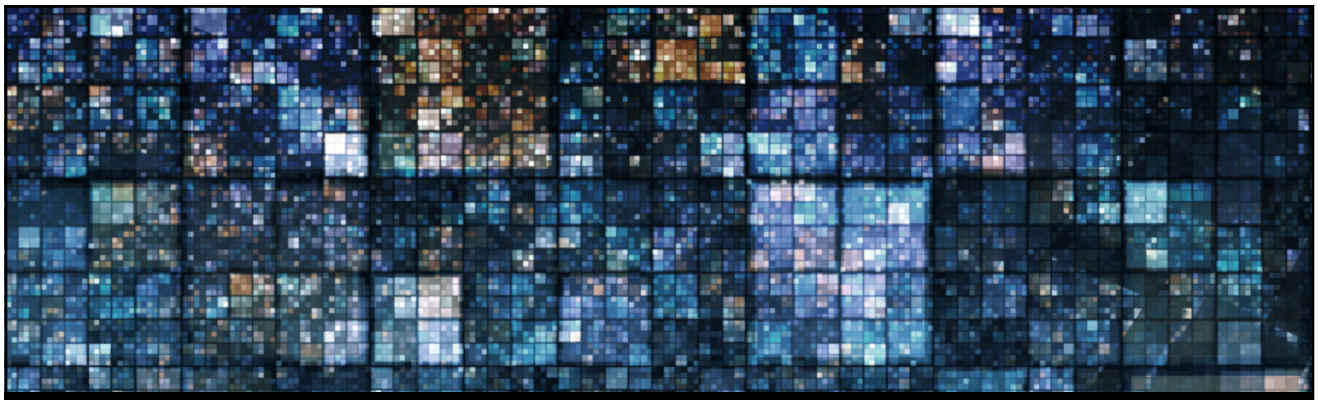
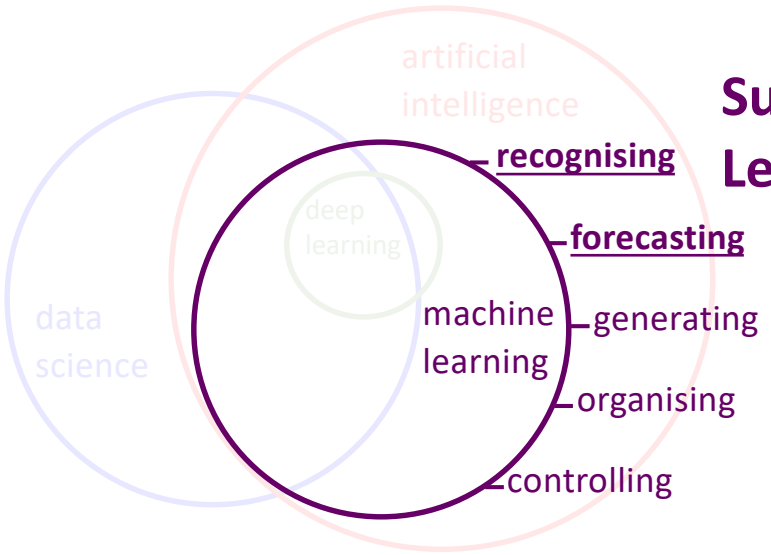


COMP47590
Advanced Machine Learning
Deep Learning - ANNs 3

Dr. Brian Mac Namee

Information

Email:	Brian.MacNamee@ucd.ie
Course Materials:	All material posted on UCD CS moodle https://csmoodle.ucd.ie/moodle/course/view.php?id=756
	Enrolment key **UCDAdvML2019**

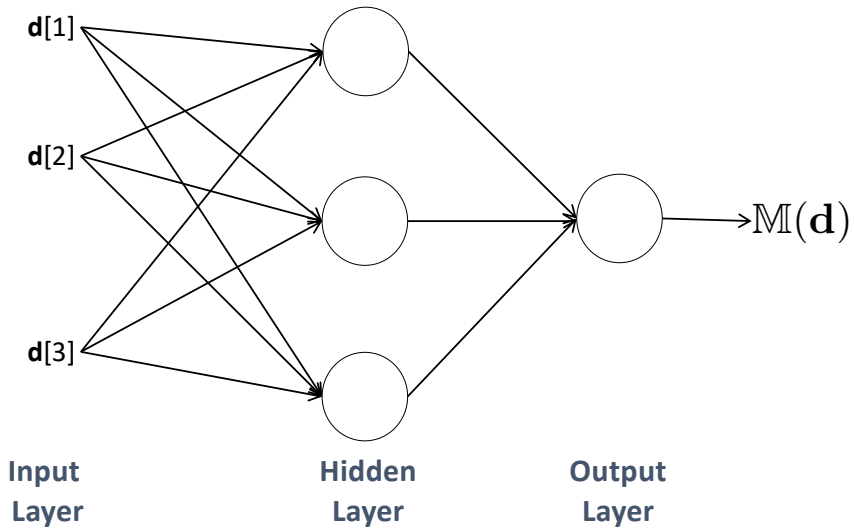


Supervised Learning

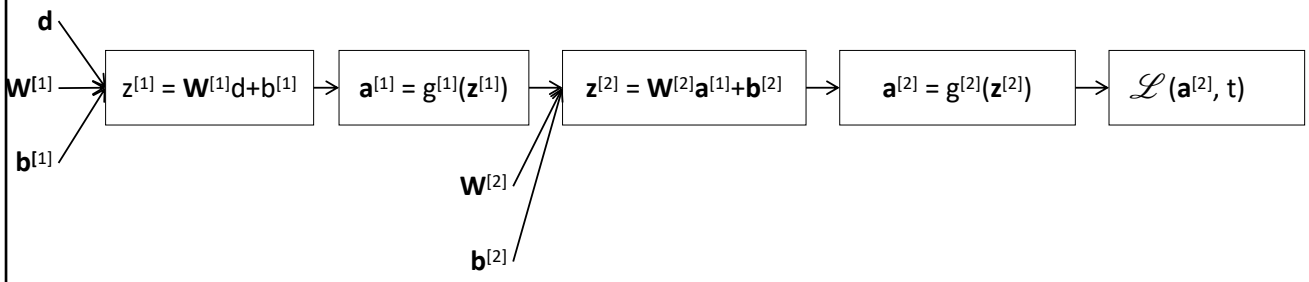
- recognising
- forecasting
- generating
- organising
- controlling

MULTI LAYER PERCEPTRON

Multi Layer Perceptron



MLP: Forward Propagation



MLP: Backward Propagation

$d\mathbf{z}^{[2]} = \mathbf{a}^{[2]} - \mathbf{t}$

$d\mathbf{W}^{[2]} = d\mathbf{z}^{[2]}\mathbf{a}^{[1]T}$

$d\mathbf{b}^{[2]} = d\mathbf{z}^{[2]}$

$d\mathbf{z}^{[1]} = \mathbf{W}^{[2]T}d\mathbf{z}^{[2]} * g^{[1]\prime}(\mathbf{z}^{[1]})$

$d\mathbf{W}^{[1]} = d\mathbf{z}^{[1]}\mathbf{d}^{T}$

$d\mathbf{b}^{[1]} = d\mathbf{z}^{[1]}$

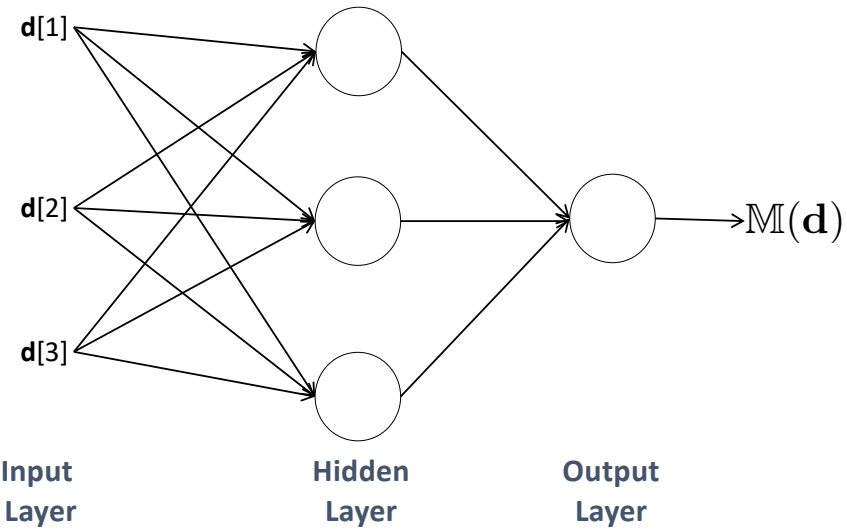
MLP: Gradient Descent

We only need to update our weight update rules to take account of the extra layers

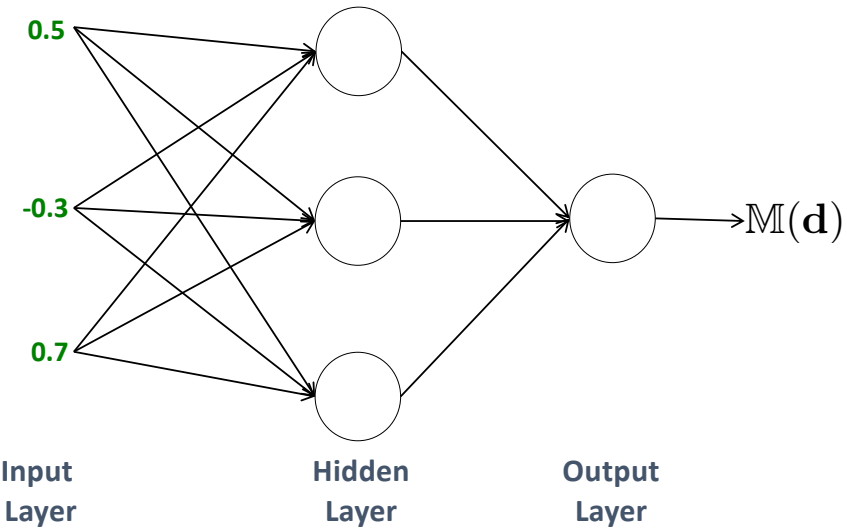
$\mathbf{W}^{[i]} = \mathbf{W}^{[i]} - \alpha\, d\mathbf{W}^{[i]}$

$\mathbf{b}^{[i]} = \mathbf{b}^{[i]} - \alpha\, d\mathbf{b}^{[i]}$

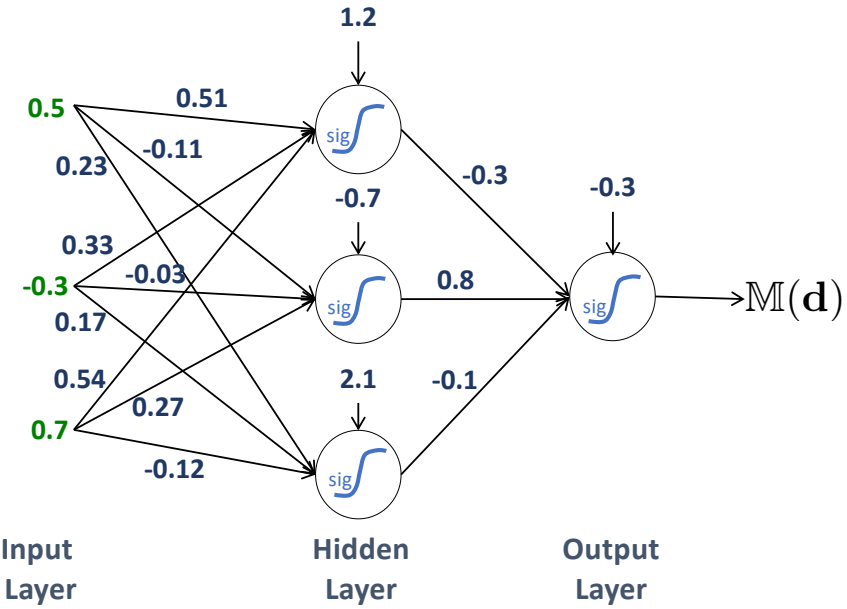
MLP: Exercise



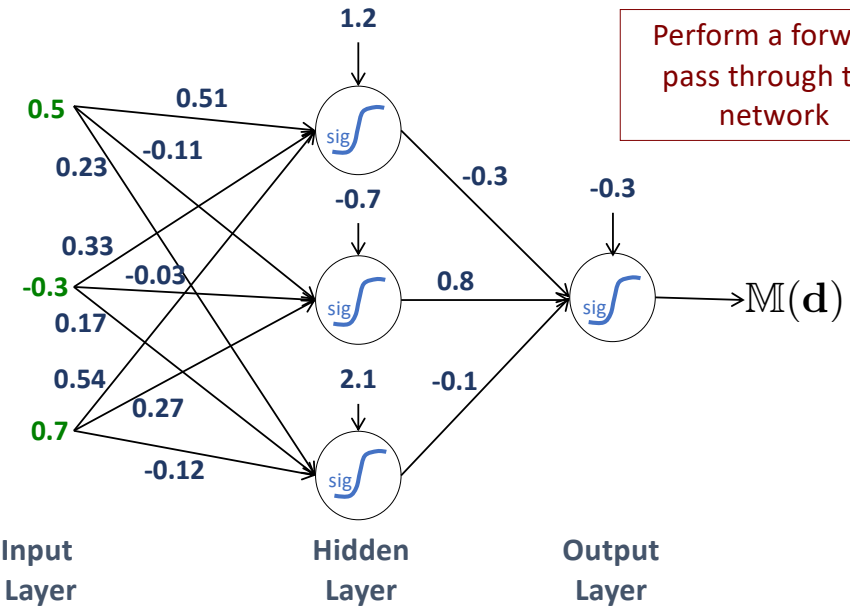
MLP: Exercise



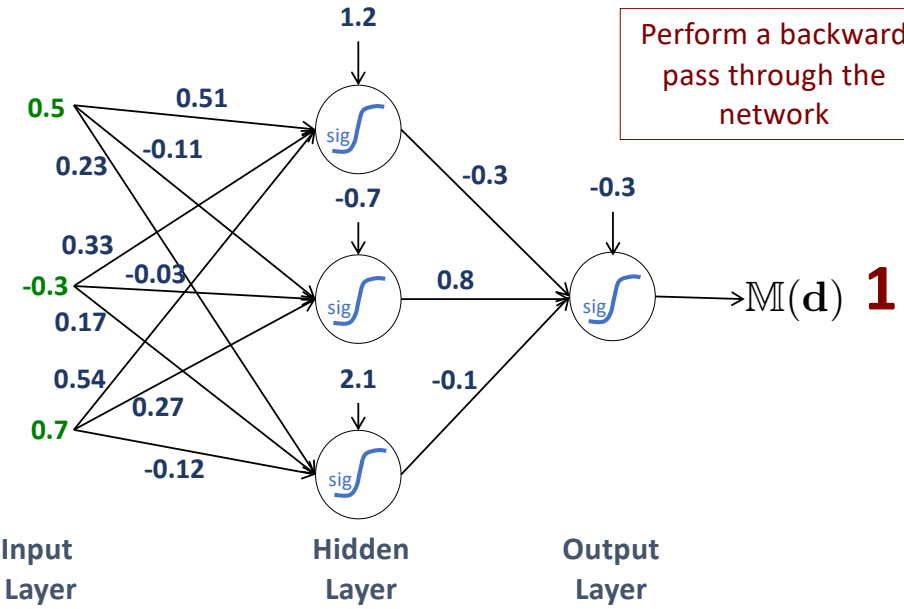
MLP: Exercise



MLP: Exercise

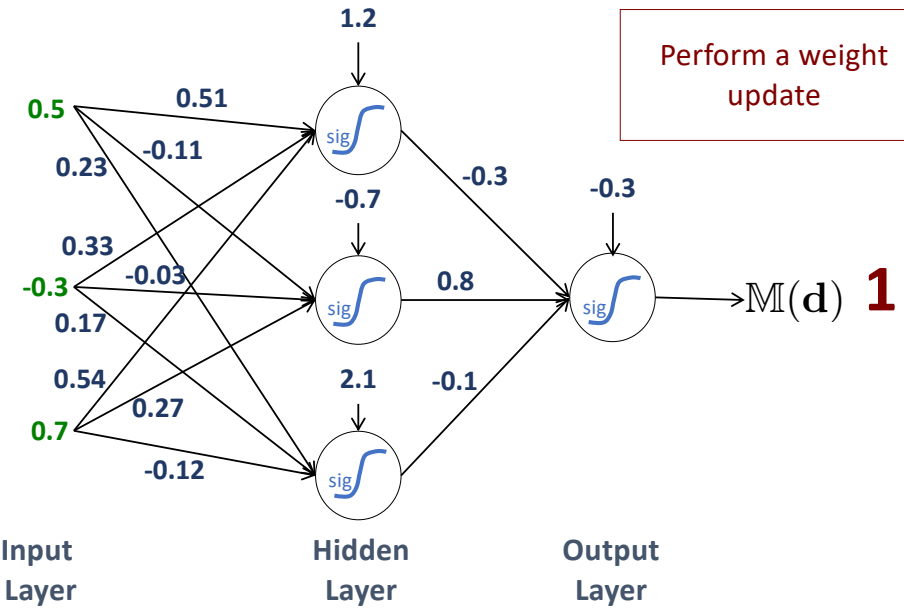


MLP: Exercise



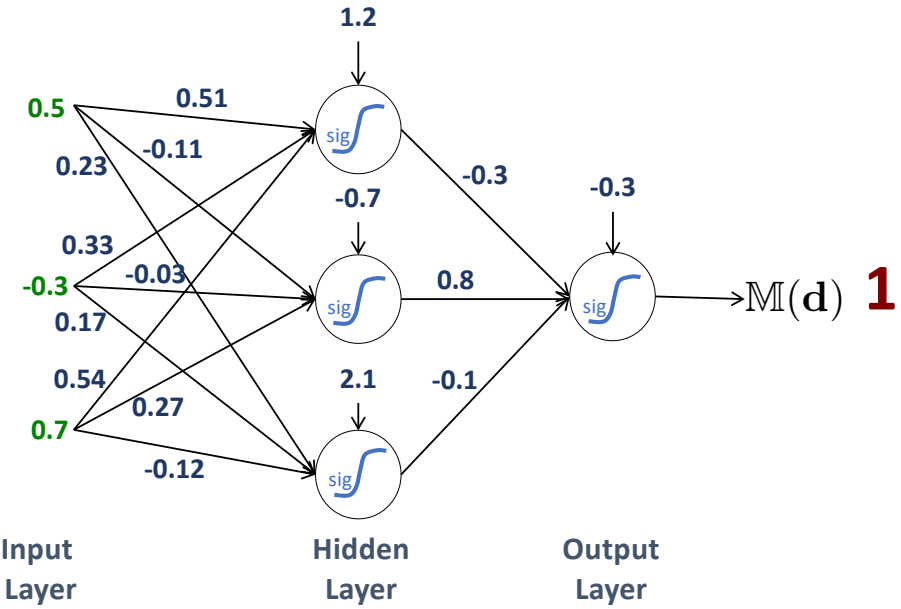
Perform a backward pass through the network

MLP: Exercise



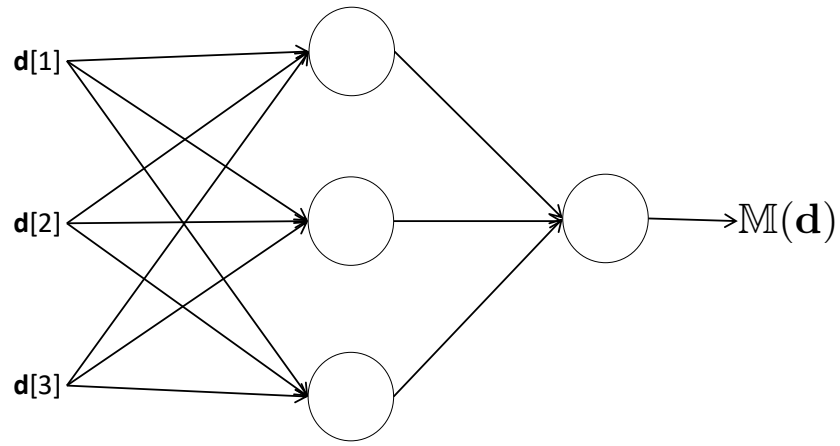
Perform a weight update

MLP: Exercise

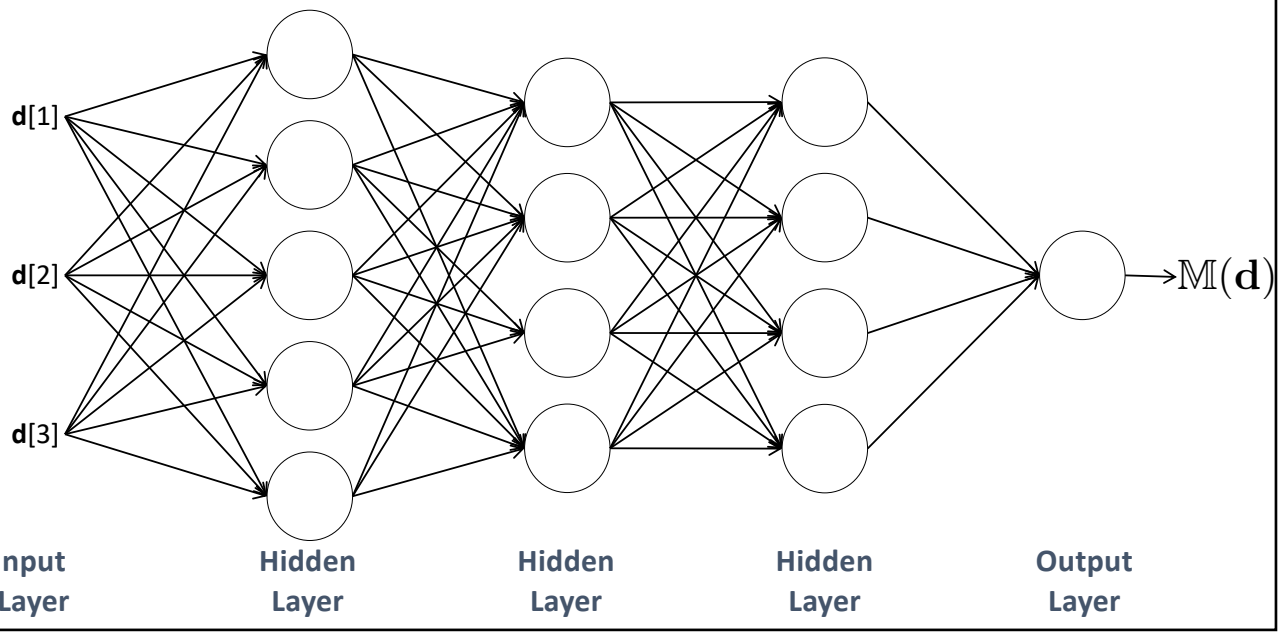


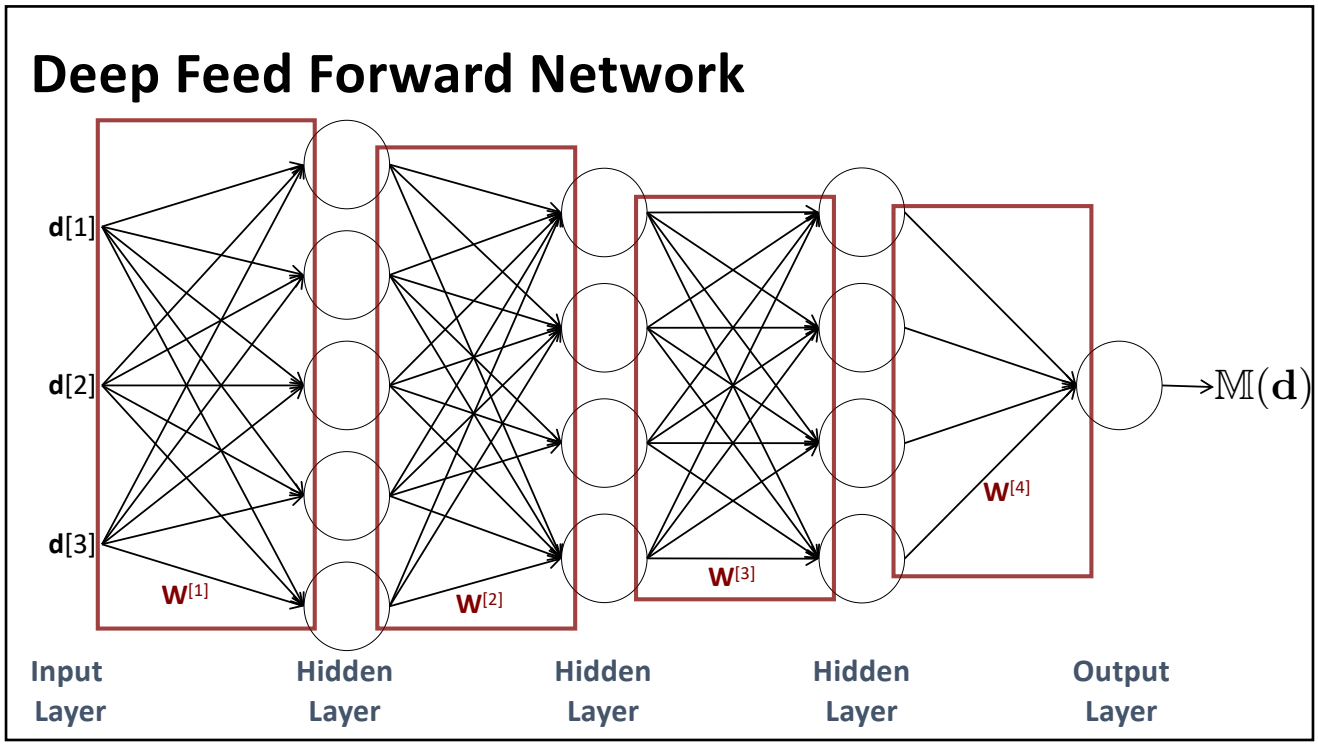
BACKPROPOGATION OF ERRORS

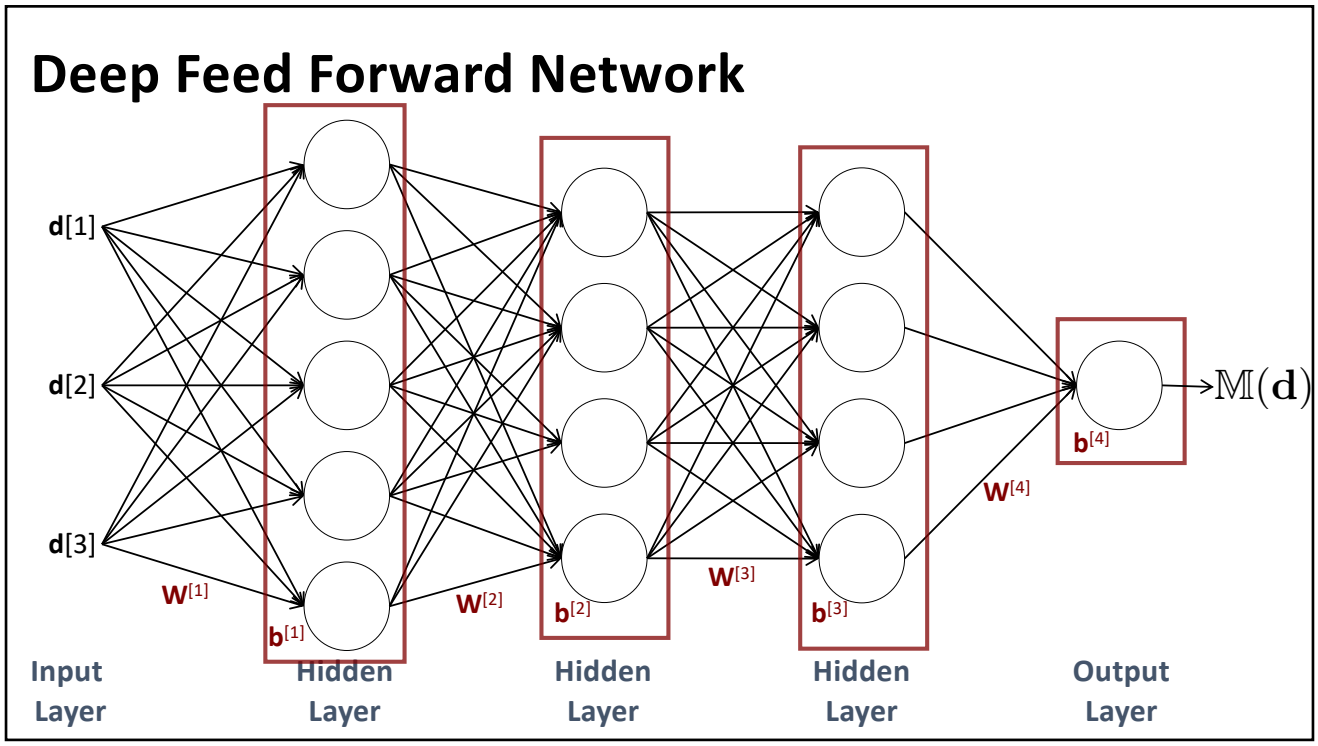
Multi Layer Perceptron

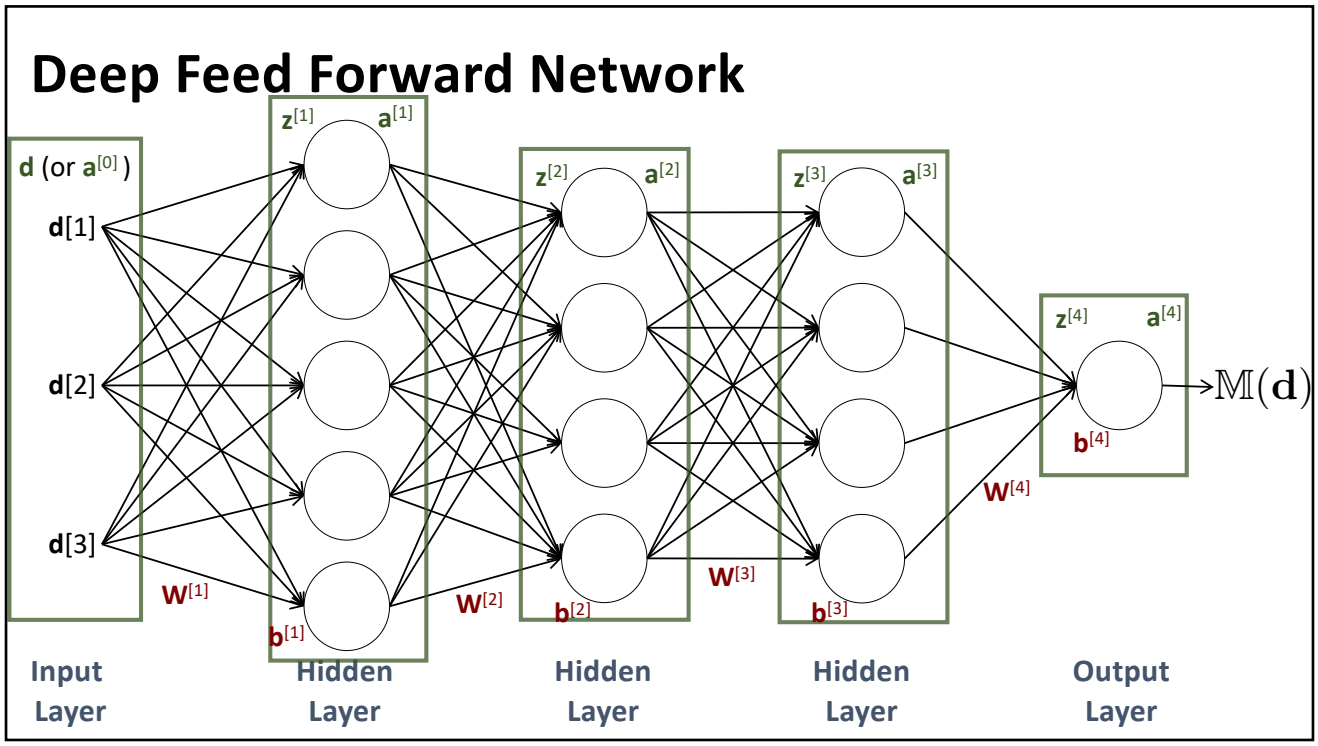


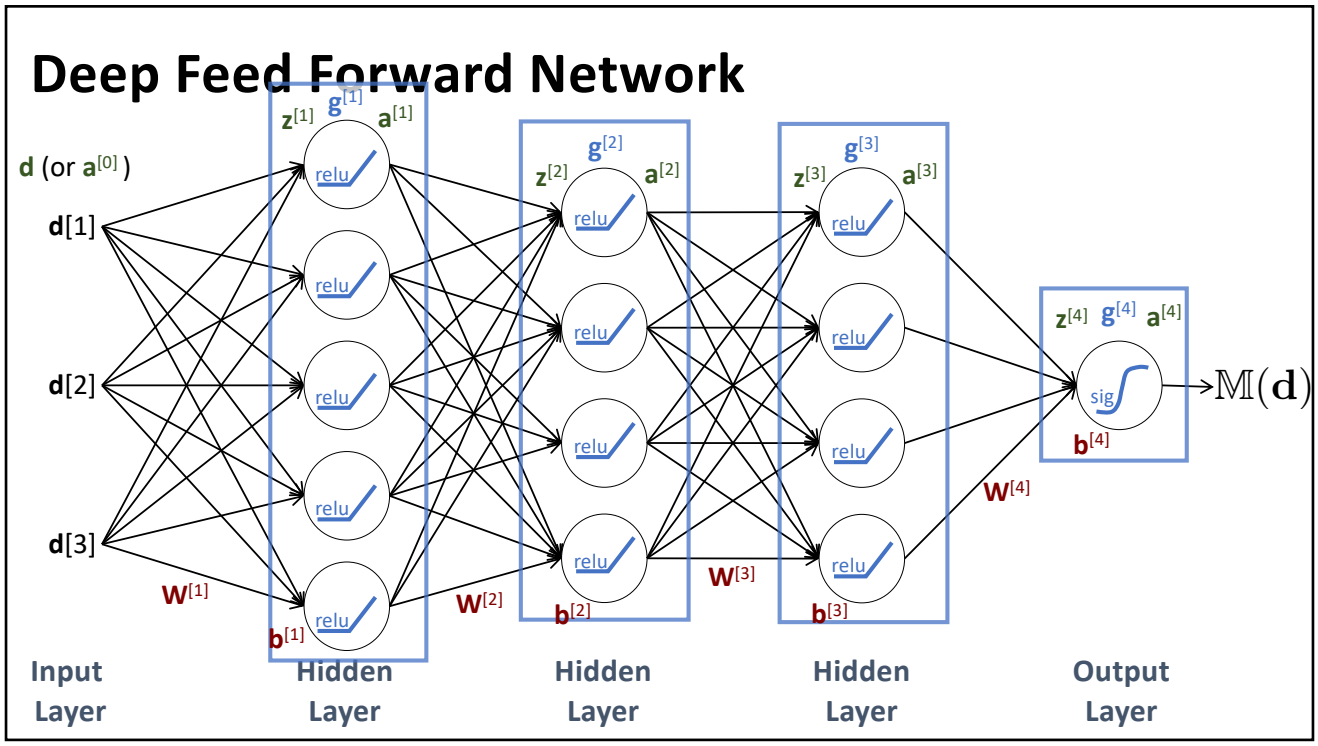
Deep Feed Forward Network



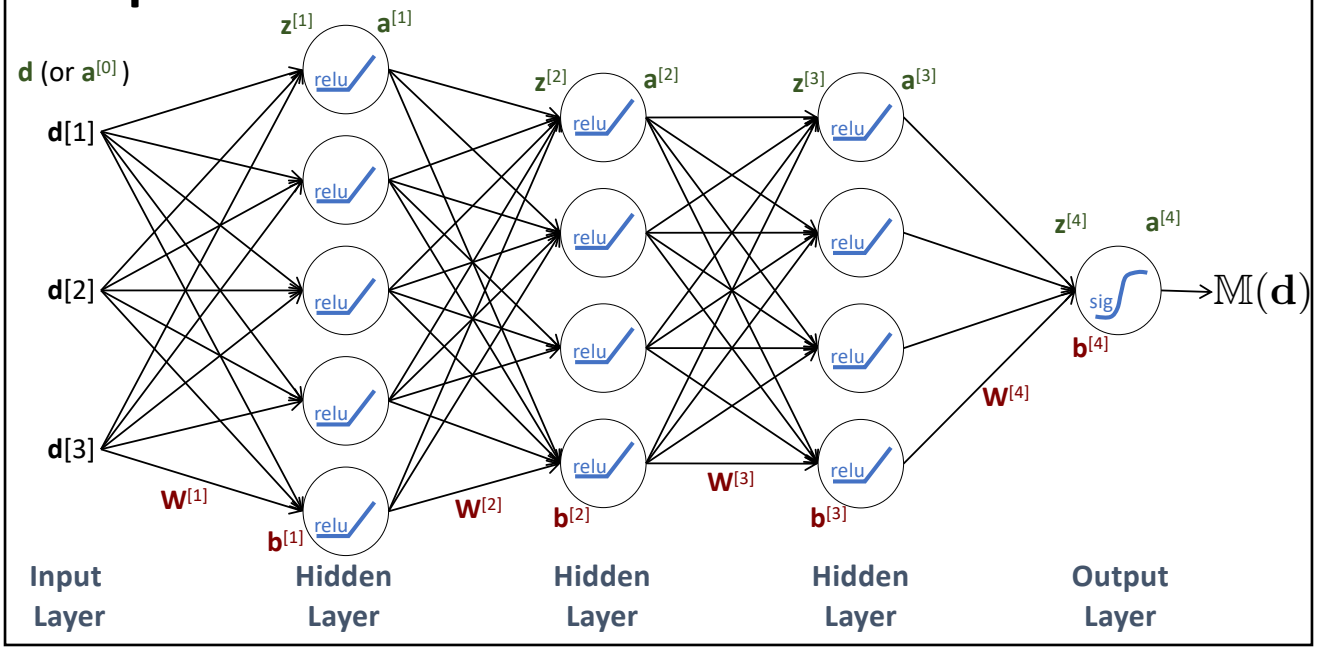

Deep Feed Forward Network
d[1]
d[2]
d[3]
W[1]
W[2]
W[3]
W[4]
$\mathbb{M}(\mathbf{d})$
Input Layer
Hidden Layer
Hidden Layer
Hidden Layer
Output Layer


Deep Feed Forward Network
d[1]
d[2]
d[3]
W[1]
b[1]
W[2]
b[2]
W[3]
b[3]
W[4]
b[4]
$\mathbb{M}(\mathbf{d})$
Input Layer
Hidden Layer
Hidden Layer
Hidden Layer
Output Layer

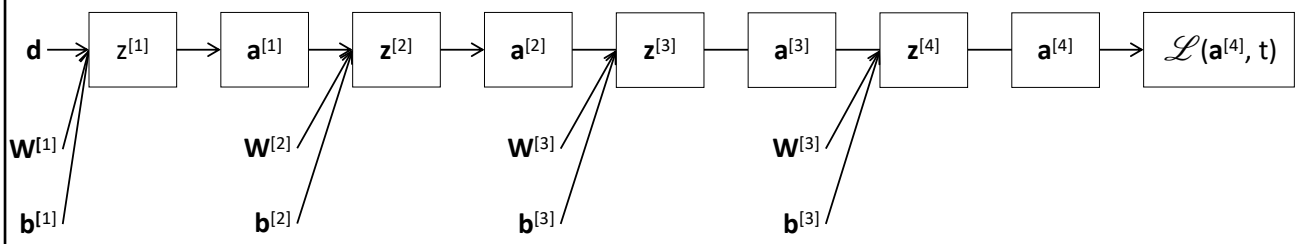

Deep Feed Forward Network
d (or a[0])
d[1]
d[2]
d[3]
z[1]
a[1]
z[2]
a[2]
z[3]
a[3]
z[4]
a[4]
W[1]
b[1]
W[2]
b[2]
W[3]
b[3]
W[4]
b[4]
𝕄(d)
Input Layer
Hidden Layer
Hidden Layer
Hidden Layer
Output Layer


Deep Feed Forward Network
d (or a[0])
d[1]
d[2]
d[3]
g[1]
z[1]
a[1]
g[2]
z[2]
a[2]
g[3]
z[3]
a[3]
g[4]
z[4]
a[4]
relu
sig
W[1]
b[1]
W[2]
b[2]
W[3]
b[3]
W[4]
b[4]
𝕄(d)
Input Layer
Hidden Layer
Hidden Layer
Hidden Layer
Output Layer

Deep Feed Forward Network



Deep Feed Forward Network



Deep Feed Forward Network
Forward Propagation

Require: L, network depth
Require: $\mathbf{W}^{[i]}$, $i \in \{1, ..., L\}$, weight matrices for each layer
Require: $\mathbf{b}^{[i]}$, $i \in \{1, ..., L\}$, bias terms for each layer
Require: $\mathbf{d}$, input descriptive features
Require: t, the target feature
$\mathbf{a}^{[0]} = \mathbf{d}$
for l = 1 to L:
    $\mathbf{z}^{[l]} = \mathbf{W}^{[l]}\, \mathbf{a}^{[l-1]} + \mathbf{b}^{[l]}$
    $\mathbf{a}^{[l]} = g^{[l]}(\mathbf{z}^{[l]})$
$\mathbb{M}(\mathbf{d}) = \mathbf{a}^{[L]}$
Calculate $\mathscr{L}(\mathbb{M}(\mathbf{d}), t)$

Deep Feed Forward Network
Backward Propagation

Require a forward pass through the network
Require: L, network depth
Require: $\mathbf{W}^{[i]}$, $i \in \{1, ..., L\}$, weight matrices for each layer
Require: $\mathbf{b}^{[i]}$, $i \in \{1, ..., L\}$, bias terms for each layer
Require: t, the target feature
Calculate $d\mathbf{a}^{[L]}$ # The derivative of the loss function
for l = L to 1:
    $d\mathbf{z}^{[l]} = d\mathbf{a}^{[l]} * g^{[l]\prime}(\mathbf{z}^{[l]})$
    $d\mathbf{W}^{[l]} = d\mathbf{z}^{[l]} \mathbf{a}^{[l-1]T}$
    $d\mathbf{b}^{[l]} = d\mathbf{z}^{[l]}$
    $d\mathbf{a}^{[l-1]} = \mathbf{W}^{[l]T} d\mathbf{z}^{[l]}$

Deep Feed Forward Network Backward Propagation

Require a forward pass through the network
Require: L, network depth
Require: $\mathbf{W}^{[i]}$, $i \in \{1, ..., L\}$, weight matrices for each layer
Require: $\mathbf{b}^{[i]}$, $i \in \{1, ..., L\}$, bias terms for each layer
Require: t, the target feature
$d\mathbf{a}^{[L]} = -t / \mathbf{a}^{[L]} + (1 - t)/(1 - \mathbf{a}^{[L]})$ # The derivative of log loss
for l = L to 1:

$d\mathbf{z}^{[l]} = d\mathbf{a}^{[l]} * g^{[l]\prime}(\mathbf{z}^{[l]})$
$d\mathbf{W}^{[l]} = d\mathbf{z}^{[l]}\mathbf{a}^{[l-1]T}$
$d\mathbf{b}^{[l]} = d\mathbf{z}^{[l]}$
$d\mathbf{a}^{[l-1]} = \mathbf{W}^{[l]T}d\mathbf{z}^{[l]}$

GRADIENT DESCENT

Gradient Descent Algorithm

Require: $\mathbb{M}_{\mathbf{W},\mathbf{b}}$,a network upon which to perform gradient descent

Require: $\mathcal{D}$, a training dataset of *m* training instances {($\mathbf{d}_1$, t_1), ($\mathbf{d}_2$, t_2), ... , ($\mathbf{d}_m$, t_m)

Require: $\mathscr{L}(\mathbb{M}(\mathbf{d}), \mathbf{t})$, a loss function to calculate the loss of a prediction

Require: $J(\mathbb{M}(\mathcal{D}_{\mathbf{d}}), \mathcal{D}_{\mathbf{t}})$, a cost function to aggregate losses

Gradient Descent Algorithm (Stochastic)

Choose random weights

Until convergence

- Shuffle all training examples
- For each training instance
  - Perform a forward pass through the network
  - Perform a backward pass through the network
  - Update weights and biases using update rule

Gradient Descent Algorithm (Stochastic) Update Rule

We apply the same simple update rule at every layer

$$\mathbf{W}^{[i]} = \mathbf{W}^{[i]} - \alpha \, d\mathbf{W}^{[i]}$$
$$\mathbf{b}^{[i]} = \mathbf{b}^{[i]} - \alpha \, d\mathbf{b}^{[i]}$$

Gradient Descent Algorithm (Batch)

Choose random weights

Until convergence

- Set all gradient sums to 0
- For each training instance
  - Perform a forward pass through the network
  - Perform a backward pass through the network
  - Update gradient sum for each weight and bias term
- Update weights and biases using update rule

Cost Function

$$J(\mathbf{w}, b) = \frac{1}{m} \sum_{i=1}^{m} \mathscr{L}\left(\mathbb{M}(\mathbf{d}_i), t_i\right)$$

$$\frac{\partial J(\mathbf{w}, b)}{\partial \mathbf{w}[j]} = \frac{1}{m} \sum_{i=1}^{m} \frac{\partial \mathscr{L}\left(\mathbb{M}(\mathbf{d}_i), t_i\right)}{\partial \mathbf{w}[j]}$$

Gradient Descent Algorithm (Batch)
Update Rule

We apply the same simple update rule at every layer

$$\mathbf{W}^{[i]} = \mathbf{W}^{[i]} - \alpha \frac{1}{m} \sum_{j=1}^{m} d\mathbf{W}^{[i]}{}_j$$

$$\mathbf{b}^{[i]} = \mathbf{b}^{[i]} - \alpha \frac{1}{m} \sum_{j=1}^{m} d\mathbf{b}^{[i]}{}_j$$

Gradient Descent Algorithm (Mini Batch)

Choose random weights

Until convergence

- Divide the training dataset into mini-batches of size *s*
- For each mini-batch $\mathcal{D}_{mb}$
  - Set all gradient sums to 0
  - For each training instance in $\mathcal{D}_{mb}$
    - Perform a forward pass through the network
    - Perform a backward pass through the network
    - Update gradient sum for each weight and bias term
  - Update weights and biases using update rule

Gradient Descent Algorithm (Batch) Update Rule

We apply the same simple update rule at every layer

$$\mathbf{W}^{[i]} = \mathbf{W}^{[i]} - \alpha \frac{1}{s} \sum_{j=1}^{s} d\mathbf{W}^{[i]}_j$$

$$\mathbf{b}^{[i]} = \mathbf{b}^{[i]} - \alpha \frac{1}{s} \sum_{j=1}^{s} d\mathbf{b}^{[i]}_j$$

Different Variants of Gradient Descent

Stochastic gradient descent is easy to implement and can result in fast learning, **but** is computationally expensive (no opportunity to speed up) and can result in a noisy gradient signal

Batch gradient descent is computationally efficient and can result in a stable gradient signal, **but** requires gradient accumulation, can result in premature convergence can require loading large datasets into memory and can become slow

Machine Learning Mastery
https://machinelearningmastery.com/gentle-introduction-mini-batch-gradient-descent-configure-batch-size/

Different Variants of Gradient Descent

Mini-batch gradient descent is relatively computationally efficient, does not require full datasets to be loaded into memory, and can result in a stable gradient signal, **but** requires gradient accumulation, and introduces a new hyper-parameter - mini-batch size

Machine Learning Mastery
https://machinelearningmastery.com/gentle-introduction-mini-batch-gradient-descent-configure-batch-size/

Different Variants of Gradient Descent

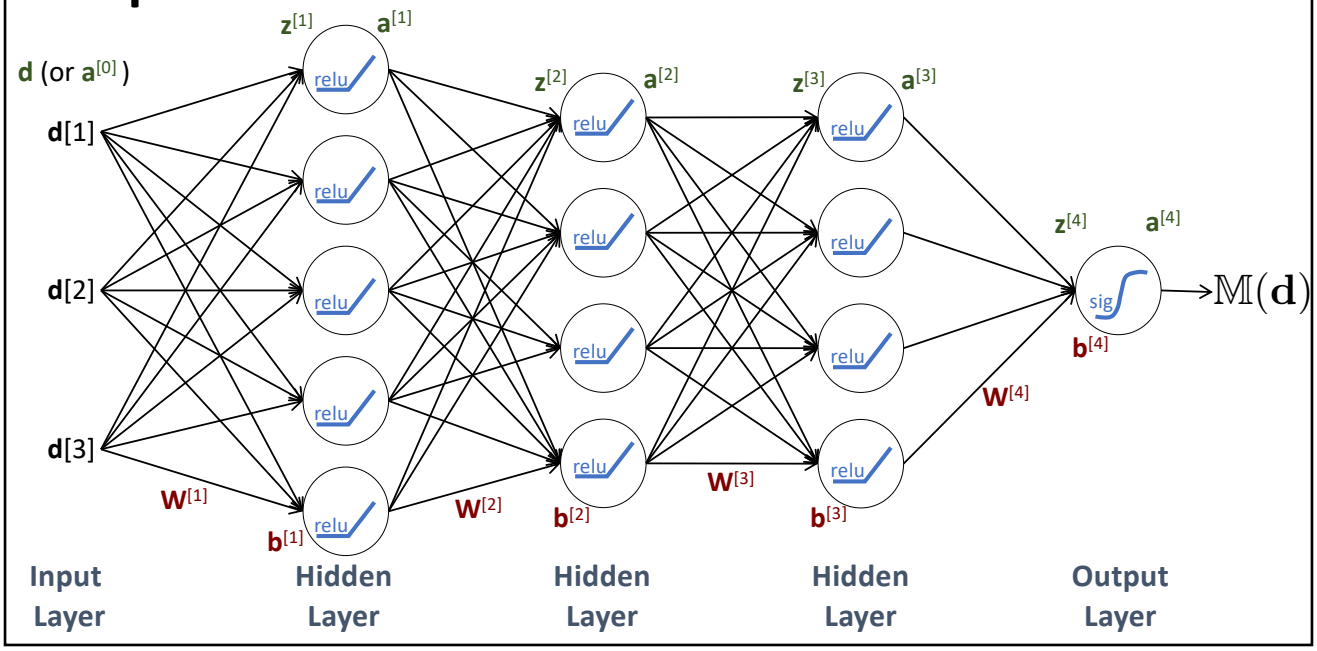
Most modern implementations use mini-batch gradient descent

Batch sizes are typically chosen to take advantage of computational infrastructure - e.g. 32, 64, 128, ...

Machine Learning Mastery
https://machinelearningmastery.com/gentle-introduction-mini-batch-gradient-descent-configure-batch-size/

OTHER LOSS FUNCTIONS

Deep Feed Forward Network



Multinomial Classification

Our combination of the sigmoid activation function and log loss loss function works fine for binary classification problems

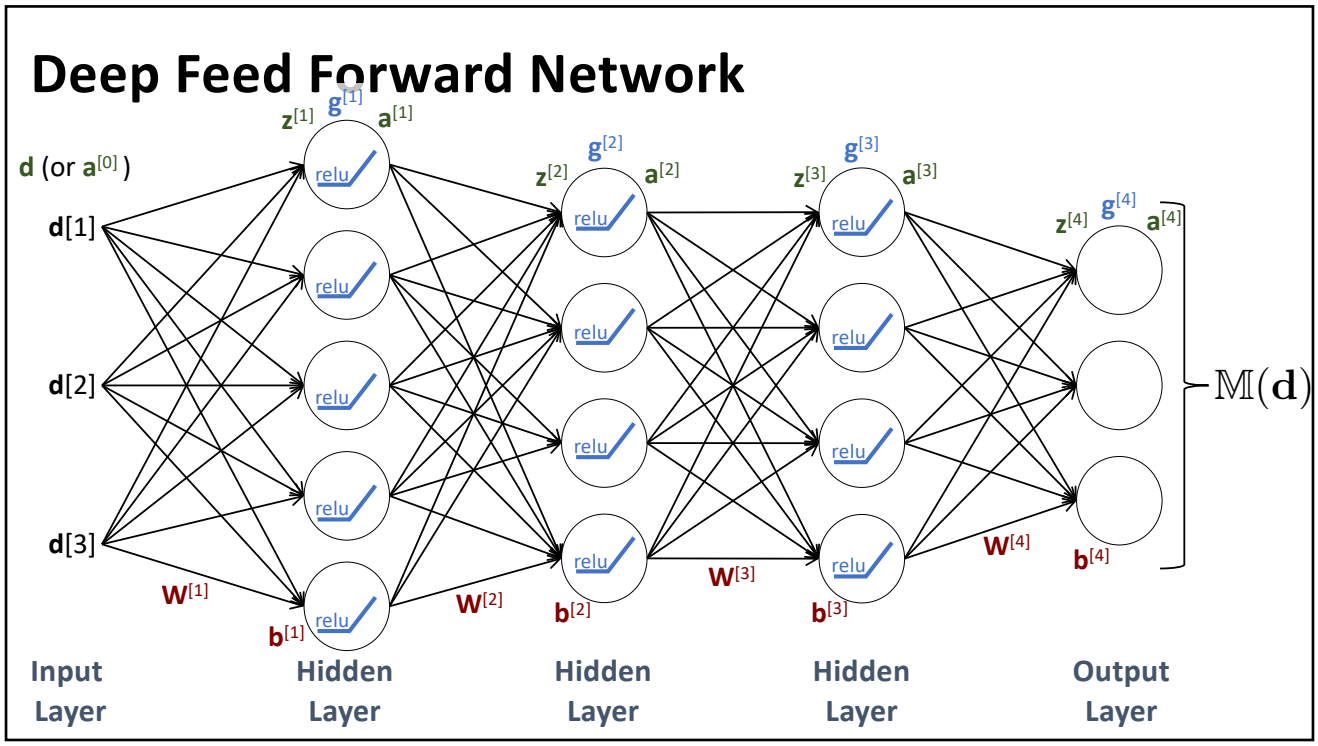
For multinomial classification problems we need to change things

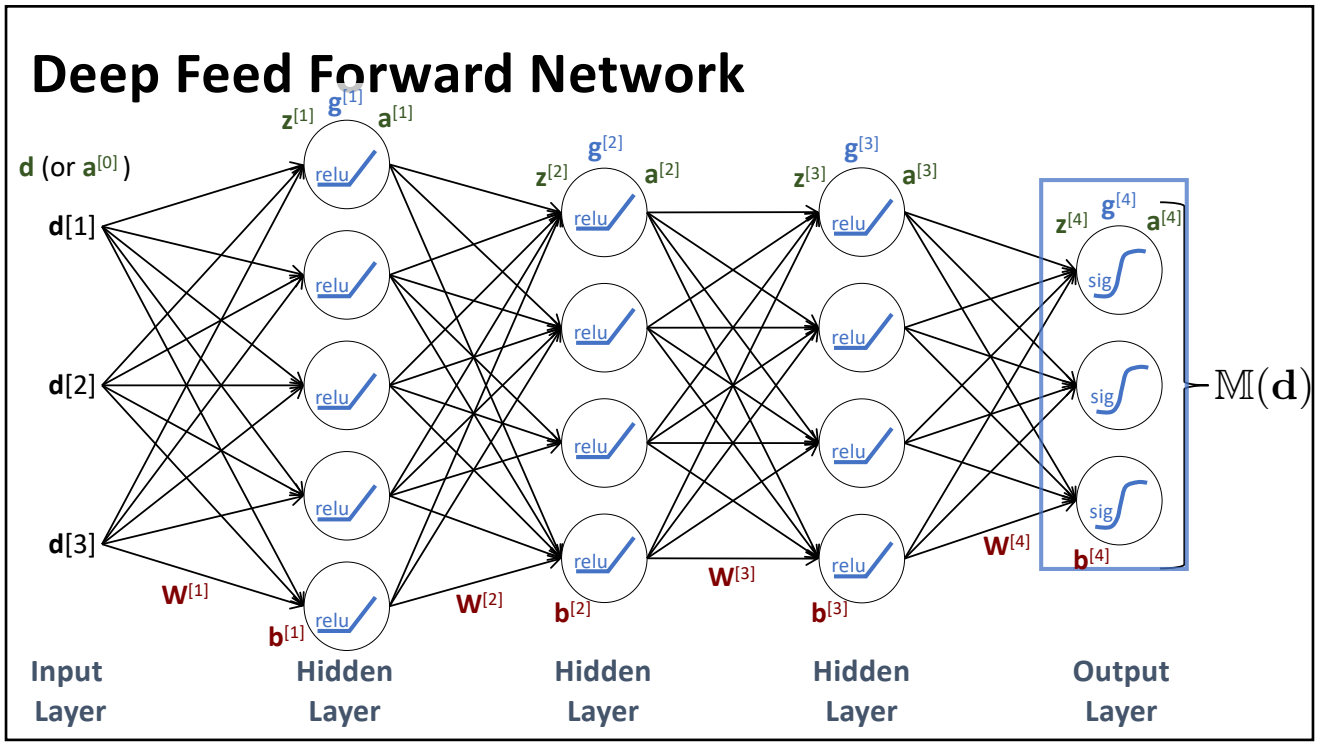
Multinomial Classification

	Descriptive Features						Target Feature
d_1	---	---	---	---	● ● ●	---	shirt
d_2	---	---	---	---		---	sneaker
			⋮				
d_{m-2}	---	---	---	---		---	trousers
d_{m-1}	---	---	---	---	● ● ●	---	shirt
d_m	---	---	---	---		---	sneaker

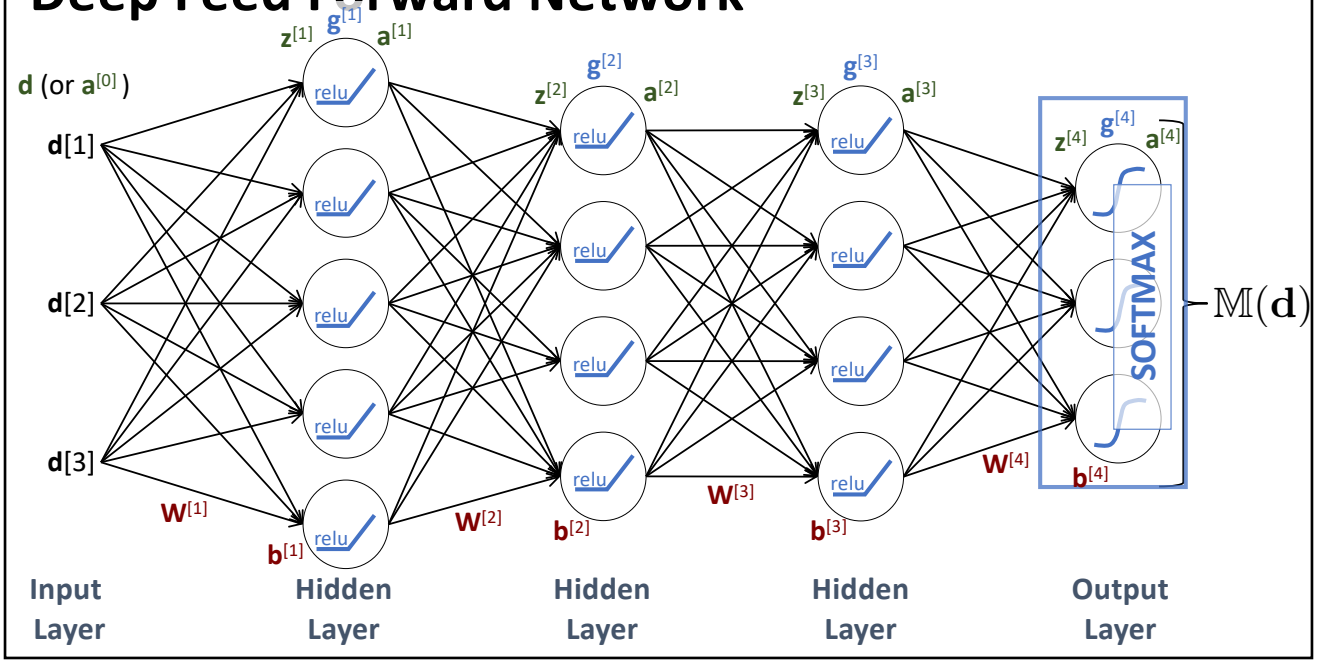
Multinomial Classification

	Descriptive Features						Target Feature		
							(shirt)	(sneaker)	(trousers)
d_1	---	---	---	---	● ● ●	---	1	0	0
d_2	---	---	---	---		---	0	1	0
			⋮						
d_{m-2}	---	---	---	---		---	0	0	1
d_{m-1}	---	---	---	---	● ● ●	---	1	0	0
d_m	---	---	---	---		---	0	1	0


Deep Feed Forward Network
d (or a[0])
d[1]
d[2]
d[3]
z[1] g[1] a[1]
z[2] g[2] a[2]
z[3] g[3] a[3]
z[4] g[4] a[4]
relu
W[1]
b[1]
W[2]
b[2]
W[3]
b[3]
W[4]
b[4]
$\mathbb{M}(\mathbf{d})$
Input Layer
Hidden Layer
Hidden Layer
Hidden Layer
Output Layer


Deep Feed Forward Network
d (or a[0])
d[1]
d[2]
d[3]
z[1] g[1] a[1]
z[2] g[2] a[2]
z[3] g[3] a[3]
z[4] g[4] a[4]
relu
sig
W[1]
b[1]
W[2]
b[2]
W[3]
b[3]
W[4]
b[4]
$\mathbb{M}(\mathbf{d})$
Input Layer
Hidden Layer
Hidden Layer
Hidden Layer
Output Layer

Deep Feed Forward Network



Activation Function: Softmax

The **softmax** (or normalized exponential) function is a generalization of the sigmoid function that *squashes* a vector of arbitrary real values a vector real values in the range [0, 1] that add up to 1

$$\text{softmax}(\mathbf{z}[i]) = \frac{e^{\mathbf{z}[i]}}{\sum_k e^{\mathbf{z}[k]}}$$

Activation Function: Softmax Example

Suppose that the output of our network is

$\mathbf{z} = [5, -1, 2]$

then

$e^{\mathbf{z}} =$

Activation Function: Softmax Example

Suppose that the output of our network is

$\mathbf{z} = [5, -1, 2]$

then

$e^{\mathbf{z}} = [148.41, 0.34, 7.34]$

then

$\text{sum}(e^{\mathbf{z}}) =$

Activation Function: Softmax Example

Suppose that the output of our network is

$\mathbf{z} = [5, -1, 2]$

then

$e^{\mathbf{z}} = [148.41, 0.34, 7.34]$

then

$\text{sum}(e^{\mathbf{z}}) = 156.17$

and

$\text{softmax}[z] =$

Activation Function: Softmax Example

Suppose that the output of our network is

$\mathbf{z} = [5, -1, 2]$

then

$e^{\mathbf{z}} = [148.41, 0.34, 7.34]$

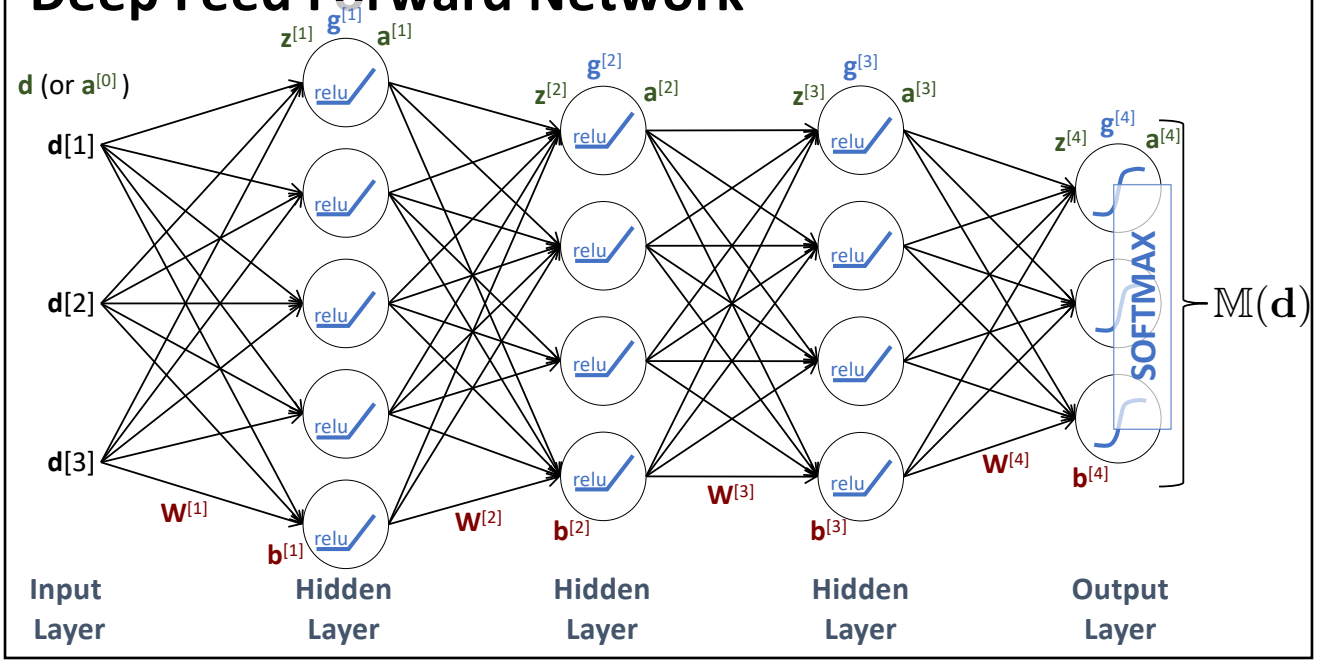
then

$\text{sum}(e^{\mathbf{z}}) = 156.17$

and

$\text{softmax}[z] = [0.950, 0.002, 0.047]$

Deep Feed Forward Network



Loss Function: Cross Entropy

Using multiple outputs requires a new loss function

The most common loss function used for this scenario is **cross entropy**

$$crossent(\mathbb{M}(\mathbf{d}), \mathbf{t}) = -\sum_{k=1}^{l} \mathbf{t}[k] log(\mathbb{M}(\mathbf{d})[k])$$

Loss Function: Cross Entropy

Log loss is just a special case of cross entropy for two target feature levels

$$\mathscr{L}\left(\mathbb{M}(\mathbf{d}), t\right) = -\left(\left(t \times log\left(\mathbb{M}\left(\mathbf{d}\right)\right) + \left(1-t\right) \times log\left(1-\mathbb{M}\left(\mathbf{d}\right)\right)\right)\right)$$

$$\mathscr{L}\left(\mathbb{M}\left(\mathbf{d}\right), \mathbf{t}\right) = -\left(\left(\mathbf{t}[1] \times log\left(\mathbb{M}\left(\mathbf{d}[1]\right)\right)\right) + \left(\mathbf{t}[2] \times log\left(\mathbb{M}\left(\mathbf{d}\right)[2]\right)\right)\right)$$

$$crossent(\mathbb{M}(\mathbf{d}), \mathbf{t}) = -\sum_{k=1}^{2} \mathbf{t}[k] log(\mathbb{M}(\mathbf{d})[k])$$

Activation Function: Softmax Example

Suppose that the output of our network after softmax is

$\mathbb{M}(\mathbf{d})$ = [0.950, 0.002, 0.047]

and the target feature vector, **t**, is

t = [1, 0, 0]

then

crossent($\mathbb{M}(\mathbf{d})$, **t**)

=

Activation Function: Softmax Example

Suppose that the output of our network after softmax is

$\mathbb{M}(\mathbf{d})$ = [0.950, 0.002, 0.047]

and the target feature vector, **t**, is

t = [1, 0, 0]

then

crossent($\mathbb{M}(\mathbf{d})$, **t**)

= -(1 log(0.950) + 0 log(0.002) + 0 log(0.047)

= -(-0.051) = 0.051

Loss Function: Cross Entropy Derivative

Conveniently the derivative of cross entropy loss with respect to the output of a unit in the final layer of a network is:

$$\frac{\partial\, crossent(\mathbb{M}(\mathbf{d}), \mathbf{t})}{\partial \mathbf{a}[i]} = \mathbb{M}(\mathbf{d})[i] - \mathbf{t}[i]$$

Cost Function: Cross Entropy

And just like we did before for batch or min-batch gradient descent we can sum the cross entropy losses across a dataset to generate a cost function

$$J_{crossent}(\mathcal{D}, \mathbb{M}) = \frac{1}{m} \sum_{i=1}^{m} crossent(\mathbb{M}(\mathbf{d}_i), \mathbf{t}_i)$$

Cost Function: Cross Entropy

And also like before for batch or min-batch gradient descent we can sum the cross entropy loss derivatives across a dataset to generate the derivative of the cost function

$$\frac{\partial J_{crossent}(\mathcal{D}, \mathbb{M})}{\partial \mathbf{w}[j]} = \frac{1}{m} \sum_{i=1}^{m} \frac{\partial \mathscr{L}\left(\mathbb{M}\left(\mathbf{d}_i\right), \mathbf{t_i}\right)}{\partial \mathbf{w}[j]}$$

Deep Feed Forward Network Backward Propagation

Require a forward pass through the network
Require: L, network depth
Require: $\mathbf{W}^{[i]}$, $i \in \{1, ..., L\}$, weight matrices for each layer
Require: $\mathbf{b}^{[i]}$, $i \in \{1, ..., L\}$, bias terms for each layer
Require: t, the target feature
Calculate $d\mathbf{a}^{[L]}$ # The derivative of the loss function
for l = L to 1:
$d\mathbf{z}^{[l]} = d\mathbf{a}^{[l]} * g^{[l]\prime}(\mathbf{z}^{[l]})$
$d\mathbf{W}^{[l]} = d\mathbf{z}^{[l]}\mathbf{a}^{[l-1]T}$
$d\mathbf{b}^{[l]} = d\mathbf{z}^{[l]}$
$d\mathbf{a}^{[l-1]} = \mathbf{W}^{[l]T}d\mathbf{z}^{[l]}$

Deep Feed Forward Network Backward Propagation

Require a forward pass through the network
Require: L, network depth
Require: $\mathbf{W}^{[i]}$, $i \in \{1, ..., L\}$, weight matrices for each layer
Require: $\mathbf{b}^{[i]}$, $i \in \{1, ..., L\}$, bias terms for each layer
Require: t, the target feature
$d\mathbf{a}^{[L]} = t/\mathbf{a}^{[L]}$ # The derivative of the loss function
for l = L to 1:
$d\mathbf{z}^{[l]} = d\mathbf{a}^{[l]} * g^{[l]\prime}(\mathbf{z}^{[l]})$
$d\mathbf{W}^{[l]} = d\mathbf{z}^{[l]}\mathbf{a}^{[l-1]T}$
$d\mathbf{b}^{[l]} = d\mathbf{z}^{[l]}$
$d\mathbf{a}^{[l-1]} = \mathbf{W}^{[l]T}d\mathbf{z}^{[l]}$

SUMMARY

Summary

Expanding the perceptron approach to multiple layers of computational units is straight-forward

The gradient descent algorithm does not require any major changes

The key is the **backpropagation of error** algorithm which allows us to propagate the gradient of the error from the output layer back through the earlier layers in a network

Questions