Bias and Variance, Under-Fitting and Over-Fitting

Shan He

School for Computational Science
University of Birmingham

Module 06-27818 and 27819:
Introduction to Neural Computation (Level 4/M)
Neural Computation (Level 3)

Outline of Topics

- What we learned from bias-variance trade-off
- Demonstration of MLPs
- How to improve generalization?
  - Cross-validation and early stopping
    - Cross-validation
    - Early stopping
  - Optimal Network Topologies
  - Regularization
    - Regularization
    - Dropout

Bias-variance decomposition

- The generalisation (also called out-of-sample or expected prediction) error: measure of how accurately an algorithm is able to predict outcome values for previously unseen data based on finite number of training samples
- The generalisation error $\mathrm{Err}(x)$:

$$\mathrm{Err}(x) = \mathrm{E}\Big[\big(t - \hat{f}(x)\big)^2\Big] = \mathrm{Bias}\big[\hat{f}(x)\big]^2 + \mathrm{Var}\big[\hat{f}(x)\big] + \sigma^2$$

Where:

$$\mathrm{Bias}\big[\hat{f}(x)\big] = \mathrm{E}\big[\hat{f}(x) - f(x)\big]$$

and

$$\mathrm{Var}\big[\hat{f}(x)\big] = \mathrm{E}[\hat{f}(x)^2] - \mathrm{E}[\hat{f}(x)]^2$$

is the variance and $\sigma = \mathrm{Var}(\epsilon)$ is the irreducible error

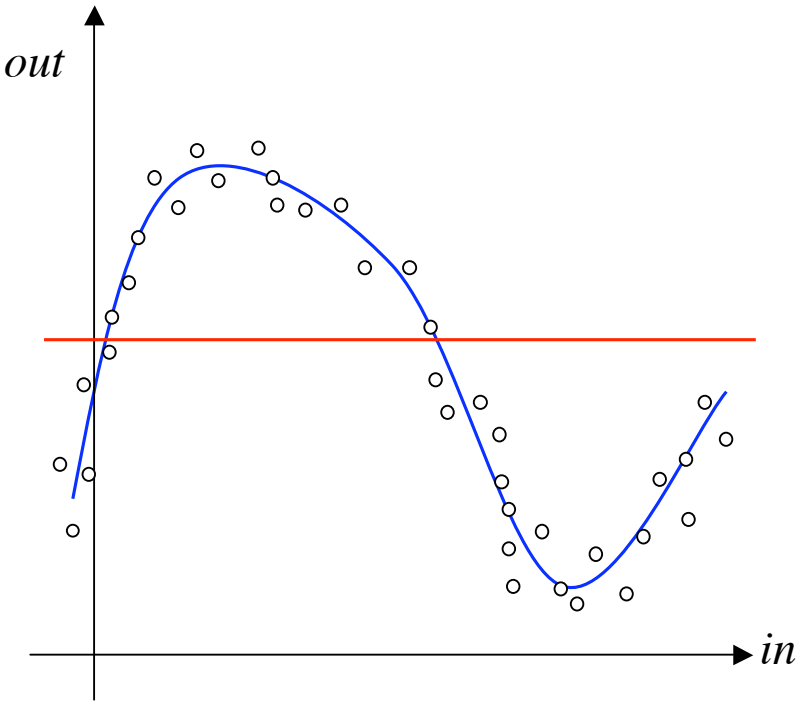
Bias-variance tradeoff

- The above equation basically means **the generalisation error from our model is either because of bias or because of variance**.
- **Question**: to reduce the generalisation error, can we train a model with **low bias AND low variance**?

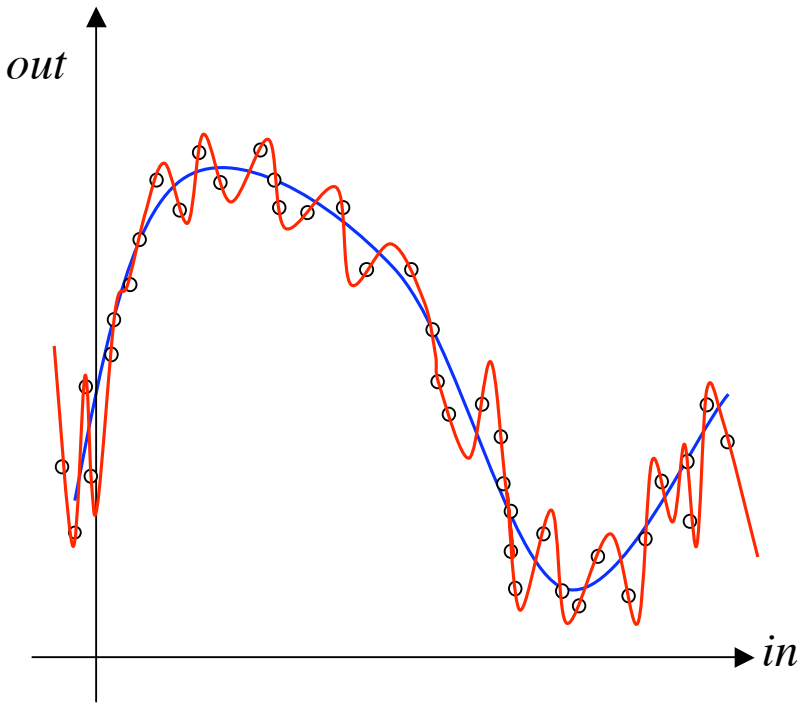
Bias-variance trade-off

- **Question**: to reduce the generalisation error, can we train a model with **low bias AND low variance** using finite number of noisy training samples?
- Answer: No. Since to reduce bias with finite number of noisy training samples, we usually increase the complexity of our model, which will increase variance

Extreme Case of Bias and Variance: Under vs Over-fitting



Ignore the data ⇒
Big approximation error (high bias)
No variation between data sets (no variance)



Fit every data point ⇒
No approximation error (zero bias)
Variation between data sets (high variance)

Let's play with the toy problem using our MLP

- The problem: a binary non-linearly separable classification problem
- Using make_moon function to generate 2d binary classification problems where data points are two interleaving half circles
- We generate 20 training samples and 100 testing samples with some noise.
- Let's try a few MLP parameters:
  - Number of hidden neurons: 2, 50
  - Number of maximum epochs: 100, 1000

Let's play with the toy problem using our MLP

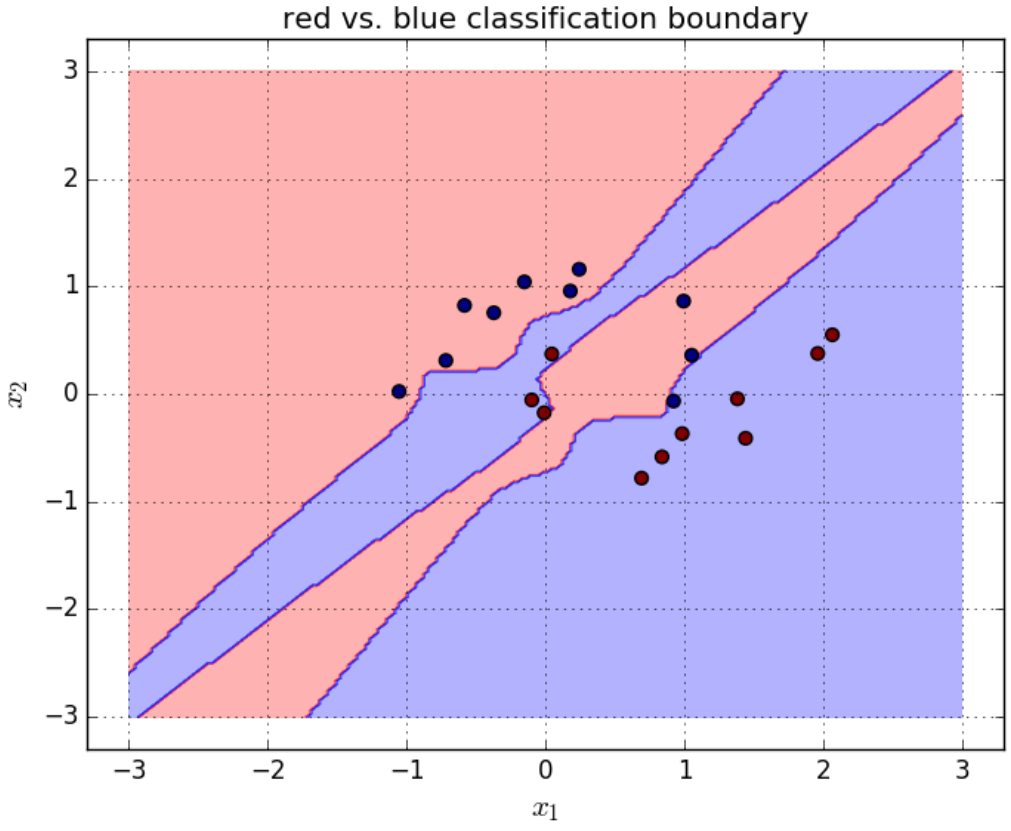


Figure : Decision boundary from a MPL with 50 neurons trained with 2000 epochs. The training error is 0

How to improve generalisation?

- To achieve minimal generalisation error, we need to build a model with optimal complexity
- But how?
  - We first need to estimate the generalisation error from training set
  - Using the estimation, we can design model with appropriate complexity

Estimating generalisation error: validation set

- Estimating generalisation error: a simple way
  - Validation set or holdout method:
    - Step 1: Divide your training set into 2 parts: validation V and (real-) training T sets.
    - Step 2: Train your model (e.g., MLP) using T and estimate the expected prediction error Err^* using V
    - Step 3: We don't want to waste, so swap V and T and go back to Step 2
    - Step 4: Average these two estimated prediction errors
  - In fact, it is called **2-fold cross-validation**
  - Problems:
    - Estimation is based on two estimated prediction errors – Not accurate
    - You model is trained with half of the training dataset – Not comparable to the model trained with the complete training dataset

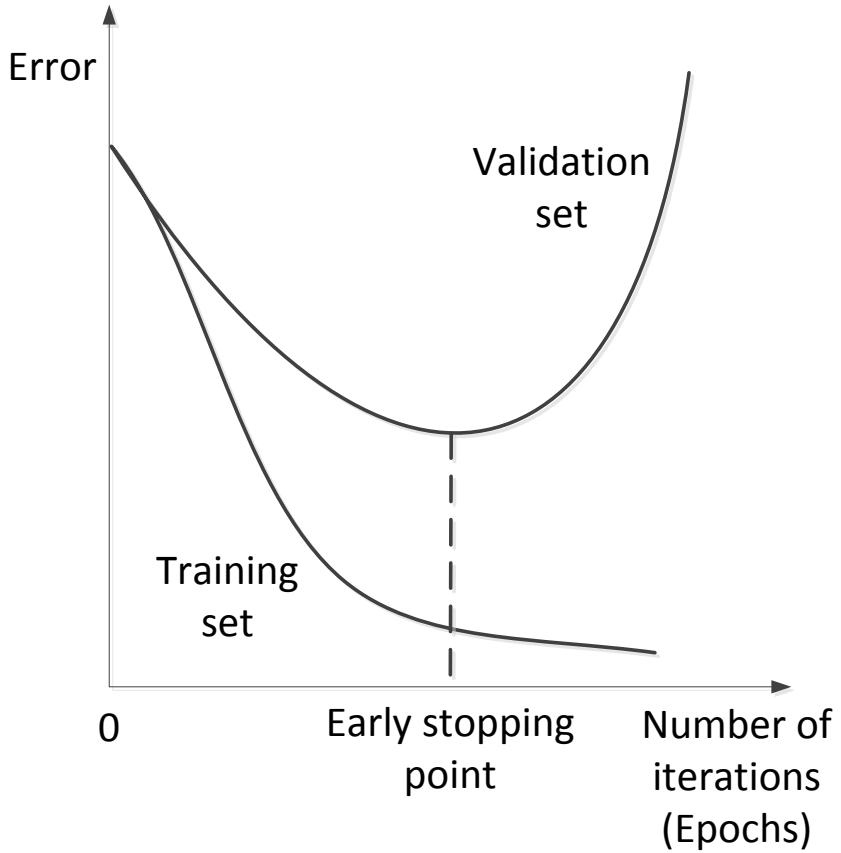
Estimating generalisation error: cross validation

- A general way to estimating generalisation error: K-fold cross-validation:
- If K is equal to the full sample size, it is called leave-one-out cross validation.

Inputs: training set
begin
Divide your training set randomly into K different parts.
for each part (denoted as k) in the K parts
Use the other $K-1$ parts together for training your model.
Use the part k for testing to estimate $\mathrm{Err^*}_k$.
end for
Calculate the average generalisation error $\overline{\mathrm{Err^*}} = \frac{1}{K}\sum_{i=1}^{K}\mathrm{Err^*}_k$
end

Early stopping

- **Early stopping**: A kind of regularisation method to optimise the amount of training to reduce the complexity of your model
- **Training set error**: decrease with increasing numbers of epochs of training
- **Validation and testing sets error**: decrease as the under-fitting is reduced, but then it will begin to increase (over-fitting).

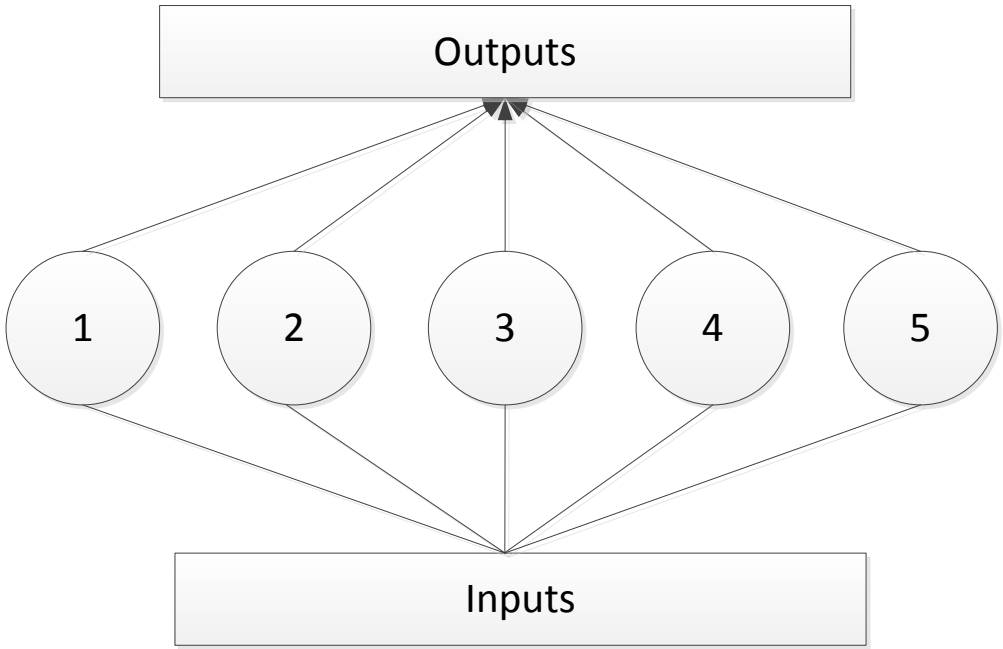


How obtain optimal network topologies

- Previously, the neural network topology is determined manually and the weights are learned by a learning process
- We can also learn network topology by algorithms automatically:
  - Constructive algorithms: start with too few hidden units, and add some more
  - Pruning algorithms: start with too many hidden units, and take some away
- You can simply rely on training set, or you can use cross-validation to estimate generalisation error to guide when to stop constructing or pruning (very time consuming, i.e., training your network K times)

Constructive algorithms

- **Idea**: keep adding further hidden neurons to an existing network and only train the new portions of it.
- **Example**: add hidden neurons in the sequence, with each additional neurons trained to deal with **the remaining incorrect training patterns**



Pruning algorithms

- **Idea**: start with too many hidden neurons and explicitly remove some connections
- **Question**: How to select connections to remove?

Pruning algorithms

- **Idea**: start with too many hidden neurons and explicitly remove some connections
- **Question**: How to select connections to remove?
- **Answer**: Remove those connections or weights that has minimum effect on training/validation error, because they are not necessary but increase the complexity of the neural network model
- **How**:
  - Define the **saliency** of each weight w_{ij}:
    $s_{ij} = E(\mathbf{w} : w_{ij} = 0) - E(\mathbf{w})$
- **Problem**: We need to compute all the saliencies – computationally expensive!
- **Better solution**: Optimal Brain Damage by Yann Le Cun

Regularization

- **Regularisation**: a process in machine learning and statistics to prevent overfitting
- **Methods**:
  - Early stopping ✓
  - Regularisation: adding a regularisation term to the cost function
  - Dropout: a new method from Hinton's group in 2014

Regularization

- **Regularisation**: Add a regularisation term to the cost function
- **Idea**: penalise over-complex network mappings by adding a penalty term Ω to the standard (e.g., sum squared error) cost function:

$$E_{reg} = E_{sse} + \lambda\Omega$$

where the regularization parameter λ controls the trade-off between reducing the error $E(\mathbf{w})$ and increasing the smoothing.

- The gradient descent update:

$$\Delta w_{ij}^{(l)} = -\eta \frac{\partial E(\mathbf{w})}{\partial w_{ij}^{(l)}} - \eta\lambda \frac{\partial \Omega(\mathbf{w})}{\partial w_{ij}^{(l)}}$$

Weight decay

- The simplest regularization function is the sum of the squares of all network weights:

$$\Omega(\mathbf{w}) = \frac{1}{2}\sum_{i,j,l}(w_{ij}^{(l)})^2$$

- The extra term in the weight updates:

$$-\eta\lambda\frac{\partial\Omega(\mathbf{w})}{\partial w_{ij}^{(l)}} = -\eta\lambda w_{ij}^{(l)}$$

- The gradient descent update:

$$w_{ij}^{(l)} = w_{ij}^{(l)} + \Delta w_{ij}^{(l)} = w_{ij}^{(l)} - \eta\frac{\partial E(\mathbf{w})}{\partial w_{ij}^{(l)}} - \eta\lambda w_{ij}^{(l)}$$

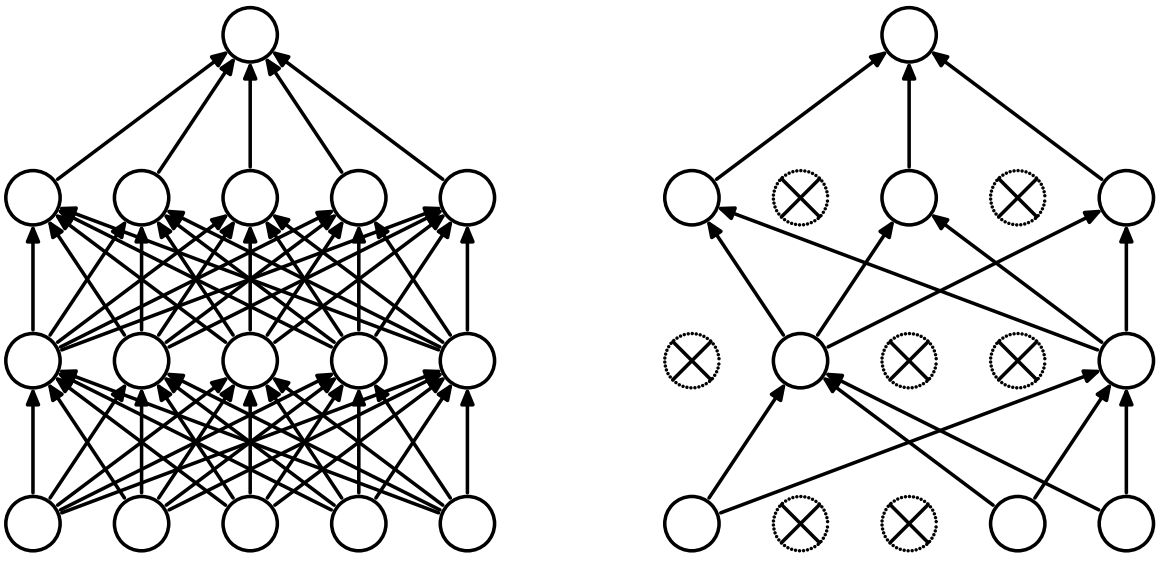
$$= (1-\eta\lambda)w_{ij}^{(l)} - \eta\frac{\partial E(\mathbf{w})}{\partial w_{ij}^{(l)}}$$

Weight decay: Python demonstration

Dropout

- Proposed by Nitish Srivastava in Hinton's group in 2014 (Paper here)
- A simple regularisation method to prevent neural networks from overfitting
- **Idea**: "*to take a large model that overfits easily and repeatedly sample and train smaller sub-models from it*"
- **How**: during training, **randomly** and **temporarily** remove neurons (hidden and visible) in a neural network, along with all its incoming and outgoing connections

Dropout

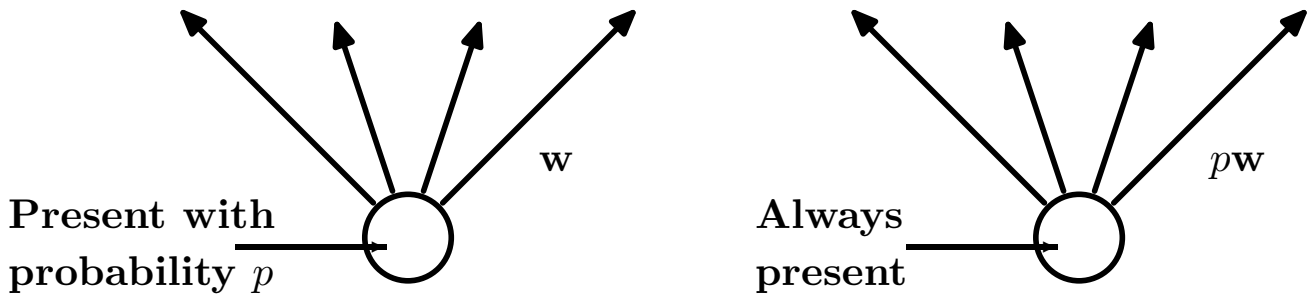


(a) Standard Neural Net (b) After applying dropout.

Figure 1: Dropout Neural Net Model. **Left**: A standard neural net with 2 hidden layers. **Right**: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

Figure : Figure taken from Nitish's 2014 Paper

Dropout



(a) At training time (b) At test time

Figure 2: **Left**: A unit at training time that is present with probability p and is connected to units in the next layer with weights $\mathbf{w}$. **Right**: At test time, the unit is always present and the weights are multiplied by p. The output at test time is same as the expected output at training time.

Figure : Figure taken from Nitish's 2014 Paper

Dropout: Python demonstration

Please download my source code from Canvas.
You can also read Nitish's code

Dropout: how to tune?

- One hyper-parameter to tune: Dropout rate p, **the probability of retaining a neuron**.
  - Smaller p requires large number of neurons which slows down the training and leads to underfitting.
  - Large p may not produce enough dropout to prevent overfitting.
  - Default value of p: for hidden layer is in the range of 0.5 to 0.8, for input layer larger than 0.8
- Hinton himself advocate tuning dropout in conjunction with tuning the size of your hidden layer (Improving neural networks by preventing co-adaptation of feature detectors and this website)
  - Step 1: Increase your hidden layer size(s) with dropout turned off until you perfectly fit your data
  - Step 2: using the same hidden layer size, train with dropout turned on.

Conculsion and further reading

- Recalling the aim of good generalization.
- Validation and cross-validation for estimating generalization based on training data.
- We introduced a few methods to improve generalisation:
  - Optimal Network topologies:
    - Constructive method
    - Pruning methods (Optimal Brain Damage)
  - Regularisation:
    - Early stopping
    - Regularisation (Weight decay)
    - Dropout
  - Reading list
    - Bishop: Sections 9.2, 9.3, 9.4, 9.5, 9.8
    - Haykin-2009: Sections 4.11, 4.13, 4.14
    - Nitish's 2014 Paper and Hinton's 2012 paper