Chemical Engineering 4H03

Artificial Neural Networks (ANNs)

Deep Learning

Jake Nease
McMaster University

https://allisonmaruska.files.wordpress.com/2015/10/we-need-to-go-deeper.jpg?w=640

Where are We?

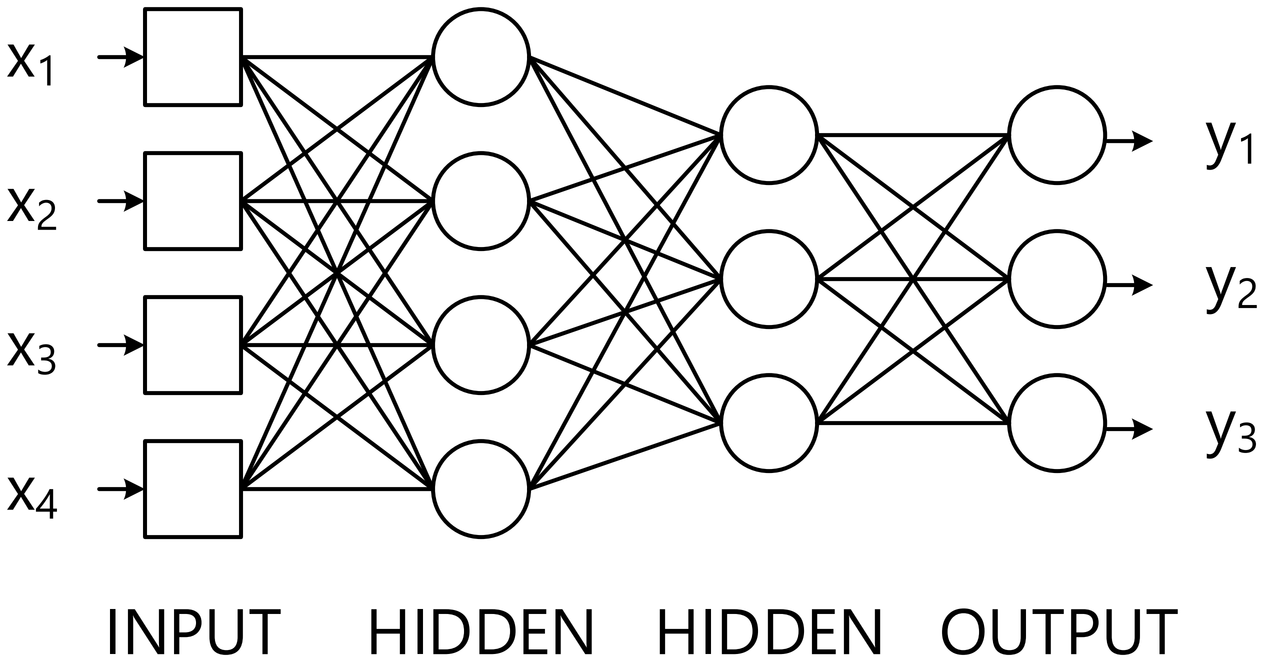
- We have looked at "simple" single-layer ANNs with several input nodes and a single output node

- Where to now?
  - Multiple input AND output nodes
  - Addition of more hidden layers
  - Underlying math and analysis
  - Coded examples

- Where will we end up?
  - Discussion on more advanced deep networks
  - Applications of deep networks (image/speech recognition)

Deep Learning

- **Deep Learning** is the phrase coined for having multiple hidden layers in an ANN
  - The hidden layers can have any number of nodes
  - Each node in every hidden layer has an activation function



Quantifying Activations

- Recall that each layer's output corresponds to the **input** of the subsequent layer:

$$\boldsymbol{y}^{(l)} = \varphi\left(\boldsymbol{v}^{(l)}\right)$$

The l means "layer l"

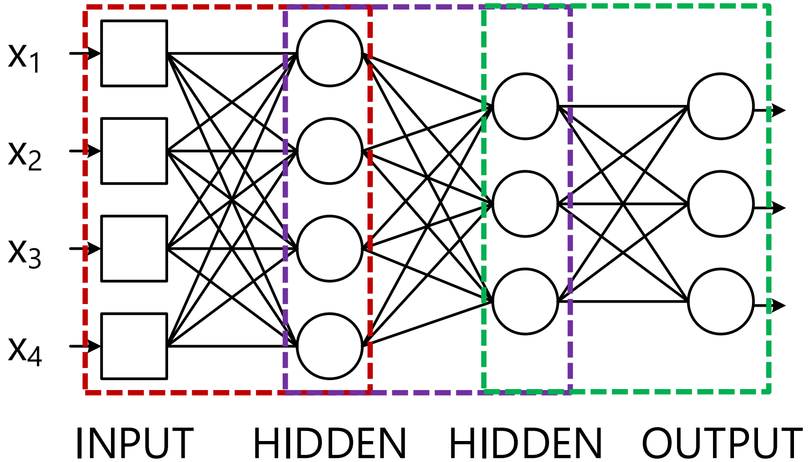
Note that $y^{(0)}$ is the OUTPUT of the input layer (layer 0)

$$\boldsymbol{y}^{(1)} = \varphi\left(W^{(1)}\boldsymbol{y}^{(0)} + \boldsymbol{b}^{(1)}\right)$$

Note that $b^{(l)}$ is the BIAS of layer l

$$\boldsymbol{y}^{(2)} = \varphi\left(W^{(2)}\boldsymbol{y}^{(1)} + \boldsymbol{b}^{(2)}\right)$$

$$\boldsymbol{y}^{(3)} = \varphi\left(W^{(3)}\boldsymbol{y}^{(2)} + \boldsymbol{b}^{(3)}\right)$$



Quantifying The First Layer

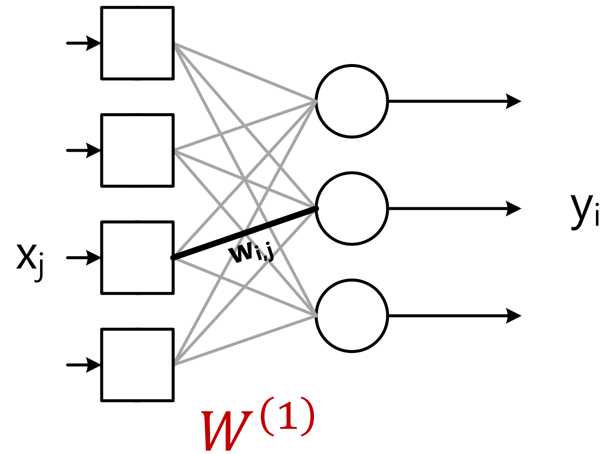
- Recall that each layer's output corresponds to the **input** of the subsequent layer:

$$\boldsymbol{y}^{(1)} = \varphi\{\boldsymbol{v}^{(1)}\} \qquad \boldsymbol{v}^{(1)} = W^{(1)}\boldsymbol{x}^{(1)} + \boldsymbol{b}^{(1)}$$

Note here that each activation function can be applied to each row (or all be the same)

$$\begin{bmatrix} y_1^{(1)} \\ y_2^{(1)} \\ y_3^{(1)} \end{bmatrix} = \varphi\left\{ \begin{bmatrix} w_{11}^{(1)} & w_{12}^{(1)} & w_{13}^{(1)} & w_{14}^{(1)} \\ w_{21}^{(1)} & w_{22}^{(1)} & w_{23}^{(1)} & w_{24}^{(1)} \\ w_{31}^{(1)} & w_{32}^{(1)} & w_{33}^{(1)} & w_{34}^{(1)} \end{bmatrix} \begin{bmatrix} x_1^{(1)} \\ x_2^{(1)} \\ x_3^{(1)} \\ x_4^{(1)} \end{bmatrix} + \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \\ b_3^{(1)} \end{bmatrix} \right\}$$

$$\begin{bmatrix} y_1^{(1)} \\ y_2^{(1)} \\ y_3^{(1)} \end{bmatrix} = \varphi\left\{ \begin{bmatrix} w_{11}^{(1)} & w_{12}^{(1)} & w_{13}^{(1)} & w_{14}^{(1)} \\ w_{21}^{(1)} & w_{22}^{(1)} & w_{23}^{(1)} & w_{24}^{(1)} \\ w_{31}^{(1)} & w_{32}^{(1)} & w_{33}^{(1)} & w_{34}^{(1)} \end{bmatrix} \begin{bmatrix} y_1^{(0)} \\ y_2^{(0)} \\ y_3^{(0)} \\ y_4^{(0)} \end{bmatrix} + \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \\ b_3^{(1)} \end{bmatrix} \right\}$$



$$\boldsymbol{y}^{(1)} = \varphi\left(W^{(1)}\boldsymbol{y}^{(0)} + \boldsymbol{b}^{(1)}\right)$$

Quantifying The Second Layer

- Recall that each layer's output corresponds to the **input** of the subsequent layer:

$$\boldsymbol{y}^{(2)} = \varphi\{\boldsymbol{v}^{(2)}\}$$

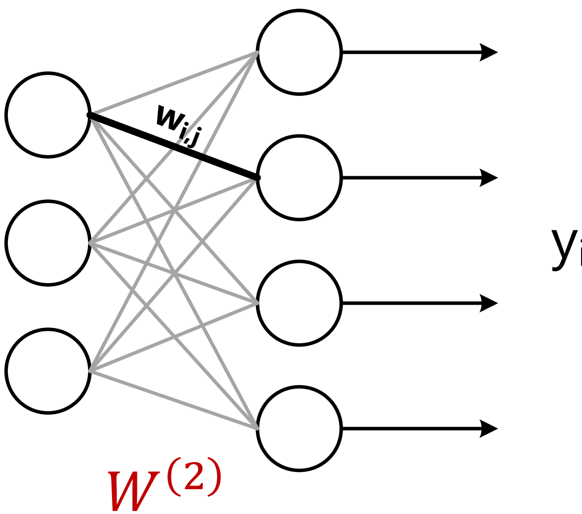
$$\boldsymbol{v}^{(2)} = W^{(2)}\boldsymbol{y}^{(1)} + \boldsymbol{b}^{(2)}$$

$$\boldsymbol{y}^{(1)} \leftarrow \boldsymbol{x}^{(2)}$$

Just consider the output from layer 1 as the input to layer 2!

$$\begin{bmatrix} y_1^{(2)} \\ y_2^{(2)} \\ y_3^{(2)} \end{bmatrix} = \varphi\left\{\begin{bmatrix} w_{11}^{(2)} & w_{12}^{(2)} & w_{13}^{(2)} \\ w_{21}^{(2)} & w_{22}^{(2)} & w_{23}^{(2)} \\ w_{31}^{(2)} & w_{32}^{(2)} & w_{33}^{(2)} \\ w_{41}^{(2)} & w_{42}^{(2)} & w_{43}^{(2)} \end{bmatrix}\begin{bmatrix} x_1^{(2)} \\ x_2^{(2)} \\ x_3^{(2)} \end{bmatrix} + \begin{bmatrix} b_1^{(2)} \\ b_2^{(2)} \\ b_3^{(2)} \\ b_4^{(2)} \end{bmatrix}\right\}$$

$$\begin{bmatrix} y_1^{(2)} \\ y_2^{(2)} \\ y_3^{(2)} \end{bmatrix} = \varphi\left\{\begin{bmatrix} w_{11}^{(2)} & w_{12}^{(2)} & w_{13}^{(2)} \\ w_{21}^{(2)} & w_{22}^{(2)} & w_{23}^{(2)} \\ w_{31}^{(2)} & w_{32}^{(2)} & w_{33}^{(2)} \\ w_{41}^{(2)} & w_{42}^{(2)} & w_{43}^{(2)} \end{bmatrix}\begin{bmatrix} y_1^{(1)} \\ y_2^{(1)} \\ y_3^{(1)} \end{bmatrix} + \begin{bmatrix} b_1^{(2)} \\ b_2^{(2)} \\ b_3^{(2)} \\ b_4^{(2)} \end{bmatrix}\right\}$$



$$\boldsymbol{y}^{(2)} = \varphi\left(W^{(2)}\boldsymbol{y}^{(1)} + \boldsymbol{b}^{(2)}\right)$$

Quantifying The _______ Layer

- And so on

- SO – our method of training each layer using the delta rule can be applied to **all** layers of the network sequentially!
  - One question though… We only have the error ϵ for the output layer, so how do we relate that to the hidden layers?
  - We **KNOW** that the inputs to layer l can be influenced by weights in layer $(l-1)$, but what "should" those weights be?
    - We don't know if we don't know the desired inputs!

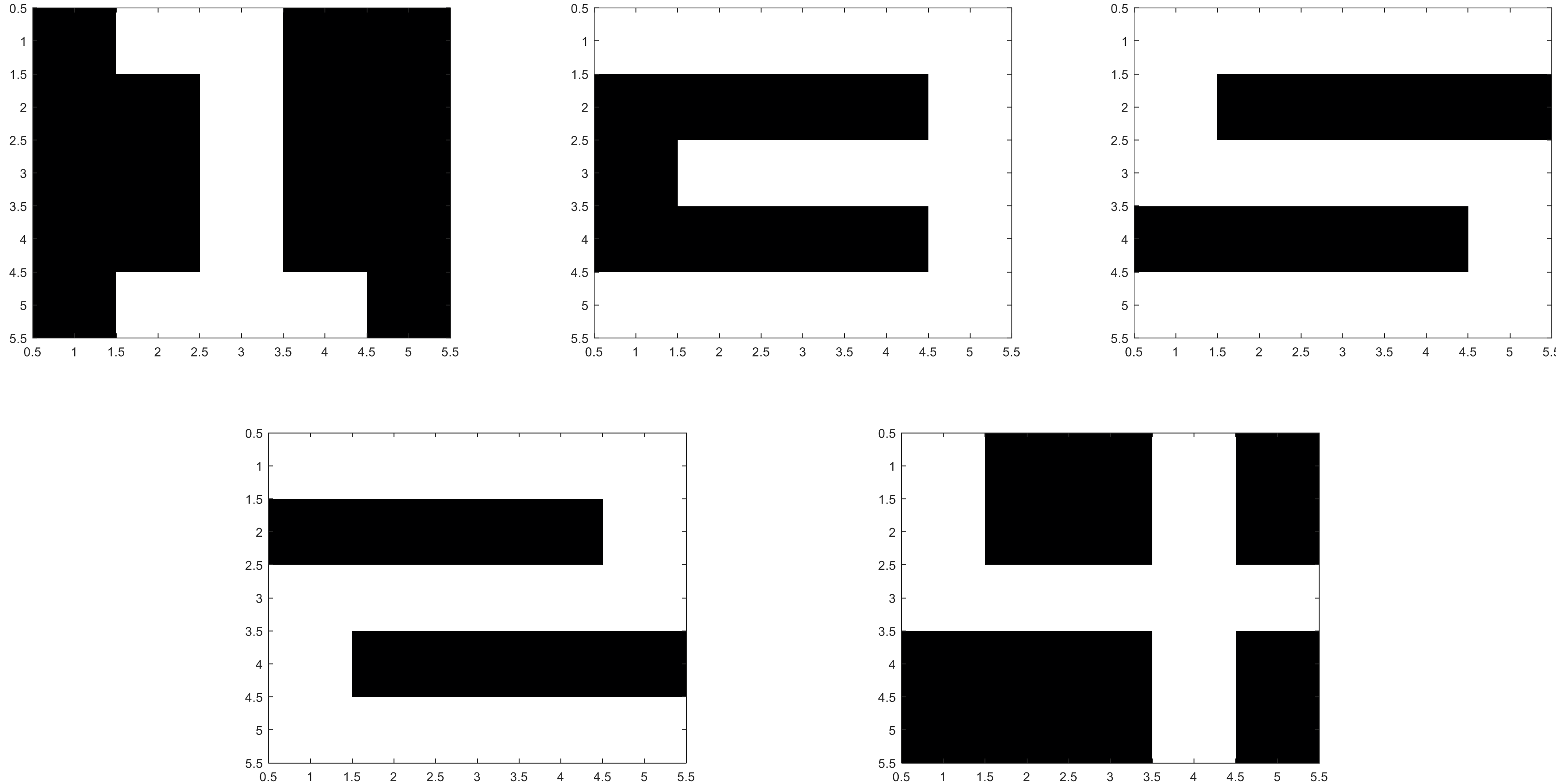
- Our strategy is actually quite simple – we will treat the outputs of each layer as the inputs to the next!

Training Deep Networks

Here's Hoping you can FATHOM it

A MOTIVATING EXAMPLE

- Consider the following five digital digits (1…5)



A MOTIVATING EXAMPLE

- As matrices, they are 5 × 5 matrices of 0 or 1:

0	1	1	0	0
0	0	1	0	0
0	0	1	0	0
0	0	1	0	0
0	1	1	1	0

1	1	1	1	1
0	0	0	0	1
0	1	1	1	1
0	0	0	0	1
1	1	1	1	1

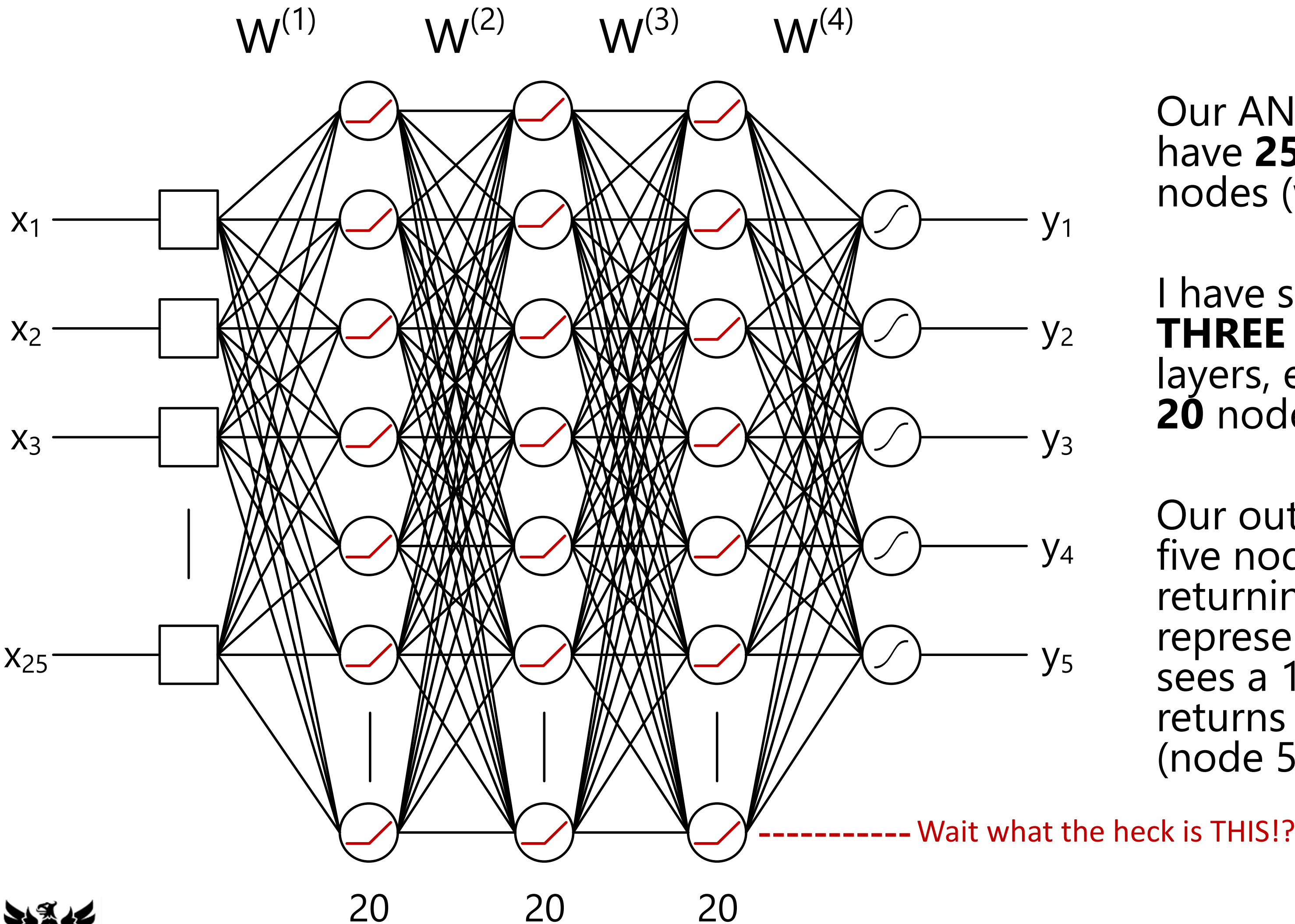
1	1	1	1	1
1	0	0	0	0
1	1	1	1	1
0	0	0	0	1
1	1	1	1	1

1	1	1	1	1
0	0	0	0	1
1	1	1	1	1
1	0	0	0	0
1	1	1	1	1

1	0	0	1	0
1	0	0	1	0
1	1	1	1	1
0	0	0	1	0
0	0	0	1	0

A MOTIVATING EXAMPLE

- We desire to train a deep ANN that recognizes these digits



Our ANN will have **25** input nodes (why?)

I have selected **THREE** hidden layers, each with **20** nodes (why?)

Our output will be five nodes, each returning 0…1, representing if it sees a 1 (node 1 returns 1) to 5 (node 5 returns 1)

We'll need some tools…

- **All ANNs** require a "vector" of inputs $\boldsymbol{x}$
  - Our 25-pixel image is not going to work
  - However, we can **unpack** the image into one vector!

"1"

0	1	1	0	0
0	0	1	0	0
0	0	1	0	0
0	0	1	0	0
0	1	1	1	0

0
0
0
0
0
1
0
0
0
1
1
1
1
1
1
0
0
0
0
1
0
0
0
0
0

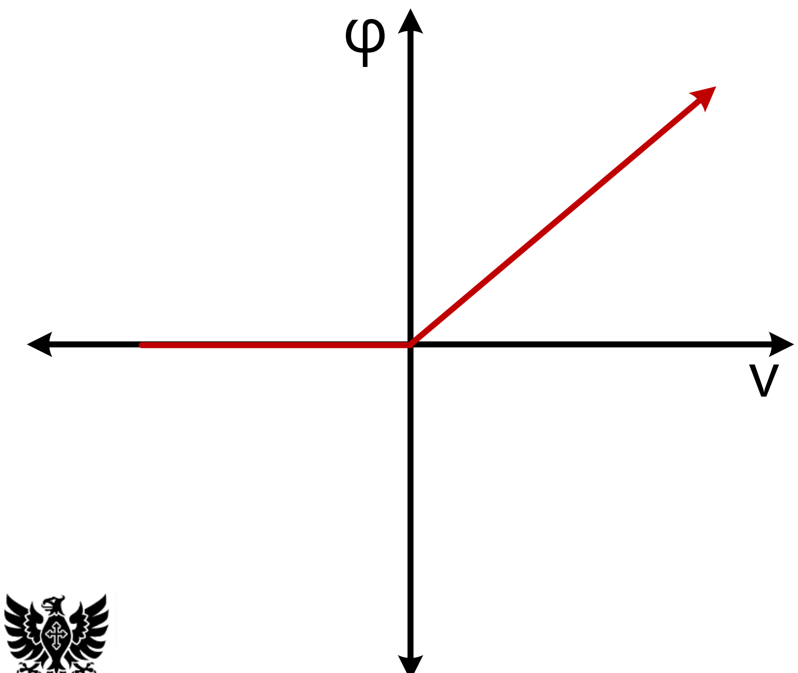
1
1
1
0
0
0
0
1
0
0
0
0
1
0
0
1
1
1
1
1
0
0
1
0
0

"4"

1	0	0	1	0
1	0	0	1	0
1	1	1	1	1
0	0	0	1	0
0	0	0	1	0

We'll need some tools…

- We have a new **activation function**
  - Deep networks tend to behave nicely with the ReLU function
  - ReLU is for "**Re**ctified **L**inear **U**nit"
  - ReLU has been shown to aid in **passing errors backwards** through the ANN
  - Sigmoids (for example) "diminish" the effects of hidden layers, thus preventing the ANN from exploiting the full value of hidden layers

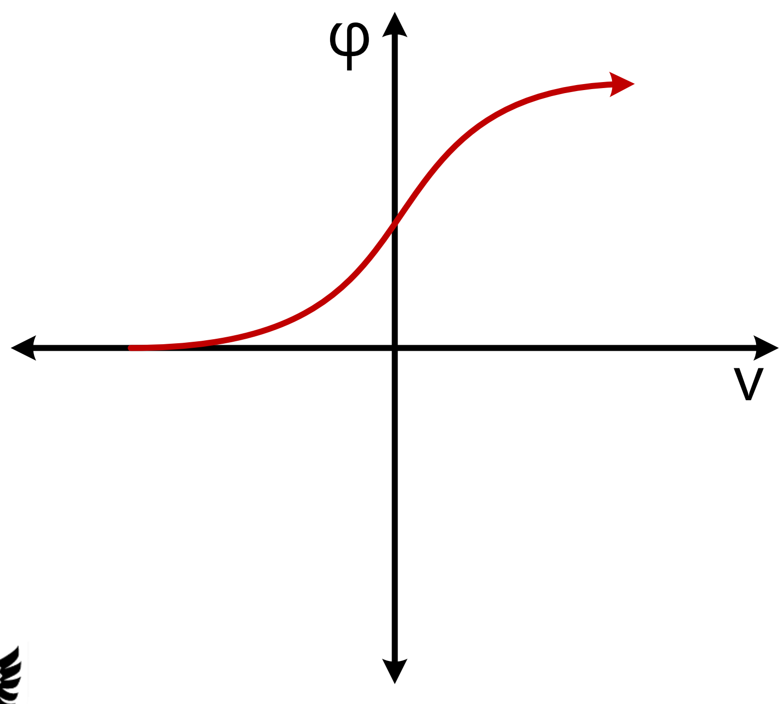


$$\varphi(v) = \begin{cases} 0; & v \leq 0 \\ v; & v > 0 \end{cases}$$

$$\varphi'(v) = \begin{cases} 0; & v \leq 0 \\ 1; & v > 0 \end{cases}$$

We'll need some tools…

- We have a new **output activation function**
  - Classification networks (such as this one) tend to use the **softmax** function
  - Softmax is a logistical function similar to sigmoid
  - Weights probabilities so that the total probability is 1.0



Here, $\boldsymbol{v}$ needs to be a vector, as opposed to the sigmoid which has its own value

$$\varphi(v_i) = \frac{e^{v_i}}{\sum_k e^{v_k}}$$

Back Propagation Procedure

- Back propagation begins with computing the OUTPUT of the ANN given initial guesses for weights $W^{(1)} \dots W^{(L)}$
  - Note that I am assuming we have L layers here (including the output)
  - In our example, we have $L = 4$ layers

1. Initialize the weights $W^{(1)} \dots W^{(L)}$
2. In a loop, until convergence
   1. FOR all data points $\boldsymbol{x}$
      1. Compute the ANN output $\hat{\boldsymbol{y}}$ through all layers
      2. FOR each layer $L \dots l \dots 1$ in the ANN (working **backwards**)
         1. Compute the error $\boldsymbol{\epsilon}^{(\boldsymbol{l})}$
         2. Compute $\boldsymbol{\delta}^{(l)}$
         3. **Propagate error backwards to $\boldsymbol{\epsilon}^{(\boldsymbol{l}-\mathbf{1})}$**
      3. Adjust all weights
   2. Check for convergence of $W^{(1)} \dots W^{(L)}$
   3. Return to (2.1)

Back-Propagation Algorithm

1. Initialize the weights
   - Most people use a normally distributed random number between 0 and 1

Back-Propagation Algorithm

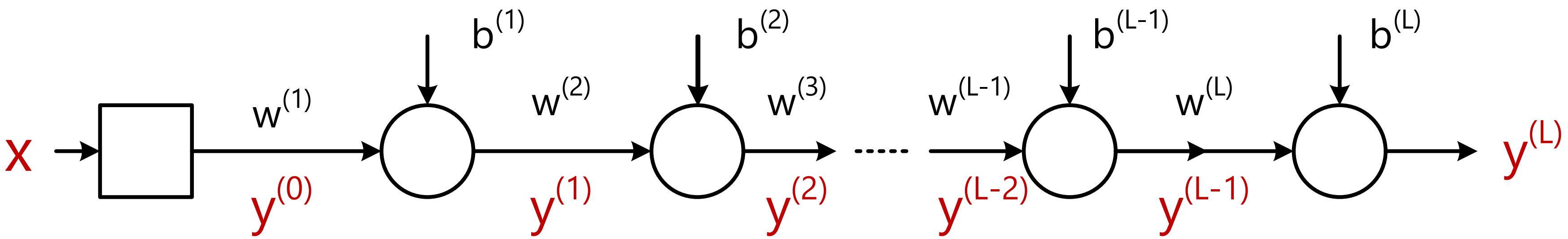
2. In a loop (for each epoch):
   1. In a loop (for each point):
      1. In a loop (for each layer l from $1 \dots L$)
         – $\boldsymbol{v}^{(l)} = W^{(l)}\boldsymbol{y}^{(l-1)} + \boldsymbol{b}^{(l)}$ ------------------------ There's the input to each layer from the ANN
         – $\boldsymbol{y}^{(l)} = \varphi\{\boldsymbol{v}^{(l)}\}$ -------------------- Layer output
      2. In a loop (for each layer l from $L \dots 1$)
         – IF $l = L$
           » $\boldsymbol{\epsilon}^{(L)} = \boldsymbol{y} - \boldsymbol{y}^{(L)}$
           » $\boldsymbol{\delta}^{(L)} = \varphi'\big(\boldsymbol{v}(L)\big)\boldsymbol{\epsilon}^{(L)}$
         – ELSE
           » $\boldsymbol{\epsilon}^{(l)} = \left(W^{(l+1)}\right)^T \boldsymbol{\delta}^{(l+1)}$ -------------------- **BACK PROPAGATION**
           » $\boldsymbol{\delta}^{(l)} = \varphi'\big(v(l)\big)\boldsymbol{\epsilon}^{(l)}$
      3. Update $W^{(l)} \leftarrow W^{(l)} + \alpha\boldsymbol{\delta}^{(l)}\left(\boldsymbol{y}^{(l-1)}\right)^T$ ---------------- **UPDATE**

Derivation of Back Propagation

Episode VI: Return of the Chain Rule

An Illustrative Example

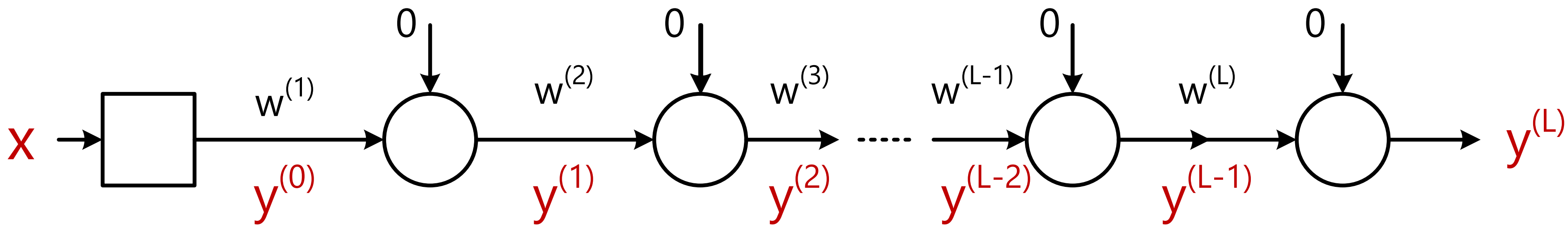
- To see how back-propagation works, consider this ANN



- We are going to make a couple of assumptions…
  - The biases for each node exist for now (removed later)
  - Each layer only has one node (relaxed later)

- What is our objective?
  - To minimize the error between $y^{(L)}$ and y

An Illustrative Example



- Let's assign an error function that accepts an input
  - For the sake of argument, let's consider the SSE

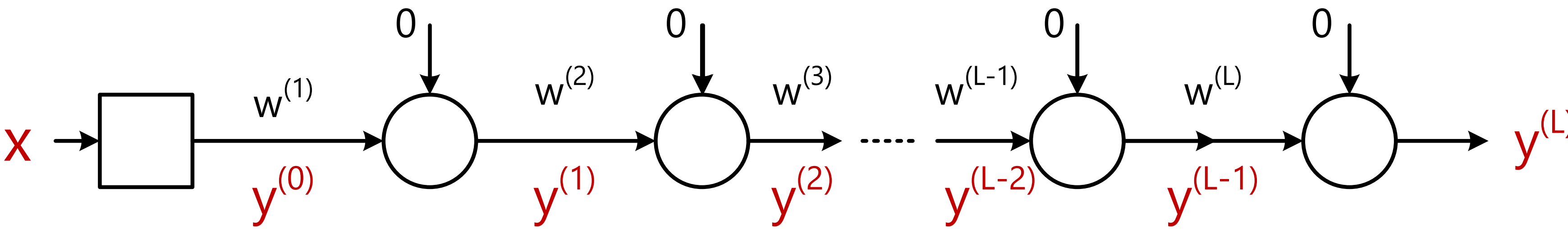
$$\mathcal{E} = \frac{1}{2}\left(y^{(L)} - y\right)^2$$

- We know that $v^{(L)}$ and $y^{(L)}$ are:

$$v^{(L)} = w^{(L)}y^{(L-1)} + b^{(L)}$$

$$y^{(L)} = \varphi\left(v^{(L)}\right)$$

An Illustrative Example



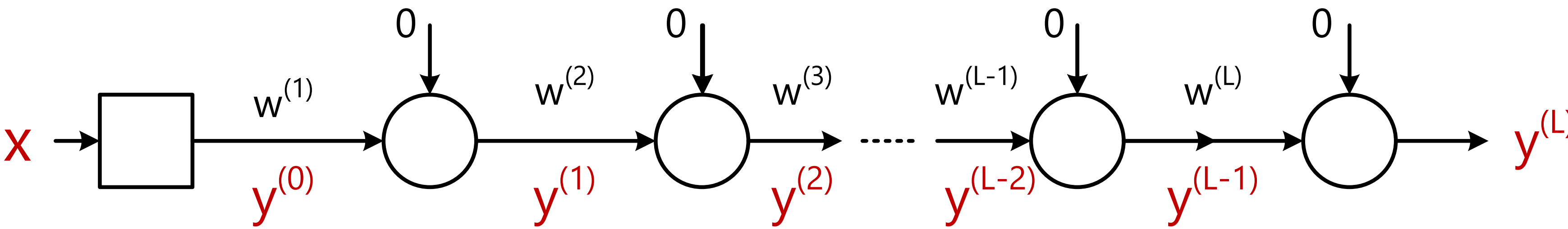
- We want to ask ourselves, what is the derivative of $\mathcal{E}$ with respect to our decision variables?
- Recall these decisions are $w^{(L)}$, $b^{(L)}$ and $y^{(L-1)}$

$$\frac{\partial \mathcal{E}}{\partial w^{(L)}} = \frac{\partial \mathcal{E}}{\partial y^{(L)}} \cdot \frac{\partial y^{(L)}}{\partial v^{(L)}} \cdot \frac{\partial v^{(L)}}{\partial w^{(L)}}$$

For those of us that might not remember, this is the fundamental application of the chain rule

- So, now we just need to figure out each piece!

An Illustrative Example



- $\frac{\partial \mathcal{E}}{\partial y^{(L)}} = \frac{\partial}{\partial y^{(L)}}\left\{\frac{1}{2}\left(y^{(L)} - y\right)^2\right\} \qquad = \left(y^{(L)} - y\right)$
- $\frac{\partial y^{(L)}}{\partial v^{(L)}} = \frac{\partial}{\partial v^{(L)}}\left\{\varphi\left(v^{(L)}\right)\right\} \qquad = \varphi'\left(v^{(L)}\right)$
- $\frac{\partial v^{(L)}}{\partial w^{(L)}} = \frac{\partial}{\partial w^{(L)}}\left\{w^{(L)} y^{(L-1)} + b^{(L)}\right\} \qquad = y^{(L-1)}$

$$\frac{\partial \mathcal{E}}{\partial w^{(L)}} = \left(y^{(L)} - y\right) \cdot \varphi'\left(v^{(L)}\right) \cdot y^{(L-1)}$$

An Illustrative Example

- Moreover, we can do the same thing *wrt* $b^{(L)}$

$$\frac{\partial \mathcal{E}}{\partial b^{(L)}} = \frac{\partial \mathcal{E}}{\partial y^{(L)}} \cdot \frac{\partial y^{(L)}}{\partial v^{(L)}} \cdot \frac{\partial v^{(L)}}{\partial b^{(L)}}$$

- $\frac{\partial \mathcal{E}}{\partial y^{(L)}} = \frac{\partial}{\partial y^{(L)}} \left\{ \frac{1}{2} \left(y^{(L)} - y \right)^2 \right\} \qquad = \left(y^{(L)} - y \right)$
- $\frac{\partial y^{(L)}}{\partial v^{(L)}} = \frac{\partial}{\partial v^{(L)}} \left\{ \varphi\left(v^{(L)} \right) \right\} \qquad = \varphi'\left(v^{(L)} \right)$
- $\frac{\partial v^{(L)}}{\partial b^{(L)}} = \frac{\partial}{\partial b^{(L)}} \left\{ w^{(L)} y^{(L-1)} + b^{(L)} \right\} \qquad = 1$

$$\frac{\partial \mathcal{E}}{\partial b^{(L)}} = \left(y^{(L)} - y \right) \cdot \varphi'\left(v^{(L)} \right) \cdot 1$$

An Illustrative Example

- One more time, we can do the same thing *wrt* $y^{(L-1)}$

$$\frac{\partial \mathcal{E}}{\partial b^{(L)}} = \frac{\partial \mathcal{E}}{\partial y^{(L)}} \cdot \frac{\partial y^{(L)}}{\partial v^{(L)}} \cdot \frac{\partial v^{(L)}}{\partial y^{(L-1)}}$$

- $\frac{\partial \mathcal{E}}{\partial y^{(L)}} = \frac{\partial}{\partial y^{(L)}}\left\{\frac{1}{2}\left(y^{(L)} - y\right)^2\right\} \qquad = \left(y^{(L)} - y\right)$
- $\frac{\partial y^{(L)}}{\partial v^{(L)}} = \frac{\partial}{\partial v^{(L)}}\left\{\varphi\left(v^{(L)}\right)\right\} \qquad = \varphi'\left(v^{(L)}\right)$
- $\frac{\partial v^{(L)}}{\partial y^{(L-1)}} = \frac{\partial}{\partial y^{(L-1)}}\left\{w^{(L)} y^{(L-1)} + b^{(L)}\right\} \qquad = w^{(L)}$

$$\frac{\partial \mathcal{E}}{\partial y^{(L-1)}} = \left(y^{(L)} - y\right) \cdot \varphi'\left(v^{(L)}\right) \cdot w^{(L)}$$

Some Remarks

- Our three derivatives should make a great deal of sense
  - $\frac{\partial \mathcal{E}}{\partial w^{(L)}} = \left(y^{(L)} - y\right) \cdot \varphi'\left(v^{(L)}\right) \cdot y^{(L-1)}$
    - The rate of change of the error *wrt* each weight is proportional to the input to layer L
  - $\frac{\partial \mathcal{E}}{\partial b^{(L)}} = \left(y^{(L)} - y\right) \cdot \varphi'\left(v^{(L)}\right) \cdot 1$
    - The rate of change of the error is unit-proportional to the bias in layer L. In other words, it only depends on how the activation function changes *w.r.t.* the input
  - $\frac{\partial \mathcal{E}}{\partial y^{(L-1)}} = \left(y^{(L)} - y\right) \cdot \varphi'\left(v^{(L)}\right) \cdot w^{(L)}$
    - The rate of change of the error *wrt* the input to the layer is proportional to weights in layer L. This means that inputs connected to nodes with higher weights will impact $\mathcal{E}$ more. **This is the fundamental relationship for back propagation**.

Let's Generalize

- First of all, let's note that what we just did was for a **single output instance**, whereas there could be many!
  - We just have to express $\mathcal{E}$ as the sum of those errors:

$$\mathcal{E} = \sum_{i=1}^{I} \frac{1}{2}\left(y_i^{(L)} - y_i\right)^2$$

- Next, let's recognize that the bias term can also be treated as the output of an activation node with an output of $\varphi(\cdot) = 1$
  - Thus, the biases will get their own weight w

$$\frac{\partial \mathcal{E}}{\partial b^{(L)}} = \left(y^{(L)} - y\right) \cdot \varphi'\left(v^{(L)}\right) \cdot 0 = 0$$

Let's Generalize

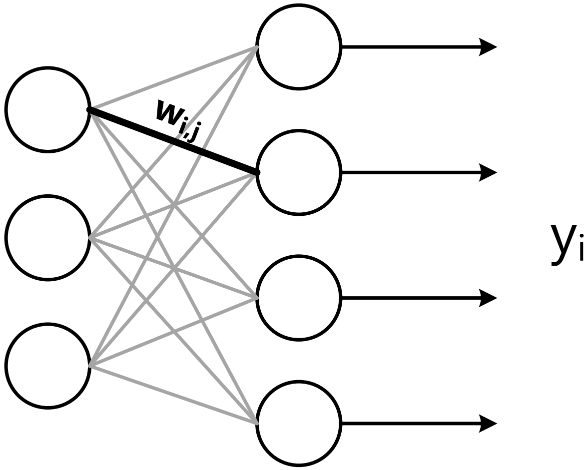
- Next, recognize that if you have multiple inputs to a given node, you need to take the derivative *wrt* each of the weights $w_{i,j}$

$$\frac{\partial \mathcal{E}}{\partial w_{i,j}^{(L)}} = \frac{\partial \mathcal{E}}{\partial y_i^{(L)}} \cdot \frac{\partial y_i^{(L)}}{\partial v_i^{(L)}} \cdot \frac{\partial v_i^{(L)}}{\partial w_{i,j}^{(L)}}$$

I'll remind you that $\frac{\partial y_i^{(L)}}{\partial v_i^{(L)}} = \varphi'(v_i)$

- Finally, let's make a sneaky notation change and use $\delta_i^{(L)} \triangleq \frac{\partial \mathcal{E}}{\partial v_i^{(L)}} = \frac{\partial \mathcal{E}}{\partial y_i^{(L)}} \cdot \frac{\partial y_i^{(L)}}{\partial v_i^{(L)}}$
- So NOWWWWW I can write

$$\frac{\partial \mathcal{E}}{\partial w_{i,j}^{(L)}} = \delta_i^{(L)} \cdot y_j^{(L-1)}$$

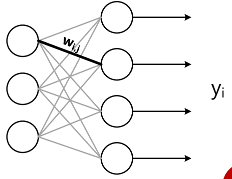


- That's for the FINAL layer, to let's make one more jump and refer to layer l instead

$$\frac{\partial \mathcal{E}}{\partial w_{i,j}^{(l)}} = \delta_i^{(l)} \cdot y_j^{(l-1)}$$

THIS is our expression for the delta-rule!

Now Apply It to Layer L



- WHEW. OK. That was fun! Let's see it work…
- Recall $\mathcal{E} = \sum_{i=1}^{I} \frac{1}{2}\left(y_i^{(L)} - y_i\right)^2$
- $\delta_i^{(L)} \triangleq \frac{\partial \mathcal{E}}{\partial v_i^{(L)}} = \frac{\partial \mathcal{E}}{\partial y_i^{(L)}} \cdot \frac{\partial y_i^{(L)}}{\partial v_i^{(L)}} = \left(y_i^{(L)} - y_i\right) \cdot \varphi'\left(v_i^{(L)}\right)$
- $\frac{\partial \mathcal{E}}{\partial w_{i,j}^{(L)}} = \delta_i^{(L)} \cdot y_j^{(L-1)}$
- … Not so bad.

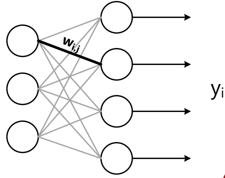
Now Apply It to Layer $L-1$

- Easy enough for the first layer since $\frac{\partial \mathcal{E}}{\partial y_i^{(L)}}$ is easy. But what about δ_i^{L-1}?
  - Note this is a bit annoying but the indices of the input/output nodes are swapped here to represent i as the "current" layer. This makes way more sense in vector form. We'll get there in a second

- $\delta_i^{(L-1)} \triangleq \frac{\partial \mathcal{E}}{\partial v_i^{(L-1)}} = \frac{\partial y_i^{(L-1)}}{\partial v_i^{(L-1)}} \cdot \frac{\partial \mathcal{E}}{\partial y_i^{(L-1)}} = \varphi'\left(v_i^{(L-1)}\right) \cdot \sum_j w_{j,i}^{(L)} \delta_j^{(L)}$

- $\frac{\partial \mathcal{E}}{\partial w_{i,j}^{(L-1)}} = \delta_i^{(L-1)} \cdot y_j^{(L-2)}$

And in General…

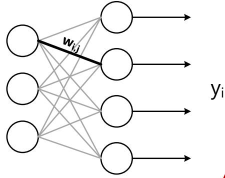


- In general, we can compute $\frac{\partial \mathcal{E}}{\partial y_i^{(l)}}$ as:

- $\delta_i^{(l)} \triangleq \frac{\partial \mathcal{E}}{\partial v_i^{(l)}} = \frac{\partial y_i^{(l)}}{\partial v_i^{(l)}} \cdot \frac{\partial \mathcal{E}}{\partial y_i^{(l)}} = \varphi'\left(v_i^{(l)}\right) \cdot \sum_j w_{j,i}^{(l+1)} \delta_j^{(l+1)}$

- $\frac{\partial \mathcal{E}}{\partial w_{i,j}^{(l)}} = \delta_i^{(l)} \cdot y_j^{(l-1)}$

As Vectors…



- IMO it is WAY easier to think of this using vectors and weight matrices for each layer:

- $\boldsymbol{\delta}^{(l)} \triangleq \nabla \mathcal{E}_{\boldsymbol{v}^{(l)}} = \nabla y^{(l)}_{\boldsymbol{v}^{(l)}} \cdot \nabla \mathcal{E}^{(l)}_{\boldsymbol{y}^{(l)}}$

- $\boldsymbol{\delta}^{(l)} = \varphi'\left(\boldsymbol{v}^{(l)}\right) \cdot \left(W^{(l+1)}\right)^T \delta^{(l+1)}$

- $\frac{\partial \mathcal{E}}{\partial W^{(l)}} = \boldsymbol{\delta}^{(l)} \cdot \left(\boldsymbol{y}^{(l-1)}\right)^T$

- And THIS is the equation we use in our algorithm!
  – It will always work since we always have $\boldsymbol{\delta}^{(l+1)}$ ☺

Annnnd Done!

- Now you just have to do that for all points!

- Once you are finished calculating $\frac{\partial \mathcal{E}}{\partial W^{(l)}}$, you can assume you are using a straight-up gradient search and consider this $\Delta W^{(l)}$ and update the weights
  - Note that you can update W every point (SGF) or take batches
  - ALSO note that this is why the training regimen is known as a modification of the *gradient search*. Each expression for $\frac{\partial \mathcal{E}}{\partial W^{(l)}}$ is in fact the gradient of the ERROR *wrt* every entry in $W^{(l)}$
  - We can use other optimization methods now, too! Since we know how to compute the gradient

Back Propagation Algorithm Again

2. In a loop (for each epoch):
    1. In a loop (for each point):
        1. In a loop (for each layer l from $1 \ldots L$)
            - $\boldsymbol{v}^{(l)} = W^{(l)}\boldsymbol{y}^{(l-1)} + \boldsymbol{b}^{(l)}$
            - $\boldsymbol{y}^{(l)} = \varphi\{\boldsymbol{v}^{(l)}\}$
        2. In a loop (for each layer l from $L \ldots 1$)
            - IF $l = L$
                - » $\boldsymbol{\epsilon}^{(L)} = \boldsymbol{y} - \boldsymbol{y}^{(L)}$
                - » $\boldsymbol{\delta}^{(L)} = \varphi'\left(\boldsymbol{v}^{(L)}\right)\boldsymbol{\epsilon}^{(L)} = \varphi'\left(\boldsymbol{v}^{(L)}\right)\left(\boldsymbol{y} - \boldsymbol{y}^{(L)}\right)$
            - ELSE
                - » $\boldsymbol{\epsilon}^{(l)} = \left(W^{(l+1)}\right)^T \boldsymbol{\delta}^{(l+1)}$
                - » $\boldsymbol{\delta}^{(l)} = \varphi'\left(\boldsymbol{v}^{(l)}\right)\boldsymbol{\epsilon}^{(l)} = \varphi'\left(\boldsymbol{v}^{(l)}\right)\left(W^{(l+1)}\right)^T \boldsymbol{\delta}^{(l+1)}$
        3. Update $W^{(l)} \leftarrow W^{(l)} + \alpha\boldsymbol{\delta}^{(l)}\left(\boldsymbol{y}^{(l-1)}\right)^T$

Now we can fundamentally see how this works!

Final Remarks

- And that, friends, is how you train your deep ANN!
- Key takeaways
  - Calculus can be used to relate changing **inner** weights to the objective function through chain rule
  - Only has to be done one layer at a time
  - First layer can be calculated directly
  - Vector/Matrix math should always be used for simplicity
- Motivating example
  - Completed in MATLAB!!