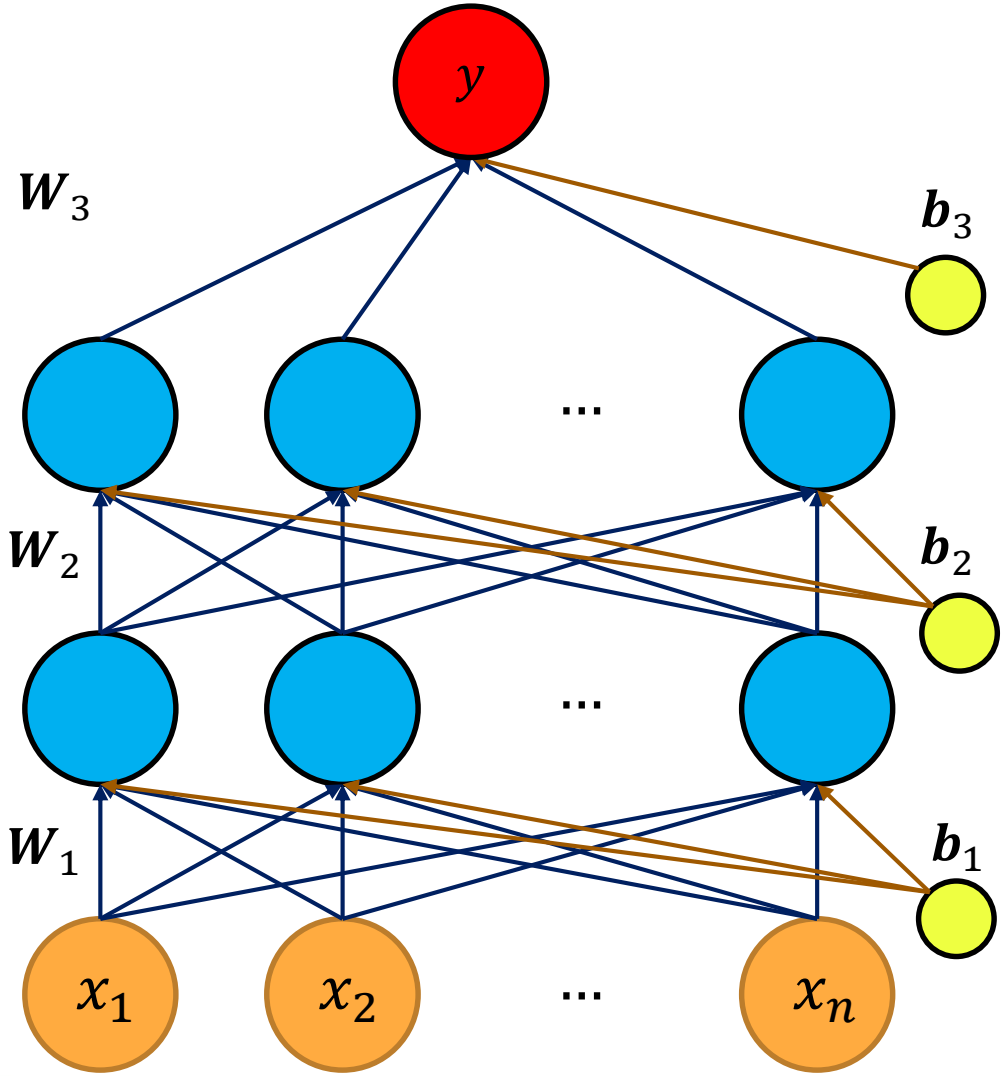
Training DNN Models

Contents

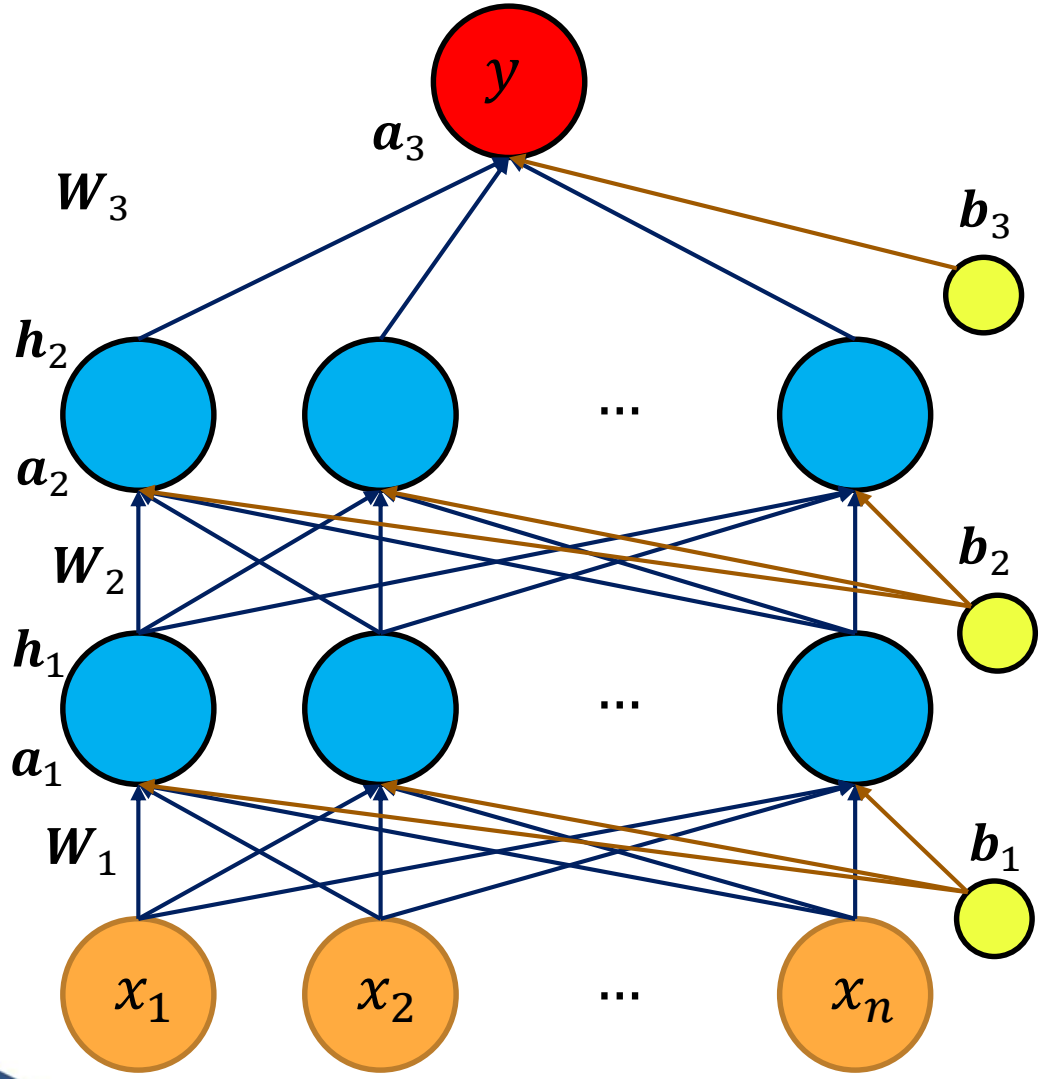
- Recap of Deep Neural Network
- Training a DNN
- Gradient Descent with Backpropagation
- Variants of Gradient Descent
- Batch Normalisation
- Regularisation

DNN Architecture



- Input layer (0^{th}) with n neurons
- $L-1$ hidden layers with m neurons each
- Output layer (L^{th}) with k neurons
- Neurons in successive layers are connected to one another
- $\boldsymbol{W}_i$ and $\boldsymbol{b}_i$ represent the weights and biases between layers $i-1$ and i $(0 < i \leq L)$
- Each neuron in hidden and output layers has an activation function
- Activation function could be linear, sigmoid, softmax, relu, tanh, etc.

DNN Feed Forward Calculation



- Input to activation function at layer 0:

$$\boldsymbol{a}_1 = \boldsymbol{W}_1\boldsymbol{x} + \boldsymbol{b}_1$$

- At hidden layer 1:

$$\boldsymbol{h}_1 = g_h(\boldsymbol{a}_1)$$

- At hidden layer i:

$$\boldsymbol{h}_i = g_h(\boldsymbol{a}_i) = g_h(\boldsymbol{W}_i\boldsymbol{h}_{i-1} + \boldsymbol{b}_i)$$

- At output layer:

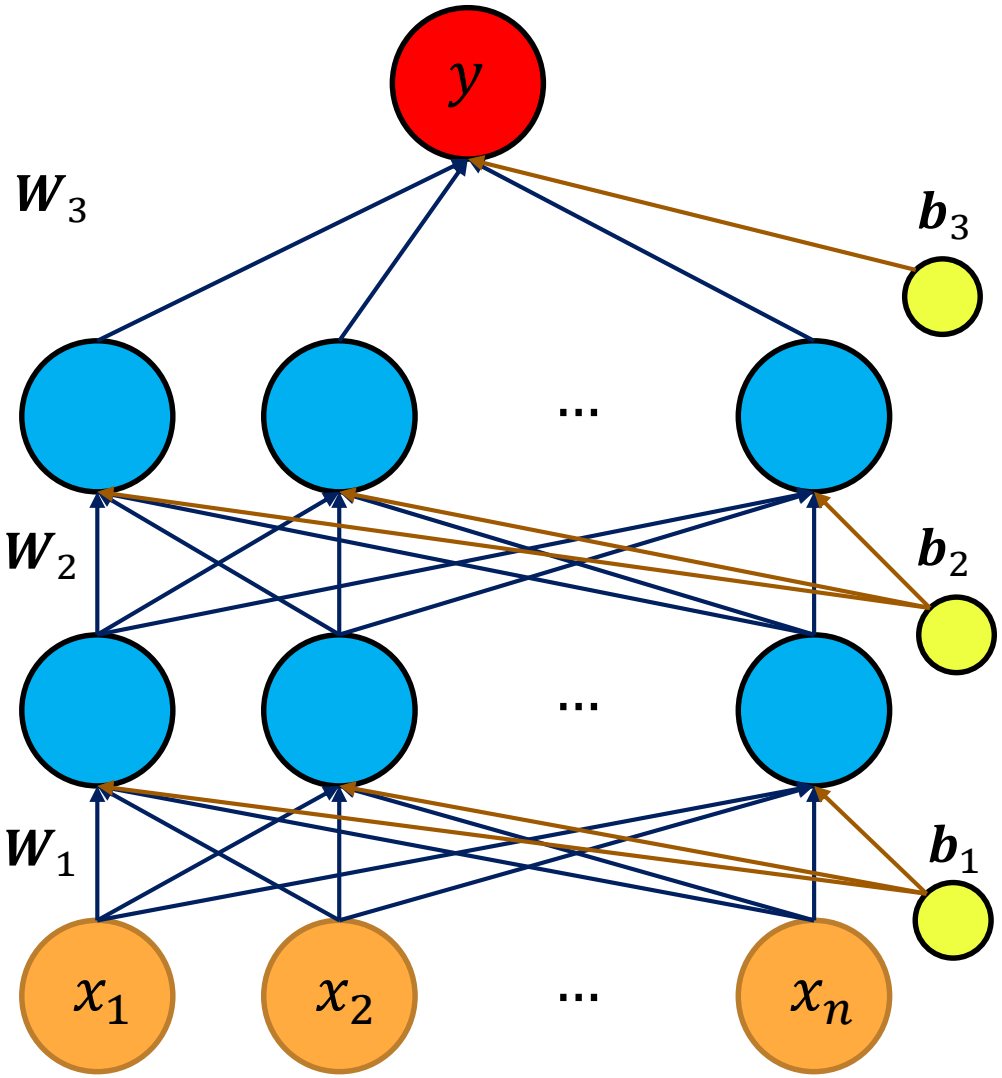
$$\hat{\boldsymbol{y}} = g_o(\boldsymbol{a}_L) = g_o(\boldsymbol{W}_L\boldsymbol{h}_{L-1} + \boldsymbol{b}_L)$$

- Model (function) being approximated by the DNN (assuming L=3):

$$\hat{\boldsymbol{y}} = g_o(\boldsymbol{W}_3(\boldsymbol{W}_2(\boldsymbol{W}_1\boldsymbol{x} + \boldsymbol{b}_1) + \boldsymbol{b}_2) + \boldsymbol{b}_3)$$

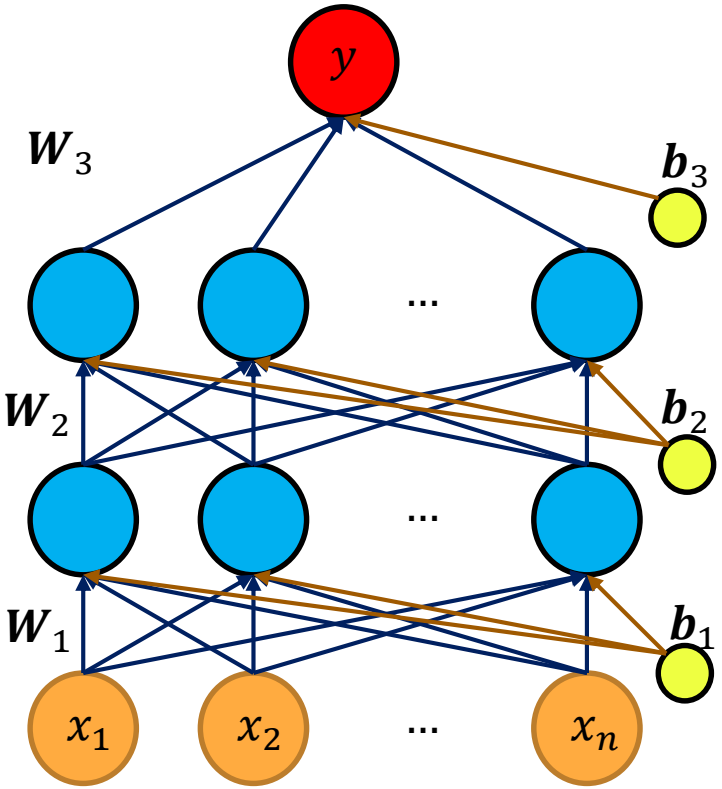
$$\hat{\boldsymbol{y}} = f(\boldsymbol{x})$$

Supervised Learning using DNN



- In supervised learning, input $(\boldsymbol{x})$ and output $(\boldsymbol{y})$ data is given to learn a function $\hat{\boldsymbol{y}} = f(\boldsymbol{x})$ such that $\hat{\boldsymbol{y}} \approx \boldsymbol{y}$
- Question: What is a suitable $f(\boldsymbol{x})$ for the given data or task ?
- Question: Can we find the weights and biases which will approximate desired $f(\boldsymbol{x})$?
- Training a DNN: Learning the parameters of the DNN (weights and biases) using the given data

Training a DNN



Steps to train a DNN using data

1. Create a neural network
layers
neuron

2. Randomly initialize weights and biases $(W_1, \ldots W_L, b_1, \ldots b_L)$

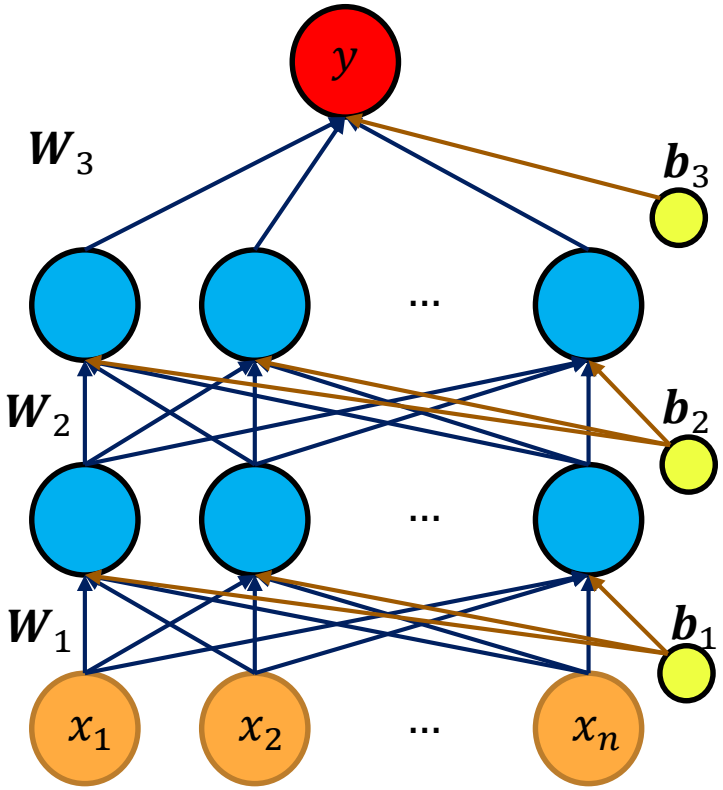
3. Pass inputs $(\boldsymbol{x})$ through the network and get output $(\hat{y})$

4. Find out difference between actual output (y) and predicted output $(\hat{y})$
- Prediction error

5. Adjust the weights to minimise the difference between y and $\hat{y}$

A DNN based model is trained on the given data
i.e., $\hat{y} \approx y$ for all inputs

Training a DNN – Step 4



Step 4: Find out difference between actual output (y) and predicted output $(\hat{y})$ - Prediction error

- Loss function or Cost function is defined to quantify the prediction error
- For regression: Mean squared error loss since output is a real number

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

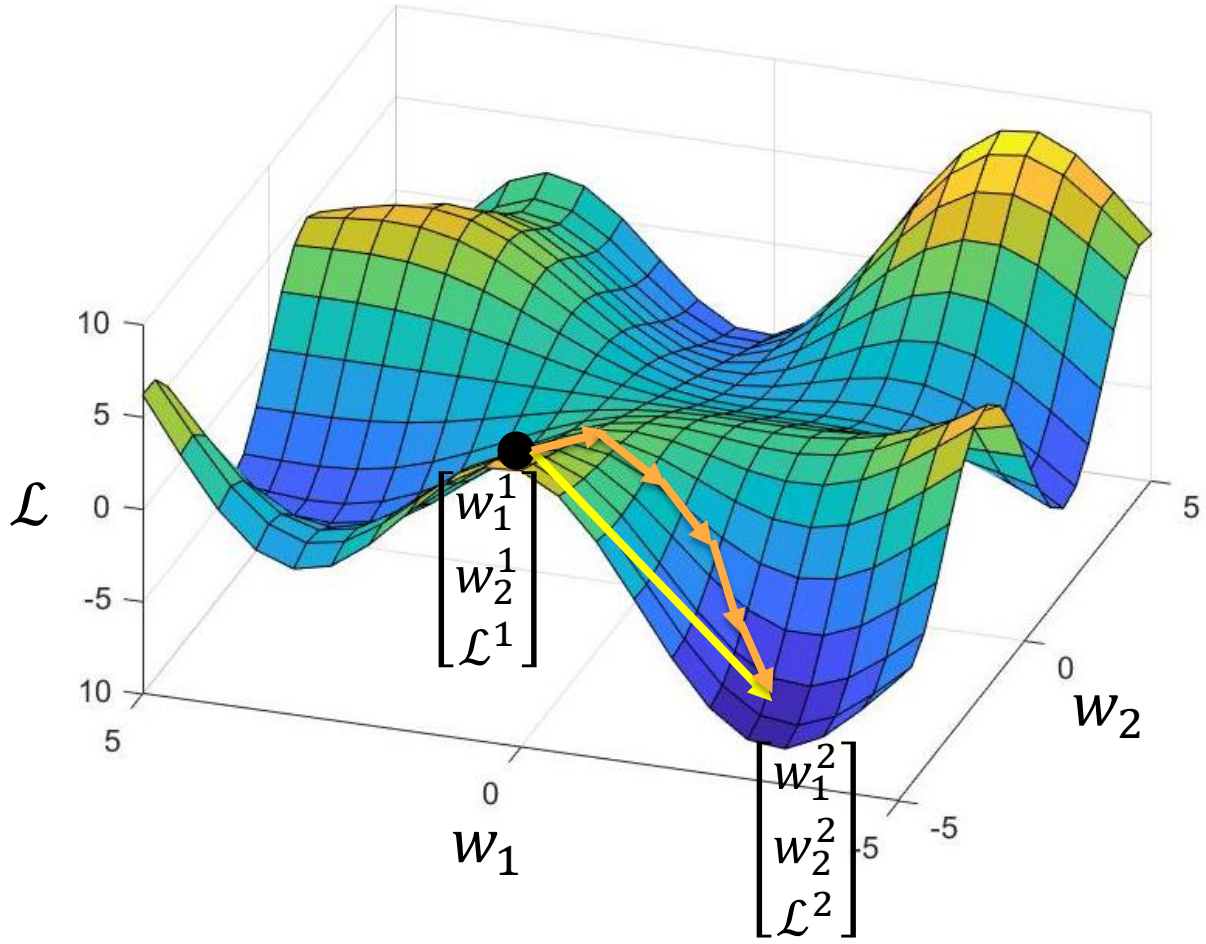
- For classification: Cross entropy loss error since output is a probabilistic value for each class (k classes)

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{1}^{k} y \log \hat{y}$$

Training a DNN – Step 5

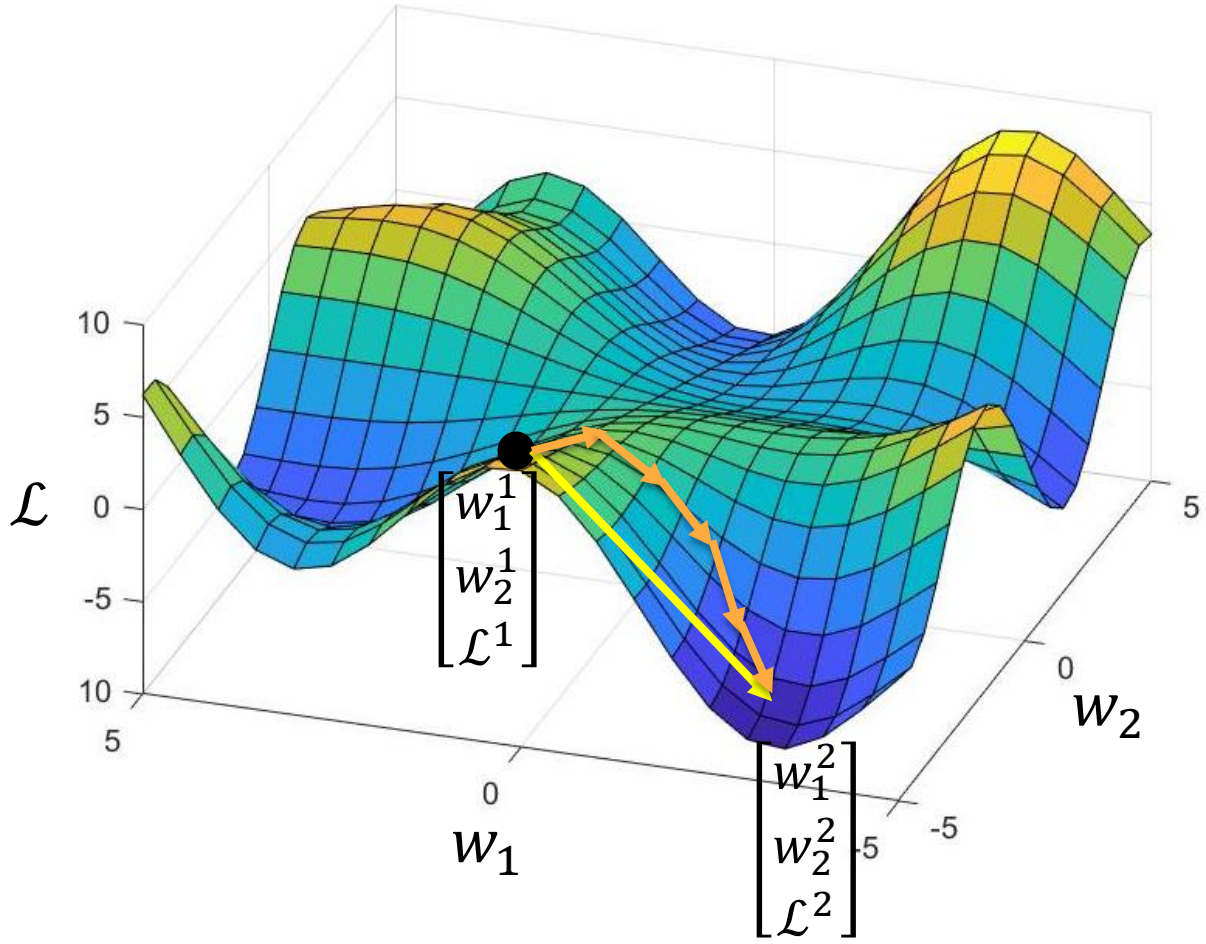
Step 5: Adjust the weights to minimise the loss function

- Question: How to adjust parameters to minimise the loss function $\mathcal{L}(y, \hat{y})$?
- Suppose there are only 2 parameters w_1 and w_2 on which the loss function is dependent
- To minimize loss, take small steps in the direction along which $\mathcal{L}$ decreases
- Implies: w_1 and w_2 are to be modified at each step such that $\mathcal{L}$ decreases
- Directions at each step are given by the gradient



Gradient Descent

- Gradient: Derivative of the loss function with respect to the parameters $\left(\frac{\partial \mathcal{L}}{\partial w_1}\right)$ and $\left(\frac{\partial \mathcal{L}}{\partial w_2}\right)$
- Interpretation: Gradient is the value by which the loss increases when the parameter increases
- $\frac{\partial \mathcal{L}}{\partial w_1} = 2 \Rightarrow$ when w_1 increases by a value of 1, $\mathcal{L}$ increases by a value of 2
- By updating w_1 as $w_1 + \frac{\partial \mathcal{L}}{\partial w_1}$ and w_2 as $w_2 + \frac{\partial \mathcal{L}}{\partial w_2}$, the value of $\mathcal{L}$ increases



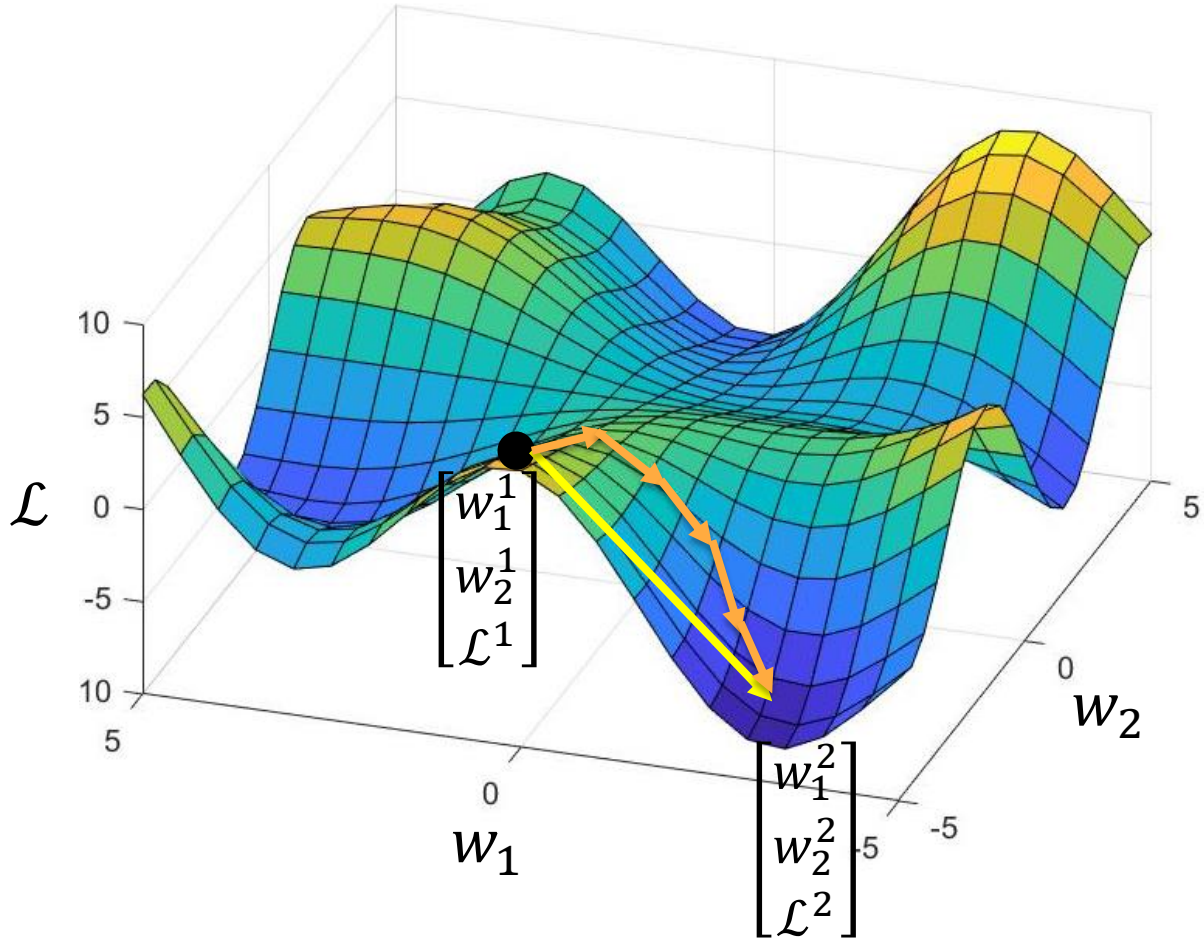
Gradient Descent

- Since the objective is to decrease, the parameters are updated as follows:

$$w_1(new) = w_1(old) - \alpha \frac{\partial \mathcal{L}}{\partial w_1}$$

$$w_2(new) = w_2(old) - \alpha \frac{\partial \mathcal{L}}{\partial w_2}$$

- α is the learning rate to decide how much to change
- Parameters are to updated iteratively until the loss function is minimised (convergence)



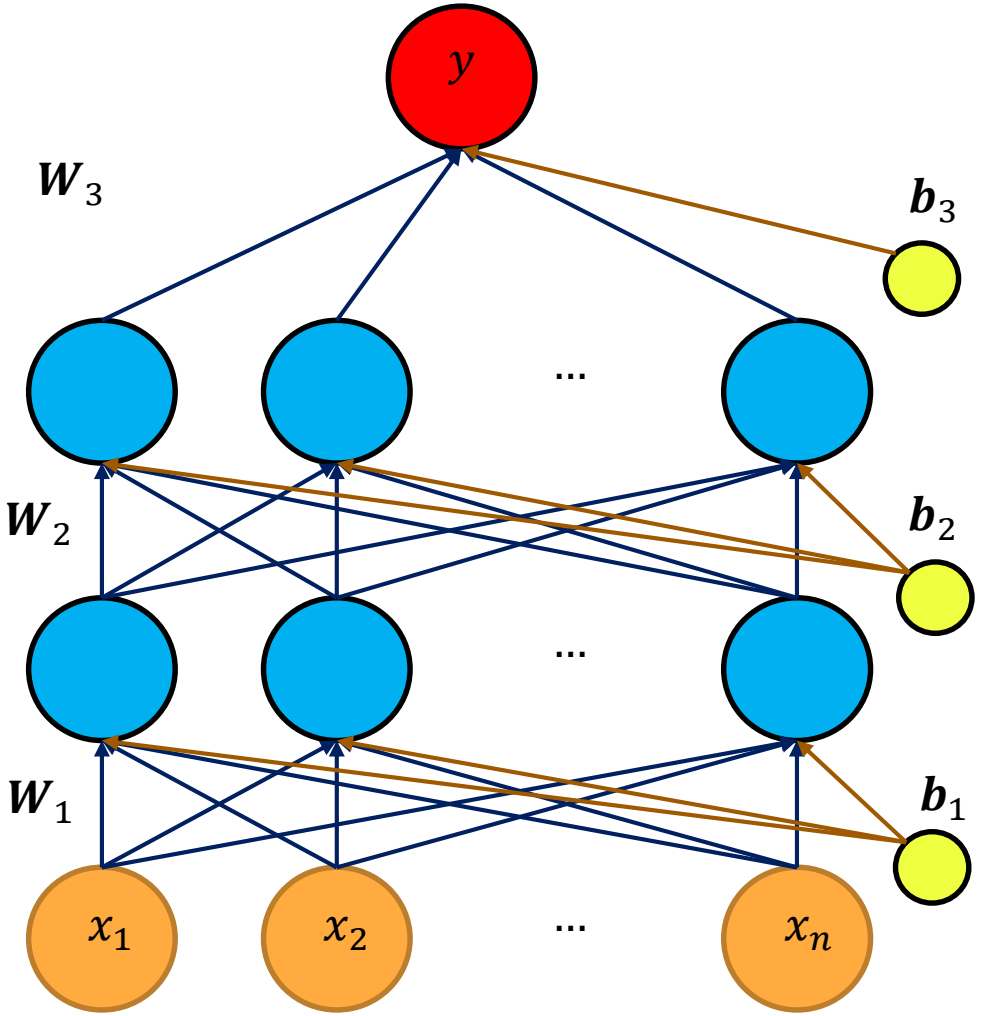
Training a DNN – Gradient Descent

- Note: In a DNN, the loss is dependent on all the parameters $\boldsymbol{W}_1, \boldsymbol{W}_2, \ldots \boldsymbol{W}_L$ and $\boldsymbol{b}_1, \boldsymbol{b}_2, \ldots, \boldsymbol{b}_L$
- So gradient needs to be calculated w.r.t. all the parameters
- General parameter update:

$$w_{new} = w_{old} - \alpha \frac{\partial \mathcal{L}}{\partial w}$$

$$b_{new} = b_{old} - \alpha \frac{\partial \mathcal{L}}{\partial b}$$

- Question: How to calculate the gradient of $\mathcal{L}$ w.r.t. all the parameters in the DNN ?
- Answer: Gradient descent with backpropagation



Gradient Descent with Backpropagation

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \text{ (or)}$$

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{1}^{k} y \log \hat{y}$$

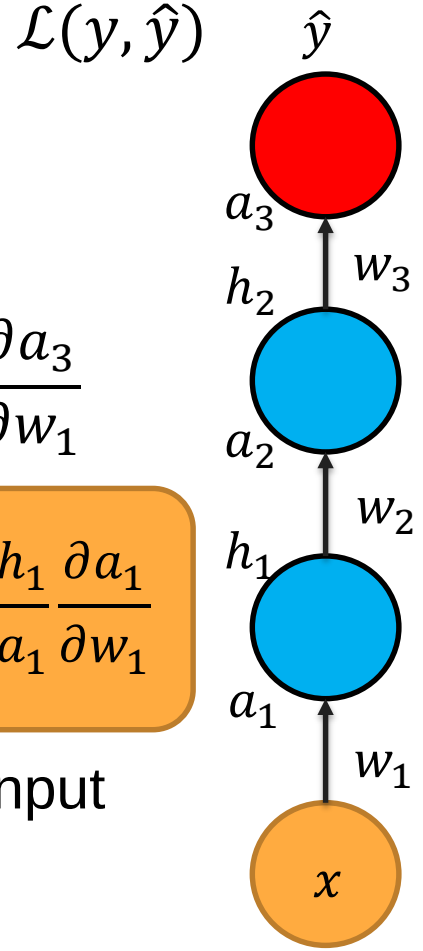
$$\hat{\boldsymbol{y}} = g_o(\boldsymbol{W}_3(\boldsymbol{W}_2(\boldsymbol{W}_1\boldsymbol{x} + \boldsymbol{b}_1) + \boldsymbol{b}_2) + \boldsymbol{b}_3)$$

- Loss function is connected to the parameters through $\hat{\boldsymbol{y}}$
- Issue: Finding loss function in terms of each of the parameters is a complex process
- Solution: Chain rule can be used to compute the gradient w.r.t each of the parameters
- So the loss is being backpropagated through the chain of gradients

$$\frac{\partial \mathcal{L}}{\partial w_1} = \frac{\partial \mathcal{L}}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial w_1} = \frac{\partial \mathcal{L}}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial a_3}\frac{\partial a_3}{\partial w_1}$$

$$\frac{\partial \mathcal{L}}{\partial w_1} = \frac{\partial \mathcal{L}}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial a_3}\frac{\partial a_3}{\partial h_2}\frac{\partial h_2}{\partial a_2}\frac{\partial a_2}{\partial h_1}\frac{\partial h_1}{\partial a_1}\frac{\partial a_1}{\partial w_1}$$

Note: $\frac{\partial \mathcal{L}}{\partial w_1}$ will vary for each input and output



Training a DNN – Summary of Step 5

a) Known:
- $(\boldsymbol{x}, \boldsymbol{y}) - N$ Samples
- $\boldsymbol{W}_1, \boldsymbol{W}_2, \ldots \boldsymbol{W}_L$ and $\boldsymbol{b}_1, \boldsymbol{b}_2, \ldots, \boldsymbol{b}_L$ (initialised values)
- $\hat{\boldsymbol{y}}$ for each sample and and overall loss $\mathcal{L}$

b) Calculate the gradients of loss function w.r.t. all the weights and biases using chain rule

Note: Gradient calculation involves a summation over all samples in the data

c) Update the weights and biases:

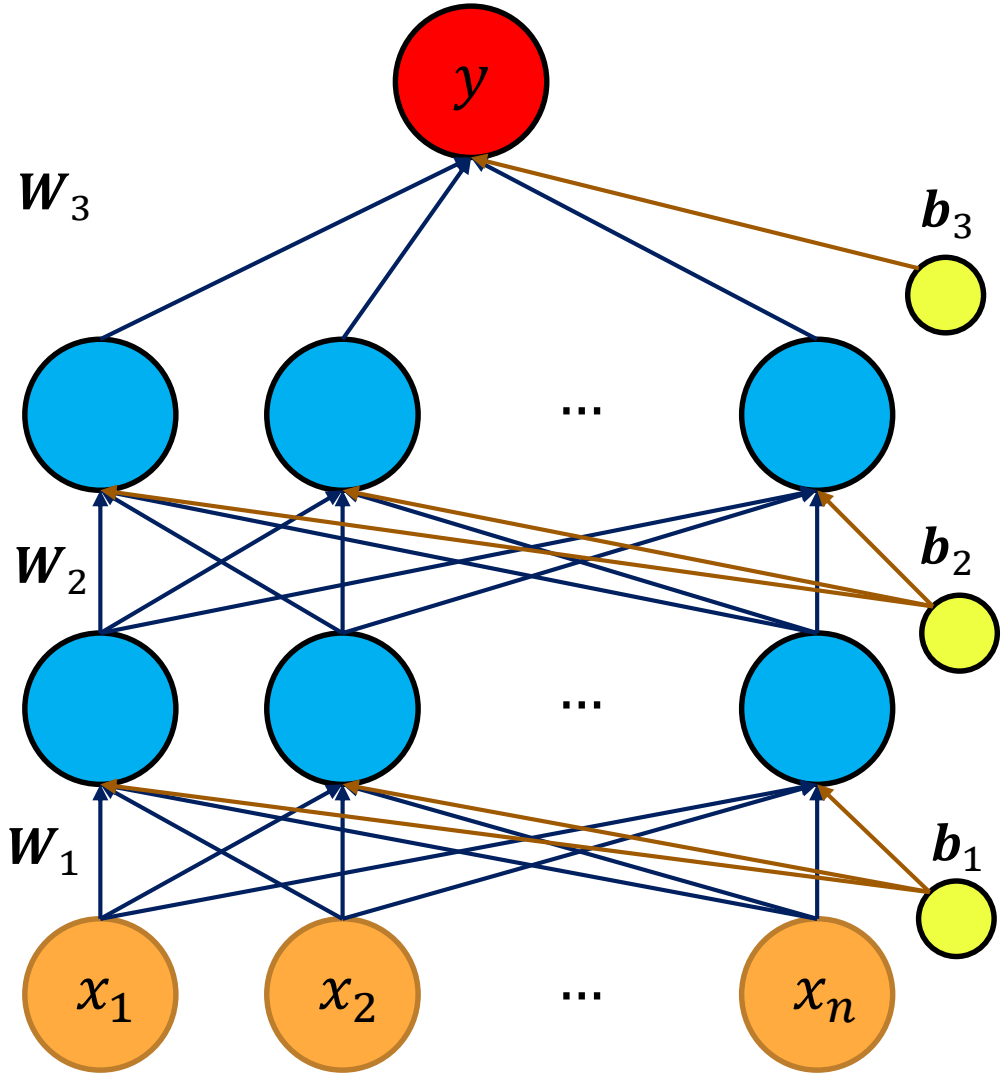
$$w_{new} = w_{old} - \alpha \frac{\partial \mathcal{L}}{\partial w}$$

$$b_{new} = b_{old} - \alpha \frac{\partial \mathcal{L}}{\partial b}$$

d) With new parameter, compute $\hat{\boldsymbol{y}}$ for each sample and and overall loss $\mathcal{L}$

e) Repeat (b), (c) and (d) for many iterations – until loss is minimised

DNN Model Training - Components



- Data (Given): $\{\boldsymbol{x}_i, \boldsymbol{y}_i\}; i = 1 \ldots N$
- Model (Chosen):

$$\hat{\boldsymbol{y}} = f(\boldsymbol{x}, \boldsymbol{W}_1, \ldots \boldsymbol{W}_L, \boldsymbol{b}_1, \ldots \boldsymbol{b}_L)$$

- Parameters (To be learnt):

$$\boldsymbol{W}_1, \ldots, \boldsymbol{W}_L, \boldsymbol{b}_1, \ldots, \boldsymbol{b}_L$$

- Loss Function: Mean squared error for regression and cross entropy for classification
- Training Algorithm: Gradient descent with back propagation

Variants of Gradient Descent

Types of Gradient Descent

- Depending on the number of samples used to estimate the gradient of loss w.r.t. the parameters, there are 3 types of gradient descent:
  - Batch Gradient Descent
  - Stochastic Gradient Descent
  - Mini-Batch Gradient Descent
- Reasons for these 3 types of gradient descent:
  - Computational efficiency
  - Accuracy of estimated gradient

Some Terminology

- Epoch – One epoch of training is said to be complete if every sample in the training dataset is used for gradient calculation and parameter updation.
- Batch size (b) – Number of samples used in gradient computation.

Batch Gradient Descent

- Parameters are updated by estimating the gradient using all the N samples i.e., batch size, $b = N$

$$w_{1(new)} = w_{1(old)} - \alpha\left(\frac{\partial \mathcal{L}}{\partial w_1}\right) = w_{1(old)} - \alpha\left[\left(\frac{\partial \mathcal{L}}{\partial w_1}\right)_1 + \left(\frac{\partial \mathcal{L}}{\partial w_1}\right)_2 + \cdots + \left(\frac{\partial \mathcal{L}}{\partial w_1}\right)_N\right]$$

- In this case, the parameters are updated only once in a epoch
- Advantages: Convergence is guaranteed in this case
- Disadvantage: Slow to converge with large datasets

Stochastic Gradient Descent

- Parameters are updated by estimating the gradient using a single sample
  i.e., batch size, $b = 1$

$$w_{1(new)} = w_{1(old)} - \alpha\left(\frac{\partial \mathcal{L}}{\partial w_1}\right) = w_{1(old)} - \alpha\left[\left(\frac{\partial \mathcal{L}}{\partial w_1}\right)_1\right]$$

$$\vdots$$

$$w_{1(new)} = w_{1(old)} - \alpha\left(\frac{\partial \mathcal{L}}{\partial w_1}\right) = w_{1(old)} - \alpha\left[\left(\frac{\partial \mathcal{L}}{\partial w_1}\right)_N\right]$$

- In this case, the gradient is computed using a single training sample.
- Advantage: Very fast to minimise the loss function.
- Disadvantages: Too much variance in gradient calculation and learning might not be stable.
- Convergence cannot be guaranteed.

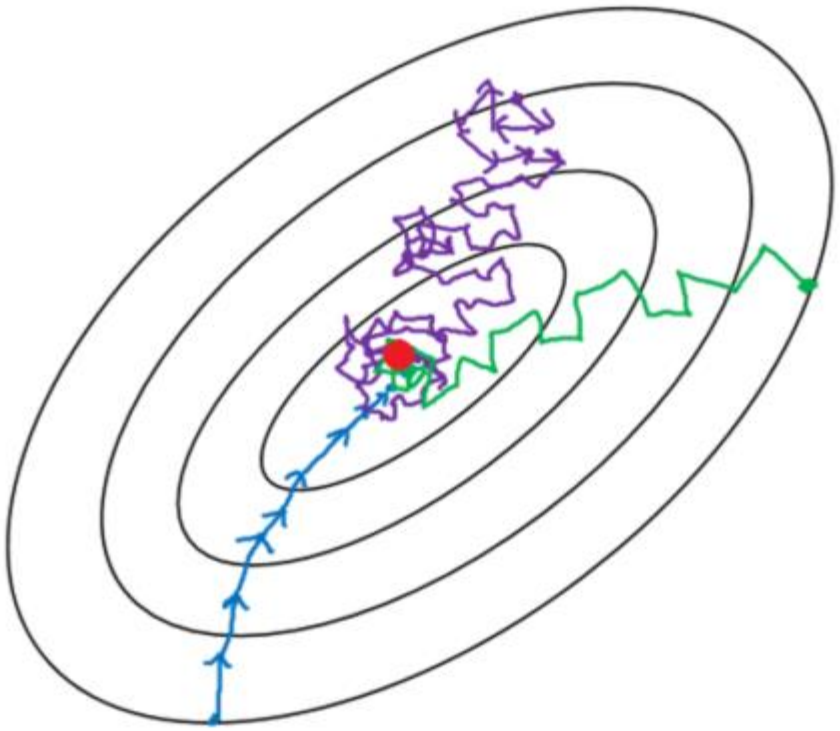
Mini-Batch Gradient Descent

- A balance between batch and stochastic gradient descent
- Parameters are updated by using the gradient computed at a small batch of samples i.e., $1 < b < N$

$$w_{1(new)} = w_{1(old)} - \alpha\left(\frac{\partial \mathcal{L}}{\partial w_1}\right) = w_{1(old)} - \alpha\left[\left(\frac{\partial \mathcal{L}}{\partial w_1}\right)_1 + \cdots + \left(\frac{\partial \mathcal{L}}{\partial w_1}\right)_b\right]$$

- Advantages: Faster than batch gradient descent
- Lesser variance and more stability compared to stochastic gradient descent
- Convergence cannot be guaranteed but most preferred

Variants of Gradient Descent - Visualisation



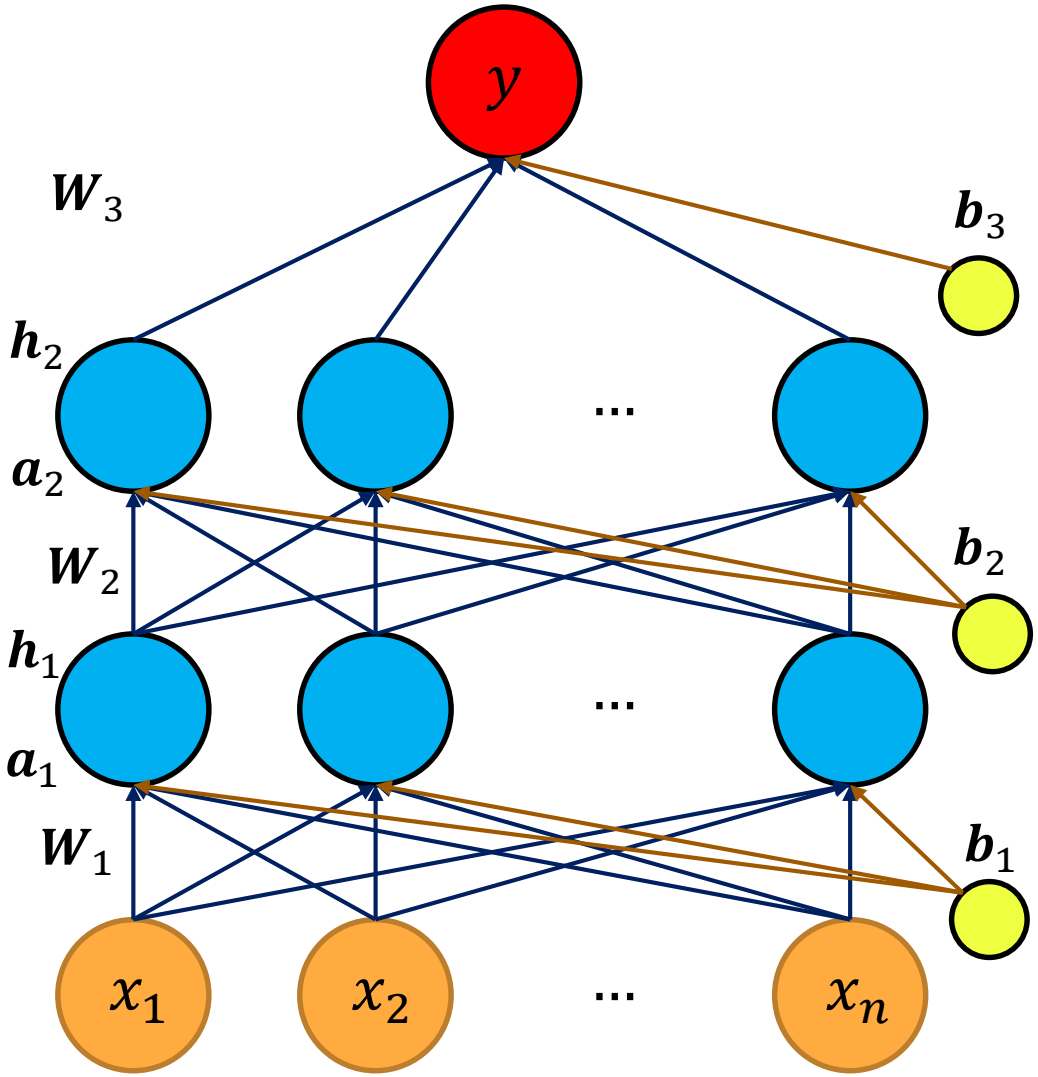
Batch Gradient descent
Mini-Batch Gradient descent
Stochastic Gradient descent

Image Source: medium.com

- Ellipse with higher radius indicates higher loss function

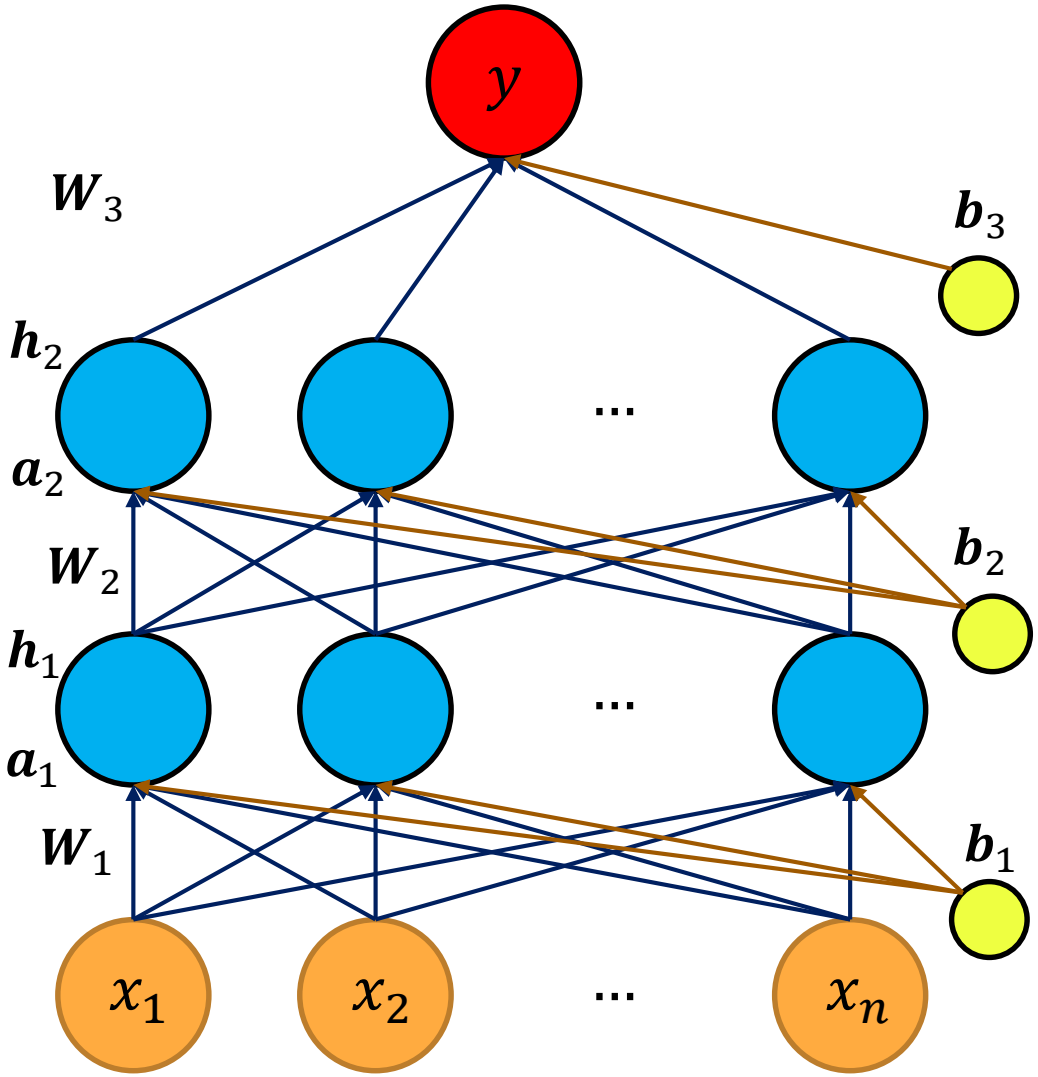
Batch Normalisation

Why Batch Normalisation?



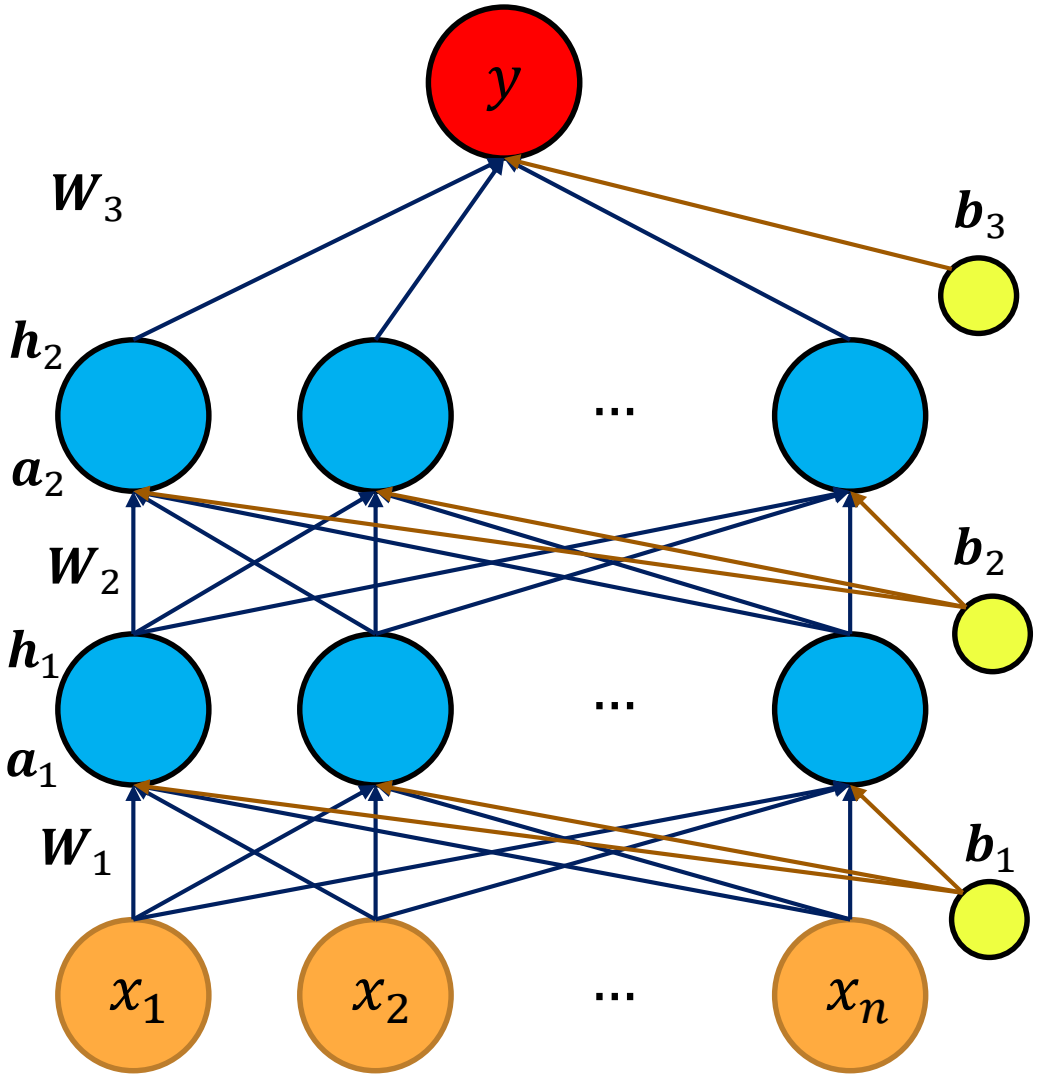
- Intuition: All the samples can be considered to be drawn from a multi-variate distribution
- If batch gradient descent is performed, then the distribution of samples for each batch of inputs remains the same
- Issue: However, with stochastic and mini-batch gradient descent, the distribution varies from one batch to another.

Why Batch Normalisation?



- Illustration: Consider the 2nd hidden layer to which h_1 acts as input.
- Suppose the distribution of h_1 changes with every batch.
- Learning would be hard because the weights do not know which distribution to adjust to.
- How to overcome this issue? – Batch Normalisation

Applying Batch Normalisation



- Inputs to each layer are normalised to be unit gaussians before applying the activation function.
- Mean and variance across set of samples in that batch, are calculated.
- This normalisation step is differentiable and hence we can backpropagate through it.
- Advantage: Learning is much faster and leads to be better convergence.

Ref: Article by Johann Huber - Batch normalization in 3 levels of understanding

Regularisation

Regularisation – Motivation

DNN Model Training Objectives:

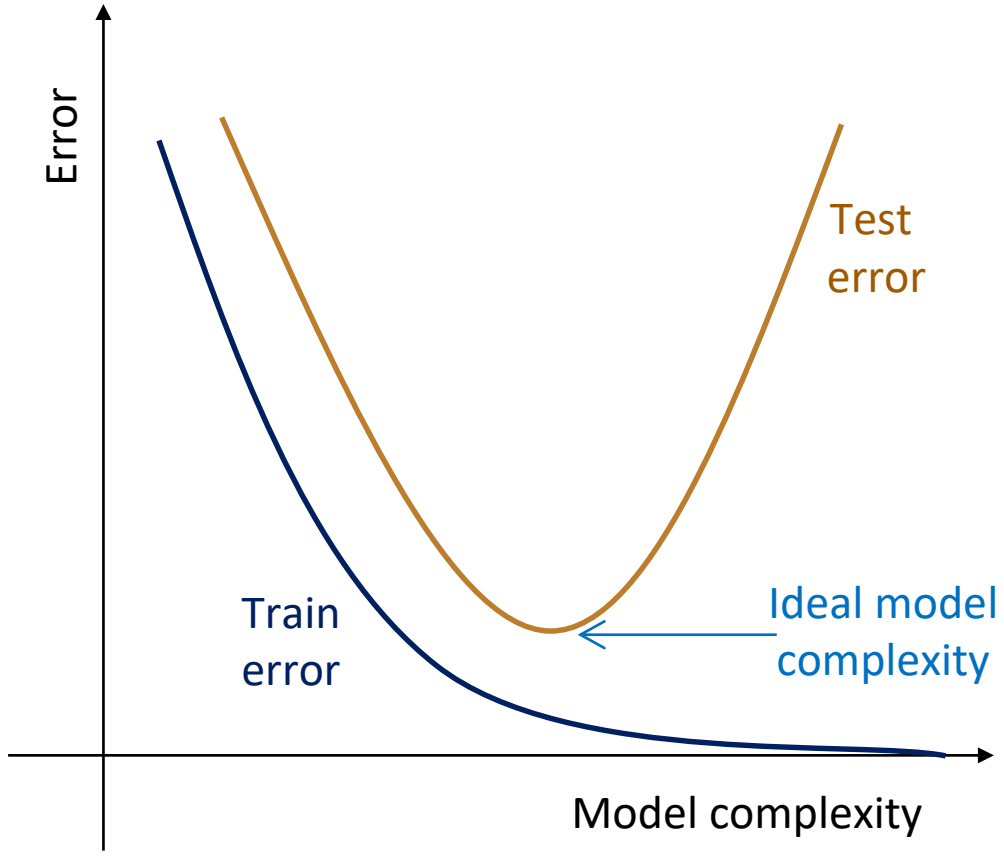
- Finding a model which is a good fit to the given data.
- Model should be able to generalise over unseen data (Test data).
- Prediction error should be low both on training and test data.

Issues:

- In DNNs, there is the issue of overfitting due to many parameters.
- Test error could be high due to overfitting.

Solution:

- To obtain a balanced model, regularisation is performed

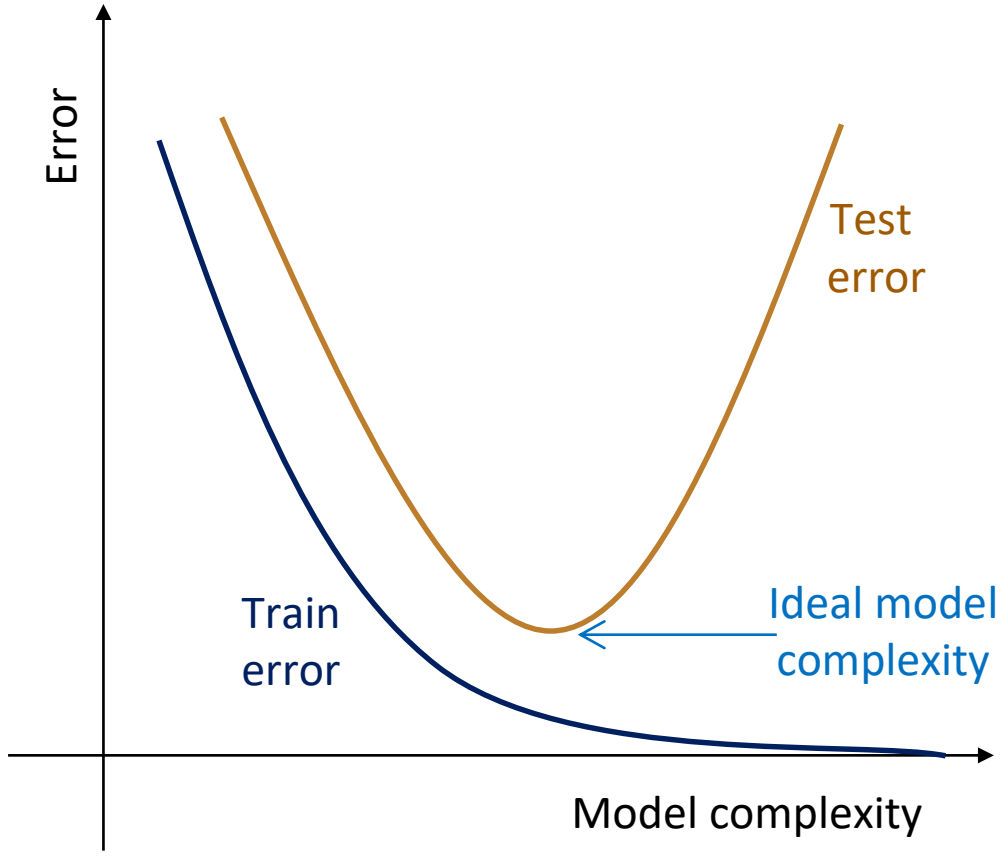


Finding Ideal Model Complexity

- To get near ideal model complexity, an additional component is added to the loss function

$$\mathcal{L}(\theta) + \lambda\Omega(\theta)$$

- $\Omega(\theta)$ is the regularisation term which regularises the model.
- $\Omega(\theta)$ ensures that the model is neither too complex nor too simple.
- λ is the regularisation rate (hyperparameter)
- λ determines how much the model is to be regularised.



Types of Regularization

- l_1 and l_2 regularization
- Early stopping
- Ensemble Methods
- Dropout

l_1 and l_2 Regularisation

- Regularisation term is the l_1 or l_2 norm of the vectors of weights in the neural network.
- This introduces a constraint over the parameters.
- Pushes some of the weights towards zero.
- Some neuron connections become negligible (no impact on output) and overall complexity reduces.

Loss function with l_1 regularization term:

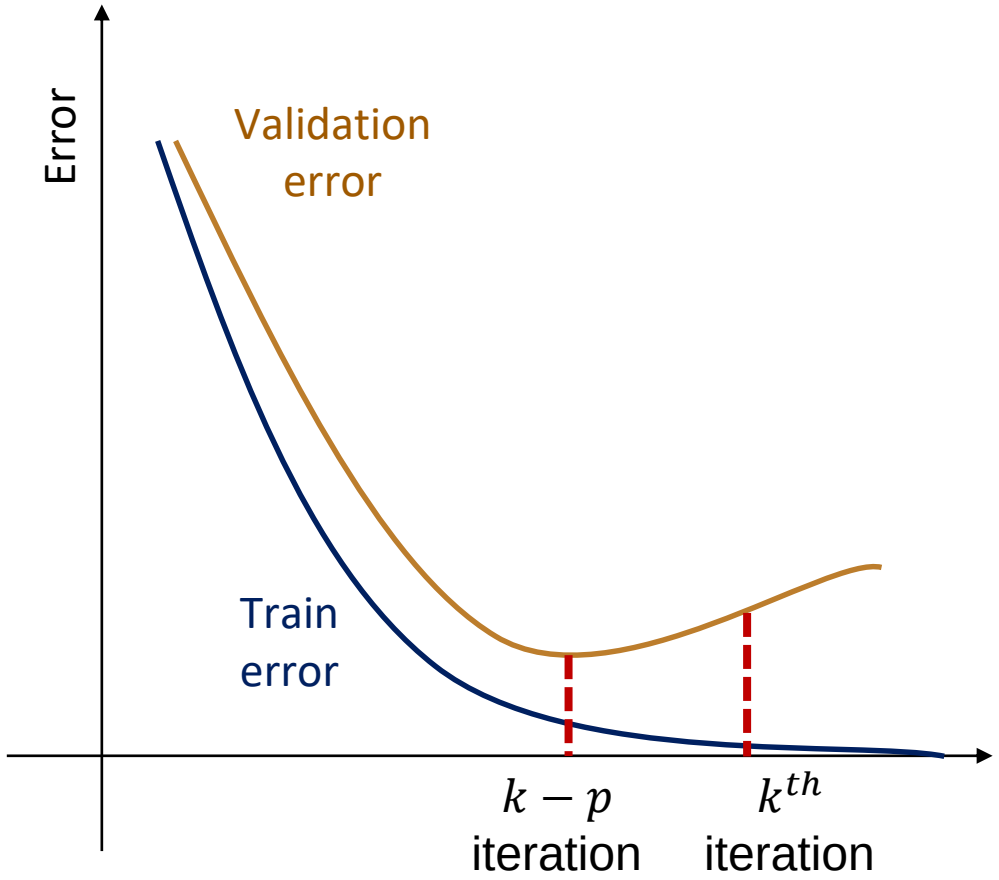
$$\bar{\mathcal{L}}(\theta) = \mathcal{L}(\theta) + \lambda \|W\|_1$$

Loss function with l_2 regularization term:

$$\bar{\mathcal{L}}(\theta) = \mathcal{L}(\theta) + \frac{\lambda}{2} \|W\|_2$$

Early Stopping

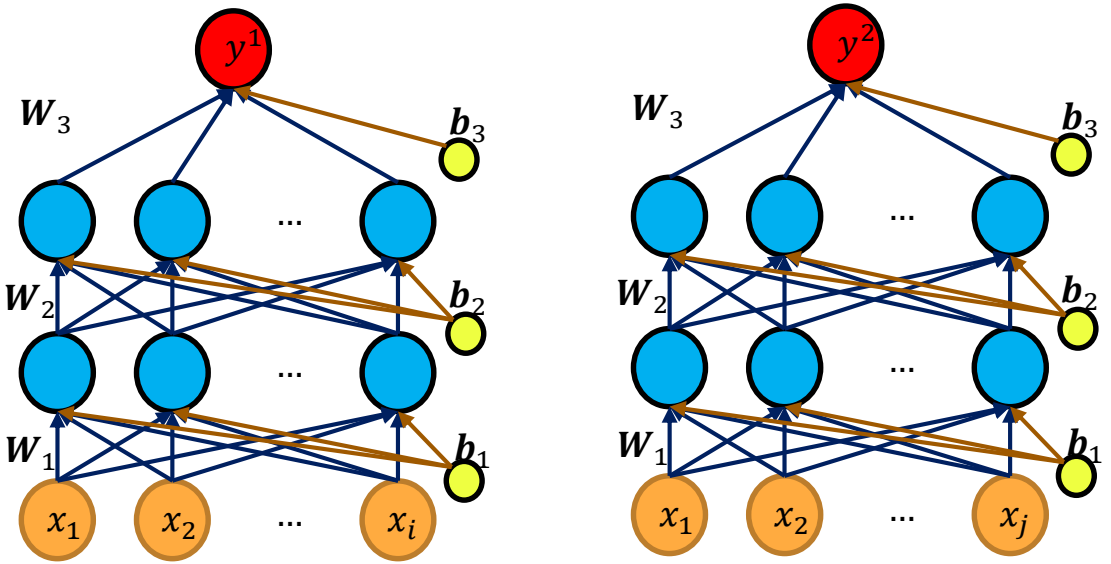
- Prediction error on a validation set is tracked.
- New hyperparameter called patience parameter p is introduced.
- Check if there is improvement in the validation error for p continuous iterations.
- If not stop and take the model prior to those p iterations.



Ensemble Methods

- Different models are trained for the same task using different features, hyperparameters, samples, etc.
- Outputs of these models are combined to reduce the prediction error.
- Computationally very expensive and hence not preferred.

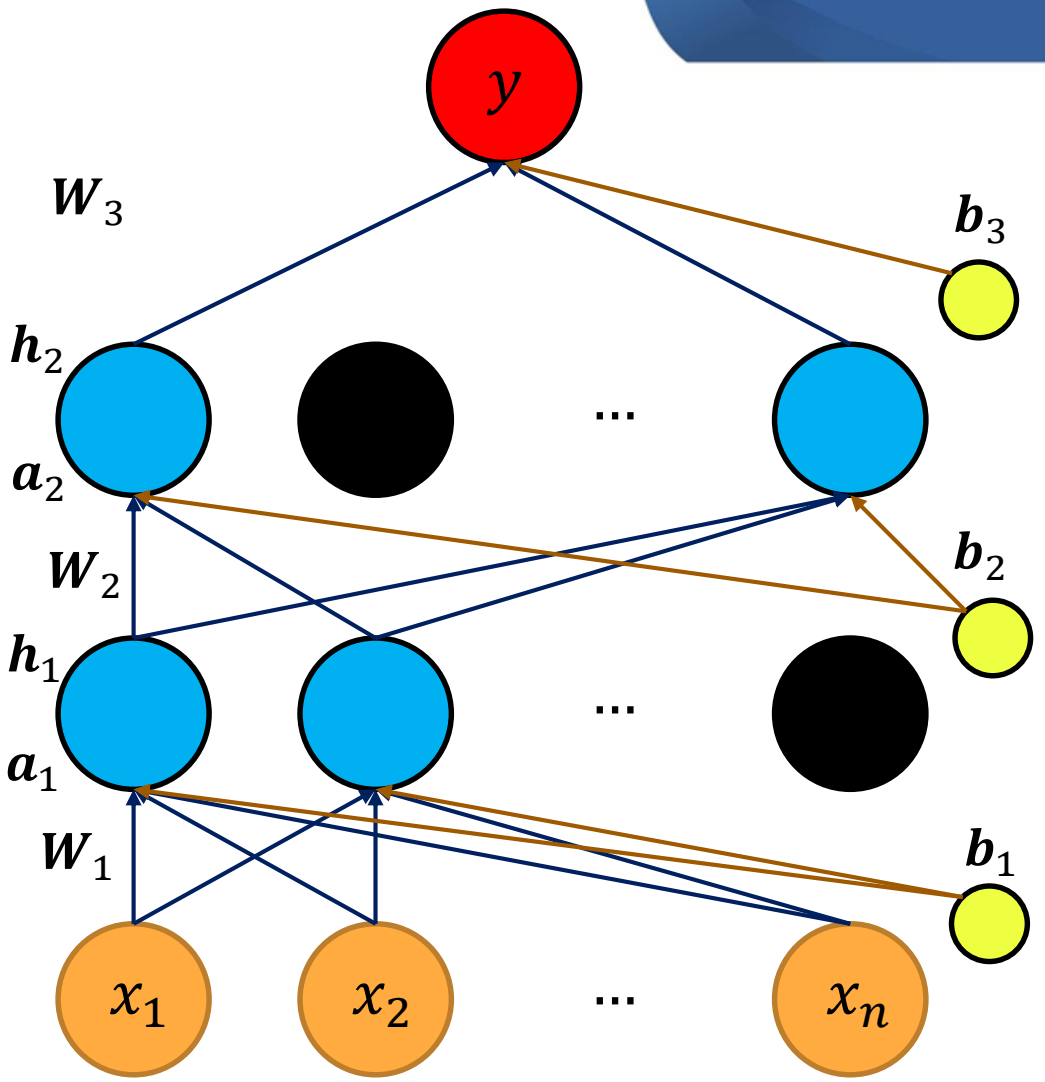
Ensemble of 2 DNN models



$$\hat{y} = \frac{y^1 + y^2}{2}$$

Dropout

- Refers to dropping out of neurons during training.
- For an iteration, some neurons with all their connections are removed (inactive).
- Feed forward calculation and backpropagation happens only with the active connections.
- Update the weights and biases of only active connections.
- Effectively, learning is happening on a different neural network in each iteration.



At i^{th} iteration

DNN Training - Demonstration

- https://playground.tensorflow.org/

Summary

- ✓ Training a DNN implies learning the parameters of the neural network (weights and biases) for modelling a particular function (input-output relation).
- ✓ Training is performed by adjusting the parameters such that the loss between actual and predicted outputs is minimised.
- ✓ Loss is backpropagated through the layers of the neural network following a direction of gradient descent.
- ✓ Different types of gradient descent approaches have been proposed to balance between computational efficiency and accuracy of gradients estimated.
- ✓ Batch Normalisation is performed to ensure stable learning of parameters.
- ✓ To avoid overfitting of DNN models, regularisation is performed during training of DNN models.

More information…

- Blogs – analyticsvidya, towardsdatascience, medium.
- Book - Pattern Recognition and Machine Learning – Christopher Bishop.


operation == "MIRROR_X":
irror_mod.use_x = True
irror_mod.use_y = False
irror_mod.use_z = False
operation == "MIRROR_Y":
irror_mod.use_x = False
irror_mod.use_y = True
irror_mod.use_z = False
operation == "MIRROR_Z":
irror_mod.use_x = False
irror_mod.use_y = False
irror_mod.use_z = True
selection at the end -add
_ob.select= 1
er_ob.select=1
ntext.scene.objects.acti
("Selected" + str(modifier
irror_ob.select = 0
bpy.context.selected_ob
ata.objects[one.name].se
int("please select exact

THANK YOU