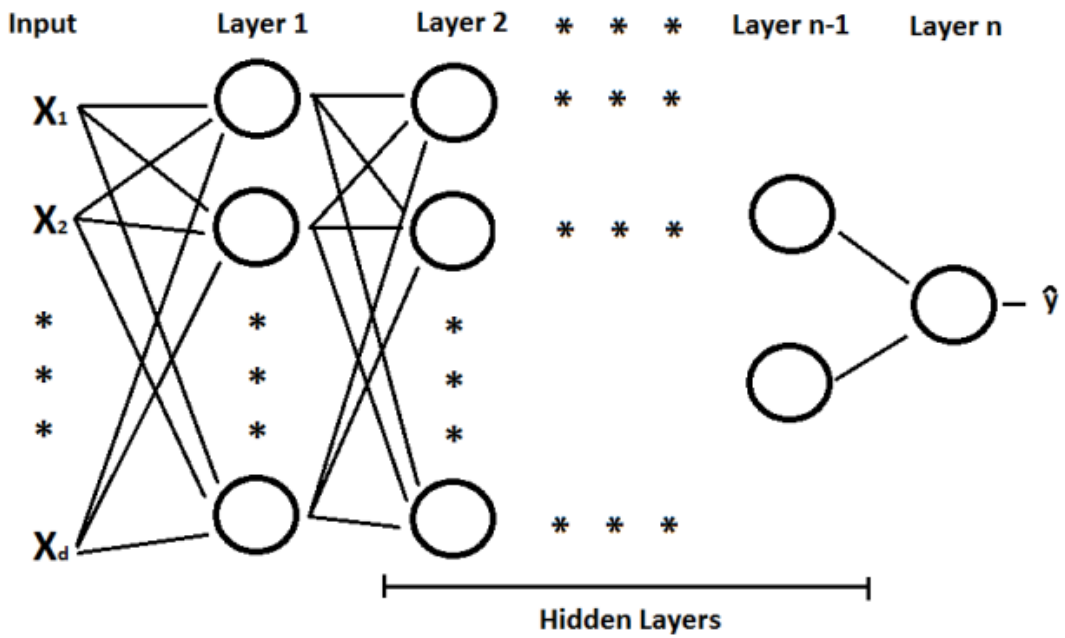
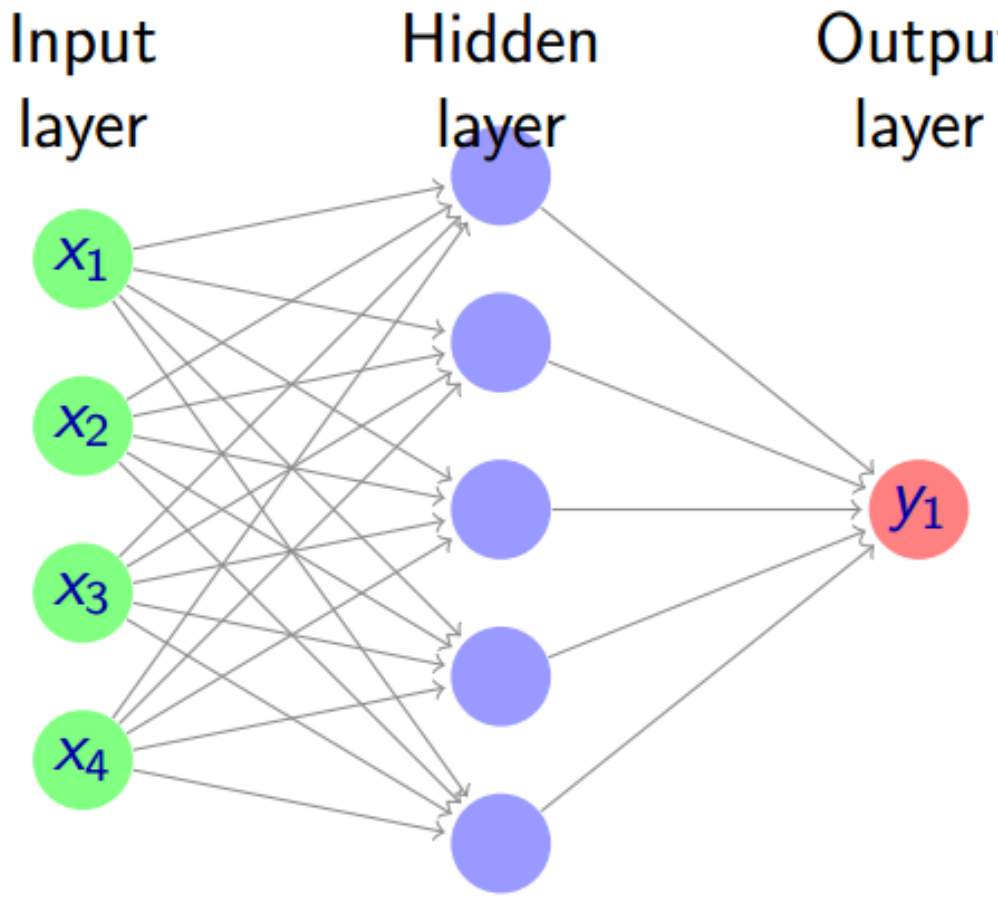
Neural Network

- A neural network is a multistate regression model which is typically represented by a network diagram.

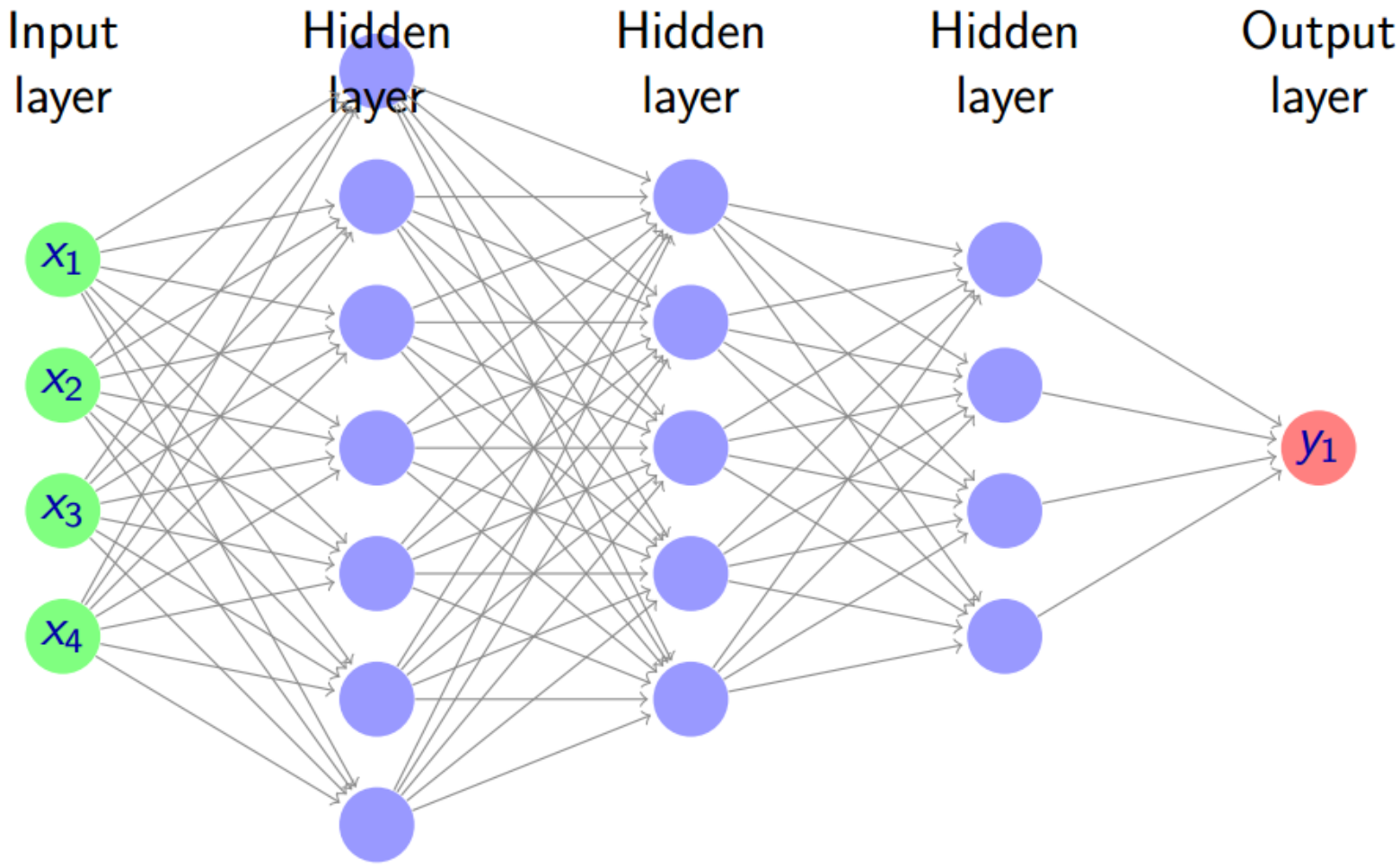


Feed Forward Neural Network

Feedforward Neural Network

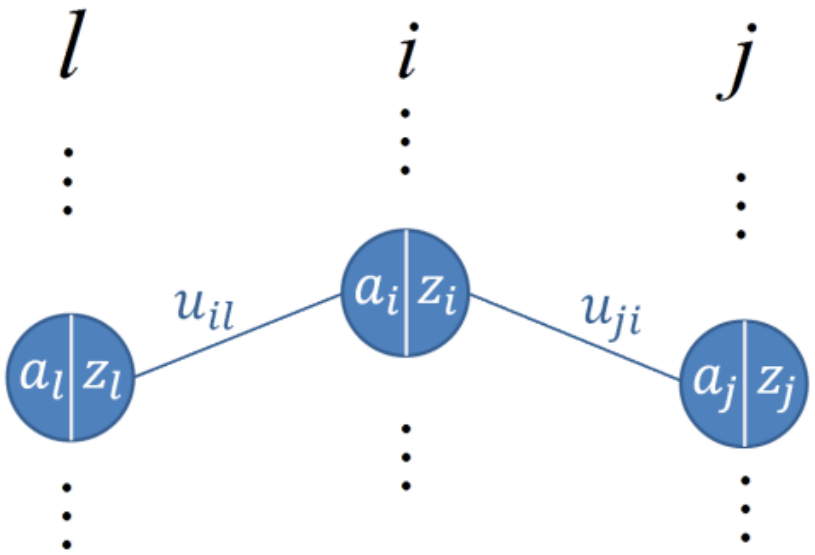


Feedforward Deep Networks



- For regression, typically k $= 1$ (the number of nodes in the last layer), there is only one output unit y_1 at the end.
- For c-class classification, there are typically c units at the end with the c^{th} unit modelling the probability of class c, each y_c is coded as 0-1 variable for the cth class.

Backpropagation



Nodes from three hidden layers within the neural network. Each node is divided into the weighted sum of the inputs and the output of the activation function.

$$a_i = \sum_l z_l u_{il}$$
$$z_i = \sigma(a_i)$$
$$\sigma(a) = \frac{1}{1+e^{-a}}$$

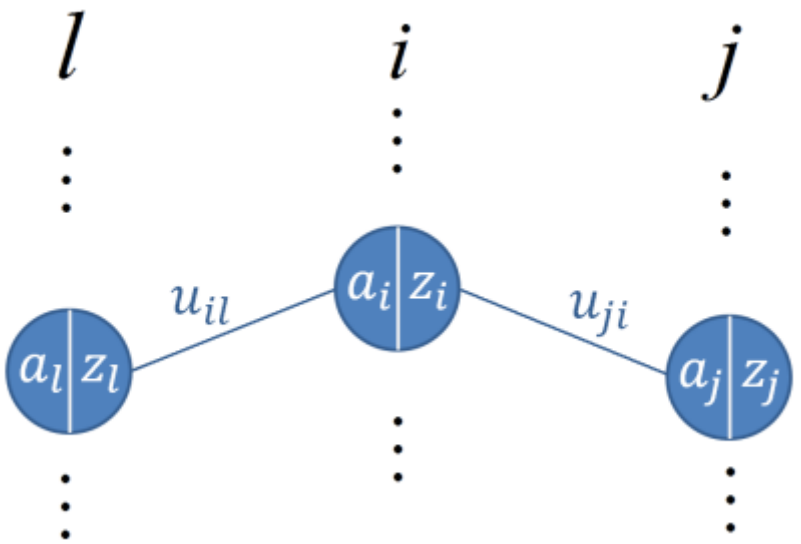
Backpropagation

Take the derivative with respect to weight u_{il}

$$\frac{\partial |y - \hat{y}|^2}{\partial u_{il}}$$

$$\frac{\partial |y - \hat{y}|^2}{\partial u_{il}} = \delta_i \cdot z_l$$

where $\delta_i = \frac{\partial |y - \hat{y}|^2}{\partial a_i}$



Backpropagation

$$\delta_i = \frac{\partial |y - \hat{y}|^2}{\partial a_i} =$$

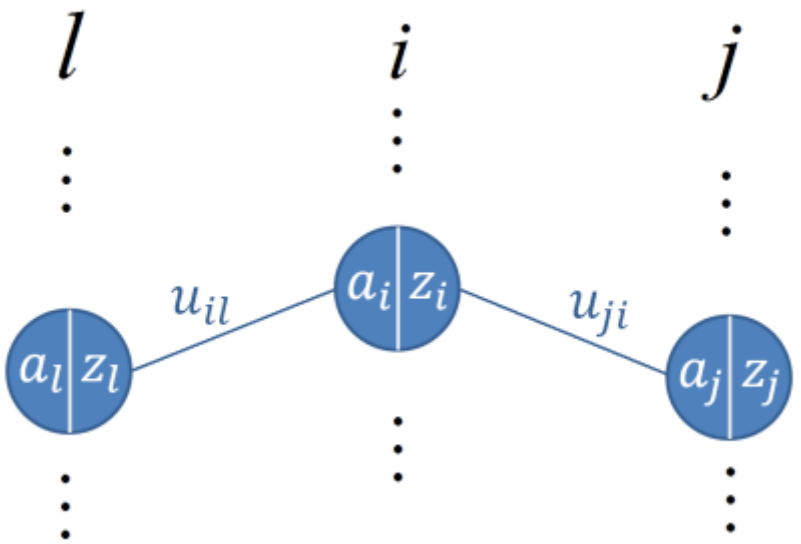
$$\sum_j \frac{\partial |y - \hat{y}|^2}{\partial a_j} \cdot \frac{\partial a_j}{\partial a_i}$$

$$\delta_i = \sum_j \delta_j \cdot \frac{\partial a_j}{\partial z_i} \cdot \frac{\partial z_i}{\partial a_i}$$

$$\delta_i = \sum_j \delta_j \cdot u_{ji} \cdot \sigma'(a_i)$$

where

$$\delta_j = \frac{\partial |y - \hat{y}|^2}{\partial a_j}$$



Backpropagation

Note that if $\sigma(x)$ is the sigmoid function, then
$\sigma'(x) = \sigma(x)(1 - \sigma(x))$

The recursive definition of δ_i
$\delta_i = \sigma'(a_i) \sum_j \delta_j \cdot u_{ji}$

Backpropagation

Now considering δ_k for the output layer:

$$\delta_k = \frac{\partial (y - \hat{y})^2}{\partial a_k} .$$

where $a_k = \hat{y}$

Assume an activation function is not applied in the output layer.

$$\delta_k = \frac{\partial (y - \hat{y})^2}{\partial \hat{y}}$$

$$\delta_k = -2(y - \hat{y})$$

Backpropagation

$$u_{il} \leftarrow u_{il} - \rho \frac{\partial (y - \hat{y})^2}{\partial u_{il}}$$

The network weights are updated using the backpropagation algorithm when each training data point x is fed into the feed forward neural network (FFNN).

Backpropagation

Backpropagation procedure is done using the following steps:

- First arbitrarily choose some random weights (preferably close to zero) for your network.

Backpropagation

- Apply x to the FFNN's input layer, and calculate the outputs of all input neurons.

Backpropagation

- Propagate the outputs of each hidden layer forward, one hidden layer at a time, and calculate the outputs of all hidden neurons.

Backpropagation

- Once x reaches the output layer, calculate the output(s) of all output neuron(s) given the outputs of the previous hidden layer.

Backpropagation

- At the output layer, compute $\delta_k = -2(y_k - \hat{y}_k)$ for each output neuron(s).

Backpropagation

Backpropagation procedure is done using the following steps:

- First arbitrarily choose some random weights (preferably close to zero) for your network.

- Apply x to the FFNN's input layer, and calculate the outputs of all input neurons.

- Propagate the outputs of each hidden layer forward, one hidden layer at a time, and calculate the outputs of all hidden neurons.

- Once x reaches the output layer, calculate the output(s) of all output neuron(s) given the outputs of the previous hidden layer.

- At the output layer, compute $\delta_k = -2(y_k - \hat{y}_k)$ for each output neuron(s).

- Compute each δ_i , starting from $i = k - 1$ all the way to the first hidden layer, where $\delta_i = \sigma'(a_i) \sum_j \delta_j \cdot u_{ji}$.

- Compute $\dfrac{\partial (y - \hat{y})^2}{\partial u_{il}} = \delta_i z_l$ for all weights u_{il} .

- Then update $u_{il}^{\text{new}} \leftarrow u_{il}^{\text{old}} - \rho \cdot \dfrac{\partial (y - \hat{y})^2}{\partial u_{il}}$ for all weights u_{il} .

- Continue for next data points and iterate on the training set until weights converge.

Epochs

It is common to cycle through the all of the data points multiple times in order to reach convergence. An epoch represents one cycle in which you feed all of your datapoints through the neural network. It is good practice to randomized the order you feed the points to the neural network within each epoch; this can prevent your weights changing in cycles. The number of epochs required for convergence depends greatly on the learning rate & convergence requirements used.

Stochastic gradient descent

Suppose that we want to minimize an objective function that is written as a sum of differentiable functions.

$Q(w) = \sum_{i=1}^{n} Q_i(w)$

Each term Q_i is usually associated with the i_th data point.

Standard gradient descent (batch gradient descent):
$w = w - \eta \nabla Q(w) = w - \eta \sum_{i=1}^{n} \nabla Q_i(w)$

where η is the learning rate (step size).

Stochastic gradient descent

Stochastic gradient descent (SGD) considers only a subset of summand functions at every iteration.

This can be quite effective for large-scale problems.

Bottou, Leon; Bousquet, Olivier (2008). The Tradeoffs of Large Scale Learning. Advances in Neural Information Processing Systems 20. pp. 161168.

The gradient of $Q(w)$ is approximated by a gradient at a single example: $w = w - \eta \nabla Q_i(w)$.

This update needs to be done for each training example.

Several passes might be necessary over the training set until the algorithm converges.

η might be adaptive.

Stochastic gradient descent

- Choose an initial value for w and η.
- Repeat until converged
  - Randomly shuffle data points in the training set.
  - For $i = 1, 2, ..., n$, do:
  - $w = w - \eta \nabla Q_i(w)$.

Example

Suppose $y = w_1 + w_2 x$

The objective function is:

$$Q(w) = \sum_{i=1}^{n} Q_i(w) = \sum_{i=1}^{n} (w_1 + w_2 x_i - y_i)^2.$$

Update rule will become:

$$\begin{bmatrix} w_1 \\ w_2 \end{bmatrix} := \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} - \eta \begin{bmatrix} 2(w_1 + w_2 x_i - y_i) \\ 2x_i(w_1 + w_2 x_i - y_i) \end{bmatrix}.$$

Example from Wikipedia

Mini-batches

Batch gradient decent uses all n data points in each iteration.

Stochastic gradient decent uses 1 data point in each iteration.

Mini-batch gradient decent uses b data points in each iteration.

b is a parameter called Mini-batch size.

Mini-batches

- Choose an initial value for w and η.
- Say $b = 10$
- Repeat until converged
  - Randomly shuffle data points in the training set.
  - For $i = 1, 11, 21, ..., n - 9$, do:
  - $w = w - \eta \sum_{k=i}^{i+9} \nabla Q_i(w)$.

Tuning η

If η is too high, the algorithm diverges.

If η is too low, makes the algorithm slow to converge.

A common practice is to make η_t a decreasing function of the iteration number t. e.g. $\eta_t = \frac{constant1}{t+constant2}$

The first iterations cause large changes in the w, while the later ones do only fine-tuning.