# **Supervised Learning (Part 1)** GE 461, Spring 202)

Slide credits: Çiğdem Gündüz Demir

## Learning

- In our lives, we take actions based on
  - What we observe in our environments
  - What we have previously learned

Face recognition Handwritten character recognition Chess playing Car driving Stock price prediction

- In order to achieve a task, we should
  - Have relevant information representing the environment
  - Know the possible set of actions
  - Know the process to take an action based on the information
    - This process relies on our past experience



## Handwritten Letter Recognition

- Obtain information representing the environment
  - Letter to be recognized
  - Preferably its adjacent letters
- Know the possible set of actions
  - Number of letters
  - Language
- Take an action, which is affected by whether or not
  - You have seen that letter before
  - You know the alphabet of that language
  - You understand the context of that language

## Machine Learning

- The goal of machine learning is to design systems that
  - Automatically achieves tasks (output) similar to us
  - Depending on the environment (input)
  - Based on the past experience (training samples)
  - With respect to some performance measures (e.g., accuracy)



#### Supervised learning:

- There is a teacher providing a label (output) for each training sample
- The task is to map an input space to an output space

#### <u>Unsupervised learning:</u>

- There is no explicit teacher that provides sample labels (outputs)
- The task is to find regularities (clusters) in the input space

## Input/Output

- We reduce the input measuring its certain properties (features), which can be numerical or non-numerical
  - Mileage (e.g., 34187)
  - Condition (e.g., poor, average, excellent)
- Deep learning aims at learning these features directly from data
- The output can be discrete or continuous
  - A, C, Z for letter recognition (**classification**)
  - 25999 TL for car price prediction (**regression**)

#### Supervised Learning

- We believe that there is a process underlying training samples (their inputs and outputs)
  - We may not identify this process completely
  - But we can construct a model approximating the process
    - A function that distinguishes discrete outputs (classification)
    - A functional description of output in terms of inputs (**regression**)

#### - Supervised learning focuses on constructing such models



## Supervised Learning



Deep learning aims at learning the features directly from data while learning the model (thus, it combines these two steps)

#### The goodness of a model depends on

- How well its approximation is
  - No model fits all problems
  - Different models have different assumptions
- How well training samples represent the real-world
  - There may exist noise and exceptions in the samples
  - Some parts may not be covered by the samples

### Model Evaluation

#### Accuracy:

- Percentage of correctly classified samples
- We may also want to consider class-based accuracies, especially when there is an unbalanced distribution among classes

#### Confusion matrix:



#### Do not use the same set of (training) samples both for learning a model and its evaluation!!!

- If available, use an independent test set
- If not, create multiple independent training and test sets by partitioning samples many times
  - ✓ Bootstrapping (draw samples with replacement)
  - ✓ K-fold cross-validation (split samples into k folds)
  - ✓ Leave-one-out (form partitions, each contains a single sample)

- It is the fundamental statistical approach in classification
- Here it is assumed that
  - 1. The decision problem is posed in probabilistic terms and
  - 2. All relevant probability values are known

- A simple decision problem: Fish classification
- Let's assume that a fish emerges nature in one of the following states

**State of nature:**  $C = \begin{cases} C_1 & \text{for hamsi} \\ C_2 & \text{for barbun} \end{cases}$ 





 To predict what type will emerge next, we consider C as a random variable, which is described probabilistically

**Prior probabilities (a priori probabilities):**  $P(C_1)$  and  $P(C_2)$  reflect our previous knowledge before the fish appears

 $P(C_1) + P(C_2) = 1$  (if no other species exist)

- Decide whether a fish is hamsi or barbun when
  - 1. We are not allowed to see the fish
  - 2. We know the prior probabilities
  - 3. The cost is the same for all incorrect decisions





**Decision rule:** Select  $\begin{cases} hamsi & \text{if } P(C_1) > P(C_2) \\ barbun & \text{otherwise} \end{cases}$ 

In this case, we always make the same decision!!!

- Fortunately, we usually have more information for making our decisions
  - E.g., we can see the fish, measure its color intensity
  - We make this measurement relying on the fact that hamsi and barbun emerge nature in different colors





 This difference can be expressed in probabilistic terms, considering color intensity x as a continuous random var, whose distribution depends on the state of nature

**Class-conditional probability density functions (likelihoods):**  $P(x|C_1)$  and  $P(x|C_2)$  are the probability of observing color intensity x when the state of nature is  $C_1$  and  $C_2$ , respectively

 Now let's combine this measurement with our previous knowledge

Joint probability  $P(C_j, x) = P(C_j | x) \cdot P(x) = P(x | C_j) \cdot P(C_j)$ 







**Posterior probabilities (a posteriori probabilities):**  $P(C_1|x)$  and  $P(C_2|x)$  reflect our beliefs of having a particular fish species when the color intensity of the fish is measured as x

- Now decide whether a fish is hamsi or barbun when
  - 1. We can see the fish and measure its color x
  - 2. We know the prior probabilities and likelihoods
  - 3. The cost is the same for all incorrect decisions

**Decision rule:** Select  $\begin{cases} hamsi & \text{if } P(C_1 \mid x) > P(C_2 \mid x) \\ barbun & \text{otherwise} \end{cases}$ 

We use the Bayes' decision rule to minimize the probability of error

$$P(error) = \int_{-\infty}^{\infty} P(error, x) \, dx = \int_{-\infty}^{\infty} P(error \mid x) \, P(x) \, dx$$

For every x, keep P(error|x) as small as possible, by selecting the state of nature (class) with the highest posterior probability

- Now decide whether a fish is hamsi or barbun when
  - 1. We can see the fish and measure its color x
  - 2. We know the prior probabilities and likelihoods
  - The cost is the same for all incorrect decisions 3.

**Decision rule:** Select  $\begin{cases} hamsi & \text{if } P(C_1 \mid x) > P(C_2 \mid x) \\ harbun & \text{otherwise} \end{cases}$ Select  $\begin{cases} hamsi & \text{if } P(x \mid C_1) \cdot P(C_1) > P(x \mid C_2) \cdot P(C_2) \\ barbun & \text{otherwise} \end{cases}$ 

- Evidence is unimportant since it is the same for all states of nature ٠
- *Equal priors*  $\rightarrow$  Observing each state of nature is equally likely
- *Equal likelihoods*  $\rightarrow$  Measurement x gives no information

Now let's generalize the decision problem

States of nature $\{C_1, C_2, \dots, C_c\}$ Possible actions $\{\alpha_1, \alpha_2, \dots, \alpha_a\}$ Loss function $\lambda(\alpha_i | C_j)$ 

Let  $x \in \mathbb{R}^d$  be a feature vector in a d - dimensional space For this x, we would take the action  $\alpha_i$  that minimizes the loss  $\lambda(\alpha_i | C_j)$  if we knew  $C_j$  is its true state of nature

However, we do not know the true state of nature **Thus, we will take the action based on expectation** 

• The expected loss associated with taking action  $\alpha_i$ 

$$\underbrace{R(\alpha_i \mid x)}_{Conditional} = \sum_{j=1}^{C} P(C_j \mid x) \cdot \lambda(\alpha_i \mid C_j)$$

risk

$$P(C_j \mid x) = \frac{P(x \mid C_j) \cdot P(C_j)}{P(x)}$$

We take the action that minimizes the conditional risk

$$\alpha^* = \underset{i}{\operatorname{argmin}} R(\alpha_i \mid x)$$
*Optimal action*

The resulting minimum risk R\* is called *Bayes risk* 

### Minimum Error-Rate Classification

- In multi-class classification
  - Each state of nature is usually associated with a class
  - Each action is usually interpreted as deciding on a class (sometimes other actions —e.g., reject action, are defined)
  - Zero-one loss function is commonly used

 $\lambda(\alpha_i | C_j) = \begin{cases} 0 & \text{if } i = j \quad \text{(correct classification)} \\ 1 & \text{if } i \neq j \quad \text{(all incorrect classifications)} \end{cases}$ 

The optimal action is  $\alpha^* = \underset{i}{\operatorname{arg\,max}} P(C_i | x)$  When zero-one loss function is used, selecting the action that minimizes conditional risk is equivalent to selecting the action that maximizes posterior probability

#### **Classifiers and Discriminant Functions**

- A classifier is represented with a set of discriminant functions g<sub>j</sub>(x) for j = 1, 2, ... C
- A given instance x is then classified with the class C<sub>j</sub> for which the discriminant function g<sub>j</sub>(x) is the maximum
  - Likelihood-based approaches
     Discriminant-based approaches

#### Likelihood-Based Approaches

- They estimate class probabilities on training samples and then use them to define the discriminant functions
- Bayes classifier
  - Defines a discriminant function using the conditional risk



$$= \frac{\hat{P}(x \mid C_j) \cdot \hat{P}(C_j)}{\hat{P}(x)}$$

$$= \hat{P}(x \mid C_j) \cdot \hat{P}(C_j)$$

For each class, estimate the prior and the likelihood on the training samples that belong to this class

## Likelihood-Based Approaches

#### Parametric approach

- Assumes a parametric form on the probability distributions and estimate their parameters on training samples
- For a given instance x, it estimates its class probabilities using these distributions



- Maximum likelihood estimation and Bayesian estimation

#### Nonparametric approach

- Does not have such assumption
- It estimates the class probabilities of the instance x using the nearby points of this instance
- Parzen windows, k-nearest neighbors

## **Discriminant-Based Approaches**

- They learn discriminant functions directly on training samples
- They make an assumption on the form of discriminant functions and learn their parameters on training samples without estimating class probabilities
- Linear discriminants assume that each discriminant function is a linear combination of the input features



They define g<sub>j</sub>(x) as a linear combination of the input features

$$g_{j}(x \mid W_{j}) = \sum_{i=1}^{d} W_{ij} x_{i} + W_{0j}$$
weight vector for

the *j*-th class

let's define  $x_0 = 1$ 

$$g_j(x \mid W_j) = \sum_{i=0}^d W_{ij} x_i$$

- Learning involves learning the parameters (weights) W<sub>j</sub> for each class C<sub>j</sub> from the training samples
- For that, we will define a criterion function and learn the weights that minimize/maximize this function

They yield hyperplane decision boundaries

Consider two-class classification

$$g_{1}(x) = \sum_{i=1}^{d} W_{i1} x_{i} + W_{01}$$

$$g_{1}(x) = W_{1}^{T} x + W_{01}$$

$$g_{2}(x) = W_{2}^{T} x + W_{02}$$

$$g(x) = g_{1}(x) - g_{2}(x) \text{ choose } C_{1} \text{ if } g(x) \ge 0$$

$$C_{2} \text{ otherwise}$$

$$g(x) = (W_1 - W_2)^{\mathsf{T}} x + (W_{01} - W_{02})$$
$$g(x) = W^{\mathsf{T}} x + W_0$$
$$This is another linear function$$

Consider two-class classification



Let's take two points on the decision plane

$$g(x^a) = g(x^b)$$

$$W^{\mathsf{T}}x^a + W_0 = W^{\mathsf{T}}x^b + W_0$$

$$W^{\mathsf{T}}(x^a - x^b) = 0$$

W determines the hyperplane's orientation (W is normal to any vector on the hyperplane)

 $W_0$  determines the hyperplane's location with respect to the origin

We consider multi-class classification as



The most common way to resolve ambiguities is to select the class for which the discriminant is highest (one may also take the reject action for ambiguities)

 $g_3(x)$ 

g<sub>2</sub>(x

#### How to Learn?

Although we will use linear discriminants for classification, let's first consider a linear regression problem



construct a linear model

 $f(x) = Wx + W_0$ 

define a criterion function

$$\underbrace{loss(W, W_0)}_{t} = \frac{1}{2} \sum_{t} (f(x^t) - y^t)^2$$

Sum of squared errors

.

select W and  $W_0$  that minimize this error on the training samples

$$\frac{\partial loss}{\partial W} = 0 \qquad \frac{\partial loss}{\partial W_0} = 0$$

#### **Analytical Solution**

$$loss(W, W_{0}) = \frac{1}{2} \sum_{t} (f(x^{t}) - y^{t})^{2}$$

$$f(x) = Wx + W_{0}$$

$$\frac{\partial loss}{\partial W} = \frac{1}{2} 2 \sum_{t} (Wx^{t} + W_{0} - y^{t}) x^{t} = W \sum_{t} x^{t^{2}} + W_{0} \sum_{t} x^{t} + \sum_{t} x^{t} y^{t} = 0$$

$$\frac{\partial loss}{\partial W_{0}} = \frac{1}{2} 2 \sum_{t} (Wx^{t} + W_{0} - y^{t}) = W \sum_{t} x^{t} + W_{0} N + \sum_{t} y^{t} = 0$$



Sometimes it is hard to analytically solve OR There may be no analytical solution at all (if the linear system has a singular matrix, no solution or multiple solutions exist)



### Gradient Descent Algorithm

- One commonly used iterative optimization method
- Goal is to find the parameters that minimize the loss
  - Starting with random parameters, it iteratively updates them in the direction of the steepest descent (in the opposite direction of the gradient) until the gradient is zero (or small enough)

start with random weights  $W_i$ 

do

$$\Delta W_i = -\eta \frac{\partial loss_{ALL}}{\partial W_i} \quad \text{for all } i$$

$$W_i = W_i + \Delta W_i$$
 for all  $i$ 

until convergence

 $\eta$  is the learning rate, which determines how much to move in the direction of the steepest descent

- $\rightarrow$  if it is too small, convergence is slow
- → if it is too large, we may overshoot the minimum (divergence might occur)

This method finds the nearest minimum, which could be local. It does not guarantee to find the global minimum

#### Regression

Let's derive the update rules for regression



$$loss_{ALL}(W) = \sum_{t} loss^{t}$$
$$loss = \frac{1}{2} (f(x) - y)^{2}$$

f(x) = net

$$net = \sum_{i} x_i W_i$$

$$\Delta W_{i} = -\eta \frac{\partial loss_{ALL}(W)}{\partial W_{i}}$$
$$\frac{\partial loss}{\partial W_{i}} = \frac{\partial loss}{\partial net} \frac{\partial net}{\partial W_{i}}$$
$$\frac{\partial loss}{\partial W_{i}} = \delta x_{i}$$
$$\delta = (net - y)$$
$$\Delta W_{i} = -\eta \sum_{t} \delta x_{i}$$
$$\Delta W_{i} = \eta \sum_{t} (y^{t} - net^{t}) x^{t}_{i}$$

## Classification (Logistic Regression)

Let's derive the update rules for 2-class classification





## Classification (Logistic Regression)

Let's derive the update rules for 2-class classification



$$loss_{ALL}(W) = \sum_{t} loss^{t}$$

$$\underbrace{loss = -y \log f(x)}_{\text{Cross entropy}} - (1 - y) \log (1 - f(x))$$

$$loss = \frac{1}{2} \left( f(x) - y \right)^2$$

Squared error

$$f(x) = \sigma(net)$$

$$net = \sum_{i} x_{i} W_{i}$$

$$\frac{\partial loss}{\partial W_{i}} = \frac{\partial loss}{\partial net} \quad \frac{\partial net}{\partial W_{i}}$$

$$\frac{\partial loss}{\partial W_{i}} = \delta x_{i}$$

When squared error is used  $\delta = (\sigma(net) - y) \sigma'(net)$ 

$$\Delta W_i = \eta \sum_t \left( y^t - \sigma(net^t) \right) \sigma(net^t) \left( 1 - \sigma(net^t) \right) x_i^t$$

#### Classification

Let's derive the update rules for multiclass classification



#### Classification

Let's derive the update rules for multiclass classification



$$loss_{ALL}(W) = \sum_{t} loss^{t}$$

$$loss = -\sum_{k} y_{k} \log f_{k}(x)$$

Cross entropy

$$loss = \frac{1}{2} \sum_{k} (f_k(x) - y_k)^2$$
  
Squared error

$$f_{j}(x) = softmax(net_{j})$$

$$net_{j} = \sum_{i} x_{i} W_{ij}$$

$$\frac{\partial loss}{\partial W_{ij}} = \frac{\partial loss}{\partial net_{j}} \frac{\partial net_{j}}{\partial W_{ij}}$$

$$\frac{\partial loss}{\partial W_{ij}} = \delta_{j} x_{i}$$

When squared error is used

$$\delta_{j} = \sum_{k} \left( softmax \left( net_{k} \right) - y_{k} \right) \frac{\partial softmax \left( net_{k} \right)}{\partial net_{j}}$$

$$\Delta W_{ij} = \eta \sum_{t} \sum_{k} \left( y_k^t - s(net_k^t) \right) s(net_j^t) \left( \delta_{jk} - s(net_k^t) \right) x_i^t$$

 $softmax(net_k^t)$ 

start with random weights 
$$W_{ij}$$
  
do  

$$\int_{0}^{0} (compute f_{j}(x') = softmax \left(\sum_{i} x'_{i} W_{ij}\right) \quad \text{for all } t \text{ and } j$$
  
indicate  $W_{ij} = -\eta \sum_{i} \delta_{j} x_{i}^{i} \quad \text{for all } i \text{ and } j$   
indicate  $W_{ij} = W_{ij} + \Delta W_{ij} \quad \text{for all } i \text{ and } j$   
indicate  $W_{ij} = W_{ij} + \Delta W_{ij} \quad \text{for all } i \text{ and } j$   
indicate  $W_{ij} = W_{ij} + \Delta W_{ij} \quad \text{for all } i \text{ and } j$   
indicate  $W_{ij} = W_{ij} + \Delta W_{ij} \quad \text{for all } i \text{ and } j$   
indicate  $W_{ij} = M_{ij} + \Delta W_{ij} \quad \text{for all } i \text{ and } j$   
indicate  $W_{ij} = -\eta \delta_{j} x_{i}^{i} \quad \text{for all } i \text{ and } j$   
indicate  $W_{ij} = -\eta \delta_{j} x_{i}^{i} \quad \text{for all } i \text{ and } j$   
indicate  $W_{ij} = W_{ij} + \Delta W_{ij} \quad \text{for all } i \text{ and } j$   
indicate  $W_{ij} = W_{ij} + \Delta W_{ij} \quad \text{for all } i \text{ and } j$   
indicate  $W_{ij} = W_{ij} + \Delta W_{ij} \quad \text{for all } i \text{ and } j$   
indicate  $W_{ij} = W_{ij} + \Delta W_{ij} \quad \text{for all } i \text{ and } j$   
indicate  $W_{ij} = W_{ij} + \Delta W_{ij} \quad \text{for all } i \text{ and } j$   
indicate  $W_{ij} = W_{ij} + \Delta W_{ij} \quad \text{for all } i \text{ and } j$   
indicate  $W_{ij} = W_{ij} + \Delta W_{ij} \quad \text{for all } i \text{ and } j$   
indicate  $W_{ij} = W_{ij} + \Delta W_{ij} \quad \text{for all } i \text{ and } j$   
indicate  $W_{ij} = W_{ij} + \Delta W_{ij} \quad \text{for all } i \text{ and } j$   
indicate  $W_{ij} = W_{ij} + \Delta W_{ij} \quad \text{for all } i \text{ and } j$   
indicate  $W_{ij} = W_{ij} + \Delta W_{ij} \quad \text{for all } i \text{ and } j$   
indicate  $W_{ij} = W_{ij} + \Delta W_{ij} \quad \text{for all } i \text{ and } j$   
indicate  $W_{ij} = W_{ij} + \Delta W_{ij} \quad \text{for all } i \text{ and } j$   
indicate  $W_{ij} = W_{ij} + \Delta W_{ij} \quad \text{for all } i \text{ and } j$   
indicate  $W_{ij} = W_{ij} + \Delta W_{ij} \quad \text{for all } i \text{ and } j$   
indicate  $W_{ij} = W_{ij} + \Delta W_{ij} \quad \text{for all } i \text{ and } j$   
indicate  $W_{ij} = W_{ij} + \Delta W_{ij} \quad \text{for all } i \text{ and } j$   
indicate  $W_{ij} = W_{ij} + \Delta W_{ij} \quad \text{for all } i \text{ and } j$   
indicate  $W_{ij} = W_{ij} + \Delta W_{ij} \quad \text{for all } i \text{ and } j$   
indicate  $W_{ij} = W_{ij} + \Delta W_{ij} \quad \text{for all } i \text{ and } j$   
indicate  $W_{ij} = W_{ij} + \Delta W_{ij} \quad \text{for all } i \text{$ 

## Adding Nonlinearity

- Linear discriminants yield hyperplane decision boundaries
- If they are not sufficient to construct a "good" model
  - Transform the space into a new one using nonlinear mappings and construct linear discriminants on the transformed space → Support vector machines
  - Learn nonlinearity at the same time as you learn the linear discriminants → Neural networks

#### **XOR Problem**



Support vector machines use the idea of nonlinear mapping to find a linearly separable space

#### **XOR Problem**



Neural networks learn the nonlinearity at the same time as they learn the linear discriminants (learn all the weights at the same time)

# **Neural Networks**

### **Multilayer Perceptrons**

Also contain hidden layers in addition to input and output layers



Hidden units  $h_j$ 's can be viewed as new "features" obtained by combining  $x_i$ 's

A deeper architecture with nonlinear activations is more expressive than a shallow one In this network

- 1. Each hidden unit computes its net activation  $net_j = \sum x_i W_{ij}$
- 2. Each hidden unit emits an output that is a nonlinear function of its activation  $h_i = \sigma(net_i)$
- 3. Each output unit computes its net activation  $net_{k} = \sum_{j} h_{j} W_{jk}$
- 4. Each output units emits an output  $y_k = g(net_k)$

#### How to Learn?

- In linear discriminants, we select the weights to minimize a loss function defined on the difference between the actual and computed outputs
- In multilayer structures, we can also select the hiddento-output-layer weights to minimize a loss function defined on the actual and computed outputs
- However, we cannot select the input-to-hidden-layer weights in a similar way since we do not know the actual values of the hidden units
- Thus, to learn the input-to-hidden-layer weights, we propagate the loss function (defined on the outputs) from the output layer to the corresponding hidden layer
   → BACKPROPAGATION ALGORITHM

#### **Backpropagation Algorithm**

Let's derive the update rules for multiclass classification

 $net_{j} = \sum_{i} x_{i} W_{ij}$   $h_{j} = \sigma(net_{j})$   $net_{k} = \sum_{j} h_{j} W_{jk}$   $f_{k}(x) = softmax(net_{k})$  $loss_{ALL}(W) = \sum_{t} loss^{t}$  $loss = \frac{1}{2} \sum_{k} \left( f_{k}(x) - y_{k} \right)^{2}$ Squared error  $\Delta W_{jk} = -\eta \frac{\partial loss_{ALL}(W)}{\partial W_{jk}}$  $\frac{\partial loss}{\partial W_{jk}} = \frac{\partial loss}{\partial net_k} \quad \frac{\partial net_k}{\partial W_{jk}}$  $\frac{\partial loss}{\partial W_{jk}} = \delta_k h_j$  $\delta_k = \sum_{m} \left( softmax \left( net_m \right) - y_m \right) \frac{\partial softmax \left( net_m \right)}{\partial net_k}$ 

Hidden-to-outputlayer weights

#### **Backpropagation Algorithm**

Let's derive the update rules for multiclass classification

 $net_{j} = \sum_{i} x_{i} W_{ij}$   $h_{j} = \sigma(net_{j})$   $net_{k} = \sum_{j} h_{j} W_{jk}$  $loss_{ALL}(W) = \sum_{t} loss^{t}$  $loss = \frac{1}{2} \sum_{k} (f_{k}(x) - y_{k})^{2}$ Squared error  $f_k(x) = softmax(net_k)$  $\Delta W_{ij} = -\eta \frac{\partial loss_{ALL}(W)}{\partial W_{ii}}$ Input-to-hidden  $\begin{aligned} \frac{\partial loss}{\partial W_{ij}} &= \frac{\partial loss}{\partial net_j} \quad \frac{\partial net_j}{\partial W_{ij}} \\ \frac{\partial loss}{\partial W_{ij}} &= \delta_j x_i \\ \delta_j &= \sum_k \frac{\partial loss}{\partial net_k} \quad \frac{\partial net_k}{\partial net_j} = \left[\sum_k \delta_k W_{jk}\right] \sigma'(net_j) \end{aligned}$ layer weights

#### More Hidden Layers



$$\frac{\partial loss}{\partial W_{ij}} = \frac{\partial loss}{\partial net_j} \frac{\partial net_j}{\partial W_{ij}}$$
$$\frac{\partial loss}{\partial W_{ij}} = \delta_j x_i$$
$$\delta_j = \sum_{(j+1)} \frac{\partial loss}{\partial net_{(j+1)}} \frac{\partial net_{(j+1)}}{\partial net_j}$$
$$\delta_j = \left[\sum_{(j+1)} \delta_{(j+1)} W_{j(j+1)}\right] \sigma'(net_j)$$

 $\delta_j$  may vanish after repeated multiplication. This makes deep architectures hard to train (when initial weights are not "good" enough)

#### Approaches for alleviating underfitting and overfitting problems

- Better network designs: Sparse connections, weight sharing, convolutional nets, long/short skip connections, activation functions, …
- Better network training: regularization, loss function definitions, larger datasets, data augmentation, ...
- Previously, layerwise pretraining (restricted Boltzmann machines, autoencoders)

# Network Topology

#### The number of hidden units and hidden layers

- It controls the expressive power of the network
- Thus, the complexity of the decision boundary
- No foolproof method to set them before training
  - Few hidden units/layers will be enough if samples are well-separated
  - More will be necessary if samples have complicated densities



#### Hidden units more than necessary

- Network is tuned to the particular training set (overfitting)
- Training error can become small, but test error is unacceptably high

#### Too few hidden units

- Network does not have enough free parameters to fit the training set well
- Training and test errors are high

#### You will see different deep network architectures next week!!!

#### Some Practical Issues

Commonly used activation functions



Logarithmic sigmoid function



When to stop?



Validation data can be used:

- Training error ultimately reaches an asymptotic value
- The error on an independent test set is expected to be higher
  - Although it usually decreases, it can also increase or oscillate

#### Some Practical Issues

- 1. Unbalanced class distributions
  - Classifiers typically favor the majority class(es)
- 2. Small training sets
  - Data augmentation is typically useful
- 3. Features with different orders of magnitudes
  - A neural network adjusts weights in favor of features with higher magnitudes
  - Normalization/scaling is typically useful
- 4. High feature values
  - May cause the exploding gradient problem
  - Normalization/scaling is typically useful

Many of them are indeed issues not only for neural networks but also for many other classifiers

#### **Some Practical Issues**

<u>Regularization</u> reduces sensitivity to training samples and decreases the risk of overfitting  $1 \sum \frac{1}{2} \sum \frac{1}{$ 

$$loss_{ALL}(W) = \frac{1}{T} \sum_{t} loss^{t} + ||W||$$
  

$$loss_{ALL}(W) = \frac{1}{2T} \sum_{t} \sum_{k} \left( f_{k}(x^{t}) - y_{k}^{t} \right)^{2} + \frac{\lambda}{2T} ||W||_{2}^{2} \quad \text{where } ||W||_{2}^{2} = \sum_{i} W_{i}^{2}$$
  
Mean squared error L2-regularization term

#### Dropout regularization

- During training, in each iteration, randomly drop out units (also their incoming and outgoing connections) with probability p to sample a "thinned" network and train it
- Training can be seen as training a collection of different thinned networks with extensive weight sharing
- In testing, consider the entire network where the weights are scaled down by multiplying them a factor of 1 – p