MULTI-LAYER PERCEPTRON

INT301 Bio-computation, Week 6, 2021

Outline

- Revision of BP training
- Practical issues in BP training

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

Backpropagation Training Algorithm

Backward

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

$$\beta_k = o_k(1-o_k)[y_k - o_k]$$

effects from the output nodes

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

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

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

effects from the output of the neuron

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

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

effects from multiple nodes in the next layer

Backpropagation Training Algorithm

- compute the changes for the weights $i \rightarrow j$ on connections to nodes in the hidden layer:

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

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

- *update the weights* by the computed changes:

$$w = w + \Delta w$$

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.

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**
  - *for each training example (x, y)*
  - *forward propagate the example and calculate the outputs using the sigmoid function*
  - *backward propagate the error to calculate the benefits*
  - *after processing all examples update the weights by the computed changes:*

$$w = w + \Delta w$$

- **until** *termination condition is satisfied.*

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

- 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

- 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 converges to the minimum solution.

Issues of Backpropagation: Momentum

- 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

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

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

- 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

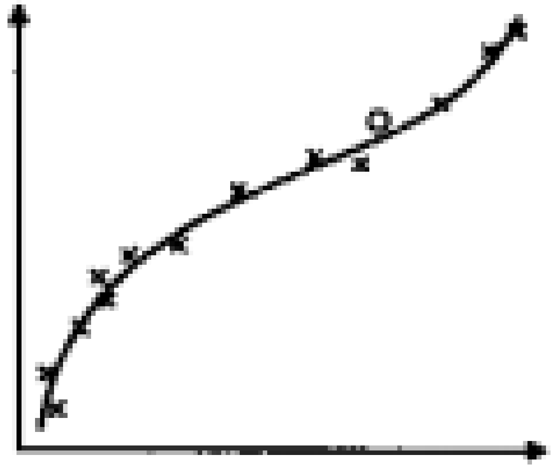
➢ ⇒ the network has memorized the training examples but has not learned to generalize to new situations!

When Does Overfitting Occur?

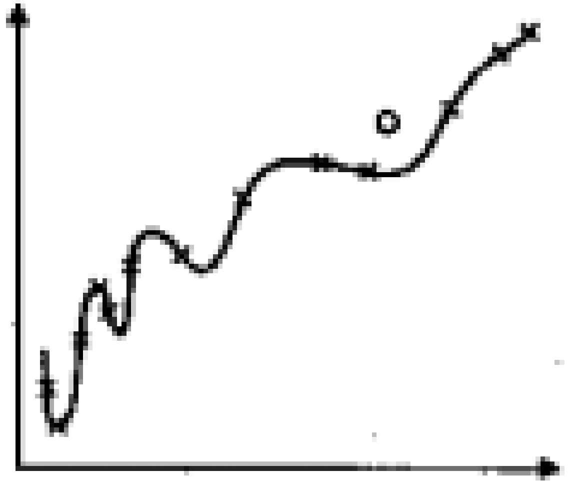
- Training examples are noisy

Example: x- training set, o-testing set

A good fit to noisy data



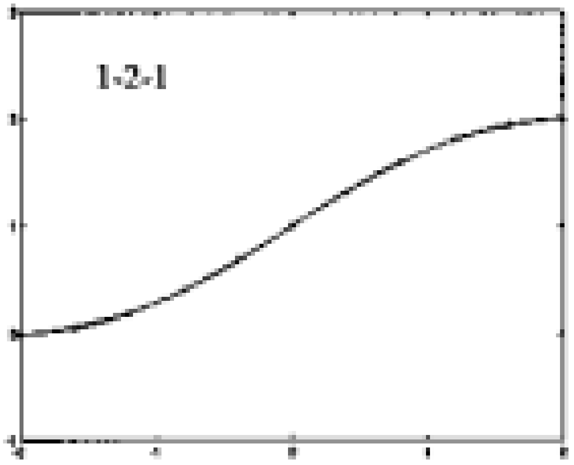
Overfitting

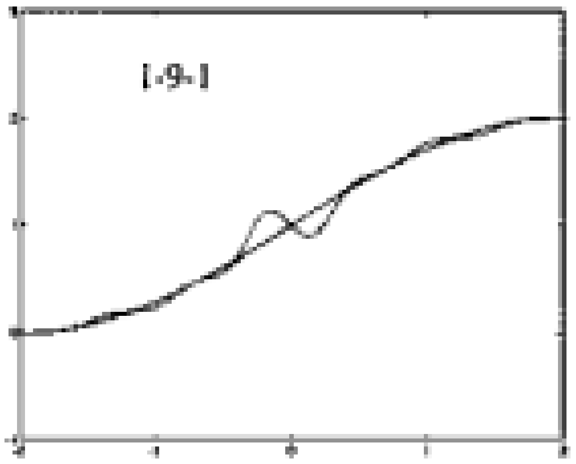


When Does Overfitting Occur? – cont.

- **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



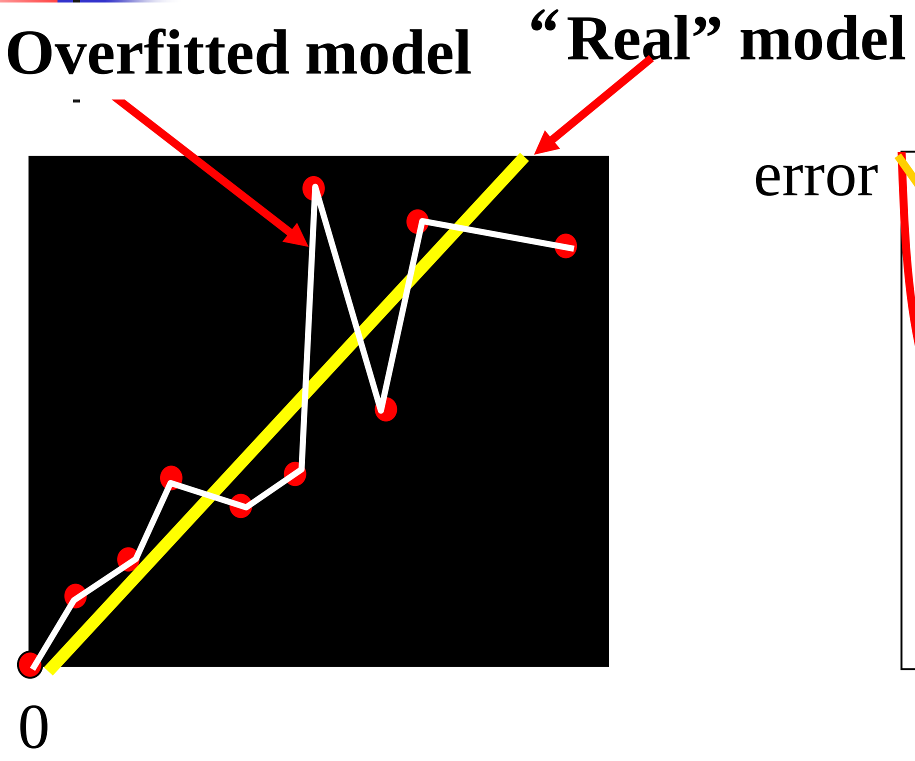


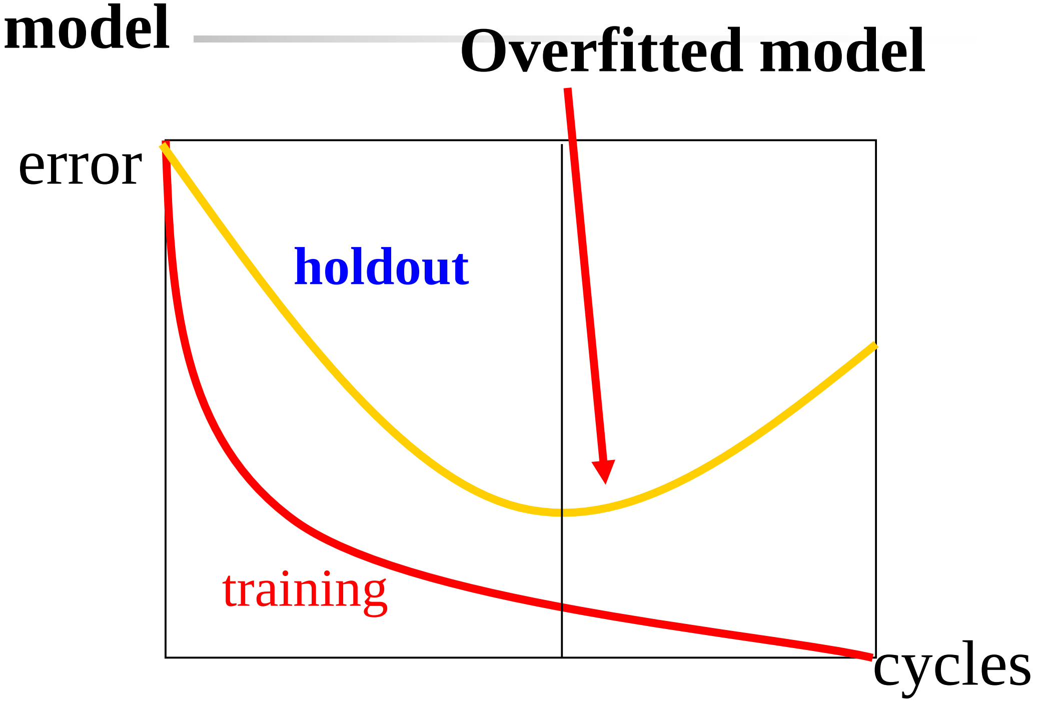
Preventing Overtraining

- 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

- ***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

penalizes large weights

$$MSE_{reg} = MSE + \gamma \cdot MSW$$

More on Weight decay

- 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 the learning is irreversibly stopped.

Techniques to overcome overfitting

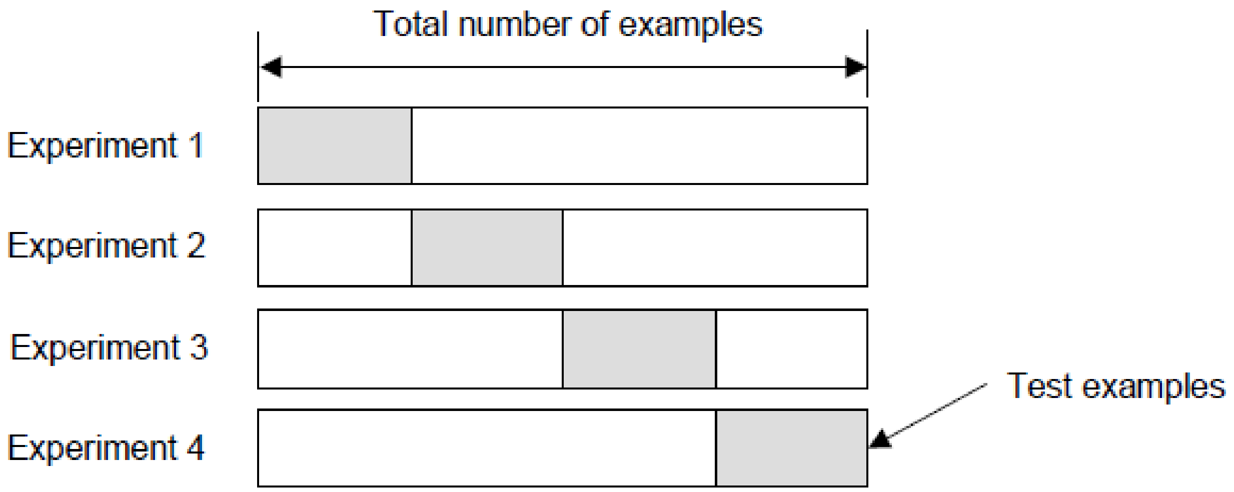
- **Cross-validation:** a set of **validation data** in addition to the training data.
- The algorithm **monitors** the error *w.r.t.* this validation data while using the training set to drive the gradient descent search.
  - How many weight-tuning iterations should the algorithm perform? It should 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 weights are returned.

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} epochs

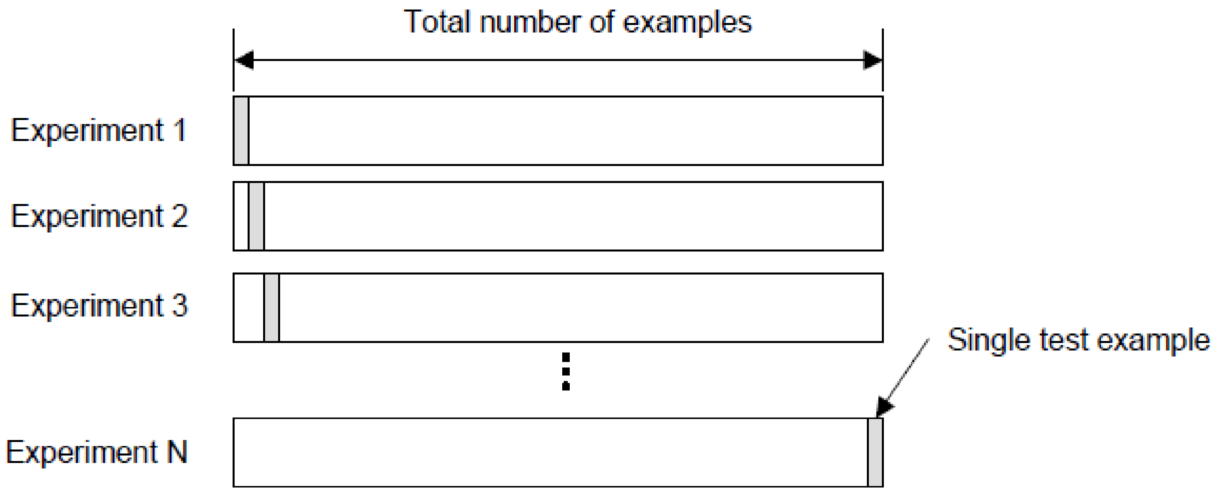
More on k-fold cross-validation

- 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
西交利物浦大學