MULTI-LAYER PERCEPTRON

INT301 Bio-computation, Week 4, 2025

Multilayer Perceptrons

- The **multilayer perceptron** (**MLP**) is a hierarchical structure of several perceptrons, which overcomes the shortcomings of the single-layer networks.

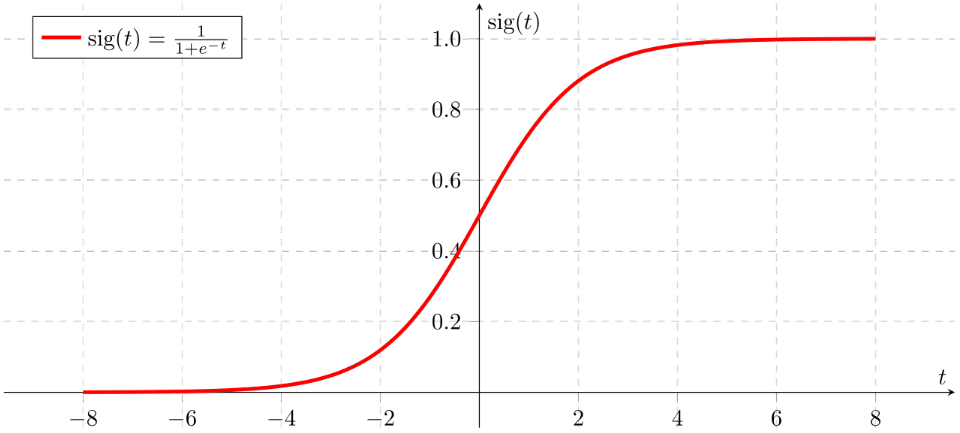
  缺点

- The MLP neural network is able to learn nonlinear function mappings.
  - learning a rich variety of nonlinear decision surfaces.
- Nonlinear functions can be represented by MLPs with units that use nonlinear activation functions.
  - Multiple layers of cascaded linear units still produce only linear mappings!

Differentiable Activation Functions

- Training algorithms for MLP require *differentiable, continuous nonlinear activation functions*.
- Such a function is the *sigmoid function*:

$$o = \sigma(s) = \frac{1}{1+e^{-s}}$$



where *s* is the sum:

$$s = \sum_{i=0}^{d} w_i x_i$$

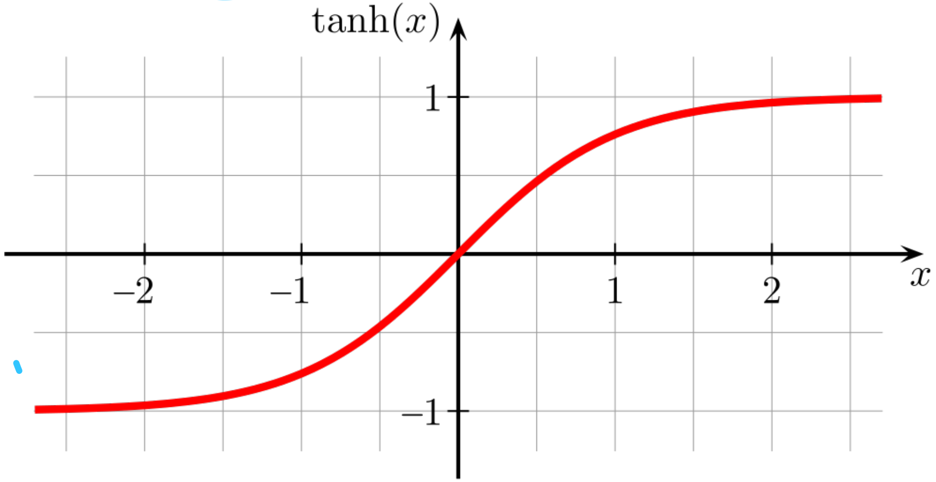
which is the products from the weights w_i and the inputs x_i.

Differentiable Activation Functions

- Another nonlinear function often used in practice is the *hyperbolic tangent*: 双曲正切函数

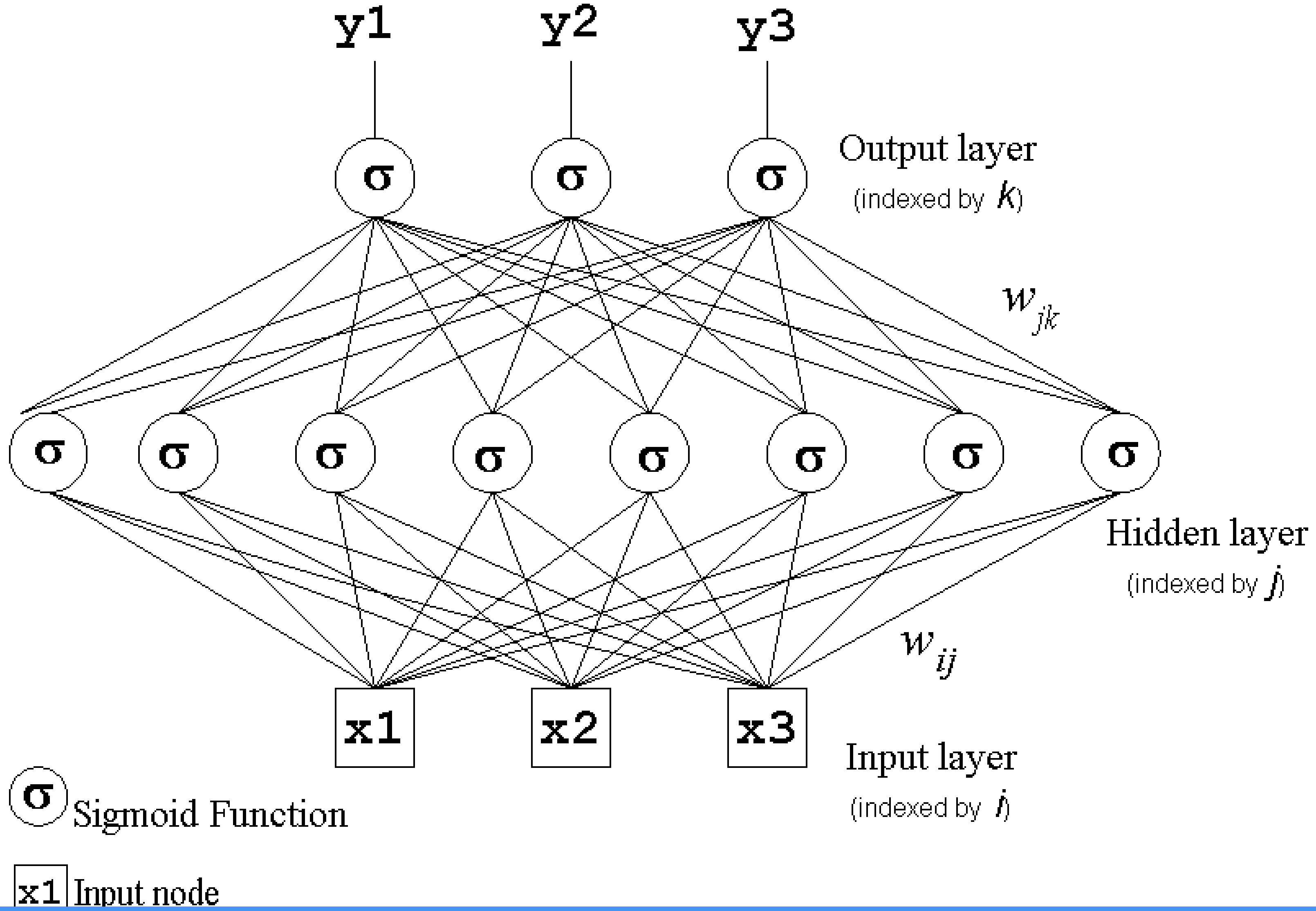
$$o = \tanh(s) = \frac{e^{s} - e^{-s}}{e^{s} + e^{-s}}$$

$$\tanh'(s) = 1 - \tanh^2(s)$$



- The mean of tanh is 0

Multilayer Network Structure



Multilayer Network Structure

- A two-layer neural network implements the function:

$$f(x) = \sigma\Big(\sum_{j=1}^{J} w_{jk} \boxed{\sigma\Big(\sum_{i=1}^{I} w_{ij} x_i + w_{oj}\Big)} + w_{ok}\Big)$$

Output from hidden layer

where: **x** is the input vector,

w_{0j} and w_{0k} are the bias terms, *

w_{ij} are the weights connecting the input with the hidden nodes

w_{jk} are the weights connecting the hidden with output nodes

σ is the sigmoid activation function.

Multilayer Network Structure

- The hidden units enable the multilayer network to learn complex tasks by extracting *progressively more meaningful information* from the input examples.
- The MLP has a highly connected topology since every input is connected to all nodes in the first hidden layer, every unit in the hidden layers is connected to all nodes in the next layer, and so on.
- The input signals, initially these are the input examples, propagate through the neural network in a forward direction on a layer-by-layer basis, that is why they are often called *feedforward multilayer networks.*

(前馈) 多层网络

Representation Power of MLP

- Properties concerning the representational power of MLP:
  - ***learning arbitrary functions*: **any function can be learned with an arbitrary accuracy by a two-layer network;
  - ***learning continuous functions*: **every bounded continuous function can be learned with a small error by a two-layer network (the number of hidden units depends on the function to be approximated);
  - ***learning Boolean functions*: **every Boolean function can be learned exactly by a two-layer network although the number of hidden units grows exponentially with the

Backpropagation Learning Algorithm

- MLP became applicable on practical tasks after the discovery of a supervised training algorithm, the ***error backpropagation learning algorithm***.

- The error backpropagation algorithm includes two passes through the network:
  - ***forward pass***, and
  - ***backward pass***.

Backpropagation Learning Algorithm

- During the backward pass the weights are adjusted in accordance with the *error correction rule*. The actual network output is subtracted from the given output in the example and the weights are adjusted so as to make the network output close to the desired one.

- The backpropagation algorithm does gradient descent as it moves in direction opposite to the gradient of the error, that is in direction of the steepest decrease of the error.

- This is the direction of most rapid error decrease by varying all the weights simultaneously:

$$\nabla E(w) = [\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \ldots, \frac{\partial E}{\partial w_{n-1}}]$$

Backpropagation Learning Algorithm

$$\nabla E(w) = [\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \ldots\ldots, \frac{\partial E}{\partial w_d}]$$

- By gradient descent search, backpropagation training algorithm minimizes a cost function E (the mean square difference between the desired and actual net outputs).
- The network is trained initially selecting small random weights and then presenting all training data incrementally.
- Weights are adjusted after every trial using side information specifying the correct class until weights converge and the cost function is reduced to an acceptable value

Backpropagation Training Algorithm

- ***Initialization*:** Examples $\{(x_e, y_e)\}_{e=1}^N$, initial weights w_i set to small random values, learning rate η
- ***Repeat***
  - For each training example (x, y)

Forward

- *calculate the outputs* using the sigmoid function:

$$o_j = \sigma(s_j) = \frac{1}{1+e^{-s_j}}, s_j = \sum_{i=0}^{d} w_{ij} o_i$$

at the hidden units j

where $o_i = x_i$

$$o_k = \sigma(s_k) = \frac{1}{1+e^{-s_k}}, s_k = \sum_{i=0}^{d} w_{jk} o_j$$

at the output units k

链式法则证明

Backpropagation Training Algorithm

$$E(W) = \frac{1}{2}\sum_e (y_e - O_e)^2 .$$

$$\frac{\partial E}{\partial W_{jk}} = (y_e - O_e)\frac{\partial(-O_k)}{\partial W_{jk}} = (y_e - O_e)\frac{\partial(-\sigma(s))}{\partial W_{jk}} .$$

$$= (y_k - O_k)\frac{\partial(\sigma(s))}{\partial s}\left(\frac{\partial s}{\partial W_{jk}}\right)$$

$$= (y_k - O_k)\,O_k(1 - O_k)(-O_j)$$

Backward

- compute the **benefit** β_k at the node *k* in the output layer:

$$* \quad \frac{\partial E}{\partial S_k} = \quad \beta_k = o_k(1-o_k)[y_k - o_k]$$

effects from the output nodes

- compute the **changes for weights** *j*→ *k* on connections to nodes in the output layer:

$$\Delta w_{jk} = \eta\beta_k o_j$$

$$\Delta w_{0k} = \eta\beta_k$$

这里的0指bias层由于$O_0 = 1$

effects from the output of the neuron

- compute the **benefit** β_j for the hidden node *j* with the formula:

$$* \quad \frac{\partial E}{\partial S_j} = \quad \beta \; = o \; (1-o \;)[\sum \; \beta \; w \;]$$

effects from multiple nodes in

Backpropagation Training Algorithm

- compute the changes for the weights *i* → *j* on connections to nodes in the hidden layer:

$$\Delta w_{ij} = \eta \beta_j o_i$$

$$\Delta w_{0j} = \eta \beta_j$$

$S_k = \sum_{j=1}^{d} w_{jk} o_j$

之前对特定的"j"

$\frac{\partial S_k}{\partial w_{jk}} = o_j$

现在相当 o_j 全部都在

$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j} \times \frac{\partial o_j}{\partial s_j} \times \frac{\partial s_j}{\partial w_{ij}}$

$= \frac{\partial E}{\partial S_k} \times \frac{\partial S_k}{\partial o_j} \times \frac{\partial o_j}{\partial s_j} \times \frac{\partial s_j}{\partial w_{ij}}$

$= (y_k - o_k)\sigma(S_k)(1-S_k) \times \sum_{j=1}^{d} w_{jk} \times o_j(1-o_j) \times o_i$

Update

update the weights by the computed changes:

$\Rightarrow \frac{\partial S_k}{\partial o_j} = \sum_{j=1}^{d} w_{jk}$

$$w = w + \Delta w$$

$\Rightarrow$ 为什么"+" $\Rightarrow w = w - \eta \frac{\partial E}{\partial w}$

上面都是"$(-o_j)$"

所以要"+"

$\Rightarrow \Delta w_{ij} = o_j(1-o_j) \times \sum_{i=1}^{d} w_{jk} \times \frac{\partial E}{\partial S_k} \times o_i$

until *termination condition is satisfied.*

On-line Training

Revision by example is called *on-line (incremental) learning*.

- **Initialization:** Examples $\{(x_e, y_e)\}_{e=1}^{N}$, initial weights w_i set to small random values, learning rate η
- **Repeat**

  pick a training example (x, y)

  - forward propagate the example and calculate the outputs using the sigmoid function
  - backward propagate the error to calculate the benefits
  - update the weights by the computed changes:

    $$w = w + \Delta w$$

until termination condition is satisfied

Derivation of Backpropagation Algorithm

- The BP training algorithm for MLP is a generalized gradient descent rule, according to which with each training example every weight is updated as:

$$w = w + \Delta w$$

where: $\Delta \mathrm{w} = -\eta \dfrac{\partial \mathrm{E_e}}{\partial \mathrm{w}}$, $\qquad \mathrm{E_e} = \dfrac{1}{2}\sum_k (y_k - o_k)^2$

- The implementation of the generalized gradient descent rule requires to derive an expression for the computation of the derivatives $\partial \mathrm{E_e}/\partial \mathrm{w}$

$$\frac{\partial E_e}{\partial w} = \frac{\partial E_e}{\partial s} \cdot \frac{\partial s}{\partial w}$$

Derivation of Backpropagation Algorithm

- The first part $\partial E_e/\partial s$ reflects the change of the error as a function of the change in the network weighted input to the unit.
- The second part $\partial s/\partial w$ reflects the change in the network weighted input as a function of the change of particular weight *w* to that node.
- Since:

$$\frac{\partial s}{\partial w} = \frac{\partial(\sum_l w_l o_l)}{\partial w} = o$$

- The expression is reduced as follows:

$$\frac{\partial E_e}{\partial w} = \frac{\partial E_e}{\partial s} \cdot o$$

Derivation of Backpropagation Algorithm

- For weights $j \rightarrow k$ on connections to nodes in the output layer:

$$\frac{\partial E_e}{\partial w_{jk}} = \frac{\partial E_e}{\partial s_k} \cdot o_j$$

$$\frac{\partial E_e}{\partial s_k} = \frac{\partial E_e}{\partial o_k} \cdot \frac{\partial o_k}{\partial s_k}$$

$$\frac{\partial E_e}{\partial o_k} = \frac{\partial(\frac{1}{2}\sum_k (y_l - o_l)^2)}{\partial o_k} = \frac{\partial(\frac{1}{2}(y_k - o_k)^2)}{\partial o_k}$$

$$= \frac{1}{2} \cdot 2 \cdot (y_k - o_k) \frac{\partial(y_k - o_k)}{\partial o_k}$$

$$= -(y_k - o_k)$$

$$\frac{\partial o_k}{\ } = \frac{\partial \sigma(s_k)}{\ } = o_k(1 - o_k)$$

Derivation of Backpropagation Algorithm

- Therefore:

$$\frac{\partial E_e}{\partial s_k} = -(y_k - o_k)o_k(1-o_k) \qquad \frac{\partial E_e}{\partial w_{jk}} = \frac{\partial E_e}{\partial s_k}\cdot o_j$$

- Then we substitute:

$$\Delta w_{jk} = -\frac{\partial E_e}{\partial w_{jk}} = \eta \beta_k o_j \qquad \beta_k = (y_k - o_k)o_k(1-o_k)$$

- The gradient descent rule in previous lecture:

$$\Delta w_i = \Delta w_i + \eta(y_e - o_e)\sigma(s)(1-\sigma(s))x_{ie}$$

Derivation of Backpropagation Algorithm

- For weights *i*→*j* on connections to nodes in the hidden layer

$$\frac{\partial E_e}{\partial w_{ij}} = \frac{\partial E_e}{\partial s_j} \cdot o_i$$

- In this case the error depends on the errors committed by all output units:

$$\frac{\partial E_e}{\partial s_j} = \sum_k \boxed{\frac{\partial E_e}{\partial s_k}} \cdot \frac{\partial s_k}{\partial s_j} = \sum_k -\beta_k \cdot \frac{\partial s_k}{\partial s_j}$$

$$= \sum_k -\beta_k \cdot \boxed{\frac{\partial s_k}{\partial o_j}} \cdot \boxed{\frac{\partial o_j}{\partial s_j}}$$

$$= \sum_k (-\beta_k) \cdot w_{jk} \cdot \frac{\partial o_j}{\partial s_j} = \sum_k (-\beta_k) \cdot w_{jk} \cdot o_j(1-o_j)$$

$$\boxed{\frac{\partial E_e}{\partial s_k} = -(y_k - o_k)o_k(1-o_k)}$$

$$\boxed{o_k = \sigma(s_k) = \frac{1}{1+e^{-s_k}}, s_k = \sum_{i=0}^{d} w_{jk} o_j}$$

$$\boxed{o_j = \sigma(s_j) = \frac{1}{1+e^{-s_j}}, s_j = \sum_{i=0}^{d} w_{ij} o_i}$$

Derivation of Backpropagation Algorithm

- For the hidden units:

$$\Delta w_{ij} = \eta \beta_j o_i$$

$$\Delta w_{0j} = \eta \beta_j$$

$$\beta_j = -\frac{\partial E_e}{\partial s_j} = o_j(1-o_j)[\sum_k \beta_k w_{jk}]$$

Note: This analysis was made for a single training pattern, but it can be generalized so that:

$$\frac{\partial E_{total}}{\partial w_{ij}} = \sum_e \frac{\partial E_e}{\partial w_{ij}}$$

Thus, we just need to sum out weight changes over the examples.

全部见前面证明手写

Batch Backpropagation Algorithms

Revision by Epoch

→ 等价（看完所有样本再更新）

- From mathematical point of view the error derivatives should be computed after each epoch, *i.e.*, after all examples in the training set have been processed.
    - This means that the error derivative is taken to be the sum of the error derivatives for all examples.
    - While this revision by epoch may have stronger theoretical motivation, revision by a particular example may yield better results and is more commonly used.
- Revision by epoch is called ***Batch Learning***.

Batch version of the backpropagation algorithm

- **Initialization:**

  Examples $\{(x_e, y_e)\}_{e=1}^{N}$, *initial weights* w_i *set to small random values, learning rate* η

- **Repeat**
  1 - *for each training example (x, y)*
      - *forward propagate the example and calculate the outputs using the sigmoid function* ↗这里 ΔW需要累加!
      - *backward propagate the error to calculate the benefits*
  2 - *after processing all examples update the weights by the computed changes:*

$$w = w + \Delta w$$

Issues of Backpropagation
Presentation order of training samples

Sequential or random presentation

- The epoch is the fundamental unit for training, and the length of training often is measured in terms of epochs.
- During a training epoch with revision after a particular example, the examples can be presented in the same sequential order, or the examples could be presented in a different random order for each epoch.
- The random presentation usually yields better results

Issues of Backpropagation: Initialization

Random initial state

- Unlike many other learning systems, the neural network begin in a random state. The network weights are initialized to some choice of random numbers with a range typically between -0.5 and 0.5 (the inputs are usually normalized to numbers between 0 and 1).
- Even with identical learning conditions, the random initial weights can lead to results that differ from one training session to another.

Issues of Backpropagation: Hidden layer

- While it may be more convenient to specify more than one layer of hidden units, *additional layers do not add representational power* to the discrimination.
  - Two-hidden-layer networks are more powerful, but one-hidden-layer networks may be sufficiently accurate for many tasks encountered in practice.
  - One-hidden layer networks assume faster training.
- A heuristic to start with:

  One hidden layer, with n hidden neurons,

  n=(inputs+output_neurons)/2

Issues of Backpropagation: Stopping Criteria

什么时候停止比如 loss <0.01

- The stopping criteria is checked at the end of each epoch:
  - The error (mean absolute or mean square) at the end of an epoch is below a threshold
    - All training examples are propagated and the mean (absolute or square) error is calculated
    - The threshold is determined heuristically – e.g. 0.01
  - Maximum number of epochs is reached
  - Early stopping using a validation set (explain later)
- It typically takes hundreds or thousands of epochs for an NN to converge

Issues of Backpropagation:

Learning rate

- While it is possible to get excellent fits to training data, the application of backpropagation is fraught with difficulties and pitfalls for the prediction of the performance on independent test data.
- Many choices to be made in applying the gradient descent method.
- The key variations of these choices are :

the learning rate and local minima

the selection of a learning rate is of critical importance in finding the true global minimum of the error distance.

The learning rate and local minima

X

- Backpropagation training with too small a learning rate will make agonizingly slow progress. Too large a learning rate will proceed much faster, but may simply produce oscillations between relatively poor solutions.
- Both of these conditions are generally detectable through experimentation and sampling of results after a fixed number of training epochs.
- Typical values for the learning rate parameter are numbers between 0 and 1:

$$0.05 < \eta < 0.75$$

- One would like to use the largest learning rate that still

Issues of Backpropagation: Momentum

动量

结合梯度下降项和一部分先前权重

- The momentum is to stabilize the weight change using a combination of the gradient decreasing term with a fraction of the previous weight change:

$$\Delta w(t) = -\eta \frac{\partial E_e}{\partial w(t)} + \alpha \Delta w(t-1)$$

where t is the index of the current weight change.

- This gives the system a certain amount of inertia since the weight vector will tend to continue moving in the same direction unless opposed by the gradient term.

Issues of Backpropagation: Momentum

- Momentum term simply makes the following change to the weight update rule, where α is the momentum term:
  - If α=0, this is the same as the regular backpropagation, where the weight update is determined purely by the gradient descent
  - If α=1, the gradient descent is completely ignored, and the update is based on the ‘momentum’, previous weight update rule
  - Typical value for α is generally between 0.6 and 0.9

（1-α）

$$\Delta w(t) = -\eta \frac{\partial E_e}{\partial w(t)} + \alpha \Delta w(t-1)$$

Issues of Backpropagation: Momentum

The momentum has the following effects:

- it smooths the weight changes and suppresses cross-stitching, that is cancels side-to-side oscillations across the error valley;
- when all weight changes are all in the same direction the momentum amplifies the learning rate causing a faster convergence;
- enables to escape from small local minima on the error surface.

Issues of Backpropagation: Generalization & Overfitting

泛化

- Supervised learning – training with finite number of examples of proper behavior: { p_i, t_i}, i=1,…,n
- Based on them the network should be able to *generalize* what it has learned to the total population of examples
- **Overtraining (overfitting):**

➢ the error on the training set is very small but when a new data is presented to the network, the error is high

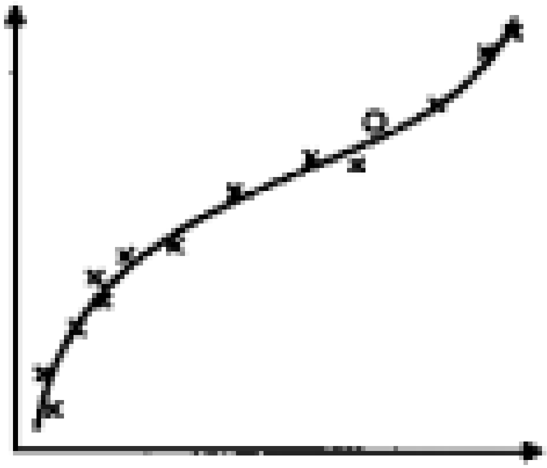
➢ ⇒ the network has memorized the training examples but has not learned to generalize to new situations!

When Does Overfitting Occur?

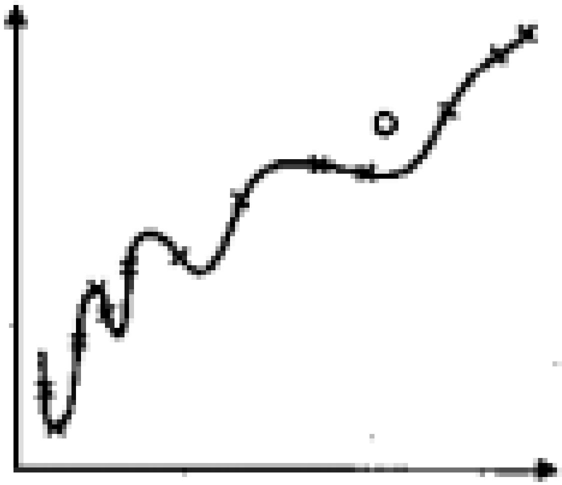
- Training examples are noisy

Example: x- training set, o-testing set

A good fit to noisy data



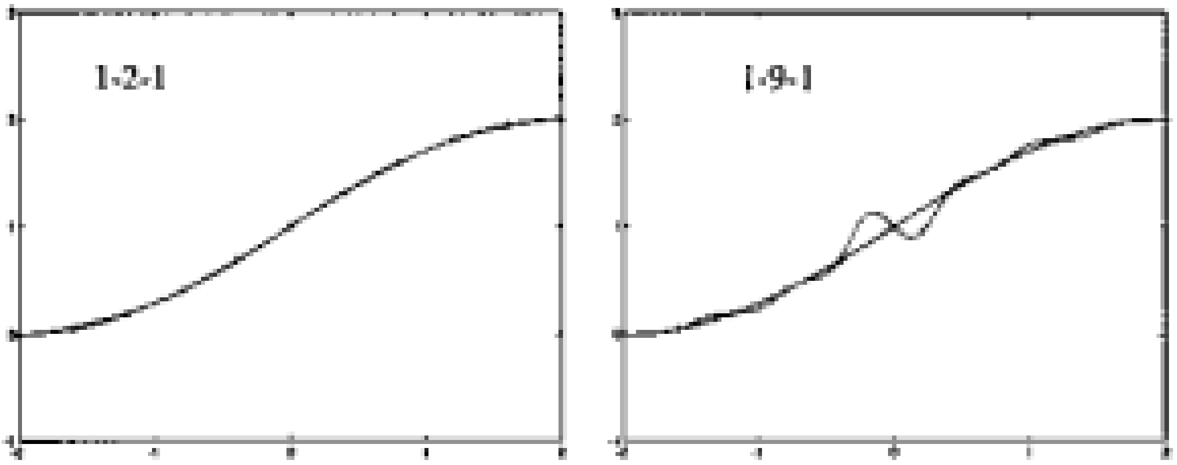
Overfitting



When Does Overfitting Occur? – cont.

- **Common reason:** number of the free parameters is bigger than the number of training examples

$f(x) = 1 + \sin\left(\frac{6\pi}{4}x\right)$ was sampled to create 11 training examples

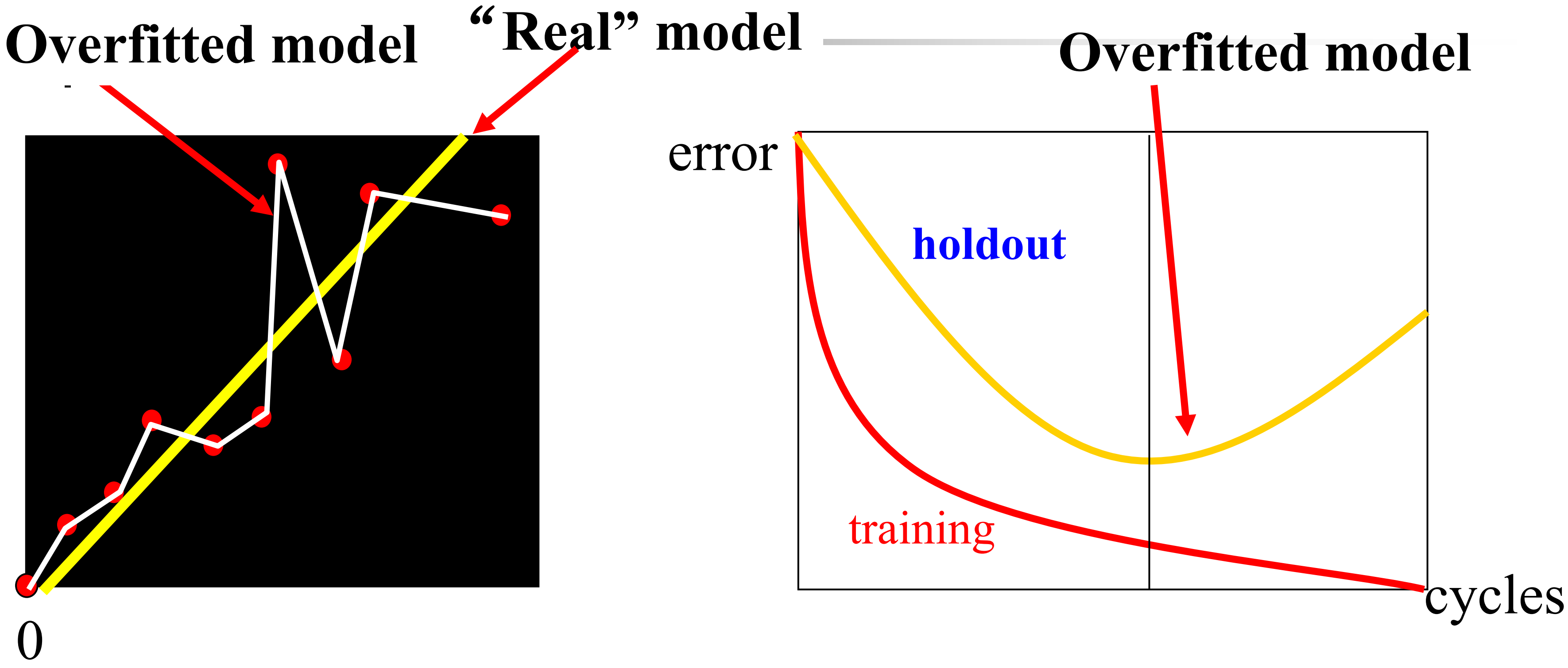


Preventing Overtraining

- Use network that is just large enough to provide an adequate fit
  - Don’t use a bigger network when a smaller one will work
  - The network should not have more free parameters than there are training examples

- However, it is difficult to know beforehand how large a network should be for a specific application

Issues of Backpropagation: Generalization & Overfitting



- With sufficient nodes MLP can classify any training set exactly;
- But it may have poor generalisation ability.
- **Overfitting** may be prevented by ***early stopping***, ***network pruning***, and applying ***regularization techniques***

Techniques to overcome overfitting

①

- *Weight decay* : **Decrease** each weight by some small factor during each iteration.
- The **motivation**: to keep weight values small.
- Add penalty term to the error function
- ✓ Penalizes large weights to reduce variance
- ✓ Standard weight decay equation

Techniques to overcome overfitting

- Weight decay penalty term causes the weights to converge to smaller absolute values than they otherwise would.
- Large weights can hurt generalization in two different ways.
  - Excessively large weights leading to hidden units can cause the output function to be too rough, possibly with near discontinuities.
  - Excessively large weights leading to output units can cause wild outputs far beyond the range of the data if the output activation function is not bounded to the same range as the data.
- The main risk with large weights is that the non-linear node outputs could be in one of the flat parts of the transfer function, where the derivative is zero. In such case

Techniques to overcome overfitting

2

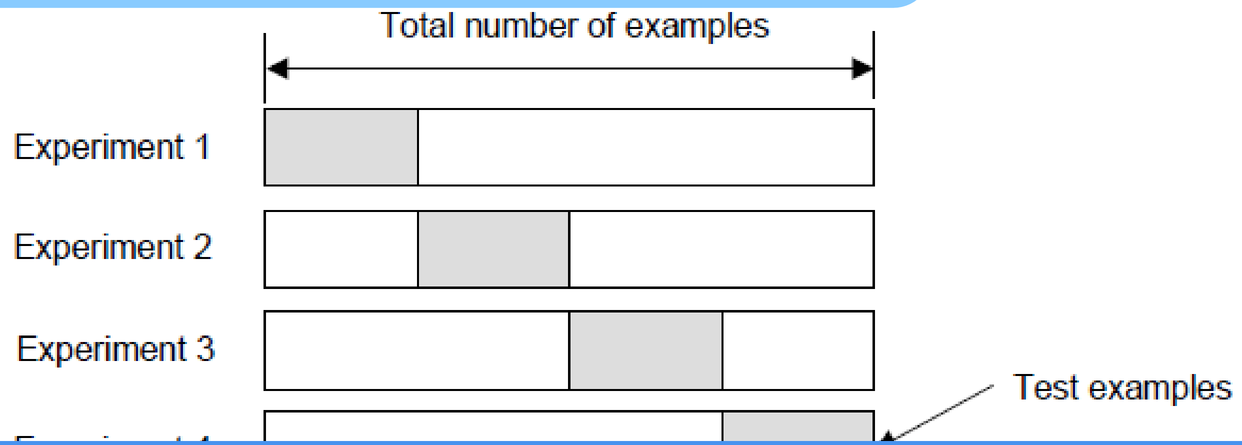
- **Cross-validation:** a set of **validation data** in addition to the training data.
- The algorithm **monitors** the error *w.r.t.* this validation data while using the training set to drive the gradient descent search.
  - How many weight-tuning iterations should the algorithm perform? It should use the number of iterations that produces the lowest error over the validation set.
  - Two copies of the weights are kept: one copy for training and a separate copy of the best weights thus far, measured by their error over the validation set.
  - Once the trained weights reach a higher error over the validation set than the stored weights, training is terminated and the stored

K-fold cross validation

- Problems with the validation set approach – small data sets
  - Not enough data may be available to provide a validation set
  - Overfitting is most severe for small data sets!
- K-fold cross validation may be used
  - Each time determine the number of epochs ep that result in best performance on the respective test partition
  - Calculate the mean of ep: ep_{mean}
  - Final run: train the network on all examples for ep_{mean}

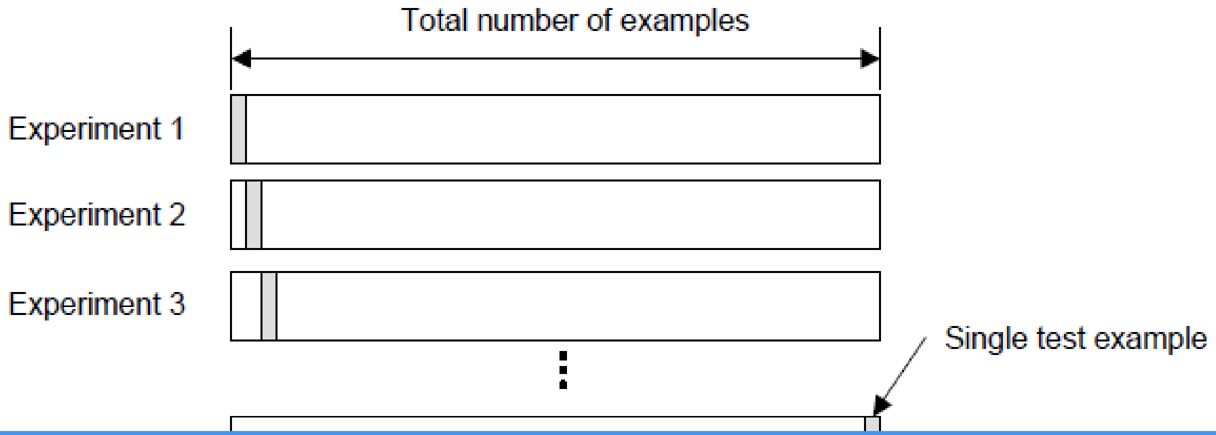
More on k-fold cross-validation

- In *K*-fold cross-validation, the original sample is randomly partitioned into *K* subsamples. Of the *K* subsamples, a single subsample is retained as the validation data for testing the model, and the remaining *K* – 1 subsamples are used as training data.
- **The cross-validation process is then repeated *K* times (the *folds*), with each of the *K* subsamples used exactly once as the validation data.**



Leave-one-out cross-validation

- leave-one-out cross-validation (**LOOCV**) involves **using a single observation from the original sample as the validation data, and the remaining observations as the training data.**
- This is repeated such that each observation in the sample is used once as the validation data.
- This is the same as a *K*-fold cross-validation with *K* being equal to the number of observations in the original sample.

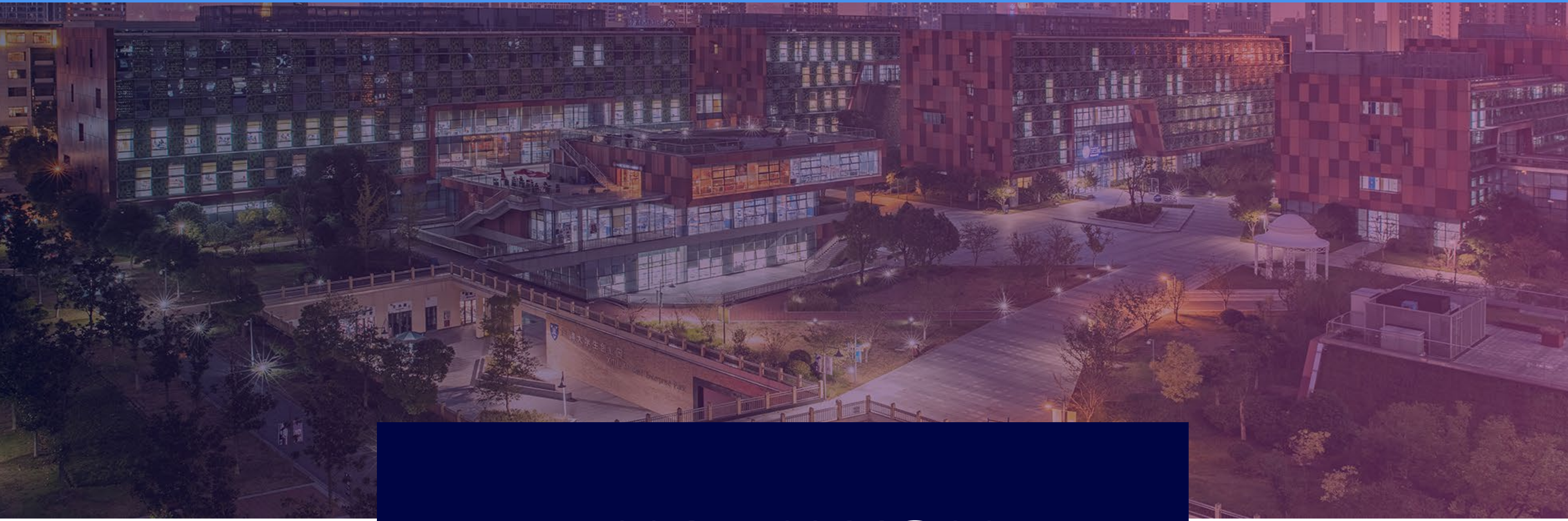


Limitations & Capabilities of MLP

- MLPs trained with backpropagation can perform function approximation and pattern classification
- Theoretically they can
  - Perform any linear and non-linear mapping
  - Can approximate any reasonable function arbitrary well
  - => Overcome the limitations of perceptrons
- In practice:
  - May not always find a solution – can be trapped in a **local minima**
  - The performance is sensitive to the starting conditions (initialization of weights)

Limitations & Capabilities of MLP

- In practice:
  - Sensitive to the number of hidden layers and neurons
    - Too few neurons – underfitting, unable to learn what you want it to learn
    - too many – overfitting, learns slowly
    - the number of hidden layers and neurons are left to the designer
  - Sensitive to the value of the learning rate
    - Too small – slow learning
    - Too big – instability or poor performance
    - The proper choices depends on the nature of examples
- Trial and error
  - Refer to the choices that have worked well in similar problems
  - Successful applications of NNs requires time and experience



THANK YOU

VISIT US

WWW.XJTLU.EDU.CN

FOLLOW US

@XJTLU

Xi’an Jiaotong-Liverpool University
西交利物浦大学