Minia University
Faculty of Computers & information

Artificial Neural Networks and Deep Learning

Slides By:

A.T. Sarah Osama Talaat

E-mail: SarahOsama.fci@gmail.com

Slides were prepared based on set of references mentioned in the last slide

Agenda

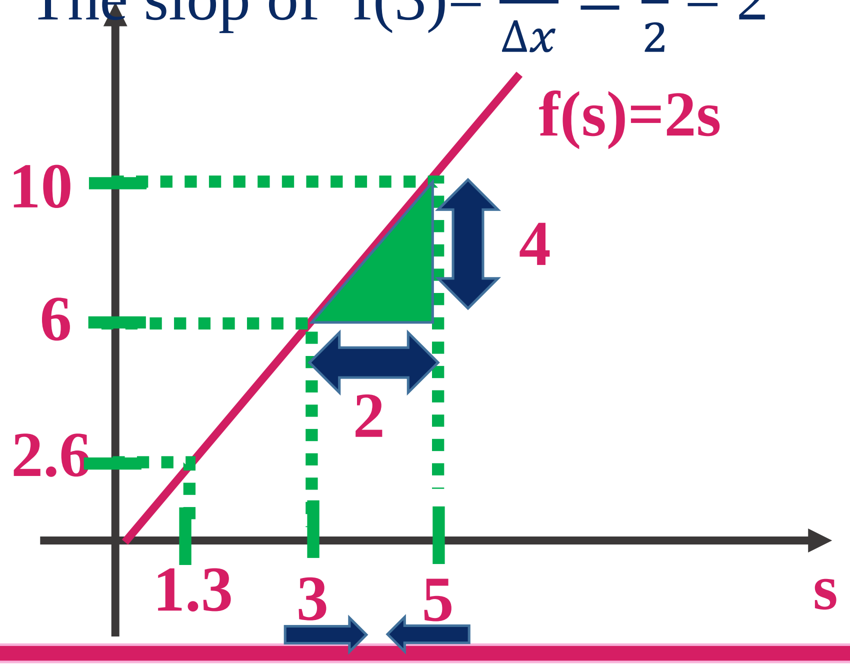
- Revision
- Mathematical Basics
  - Hadamard product ($s \odot t$)
- Supervised Learning Network Paradigms
  - Function vectorization
  - Backpropagation network

Let’s Start

Revision
Derivatives

❑ Intuition about derivatives

- Assume that, we have a following function f(s)=2s
- The slope (derivative) of a function is given by $= \frac{\Delta y}{\Delta x}$
- The slop of f(3)= $\frac{\Delta y}{\Delta x} = \frac{4}{2} = 2$



s	f(s)
3	2×3=6
5	2× 5=10
1.3	2× 1. 3=2.6
0	2×0=0

Revision

Derivatives

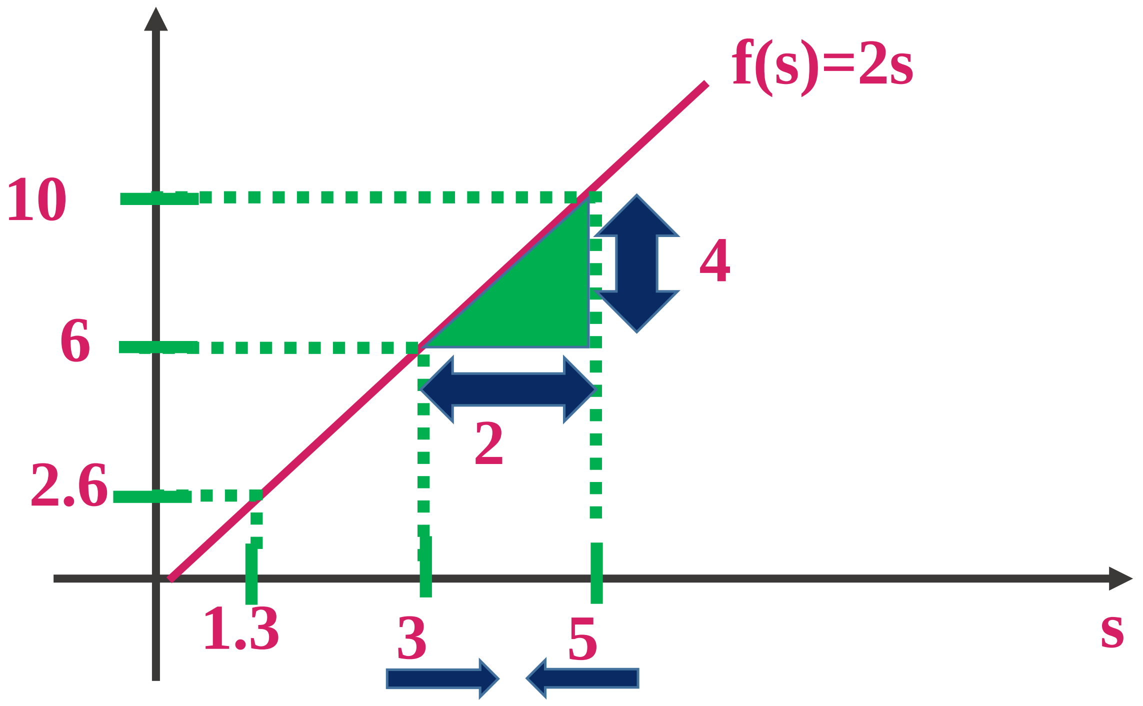
Intuition about derivatives:

- The slope (derivative) of a function f(s) is given by

  $\frac{\Delta y}{\Delta x}$

- Also, the slop is equal to = $\frac{df(s)}{ds} = \frac{d}{ds} f(s)$

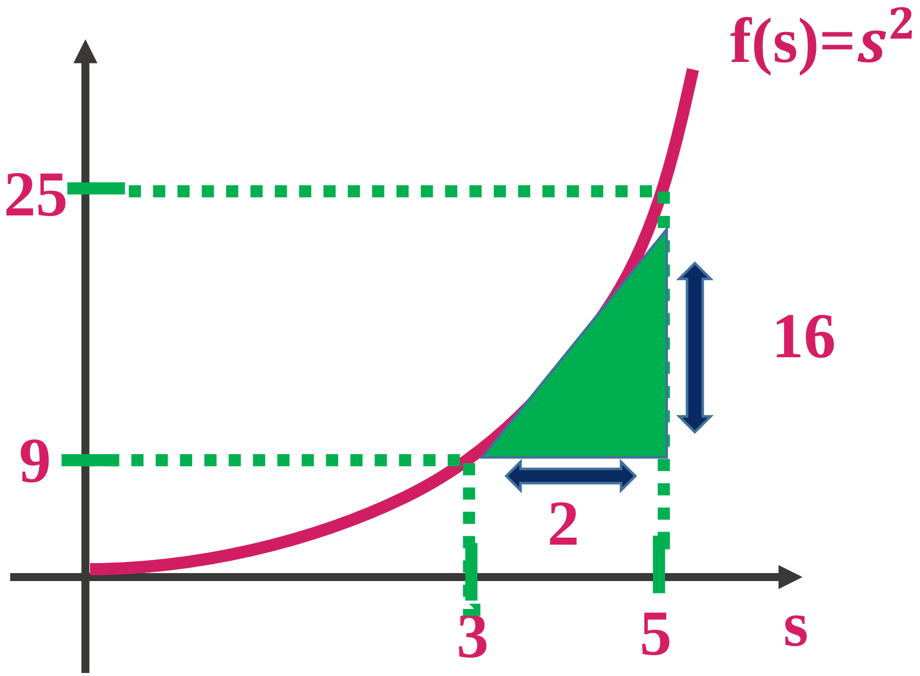
- If s=3 the slope = $\frac{df(3)}{d3} = \frac{4}{2} = 2$



Revision
Derivatives

❑ Derivatives examples

- Assume that, we have a following function f(s)=s^2



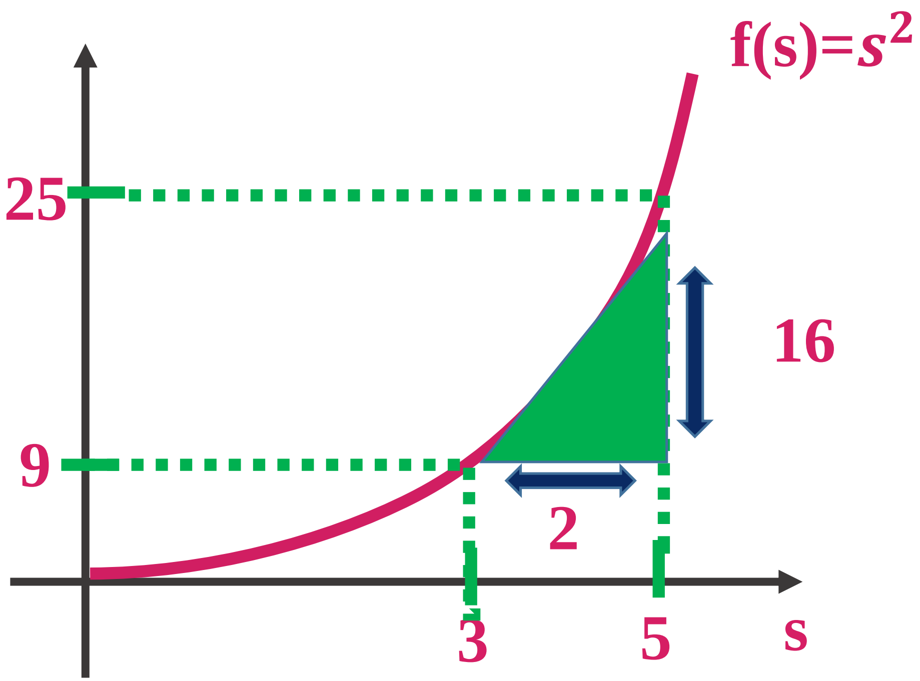
s	f(s)
3	3×3=9
5	5× 5=25
1.3	1. 3 × 1. 3=1.69

Revision

Derivatives

❑ Derivatives examples

- The slope (derivative) of a function f(s) at s= 3 =$\frac{df(3)}{d3} = 9$
- $\frac{df(s)}{ds} = 9 \; where \; s = 3$
- $\frac{df(s)}{ds} = 25 \; where \; s = 5$
- $\frac{d}{ds}\mathrm{f(s)} = \frac{d}{ds}s^2 = 2\mathrm{s}$



Revision
Derivatives

❑ Derivatives examples

- When we have a following function f(s)=s^2 the derivative will be $\frac{d}{ds}f(s) = 2a$
- $where\ \ a = 2.001\ \ the$ f(s) ≈ 4.004
- $where\ \ a = 5\ \ the$ f(s) $= 25$
- When we have a following function f(s)=s^3 the derivative will be $\frac{d}{ds}f(s) = 3s^2$
- $where\ \ a = 2.001\ \ the$ f(s) ≈ 8.012
- $where\ \ a = 5\ \ the$ f(s) $= 75$

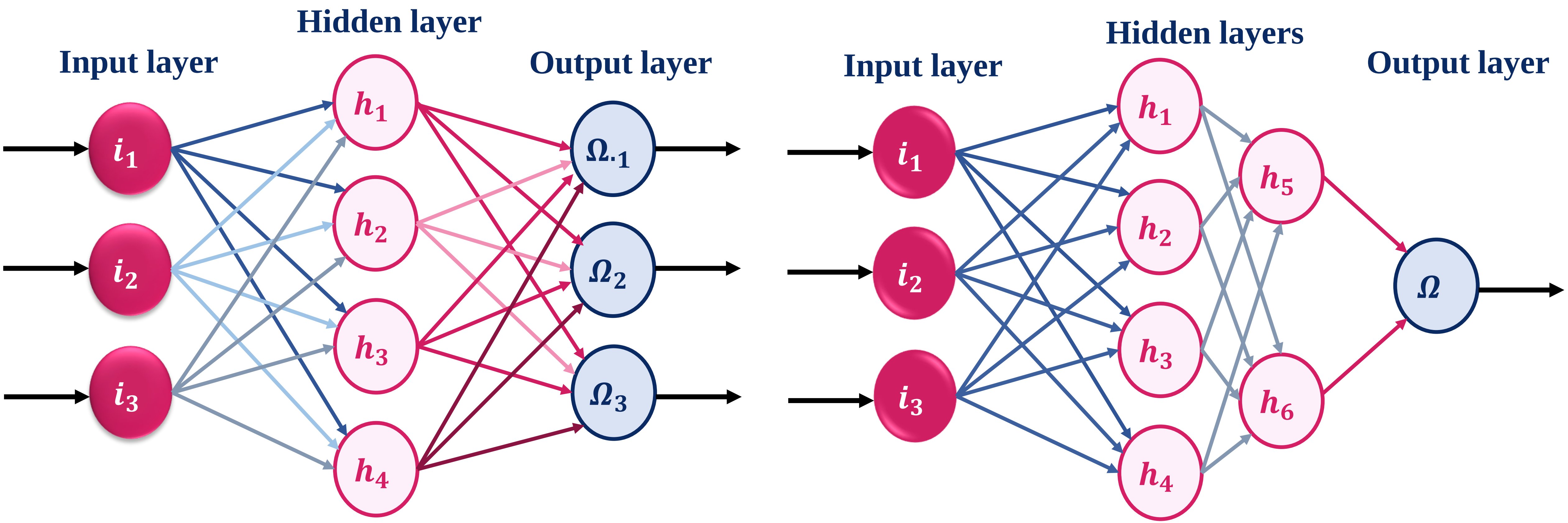
Revision

Derivatives

Definition 5.1(Multi-Layer Perceptron):

- Perceptron with more than one layer of variably weighted connections are referred to as multilayer perceptron (MLP). An n-layer or n-stage perceptron has thereby exactly n variable weight layers and n+1 neuron layers (the retina is disregarded here) with neuron layer 1 being the input layer. The MLP uses backpropagation algorithm for training process. The technical view of an MLP is shown in figure 5.3.

Revision
Multi-Layer perceptron



Figure(5.3): Left side: illustration of a single layer perceptron with four input neurons and one output neuron. Right side: illustration of a single layer perceptron with four input neurons and three output neuron.

Mathematical Basics

Hadamard product ($s \odot t$)

- **Hadamard product** is one of the common linear algebraic operations - things like vector addition, multiplying a vector by a matrix, and so on. But one of the operations is a little less commonly used.
- In particular, suppose ***s and t*** are two vectors of the same dimension. Then we use **s⊙ t** to denote the element wise product of the two vectors. Thus the components of $s \odot t = are\ just\ (s \odot t)_i = s_i t_i$. For instance,

$$\begin{bmatrix} 1 \\ 2 \end{bmatrix} \odot \begin{bmatrix} 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 1 * 3 \\ 2 * 4 \end{bmatrix} = \begin{bmatrix} 3 \\ 8 \end{bmatrix}$$

- This kind of elementwise multiplication is sometimes called the **Hadamard product**.

Mathematical Basics

Function vectorization

- The idea is that we want to apply a function such as **σ** to every element in a vector **v**. We use the obvious notation **σ(v)** to denote this kind of elementwise application of a function. That is, the components of **σ(v)** are just $\boldsymbol{\sigma}(\mathbf{v})_j = \boldsymbol{\sigma}(v_i)$. As an example, if we have the function $\boldsymbol{f(x) = x^2}$ then the vectorized form of $\boldsymbol{f}$ has the effect

$$f\left(\begin{bmatrix}2\\3\end{bmatrix}\right) = \begin{bmatrix}f(2)\\f(3)\end{bmatrix} = \begin{bmatrix}4\\9\end{bmatrix}$$

that is, the vectorized $\boldsymbol{f}$ just squares every element of the vector.

Supervised Learning Network Paradigms
Backpropagation network

- In the lecture 4 we saw how neural networks can learn their weights and biases using the **gradient descent algorithm**. In this lecture there was, a gap in our explanation: we didn't discuss how to compute the gradient descent search strategy of the cost function. After that, in the lecture 6 we saw how neural networks can learn their weights and biases using the gradient descent search strategy of the cost function to train **a single layer neural network**. In this lecture I'll explain a fast algorithm for computing such gradients , an algorithm known as **backpropagation to train a multi-layer neural network.**

Supervised Learning Network Paradigms
Backpropagation network

- Backpropagation algorithm is used the gradient descant algorithm which is a search strategy used in continuous search spaces.
- Backpropagation algorithm used the gradient descent algorithm to train multilayer artificial neural networks.
- The goal of backpropagation is to compute the partial derivatives $\frac{\partial C}{\partial w}$ ***and*** $\frac{\partial C}{\partial b}$ of the cost function ***C*** with respect to any weight **w** or bias **b** in the network.
- In the next slides I explain how Backpropagation algorithm works in general and how it applies in particular to training a **Multi-layer perceptron network**.

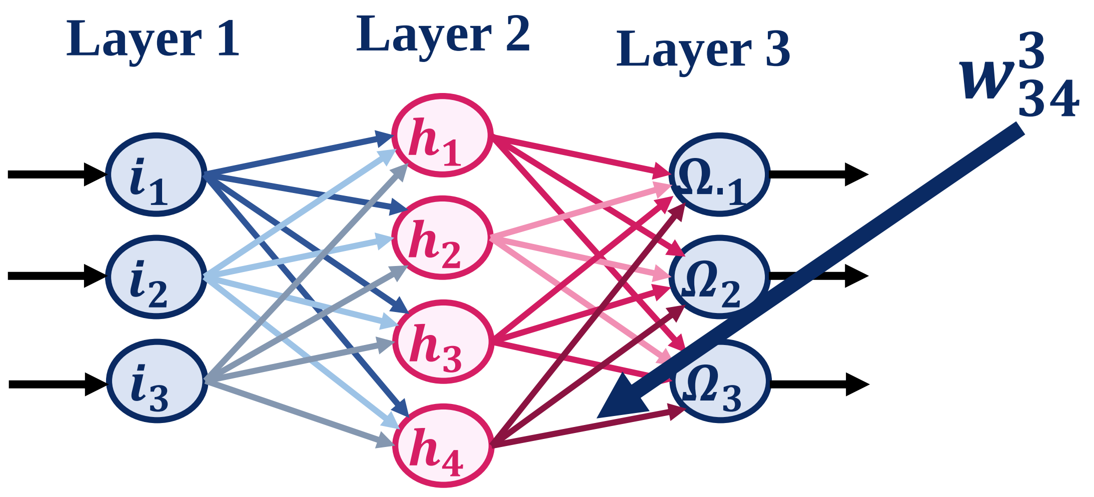
Supervised Learning Network Paradigms

Backpropagation network

- **A fast matrix-based approach to computing the output from a neural network**
  - Before discussing backpropagation, let's warm up with a fast matrix-based algorithm to compute the output from a neural network. In particular, **this is a good way of getting comfortable with the notation used in backpropagation,** in a familiar context.

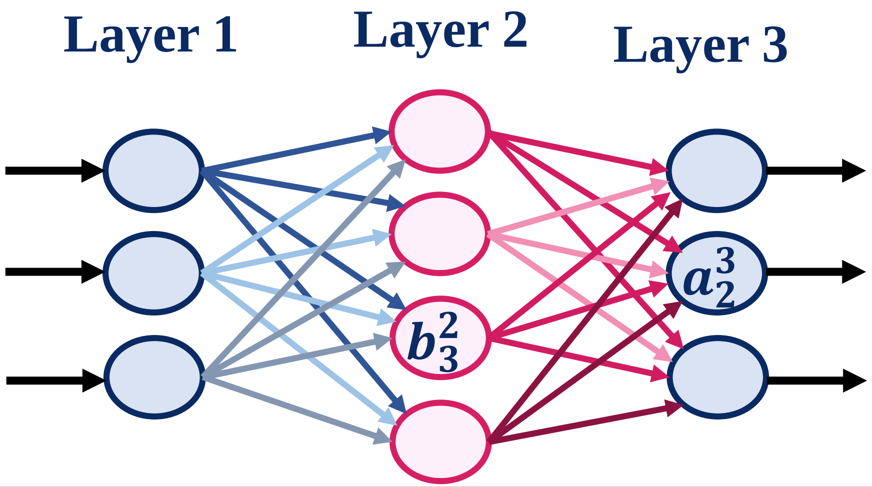
Supervised Learning Network Paradigms
Backpropagation network: notations

- Let's begin with a notation which lets us refer to weights in the network in an unambiguous way. We'll use $\boldsymbol{w_{jk}^{l}}$ to denote the weight for the connection **from** the $\boldsymbol{k^{th}}$ neuron in the $(\boldsymbol{l-1})^{\boldsymbol{th}}$ layer to the $\boldsymbol{j^{th}}$ neuron in the $\boldsymbol{l^{th}}$ layer.



Supervised Learning Network Paradigms
Backpropagation network: notations

- We use a similar notation for the network's biases and activations. Explicitly, we use b_j^l for the bias of the j^{th} neuron in the l^{th} layer. And we use a^l_j for the activation of the j^{th} neuron in the l^{th} layer. The following diagram shows examples of these notations in use:



Supervised Learning Network Paradigms

Backpropagation network: notations

- With the previous notations, the activation $a^l{}_j$ of the j^{th} neuron in the l^{th} layer is related to the activations in the $(l-1)^{th}$ layer by the equation:

$$a^l{}_j = f_{act}\left(\sum_k w^l_{jk} a^{l-1}_k + b^l_j\right),$$

- Assume that, we used the sigmoid activation function:

$$a^l{}_j = \sigma\left(\sum_k w^l_{jk}\, a^{l-1}_k + b^l_j\right),$$

where the sum is over all neurons k in the $(l-1)^{th}$ layer

Supervised Learning Network Paradigms
Backpropagation network: notations

- We need to rewrite the following equation in a matrix form which represent the idea of vectorization.

$$a^{l}_{j} = \sigma\left(\sum_k w^{l}_{jk}\, a^{l-1}_{k} + b^{l}_{j}\right),$$

- By using the previous notations and the **vectorization** definition this equation can be rewritten in the compact vectorized form

$$a^{l} = \sigma\left(w^{l}\, a^{l-1} + b^{l}\right),$$

Supervised Learning Network Paradigms
Backpropagation network: notations

- This expression

$$a^{l} = \sigma\left(w^{l} a^{l-1} + b^{l}\right),$$

- gives us a much more global way of thinking about **how the activations in one layer relate to activations in the previous layer:** we just apply the weight matrix to the activations, then add the bias vector, and finally apply the **σ** function
- That **global** view is often easier and more succinct (and involves fewer indices!) than the neuron-by-neuron view we've taken to now. **The expression is also useful in practice, because most matrix libraries provide fast ways of implementing matrix multiplication, vector addition, and vectorization.**

Supervised Learning Network Paradigms
Backpropagation network: notations

- When using this formula $\boldsymbol{a^{l}} = \boldsymbol{\sigma(w^{l}a^{l-1} + b^{l})}$ to compute $\boldsymbol{a^{l}}$ we compute the intermediate quantity $\boldsymbol{net^{l}} = \boldsymbol{w^{l}a^{l-1} + b^{l}}$, along the way. As we know, we call $\boldsymbol{net^{l}}$ the **weighted input or the net input** to the neurons in layer $\boldsymbol{l}$.
- So that, we can rewrite this formula $\boldsymbol{a^{l}} = \boldsymbol{\sigma(w^{l}a^{l-1} + b^{l})}$ as following

  $\boldsymbol{a^{l}} = \boldsymbol{\sigma(net^{l})}$.
- It's also worth noting that $\boldsymbol{net^{l}}$ has components $\boldsymbol{net^{l}_{j}} = \sum_{k} \boldsymbol{w^{l}_{jk}a^{l-1}_{k} + b^{l}_{j}}$,, that is, $\boldsymbol{net^{l}_{j}}$ is just the weighted input to the activation function for neuron $\boldsymbol{j}$ in layer $\boldsymbol{l}$.

Supervised Learning Network Paradigms
Backpropagation network

❑ Remember that

- The goal of backpropagation is to compute the partial derivatives $\frac{\partial C}{\partial w}$ ***and*** $\frac{\partial C}{\partial b}$ of the cost function ***C*** with respect to any weight **w** or bias **b** in the network.
- For backpropagation to work we need to make two main assumptions about the form of the **loss and the cost functions**.

Supervised Learning Network Paradigms
Backpropagation network: cost function

Define the cost function:

- Before stating those assumptions, though, it's useful to have an example cost function in mind. As example, we will use the **quadratic cost function**, the quadratic cost has the form

$$C(w,b) = \frac{1}{2n}\sum_x \left\| y - a^L \right\|^2,$$

where: n is the total number of training examples; the sum is over individual training examples x; $\hat{y}$ is the corresponding desired output; **L** denotes the number of layers in the network; and a^L is the vector of activations output from the network when x is input.

Supervised Learning Network Paradigms

Backpropagation network: assumptions

- **The first assumption we need about the cost function**
  - **What assumptions do we need to make about our cost function C, in order that backpropagation can be applied?**
  - The first assumption we need **is that the cost function can be written as an average** $C(w,b) = \frac{1}{n}\sum_x l(\hat{y},y)$ **,over the loss functions** $l(\hat{y},y)$ for individual training examples, $\boldsymbol{x}$. This is the case for the quadratic cost function, where the loss function is $l(\hat{y},y) = \frac{1}{2}\|y-\hat{y}\|^2 = \frac{1}{2}\|y-a^L\|^2$.

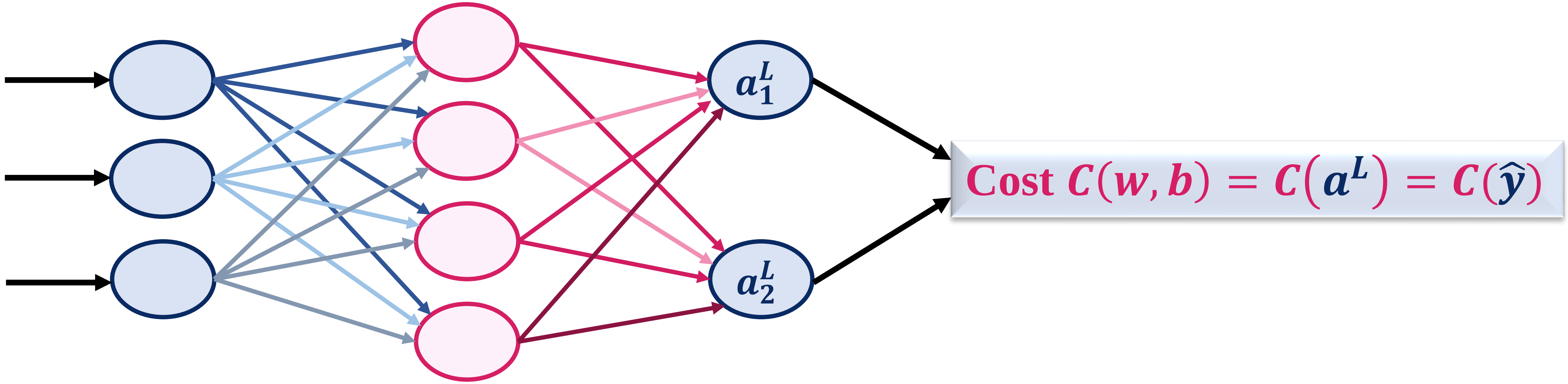
Supervised Learning Network Paradigms
Backpropagation network: assumptions

The first assumption we need about the cost function

- **The reason we need this assumption is because what backpropagation actually lets us do is** use the gradient descent algorithm to compute the partial derivatives $\frac{\partial l}{\partial w}$ ***and*** $\frac{\partial l}{\partial b}$ for loss function (a single training example). That means, the backpropagation algorithm used the Stochastic gradient optimization.
- We then recover $\frac{\partial C}{\partial w}$ ***and*** $\frac{\partial C}{\partial b}$ by averaging over training examples.
- For simplicity, with this assumption in mind, we'll suppose the training example x has been fixed, and drop the loss function and use only the cost function.

Supervised Learning Network Paradigms
Backpropagation network: assumptions

- **The second assumption we make about the cost function**
  - The cost function can be written as a function of the outputs from the neural network.



Supervised Learning Network Paradigms
Backpropagation network: assumptions

The second assumption we make about the cost function

- For example, the **quadratic cost function** satisfies this requirement, since the quadratic cost for a single training example $\boldsymbol{x}$ may be written as

$$C(w,b) = l(\hat{y},y) = \frac{1}{2}\|y - a^L\|^2 = \frac{1}{2}\sum_j (y_j - a^L{}_j)^2$$

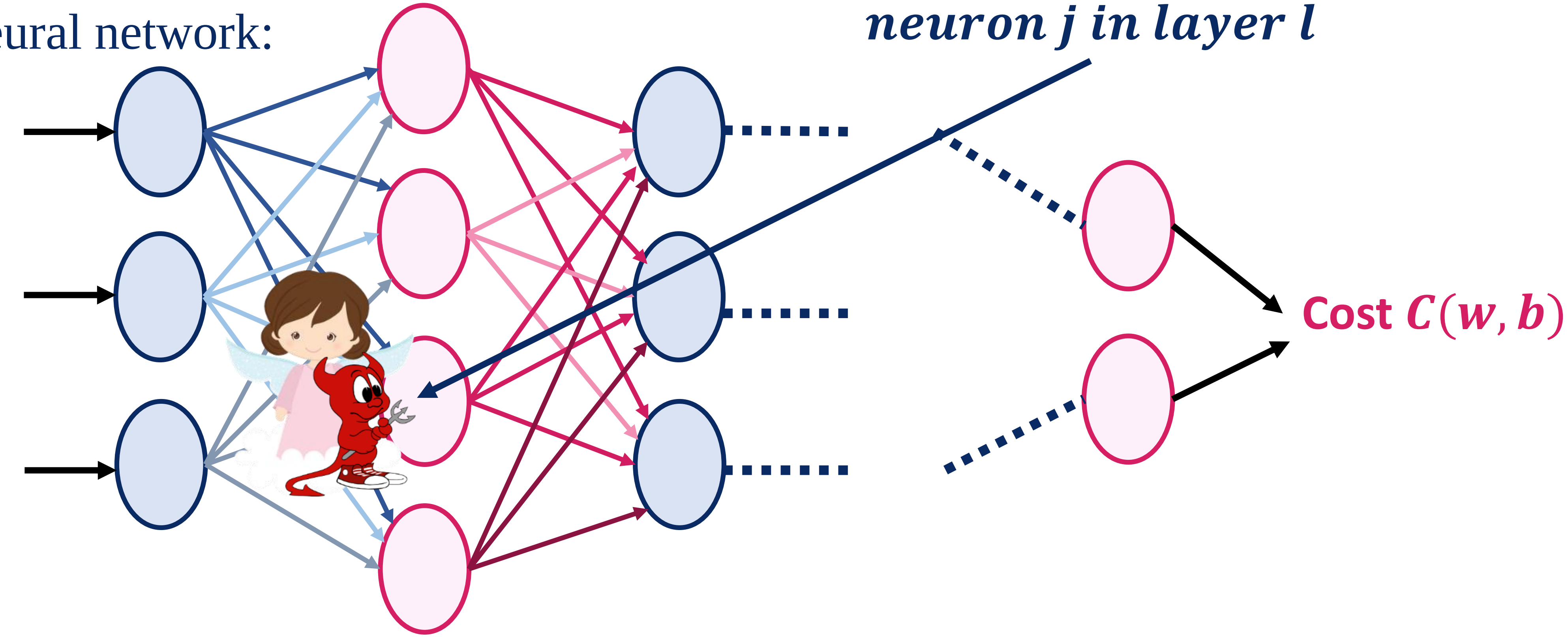
and thus is a function of the output activations. Of course, this cost function also depends on the **desired output** y, and you may wonder why we're not regarding the cost also as a function of y.

Supervised Learning Network Paradigms
Backpropagation network

- Backpropagation is about **understanding how changing the weights and biases in a network changes the cost function**. Ultimately, this means computing the partial derivatives $\frac{\partial C}{\partial w_{jk}^{l}}$ and $\frac{\partial C}{\partial b_{j}^{l}}$. But to compute those, we first introduce an intermediate quantity, δ_j^l, which we call the **error or delta term** in the j^{th} neuron in the l^{th} layer.
- Backpropagation will give us a procedure **to compute the error** δ_j^l, and then will relate δ_j^l to $\frac{\partial C}{\partial w_{jk}^{l}}$ and $\frac{\partial C}{\partial b_{j}^{l}}$

Supervised Learning Network Paradigms
Backpropagation network

- To understand how the error is defined, imagine there is someone (e.g. angle, demon) in our neural network:



Supervised Learning Network Paradigms
Backpropagation network

- The demo sits at the **j**th neuron in layer ***l***. As the input to the neuron comes in, the angle messes with the neuron's operation. It adds a little change Δnet^{l}_{j} to the neuron's weighted input(net), so that instead of outputting $\sigma(net^{l}_{j})$,the neuron instead outputs $\sigma(net^{l}_{j}+\Delta net^{l}_{j})$. This change propagates through later layers in the network, finally causing the overall cost to change by an amount $\frac{\partial C}{\partial z^{l}_{j}}\Delta net^{l}_{j}$
- **Now, this demo is a good demo which is like an angel, and is trying to help you improve the cost, i.e., they're trying to find a Δnet^{l}_{j} which makes the cost smaller.**

Supervised Learning Network Paradigms

Backpropagation network

- The angle sits at the **j**th neuron in layer ***l***. As the input to the neuron comes in, the angle messes with the neuron's operation. It adds a little change Δnet^{l}_{j} to the neuron's weighted input(net), so that instead of outputting **σ**(net^{l}_{j}),the neuron instead outputs **σ**($net^{l}_{j}+\Delta net^{l}_{j}$). This change propagates through later layers in the network, finally causing the overall cost to change by an amount $\frac{\partial C}{\partial z^{l}_{j}}\Delta net^{l}_{j}$
- **Now, this angel is a good angel, and is trying to help you improve the cost, i.e., they're trying to find a Δnet^{l}_{j} which makes the cost smaller.**

Supervised Learning Network Paradigms
Backpropagation network

- Suppose $\frac{\partial C}{\partial net_j^l}$ has a large value (either positive or negative). Then the angel can lower the cost quite a bit by choosing $\Delta net^l{}_j$ to have the opposite sign to $\frac{\partial C}{\partial net_j^l}$.
- By contrast, if $\frac{\partial C}{\partial net_j^l}$ is close to zero, then the angel can't improve the cost much at all by perturbing the weighted input $\Delta net^l{}_j$.
- So far as the angel can tell, the neuron is already pretty near optimal. This is only the case for small changes $\Delta net^l{}_j$, of course.
- **We will assume that the angel is constrained to make such small changes. And so there's a heuristic sense in which $\frac{\partial C}{\partial net_j^l}$ is a measure of the error in the neuron.**

Supervised Learning Network Paradigms
Backpropagation network

- Motivated by this story, we define the error δ_j^l of neuron j in layer *l* by:

$$\delta_j^l = \frac{\partial C}{\partial net_j^l}$$

- As per our usual conventions, we use δ^l to denote the vector of errors associated with layer *l*.
- Backpropagation will give us a way of computing δ^l for every layer, and then relating those errors to the quantities of real interest, $\frac{\partial C}{\partial w_{jk}^l}$ and $\frac{\partial C}{\partial b_j^l}$.

Supervised Learning Network Paradigms
Backpropagation network

- You might wonder why the angel is changing the weighted input $\boldsymbol{net^{l}_{j}}$. Surely it'd be more natural to imagine **the angel changing the output activation** $\boldsymbol{a^{l}_{j}}$, with the result that we'd be using $\frac{\partial C}{\partial a_j^l}$ as our measure of error.

- So we'll stick with $\boldsymbol{\delta_j^l} = \frac{\partial C}{\partial net_j^l}$ as our measure of error

Supervised Learning Network Paradigms

Backpropagation network: fundamental equations

- Backpropagation is based around **four fundamental equations**. Together, those equations give us a way of computing both the error δ^l and the **gradient of the cost function**. I state the four equations below.
  - An equation for the error in the output layer
  - An equation for the error δ^l in terms of the error in **the next layer**
  - An equation for the **rate of change of the cost** with respect to any **bias** in the network.
  - An equation for the **rate of change of the cost** with respect to any **weight** in the network

Supervised Learning Network Paradigms
Backpropagation network: fundamental equations

- **An equation for the error δ^l in the output layer :**

$$\delta_j^L = \frac{\partial C}{\partial a_j^L}\sigma'\left(net_j^L\right) \qquad \text{(BP1)}$$

- The first term on the right $\frac{\partial C}{\partial a_j^L}$,, just measures **how fast the cost is changing** as a function of the jth output activation. If, for example, C doesn't depend much on a particular output neuron, j, then δ_j^L will be small, which is what we'd expect.
- The second term on the right, $\sigma'\left(net_j^L\right)$, measures how fast the **activation function** σ is changing at net_j^L.

Supervised Learning Network Paradigms
Backpropagation network: fundamental equations

- **An equation for the error δ^l in the output layer :**

$$\delta_j^L = \frac{\partial C}{\partial a_j^L} \sigma'\left(net_j^L\right) \qquad \text{(BP1)}$$

- Equation (BP1) is a componentwise expression for δ_j^L. It's a perfectly good expression, but not **the matrix-based** form we want for backpropagation. However, it's easy to rewrite the equation in a matrix-based form, as

$$\delta^L = \nabla_a C \odot \sigma'\left(net^L\right) \quad \text{(BP1a)}$$

Supervised Learning Network Paradigms
Backpropagation network: fundamental equations

❑ An equation for the error δ^l in the output layer :

$$\boldsymbol{\delta^L = \nabla_a C \odot \sigma'\left(net^L\right)} \quad \text{(BP1a)}$$

- Here, $\boldsymbol{\nabla_a C}$ is defined to be a **vector** whose components are the partial derivatives $\frac{\partial C}{\partial a_j^L}$. You can think of $\boldsymbol{\nabla_a C}$ as expressing the rate of change of $\boldsymbol{C}$ with respect to the output activations (i.e. $\hat{y}$). It's easy to see that **Equations (BP1a) and (BP1)** are equivalent, and for that reason from now on we'll use **(BP1)** interchangeably to refer to both equations. As an example, in the case of the quadratic cost we have $\boldsymbol{\nabla_a C = (a^L - y) = (\hat{y} - y)}$, and so the fully matrix-based form of (BP1)becomes

$$\boldsymbol{\delta^L = \left(a^L - y\right) \odot \sigma'\left(net^L\right)} \quad \textbf{(BP1a)}$$

Supervised Learning Network Paradigms
Backpropagation network: fundamental equations

❑ **An equation for the error in terms of the error in the next layer, δ^{l+1} :**

$$\boldsymbol{\delta^L} = \left((w^{l+1})^{\boldsymbol{T}} \boldsymbol{\delta}^{l+1}\right) \odot \boldsymbol{\sigma'}\left(\boldsymbol{net^L}\right) \quad \text{(BP2)}$$

- where $(w^{l+1})^{\boldsymbol{T}}$ is the transpose of the weight matrix w^{l+1} for the $(l+1)^{th}$ layer. This equation appears complicated, but each element has a nice interpretation. Suppose we know the error $\boldsymbol{\delta^{l+1}}$ at the $(l+1)^{th}$ layer. When we apply the transpose weight matrix, $(w^{l+1})^{\boldsymbol{T}}$, we can think intuitively of **this as moving the error backward through the network**, giving us some sort of **measure of the error at the output of the $\boldsymbol{l^{th}}$ layer**. We then take the Hadamard product $\odot\, \boldsymbol{\sigma'}\left(\boldsymbol{net^L}\right)$. This moves the **error backward through the activation function** in layer $\boldsymbol{l}$, giving us the error $\boldsymbol{\delta^l}$ in the weighted input to layer $\boldsymbol{l}$.

Supervised Learning Network Paradigms
Backpropagation network: fundamental equations

- **An equation for the error in terms of the error in the next layer, $\boldsymbol{\delta^{l+1}}$:**

$$\boldsymbol{\delta_j^L = \frac{\partial C}{\partial a_j^L} \sigma'\left(net_j^L\right)} \qquad \text{(BP1)}$$

$$\boldsymbol{\delta^L = \left((w^{l+1})^T \delta^{l+1}\right) \odot \sigma'\left(net^L\right)} \qquad \text{(BP2)}$$

- By combining (BP2) with (BP1) we can compute the error $\boldsymbol{\delta^l}$ for any layer in the network. We start by using (BP1) to compute $\boldsymbol{\delta^l}$, then apply Equation (BP2) to compute $\boldsymbol{\delta^{l-1}}$, then Equation (BP2) again to compute $\boldsymbol{\delta^{l-2}}$, and so on, all the way back through the network.

Supervised Learning Network Paradigms
Backpropagation network: fundamental equations

- **An equation for the rate of change of the cost with respect to any bias in the network:**

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \quad \text{(BP3)}$$

- That is, the error δ_j^l is exactly equal to the rate of change $\frac{\partial C}{\partial b_j^l}$. This is great news, since (BP1) and (BP2) have already told us how to compute δ_j^l. We can rewrite (BP3) in shorthand as

$$\frac{\partial C}{\partial b} = \delta \quad \text{(BP3)}$$

- where it is understood that δ is being evaluated at the same neuron as the bias b.

Supervised Learning Network Paradigms
Backpropagation network: fundamental equations

- **An equation for the rate of change of the cost with respect to any weight in the network:**

$$\frac{\partial C}{\partial w_{jk}^{l}} = a_k^{l-1}\,\delta_j^{l} \qquad \text{(BP4)}$$

- This tells us how to compute the partial derivatives $\frac{\partial C}{\partial w_{jk}^{l}}$ in terms of the quantities δ^{l} and a^{l-1}, which we already know how to compute. The equation can be rewritten in a less index-heavy notation as

$$\frac{\partial C}{\partial w} = a_{in}\delta_{out} \qquad \text{(BP4a)}$$

Supervised Learning Network Paradigms
Backpropagation network: fundamental equations

- **An equation for the rate of change of the cost with respect to any weight in the network:**

$$\frac{\partial C}{\partial w} = a_{in}\delta_{out} \qquad \text{(BP4a)}$$

  - where it's understood that a_{in} is the activation of the neuron input to the weight w, and δ_{out} is the error of the neuron output from the weight w. Zooming in to look at just the weight w, and the two neurons connected by that weight, we can depict this as:

$$\frac{\partial C}{\partial w} = a_{in} \times \delta_{out}$$

Supervised Learning Network Paradigms

Backpropagation network: fundamental equations

- **An equation for the rate of change of the cost with respect to any weight in the network:**

$$\frac{\partial C}{\partial w} = a_{in}\delta_{out} \qquad \text{(BP4a)}$$

  - A nice consequence of Equation (BP4a) is that when the activation a_{in} is small, $a_{in} \approx 0$, the gradient term $\frac{\partial C}{\partial w}$ will **also tend to be small**. In this case, we'll say the **weight learns slowly**, meaning that it's not changing much during gradient descent. In other words, one consequence of (BP4) is that **weights output from low-activation neurons learn slowly.**

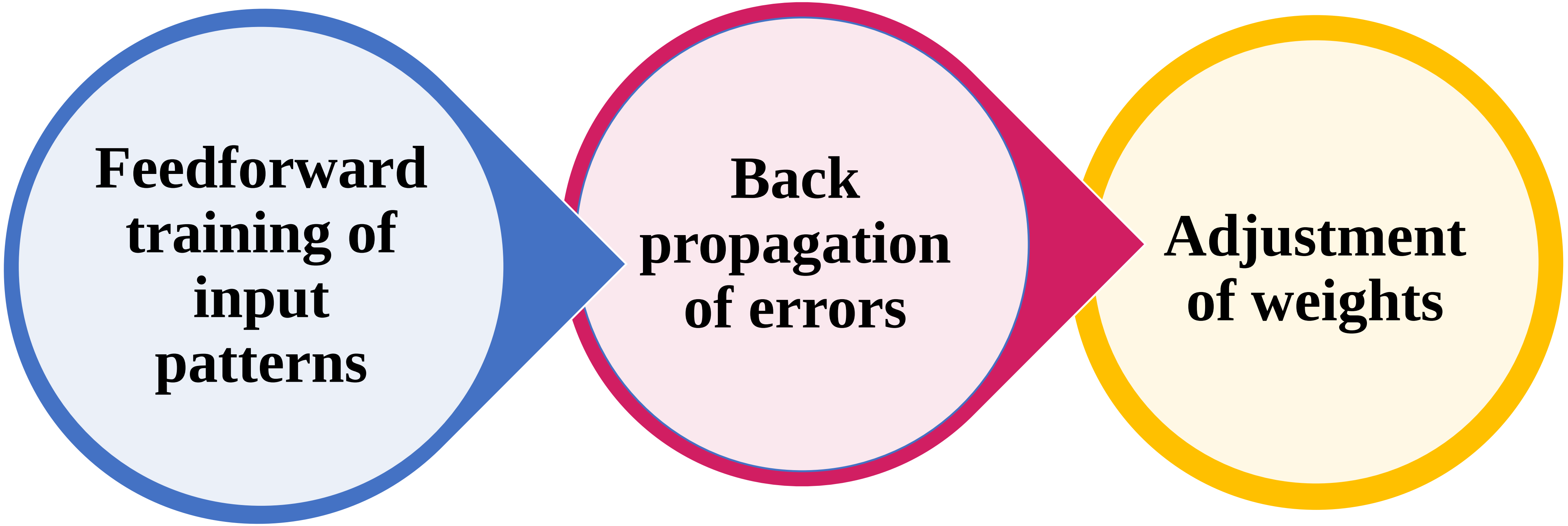
Supervised Learning Network Paradigms

Backpropagation network: fundamental equations

- The four fundamental equations turn out to hold for any activation function, not just the standard sigmoid function (that's because, as we'll see in a moment, the proofs don't use any special properties of (σ). And so we can use these equations to design activation functions which have particular desired learning properties.

Supervised Learning Network Paradigms

Backpropagation network main steps



Figure(6.1): illustration of a Backpropagation network (is a Multi-layer perceptron learning algorithm) training phase

Supervised Learning Network Paradigms
Backpropagation network algorithm

- **Algorithm 5.3: Backpropagation algorithm for single training sample(tr,y)**

Input x: set the corresponding activation a^1 for the input layer.

Output: Gradient of the cost function

1. **Initialization Initialize all weights by small random values (e.g. -0.5:0.5) and select a suitable learning rate $\eta(e.g. 0.1)$**
2. **Feedforward:**
3. **For each** $l = 2,3,4,,..,L$ **do**
4. Feed the training sample through the network and compute $net^l = w^l a^{l-1} + b^l$,
5. Then apply the activation function $a^l = \sigma(net^l)$
6. **End foreach**
7. *Foreach output unit k,compue delta* $\delta^l = \nabla_a C \odot \sigma'(net^L) = (a^L - y) \odot \sigma'(net^L)$
8. **Backpropagate the error:**
9. **For each** $l = L-1, L-2, .., 2$ **do**
10. Compute $\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(net^L)$,
11. **End foreach**
12. **Output:** The gradient of the cost function is given by $\frac{\partial C}{\partial w_{jk}^l} = a_{in}\, \delta_{out}$ and $\frac{\partial C}{\partial b_j^l} = \delta_j^l = \delta$

Supervised Learning Network Paradigms
Backpropagation network algorithm

- **Algorithm 5.3: Backpropagation algorithm for each training samples(tr,y)**

Input: set of training examples.

Output: Gradient of the cost function

1. **For each training example x: Set the corresponding input activation $a^{x,1}$, and perform the following steps:**
2. **Feedforward:**
3. **Foreach** $l = 2,3,4,,..,L$ **do**
4. Feed the training samples through the network and compute $net^{x,l} = w^{l}a^{x,l-1} + b^{l}$,
5. Then apply the activation function $a^{x,l} = \sigma(net^{x,l})$
6. **End foreach**
7. *Foreach output unit k,compue delta* $\delta^{x,L} = \nabla_a C_x \odot \sigma'(net^{x,L})$
8. **Backpropagate the error:**
9. **For each** $l = L-1, L-2,..,2$ **do**
10. Compute $\delta^{x,l} = ((w^{l+1})^{T}\delta^{x,l+1}) \odot \sigma'(net^{x,L})$,
11. **End foreach**
12. **Gradient descent: For each** $l = L-1, L-2,..,2$ update the weights according to the rule $w_{l,new} = w_{l,old} - \frac{n}{m}\sum_x \delta^{x,l}(a^{x,l-1})^{T}$, and the biases according to the rule $b_{l,new} = b_{l,old} - \frac{n}{m}\sum_x \delta^{x,l}$

References

- Kriesel, David. "A Brief Introduction to Neural Networks. 2007." URL http://www. dkriesel. com (2007).
- http://neuralnetworksanddeeplearning.com/chap2.html

Any Questions!?

Thank you