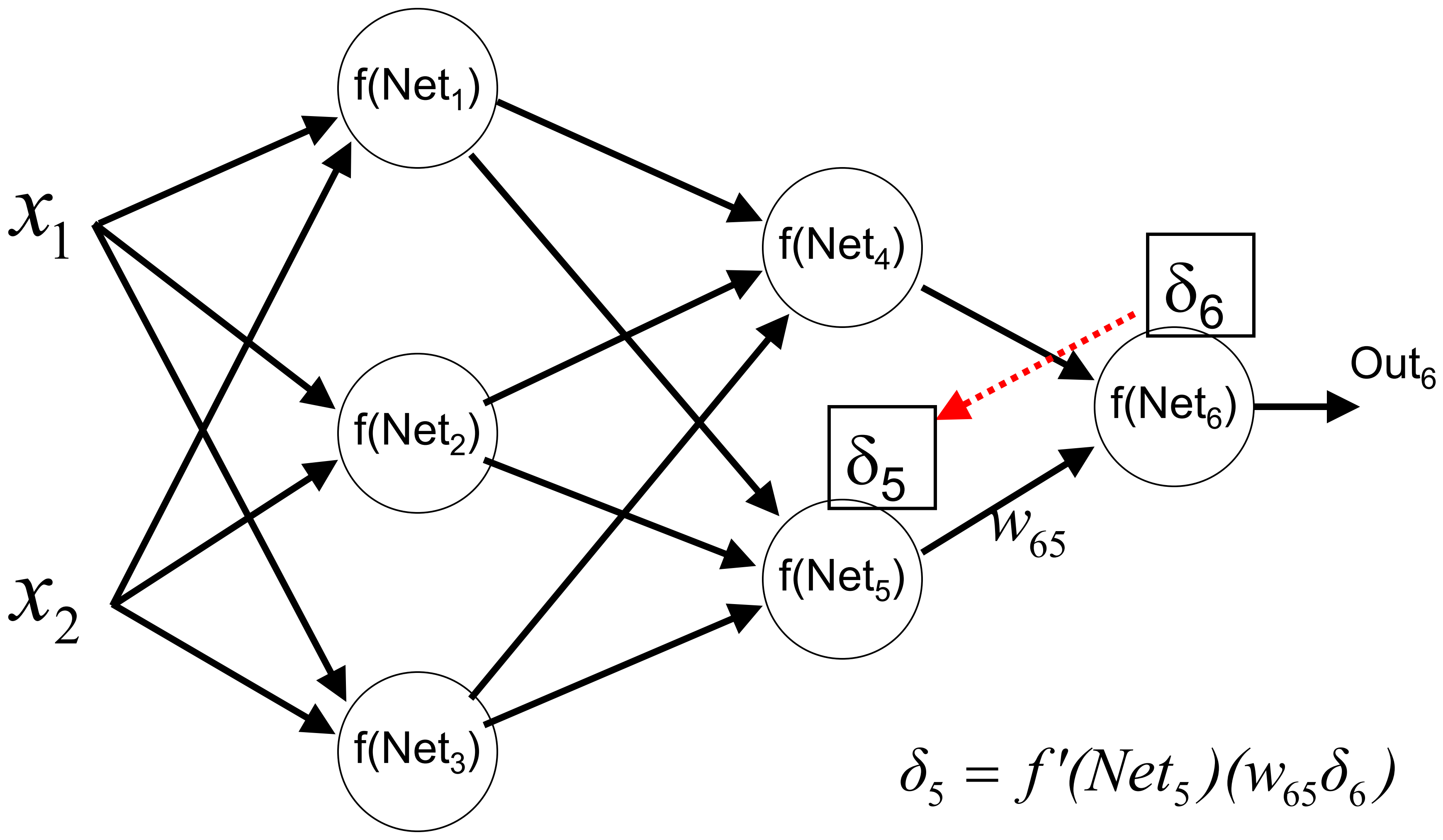
$$\delta_6 = -\frac{\partial E}{\partial Net_6} = -\left[\frac{\partial E}{\partial Out_6}\right]\left[\frac{\partial Out_6}{\partial Net_6}\right] = [d - Out_6][f'(Net_6)]$$

BP illustration – Backward phase (1)



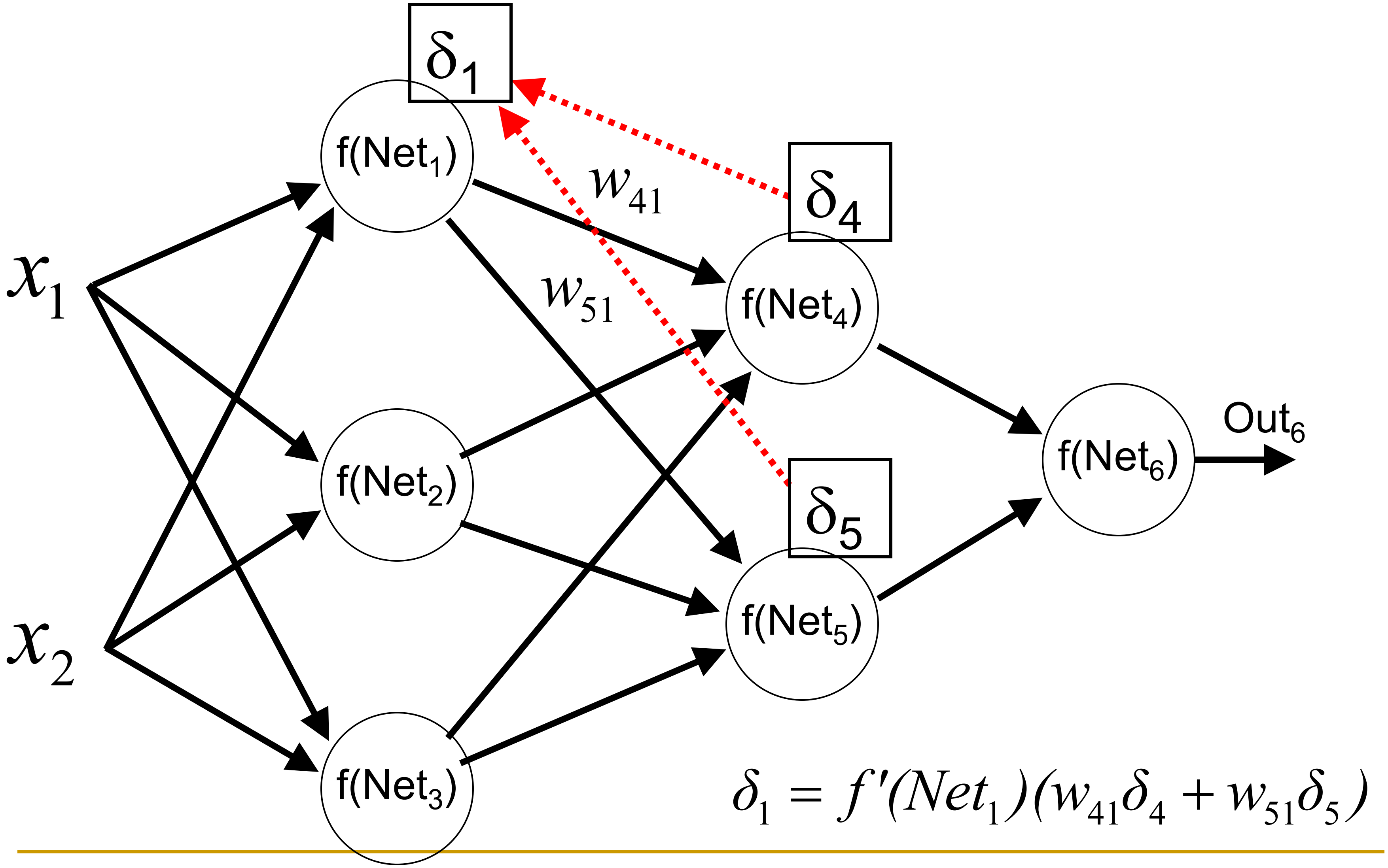
$$\delta_4 = f'(Net_4)(w_{64}\delta_6)$$

BP illustration – Backward phase (2)



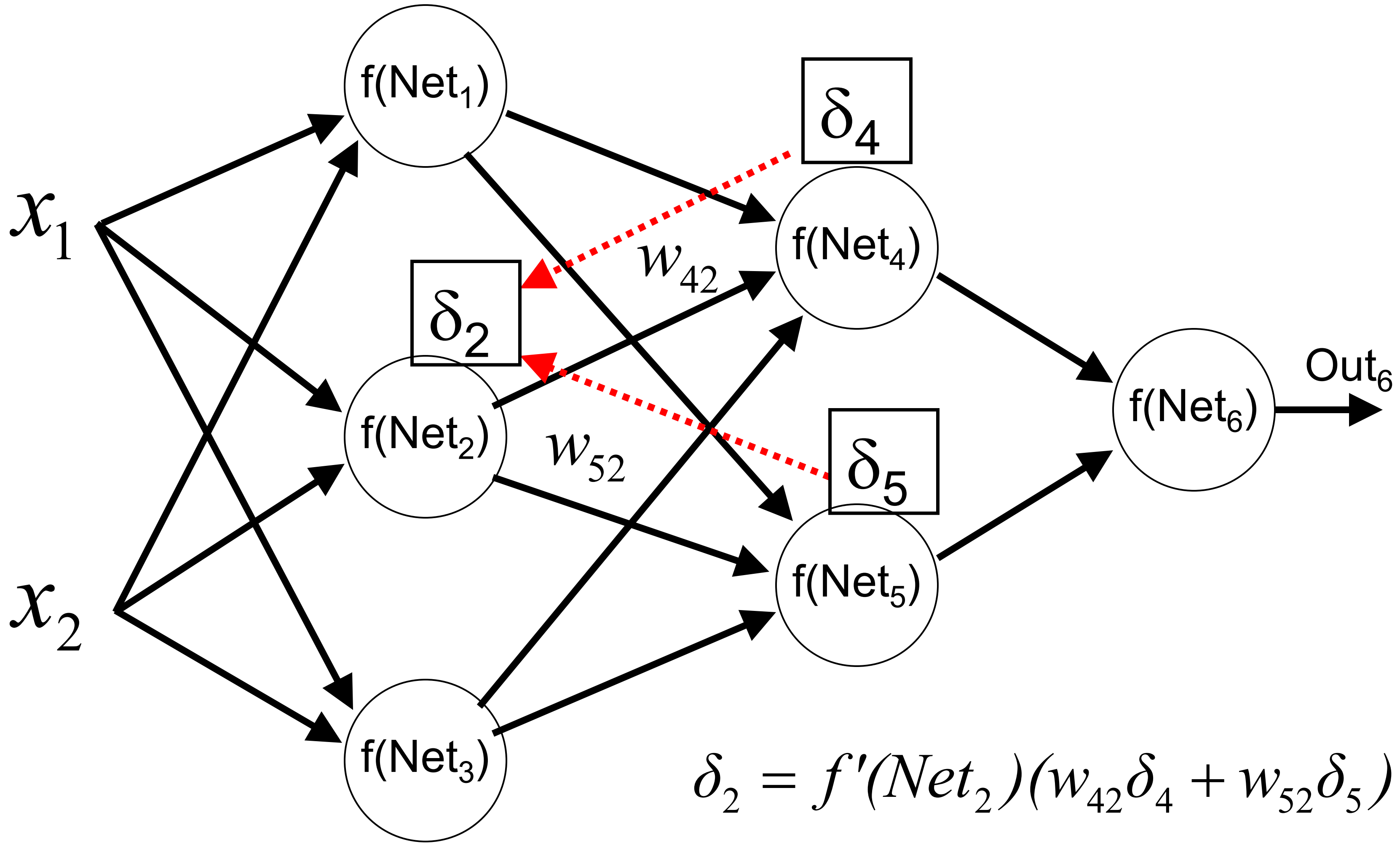
$$\delta_5 = f'(Net_5)(w_{65}\delta_6)$$

BP illustration – Backward phase (3)



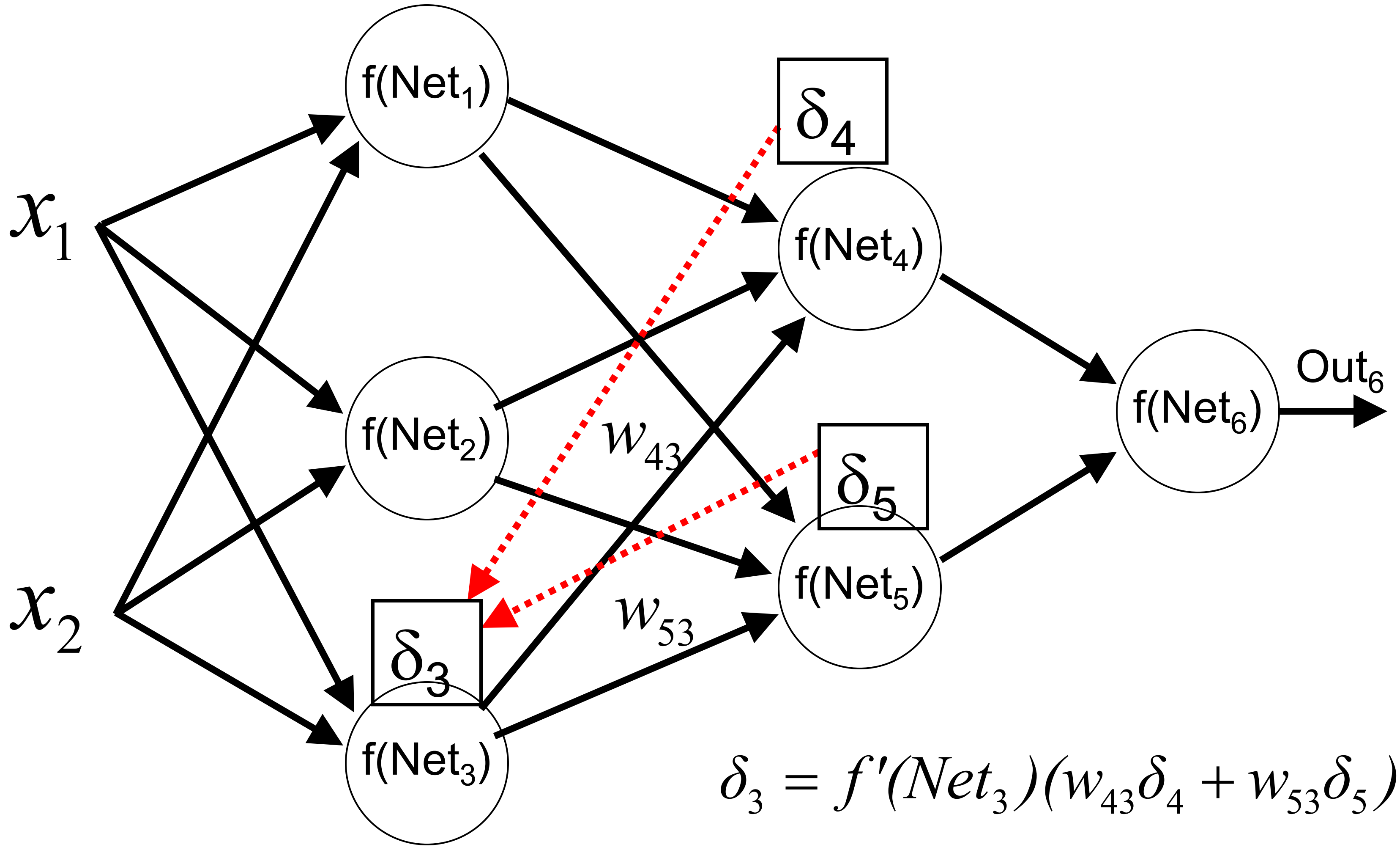
$$\delta_1 = f'(Net_1)(w_{41}\delta_4 + w_{51}\delta_5)$$

BP illustration – Backward phase (4)



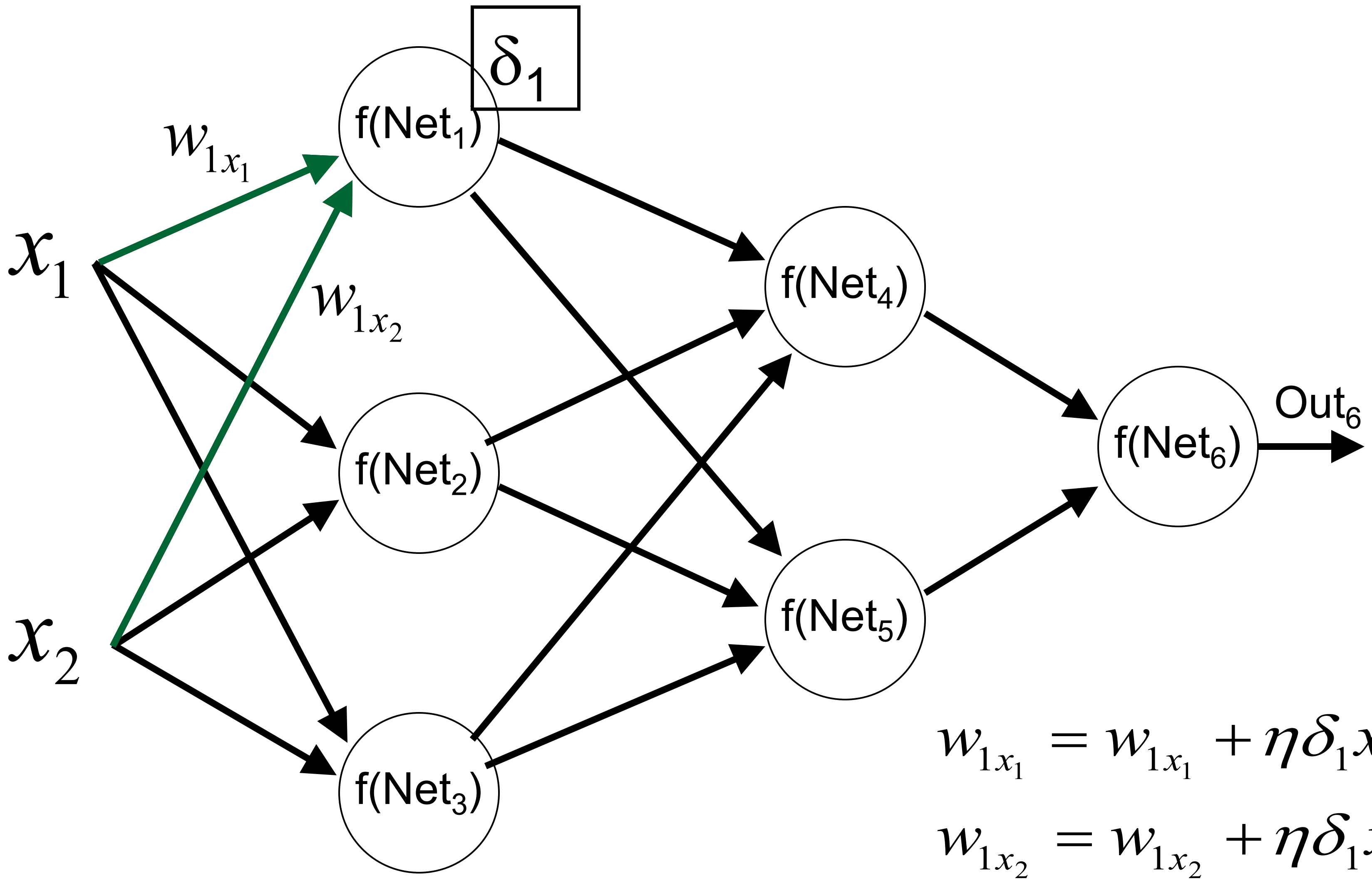
$$\delta_2 = f'(Net_2)(w_{42}\delta_4 + w_{52}\delta_5)$$

BP illustration – Backward phase (5)



$$\delta_3 = f'(Net_3)(w_{43}\delta_4 + w_{53}\delta_5)$$

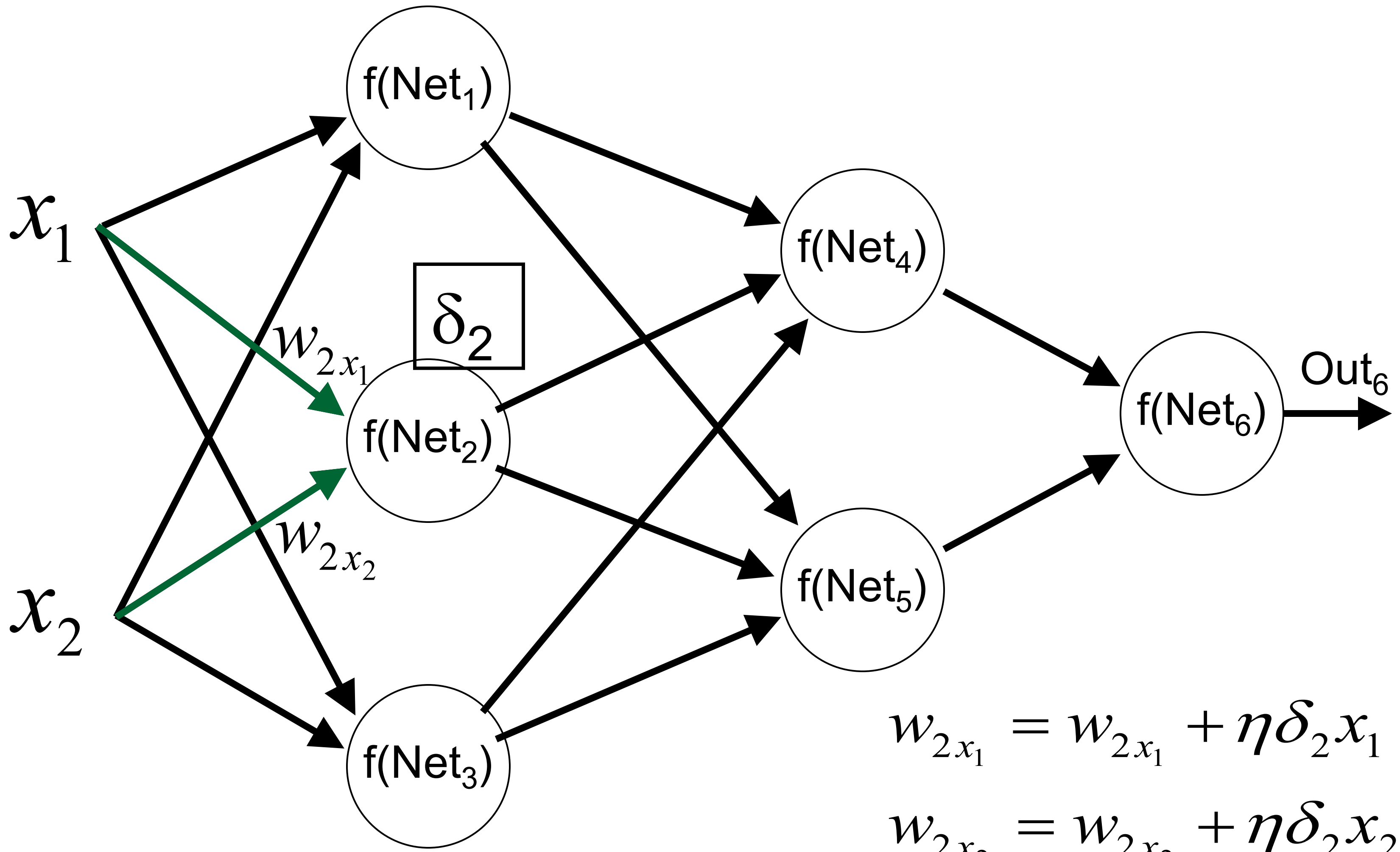
BP illustration – Weight update (1)



$$w_{1x_1} = w_{1x_1} + \eta\delta_1 x_1$$

$$w_{1x_2} = w_{1x_2} + \eta\delta_1 x_2$$

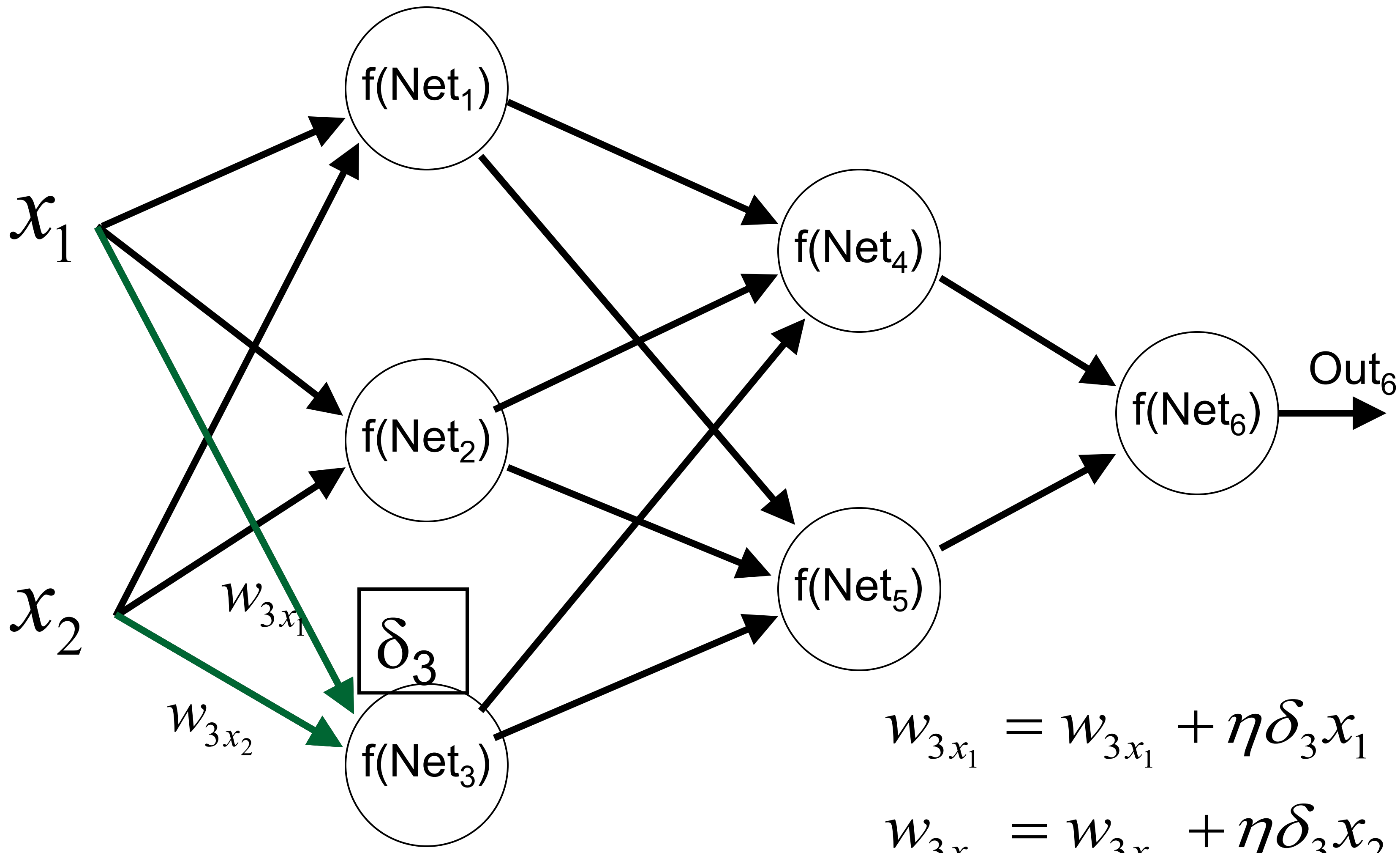
BP illustration – Weight update (2)



$$w_{2x_1} = w_{2x_1} + \eta\delta_2 x_1$$

$$w_{2x_2} = w_{2x_2} + \eta\delta_2 x_2$$

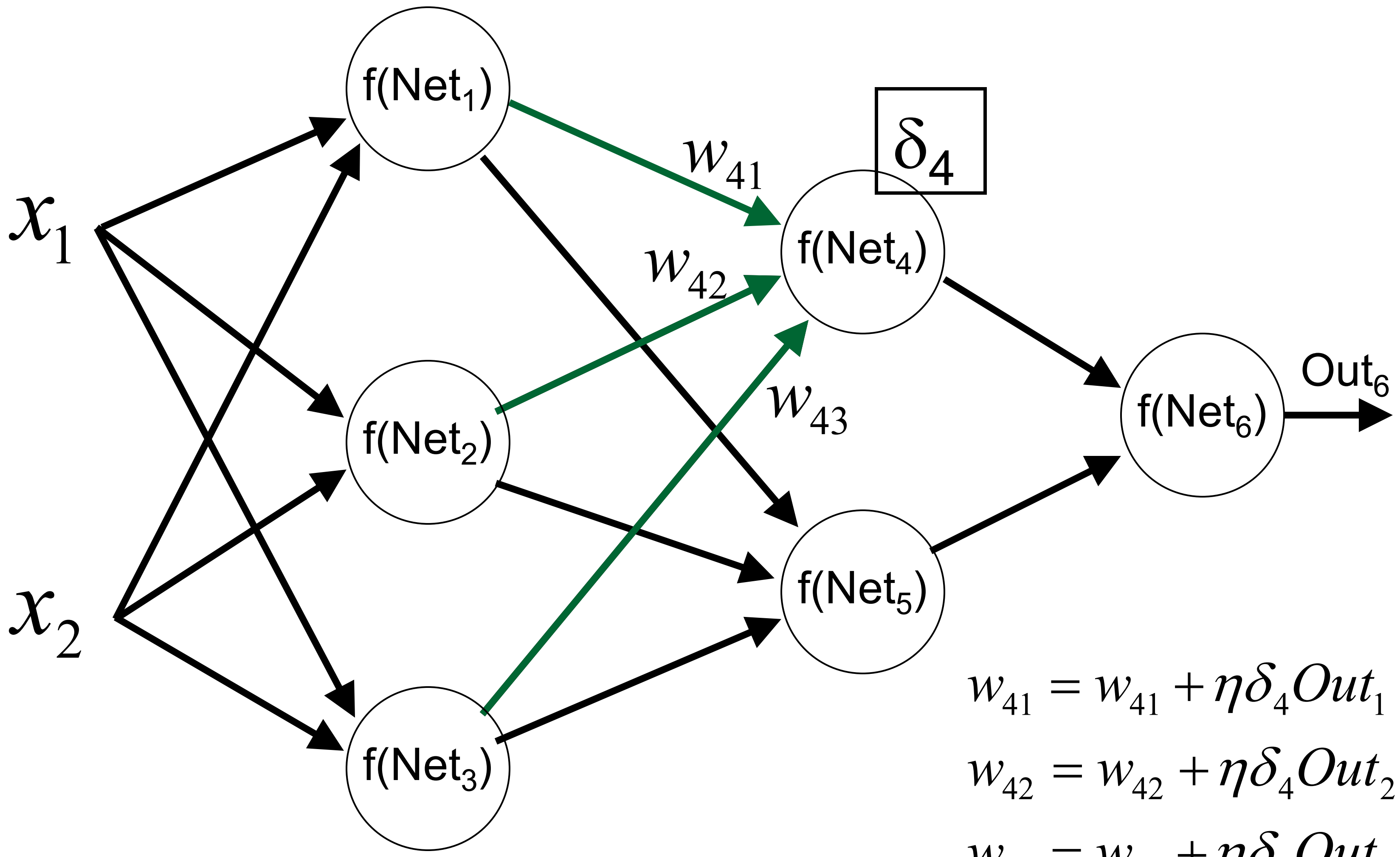
BP illustration – Weight update (3)



$$w_{3x_1} = w_{3x_1} + \eta\delta_3 x_1$$

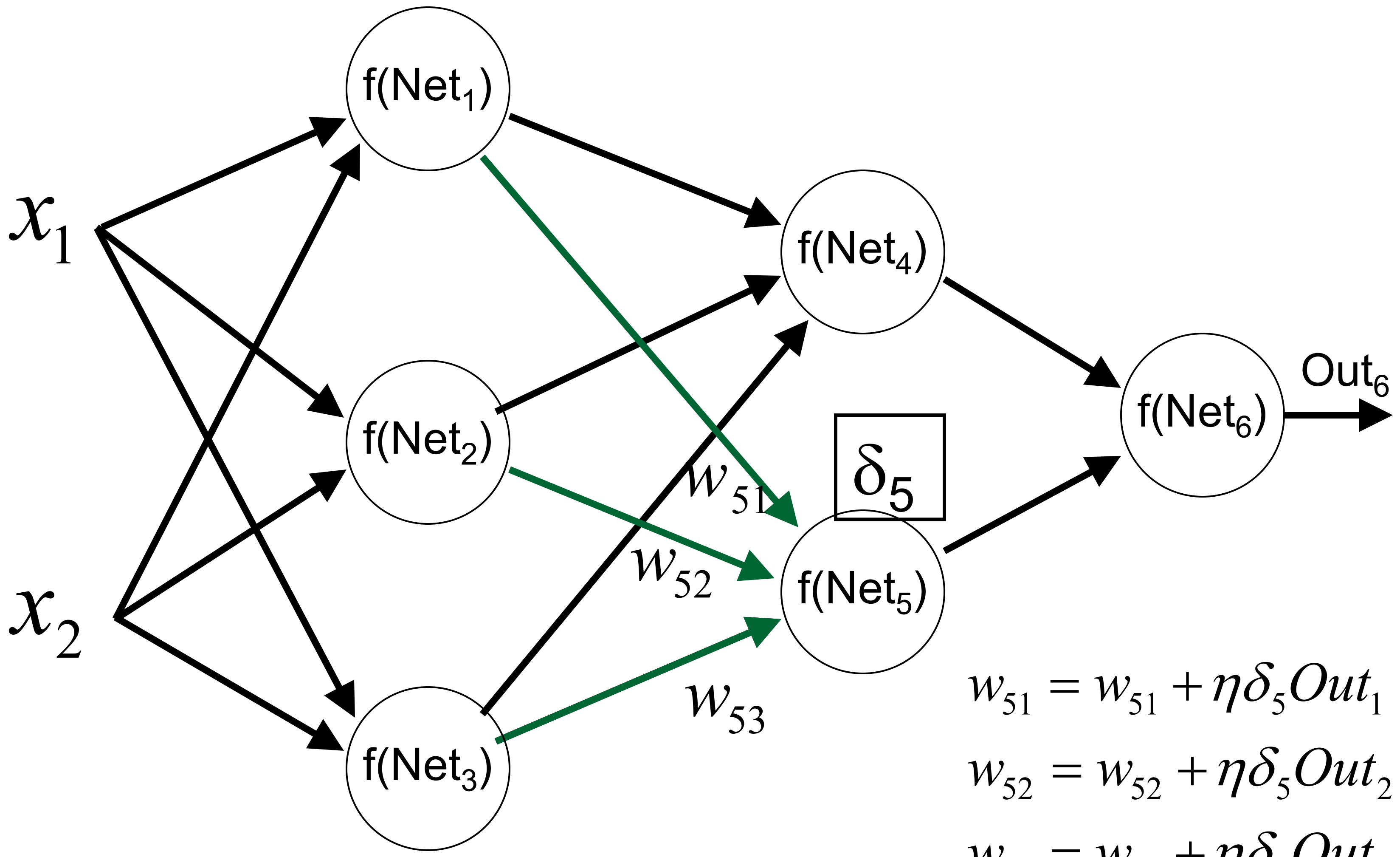
$$w_{3x_2} = w_{3x_2} + \eta\delta_3 x_2$$

BP illustration – Weight update (4)



$$w_{41} = w_{41} + \eta\delta_4 Out_1$$
$$w_{42} = w_{42} + \eta\delta_4 Out_2$$
$$w_{43} = w_{43} + \eta\delta_4 Out_3$$

BP illustration – Weight update (5)

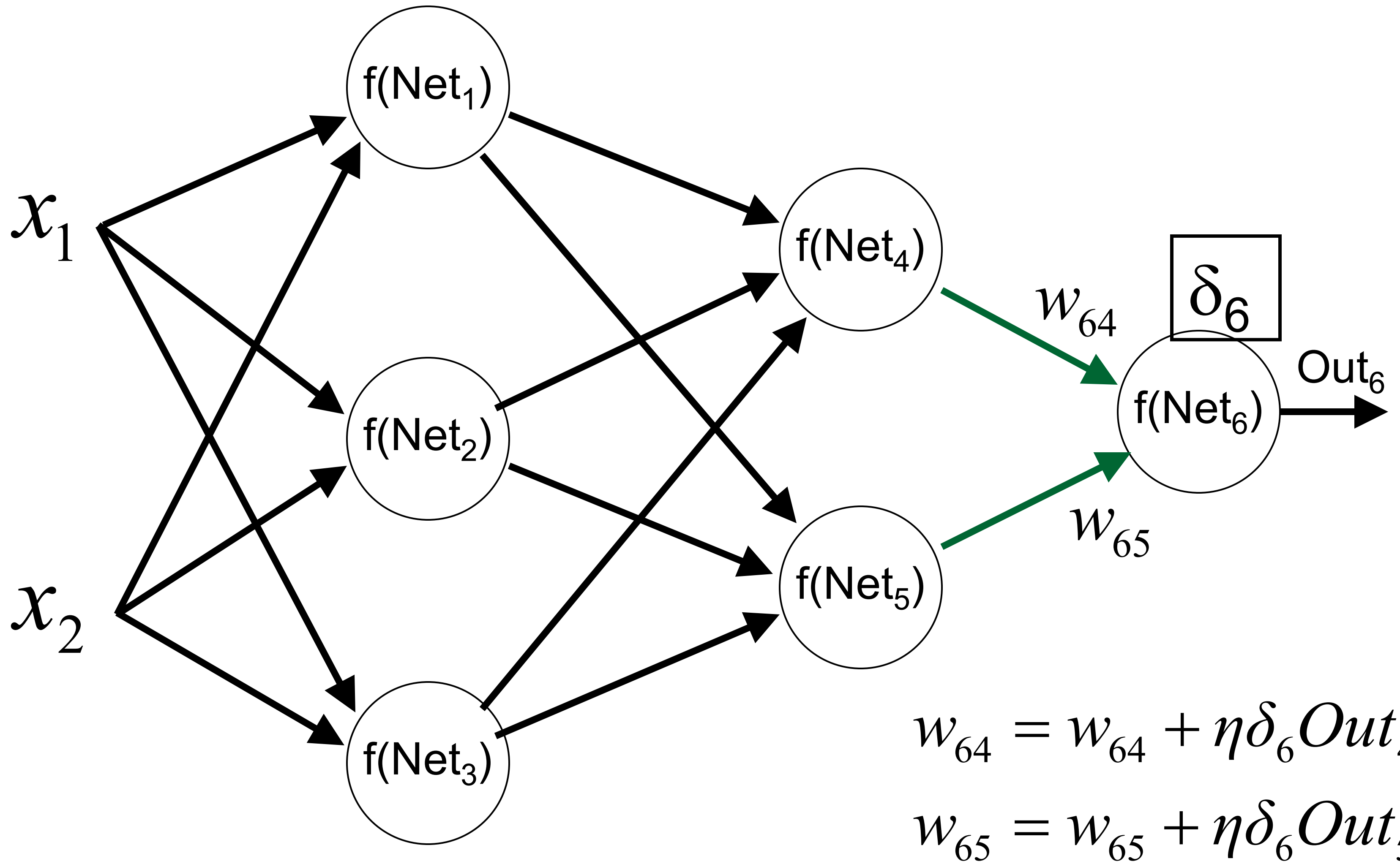


$$w_{51} = w_{51} + \eta\delta_5 Out_1$$

$$w_{52} = w_{52} + \eta\delta_5 Out_2$$

$$w_{53} = w_{53} + \eta\delta_5 Out_3$$

BP illustration – Weight update (6)



$$w_{64} = w_{64} + \eta\delta_6 Out_4$$
$$w_{65} = w_{65} + \eta\delta_6 Out_5$$

Advantages vs. Disadvantages

- Advantages
  - Massively parallel in nature
  - Fault (noise) tolerant because of parallelism
  - Can be designed to be adaptive
- Disadvantages
  - No clear rules or design guidelines for arbitrary applications
  - No general way to assess the internal operation of the network (therefore, an ANN system is seen as a “black-box”)
  - Difficult to predict future network performance (generalization)

When using ANNs?

- Input is high-dimensional discrete or real-valued
- The target function is real-valued, discrete-valued or vector-valued
- Possibly noisy data
- The form of the target function is unknown
- Human readability of result is not (very) important
- Long training time is accepted
- Short classification/prediction time is required

Reading and suggested exercises

- Chapter 18 (18.1 -> 18.7)
- Exercises 18.1, 18.9, 18.11, 18.12,

Few Good Textbooks

- Shalev-Schwartz & Ben-David. Understanding Machine Learning. Cambridge University Press. 2014.
- Hastie, Tibshirani & Friedman. The Elements of Statistical Learning: Data Mining, Inference, and Prediction, 2nd Edition. Springer. 2009.
- Bishop. Pattern Recognition and Machine Learning. Springer. 2007.
- Goodfellow, Bengio &Courville. Deep Learning. MIT Press. 2016.
- A. Burkov, The Hundred-Page Machine Learning Book, 2019