PadhAI: Backpropagation - the light math version

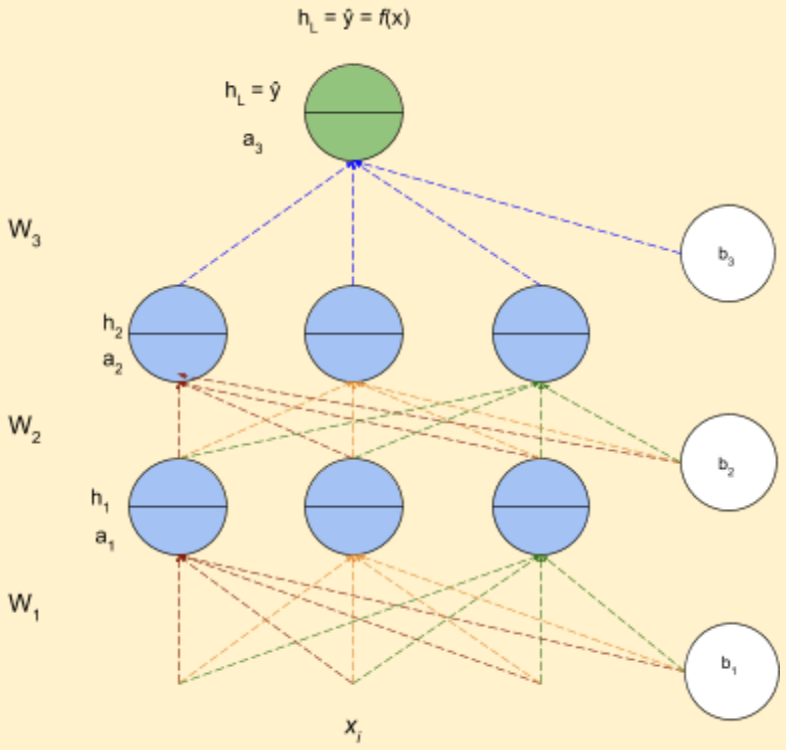
One Fourth Labs

Backpropagation (Light Math)

Setting the context

Can we use the same learning algorithm as before?

1. Here is the learning algorithm as discussed in the previous chapter, the no-math version
2. Consider the Neural Network with the following configuration



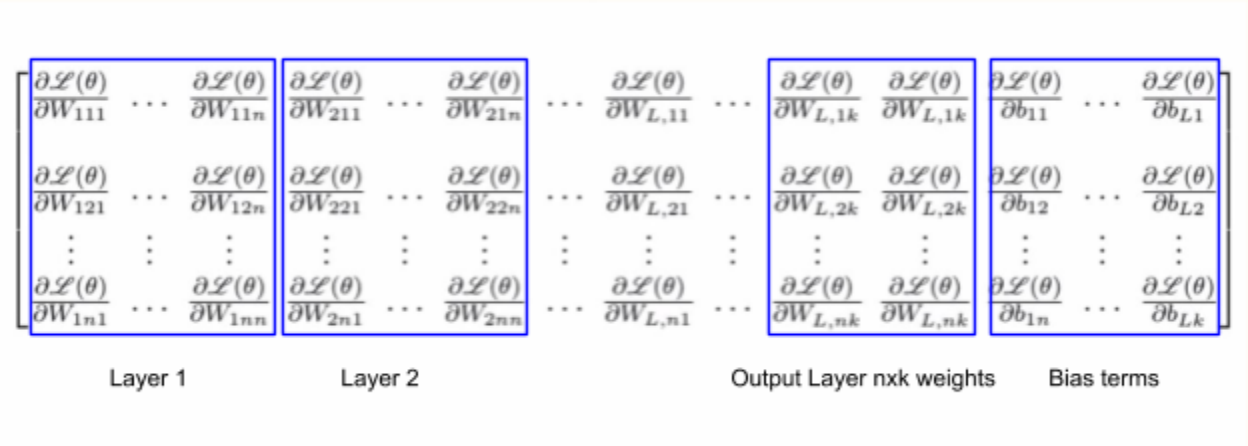
3. The algorithm
   a. **Initialise:** $w_{111}, w_{112}, \ldots w_{313}, b_1, b_2, b_3$ randomly
   b. **Iterate over data**
      i. Compute $\hat{y}$
      ii. Compute L(w,b) Cross-entropy loss function
      iii. $w_{111} = w_{111} - \eta \Delta w_{111}$
      iv. $w_{112} = w_{112} - \eta \Delta w_{112}$
      ...
      v. $w_{313} = w_{111} - \eta \Delta w_{313}$
      vi. $b_i = b_i + \eta \Delta b_i$
      vii. Pytorch/Tensorflow have functions to compute $\frac{\delta l}{\delta w}$ *and* $\frac{\delta l}{\delta b}$
   c. **Till satisfied**
      i. Number of epochs is reached (ie 1000 passes/epochs)
      ii. Continue till Loss < ε (some defined value)

PadhAI: Backpropagation - the light math version

One Fourth Labs

---

4. In this section, we will be looking at the light-math version, where we will be computing the derivatives
5. Derivatives for all layers from 1 to L



6. Once we know the gradients, we can use them in the Gradient Descent algorithm to compute the weights of the network

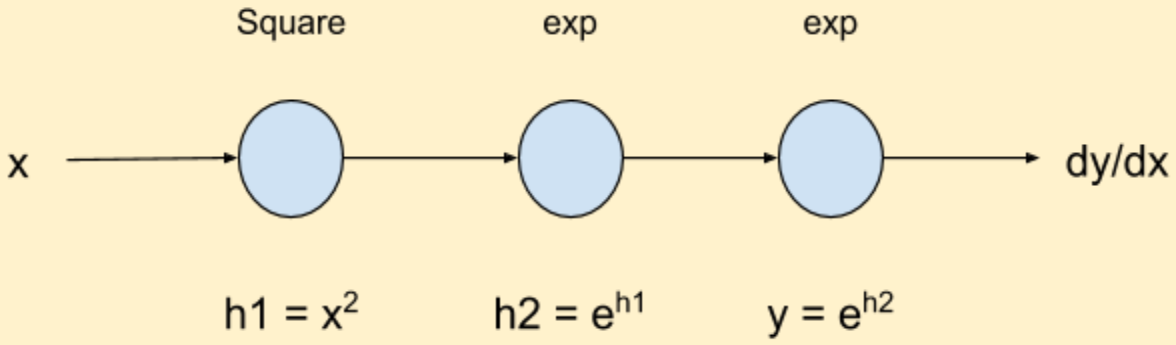
PadhAI: Backpropagation - the light math version

One Fourth Labs

Revisiting Basic Calculus

Let's do a quick recap of some basic calculus concepts

1. Here are some examples of simple derivatives
   a. $\frac{de^x}{dx} = e^x$
   b. $\frac{dx^2}{dx} = 2x$
   c. $\frac{d(\frac{1}{x})}{dx} = -\frac{1}{x^2}$
2. Now, let's look at a slightly more complicated derivative
   a. a $\frac{de^{x^2}}{dx}$
   b. Here, we break it into two parts
      i. $h = x^2$
      ii. $y = e^{(term)}$
   c. Therefore, $\frac{de^{x^2}}{dx} = \frac{dy}{dh}\frac{dh}{dx}$
   d. $\frac{dh}{dx} = 2x$
   e. $\frac{dy}{dh} = e^h$
   f. $\frac{de^{x^2}}{dx} = \frac{dy}{dh}\frac{dh}{dx} = (e^h).(2x) = (e^{x^2}).(2x) = 2xe^{x^2}$
   g. Here, the output is a composite function of the input. This process of breaking the equation into parts and solving them sequentially is known as **Chain Rule**
   h. Consider another example $\frac{de^{e^{x^2}}}{dx}$
   i. Here is the flow diagram of chain rule applied to the above equation



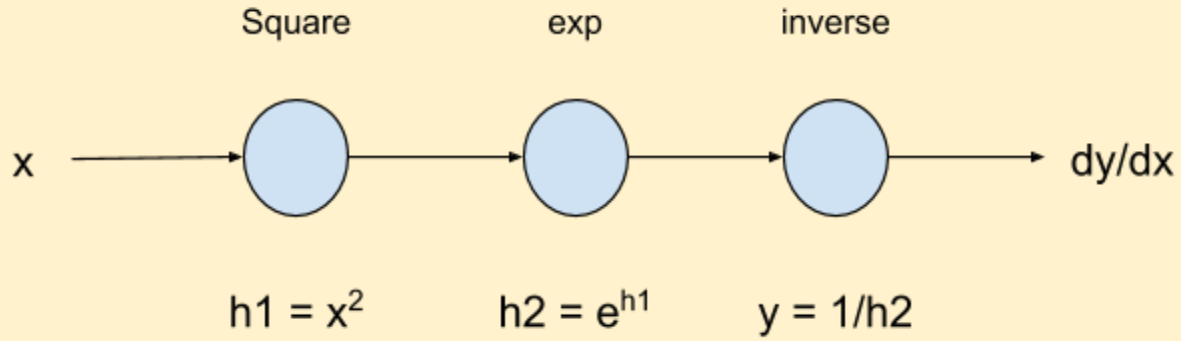
   j. $\frac{de^{e^{x^2}}}{dx} = \frac{dy}{dh2}\frac{dh2}{dh1}\frac{dh1}{dx} = (e^{h2}).(e^{h1}).(2x) = (e^{e^{x^2}}).(e^{x^2}).(2x) = 2xe^{e^{x^2}}e^{x^2}$
   k. Another example $\frac{d(\frac{1}{e^{x^2}})}{dx}$

PadhAI: Backpropagation - the light math version

One Fourth Labs

---

l. Flow diagram of chain rule



m. $$\frac{d(\frac{1}{e^{x^2}})}{dx} = \frac{dy}{dh2}\frac{dh2}{dh1}\frac{dh1}{dx} = (\frac{-1}{(h2)^2}).(e^{h1}).(2x) = (\frac{-1}{(e^{x^2})^2}).(e^{x^2}).(2x) = (\frac{-1}{(e^{x^2})^2}).2xe^{x^2}$$

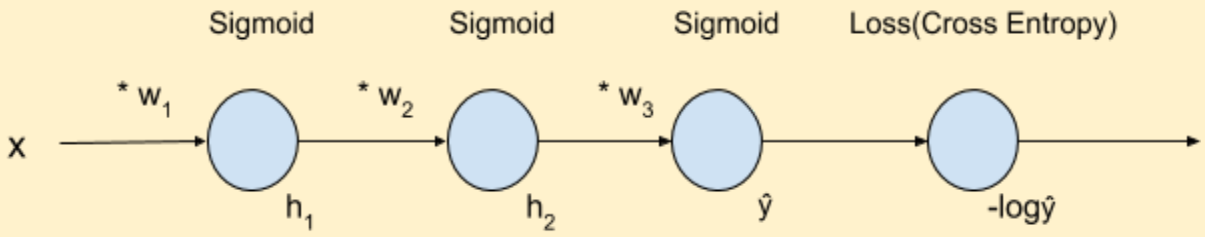
PadhAI: Backpropagation - the light math version

One Fourth Labs

Why do we care about the chain rule of derivatives

Importance of chain rule in Deep Learning

1. Let us look at a sample chain rule flow of a shallow neural network



2. Here, the output ŷ is a composite dependent on input x and all of the parameters w
3. $Loss\ function: L = f(x, w_1, w_2, w_3)$
4. Now, for the gradient, we want the derivative of the loss function with respect to the various weights $\frac{\partial L}{\partial w_i}$
5. If we want the derivative w.r.t w_2 then we do the following $\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial h_2} \frac{\partial h_2}{\partial w_2}$
6. Here, computation happens from input layer to the output layer ie forward propagation
7. Derivative calculation happens backwards from the output layer to the input, ie back propagation

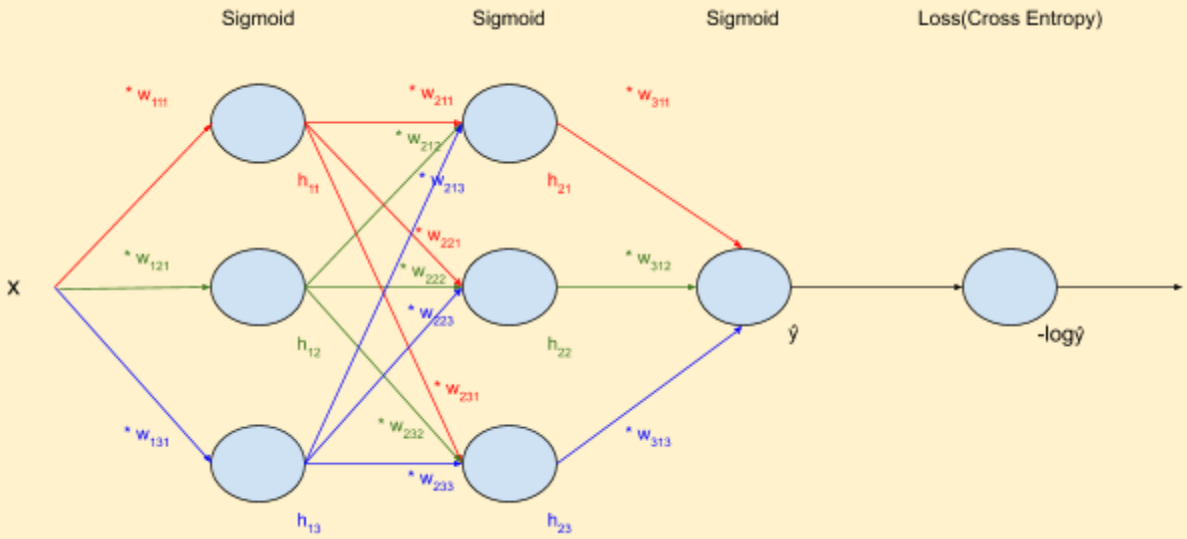
PadhAI: Backpropagation - the light math version

One Fourth Labs

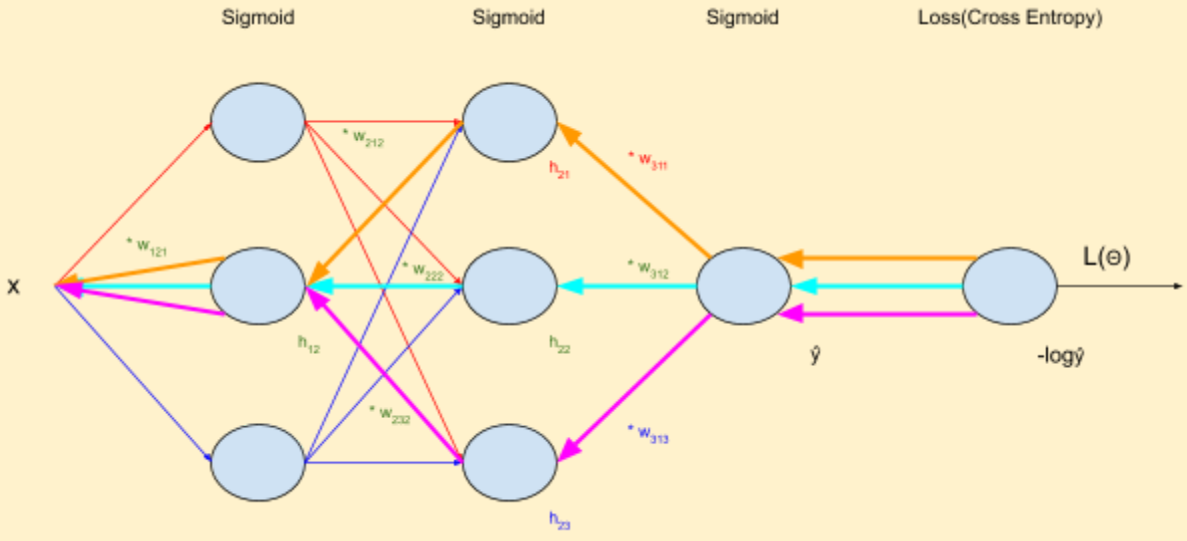
Applying chain rule across multiple paths

Importance of chain rule in deep learning

1. Let us look at a more complex neural network



2. In the shallow Neural Network from the previous example, we apply the chain rule along a straight path. However, in a more practical Neural Network as shown above, the chain rule needs to be applied across multiple parallel paths in order to find a particular gradient
3. For example, to calculate $\frac{\partial L}{\partial w_{121}}$ we need to operate along 3 different paths



4. Summing up the derivatives across the three paths (cyan, orange and pink) will give us the required derivative $\frac{\partial L}{\partial w_{121}}$
5. This scales across as many paths as there are in the neural network.
6. Here, these are not regular derivatives but partial derivatives.

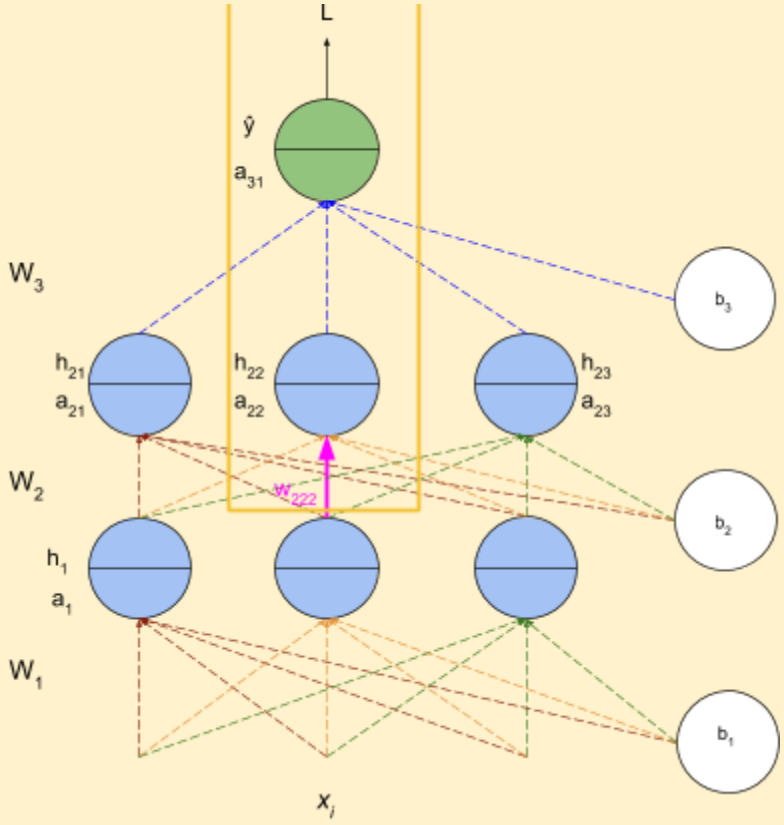
PadhAI: Backpropagation - the light math version

One Fourth Labs

Applying chain rule in a neural network

How many derivatives do we need to compute and how do we compute them?

1. Let's focus on the highlighted weight(w_{222}) of the following neural network



2. To learn this weight, we have to compute the partial derivative w.r.t loss function a

$$(w_{222})_{t+1} = (w_{222})_t - \eta * (\frac{\partial L}{\partial w_{222}})$$

3. We can calculate $\frac{\partial L}{\partial w_{222}}$ as follows
   a. $\frac{\partial L}{\partial w_{222}} = (\frac{\partial L}{\partial a_{22}}).(\frac{\partial a_{22}}{\partial w_{222}})$
   b. $\frac{\partial L}{\partial w_{222}} = (\frac{\partial L}{\partial h_{22}}).(\frac{\partial h_{22}}{\partial a_{22}}).(\frac{\partial a_{22}}{\partial w_{222}})$
   c. $\frac{\partial L}{\partial w_{222}} = (\frac{\partial L}{\partial a_{31}}).(\frac{\partial a_{31}}{\partial h_{22}}).(\frac{\partial h_{22}}{\partial a_{22}}).(\frac{\partial a_{22}}{\partial w_{222}})$
   d. $\frac{\partial L}{\partial w_{222}} = (\frac{\partial L}{\partial \hat{y}}).(\frac{\partial \hat{y}}{\partial a_{31}}).(\frac{\partial a_{31}}{\partial h_{22}}).(\frac{\partial h_{22}}{\partial a_{22}}).(\frac{\partial a_{22}}{\partial w_{222}})$
4. Thus, by breaking the partial derivative into all the subdivisions along that path and multiplying it, we will get the required solution.

PadhAI: Backpropagation - the light math version

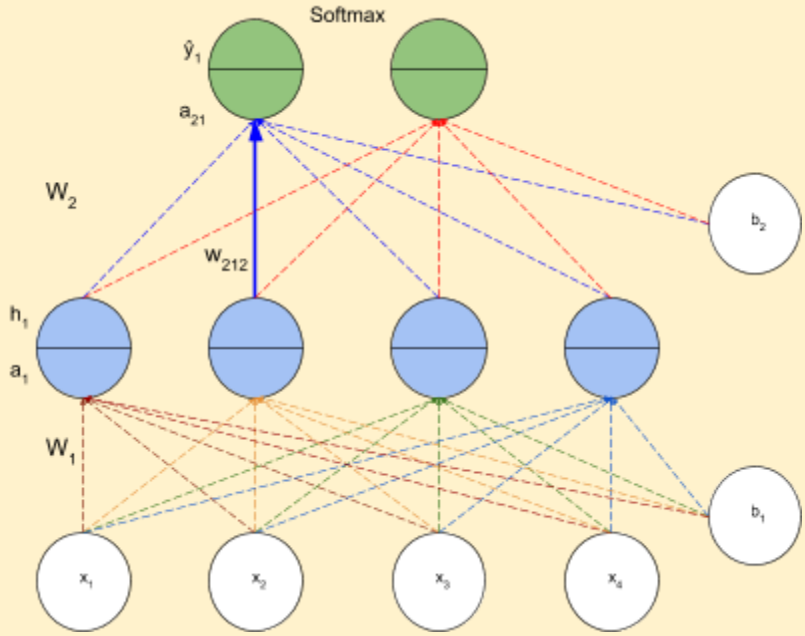
One Fourth Labs

Partial Derivatives with respect to a

Part 1

How do we compute partial derivatives

1. The following neural network will be used to demonstrate the calculations



2. Here are the parameters of the network
   a. $b = [\,0.5 \;\; 0.3\,]$
   b.

$$W_1 = \begin{bmatrix} 0.1 & 0.3 & 0.8 & -0.4 \\ -0.3 & -0.2 & 0.5 & 0.5 \\ -0.3 & 0 & 0.5 & 0.4 \\ 0.2 & 0.5 & -0.9 & 0.7 \end{bmatrix}$$

   c.

$$W_2 = \begin{bmatrix} 0.5 & 0.8 & 0.2 & 0.4 \\ 0.5 & 0.2 & 0.3 & -0.5 \end{bmatrix}$$

   d. $x = [\,2 \;\; 5 \;\; 3 \;\; 3\,]$ true distribution $y = [\,1 \;\; 0\,]$

---

3. Now, we want to find the partial derivative w.r.t w_{212} as highlighted in the figure $\frac{\partial L}{\partial w_{212}}$
4. $\frac{\partial L}{\partial w_{212}} = (\frac{\partial L}{\partial a_{21}}).(\frac{\partial a_{21}}{\partial w_{212}}) = (\frac{\partial L}{\partial \hat{y}_1}).(\frac{\partial \hat{y}_1}{\partial a_{21}}).(\frac{\partial a_{21}}{\partial w_{212}})$
5. We will solve the above equation sequentially
   a. Consider square error loss function L
   b. $\frac{\partial L}{\partial \hat{y}_1} = \Sigma^2_{i=1}(y_i - \hat{y}_i)^2$
      i. $\frac{\partial L}{\partial \hat{y}_1} = \frac{\partial}{\partial y_1}[(y_1 - \hat{y}_1)^2 + (y_2 - \hat{y}_2)^2]$
      ii. Here, the y_2 terms get cancelled, leaving $\frac{\partial}{\partial y_1}[(y_1 - \hat{y}_1)^2] = -2(y_1 - \hat{y}_1)$

PadhAI: Backpropagation - the light math version

One Fourth Labs

Partial Derivatives with respect to a

Part 2

How do we compute partial derivatives

1. Let us continue calculating the partial derivative of L w.r.t w_{212}
2. Solving the equation sequentially
   a. Let's look at the second partial derivative $\frac{\partial \hat{y}_1}{\partial a_{21}}$
      i. Here, $\hat{y}_1 = (\frac{e^{a_{21}}}{e^{a_{21}} + e^{a_{22}}})$, this is the softmax applied on a_{21}
      ii. To make it easier to compute, multiply both numerator and denominator by e^{-a21}
      iii. $\hat{y}_1 = (\frac{e^{-a_{21}}}{e^{-a_{21}}})(\frac{e^{a_{21}}}{e^{a_{21}} + e^{a_{22}}}) = \frac{1}{1 + e^{-(a_{21} - a_{22})}}$
      iv. $\frac{\partial \hat{y}_1}{\partial a_{21}} = \frac{\partial}{\partial a_{21}}(\frac{1}{1 + e^{-(a_{21} - a_{22})}})$
      v. $\frac{\partial \hat{y}_1}{\partial a_{21}} = (\frac{-1}{(1 + e^{-(a_{21} - a_{22})})^2}).(1).(e^{-(a_{21} - a_{22})}).(-1) = (\frac{1}{1 + e^{-(a_{21} - a_{22})}}).(\frac{e^{-(a_{21} - a_{22})}}{1 + e^{-(a_{21} - a_{22})}})$
      vi. Rewriting the terms $\frac{\partial \hat{y}_1}{\partial a_{21}} = \hat{y}_1(1 - \hat{y}_1)$

PadhAI: Backpropagation - the light math version

One Fourth Labs

Partial Derivatives with respect to a

Part 3

How do we compute partial derivatives?

1. Solving the equation sequentially
   a. Let's look at the third partial derivative $\frac{\partial a_{21}}{\partial w_{212}}$
      i. Here $a_{21} = w_{211}h_{11} + w_{212}h_{12} + w_{213}h_{13} + w_{214}h_{14}$
      ii. $\frac{\partial a_{21}}{\partial w_{212}} = h_{12}$, as all other terms cancel out.
2. Consider the following output values
   a. $a_1 = W_1 * x + b_1 = [2.9 \ \ 1.4 \ \ 2.1 \ \ 2.3]$
   b. $h_1 = sigmoid(a_1) = [0.95 \ \ 0.80 \ \ 0.89 \ \ 0.91]$
   c. $a_2 = W_2 * h_1 + b_2 = [1.66 \ \ 0.45]$
   d. $\hat{y} = softmax(a_2) = [0.77 \ \ 0.23]$
   e. Squared error loss $L(\Theta) = (1 - 0.77)^2 + (1 - 0.23)^2 = 0.1058$
3. Substituting these values in our formulae
   a. $\frac{\partial L}{\partial \hat{y}_1} = -2(y_1 - \hat{y}_1) = -0.46$
   b. $\frac{\partial \hat{y}_1}{\partial a_{21}} = \hat{y}_1(1 - \hat{y}_1) = 0.1771$
   c. $\frac{\partial a_{21}}{\partial w_{212}} = h_{12} = 0.8$
   d. $\frac{\partial L}{\partial w_{212}} = (-2(y_1 - \hat{y}_1)) * (\hat{y}_1(1 - \hat{y}_1)) * (h_{12}) = (-0.46) * (0.1771) * (0.8) = -0.065$
4. Now we can calculate the updated value of w_{212}
5. $w_{212} = w_{212} - \eta(\frac{\partial L}{\partial w_{212}})$
   a. $w_{212} = 0.8 - (1) * (-0.065)$
   b. $w_{212} = 0.865$
6. We can repeat this process for each weight.

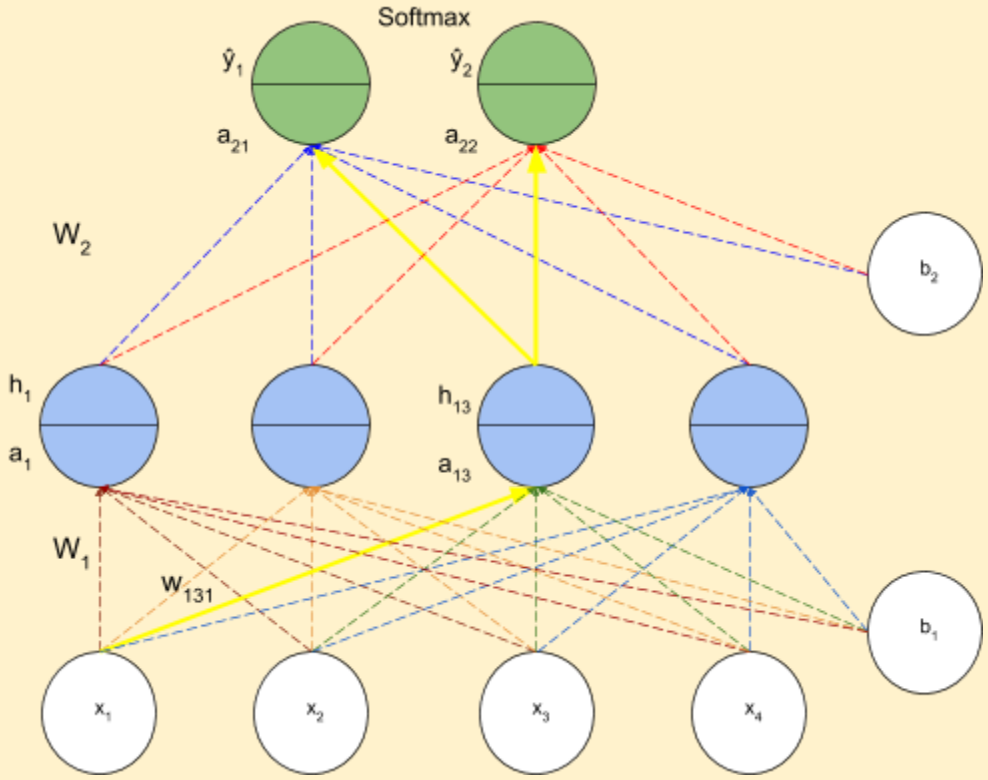
PadhAI: Backpropagation - the light math version

One Fourth Labs

Multiple Paths

Can we see one more example?

1. Let's look at a different weight from the previous example, which would require multiple paths to perform the calculations



2. Here are the parameters of the network
   a. $b = [0\ 0]$
   b.

$$W_1 = \begin{bmatrix} 0.1 & 0.3 & 0.8 & -0.4 \\ -0.3 & -0.2 & 0.5 & 0.5 \\ -0.3 & 0 & 0.5 & 0.4 \\ 0.2 & 0.5 & -0.9 & 0.7 \end{bmatrix}$$

c.

$W_2 = \begin{bmatrix} 0.5 & 0.8 & 0.2 & 0.4 \\ 0.5 & 0.2 & 0.3 & -0.5 \end{bmatrix}$

d. x = [2 5 3 3] true distribution y = [1 0]

3. Now, we want to find the partial derivative w.r.t w_{212} as highlighted in the figure $\frac{\partial L}{\partial w_{212}}$
4. $\frac{\partial L}{\partial w_{131}} = (\frac{\partial L}{\partial a_{13}}).(\frac{\partial a_{13}}{\partial w_{131}}) = (\frac{\partial L}{\partial h_{13}}).(\frac{\partial h_{13}}{\partial a_{13}}).(\frac{\partial a_{13}}{\partial w_{131}}) = (\frac{\partial L}{\partial a_{21}}.\frac{\partial a_{21}}{\partial h_{13}} + \frac{\partial L}{\partial a_{22}}.\frac{\partial a_{22}}{\partial h_{13}}).(\frac{\partial h_{13}}{\partial a_{13}}).(\frac{\partial a_{13}}{\partial w_{131}})$ a
5. The final split is $\frac{\partial L}{\partial w_{131}} = (\frac{\partial L}{\partial \hat{y}_1}.\frac{\partial \hat{y}_1}{\partial a_{21}}.\frac{\partial a_{21}}{\partial h_{13}} + \frac{\partial L}{\partial \hat{y}_2}.\frac{\partial \hat{y}_2}{\partial a_{22}}.\frac{\partial a_{22}}{\partial h_{13}}).(\frac{\partial h_{13}}{\partial a_{13}}).(\frac{\partial a_{13}}{\partial w_{131}})$
6. Let us sequentially solve both splits

$\frac{\partial L}{\partial \hat{y}_1} = -2(y_1 - \hat{y}_1) = -0.46$	$\frac{\partial L}{\partial \hat{y}_2} = -2(y_2 - \hat{y}_2) = 0.46$
$\frac{\partial \hat{y}_1}{\partial a_{21}} = \hat{y}_1(1 - \hat{y}_1) = 0.1771$	$\frac{\partial \hat{y}_2}{\partial a_{22}} = \hat{y}_2(1 - \hat{y}_2) = 0.1771$
$\frac{\partial a_{21}}{\partial h_{13}} = w_{213} = 0.2$	$\frac{\partial a_{22}}{\partial h_{13}} = w_{223} = 0.3$
$\frac{\partial h_{13}}{\partial a_{13}} = h_{13} * (1 - h_{13}) = 0.0979$	$\frac{\partial h_{13}}{\partial a_{13}} = h_{13} * (1 - h_{13}) = 0.0979$
$\frac{\partial a_{13}}{\partial w_{131}} = x_1 = 2$	$\frac{\partial a_{13}}{\partial w_{131}} = x_1 = 2$
Path1: $(-0.46 * 0.1771 * 0.2 * 0.0979 * 2) = -0.003190$	Path1: $(0.46 * 0.1771 * 0.3 * 0.0979 * 2) = 0.004785$
Sum of the paths is $\frac{\partial L}{\partial w_{131}} = 0.001595$	

7. Now we can calculate the updated value of w_{212}
8. $w_{131} = w_{131} - \eta(\frac{\partial L}{\partial w_{131}})$
   a. $w_{131} = -0.3 - (1) * (0.001595)$
   b. $w_{131} = -0.301595$
9. We can repeat this process for each weight

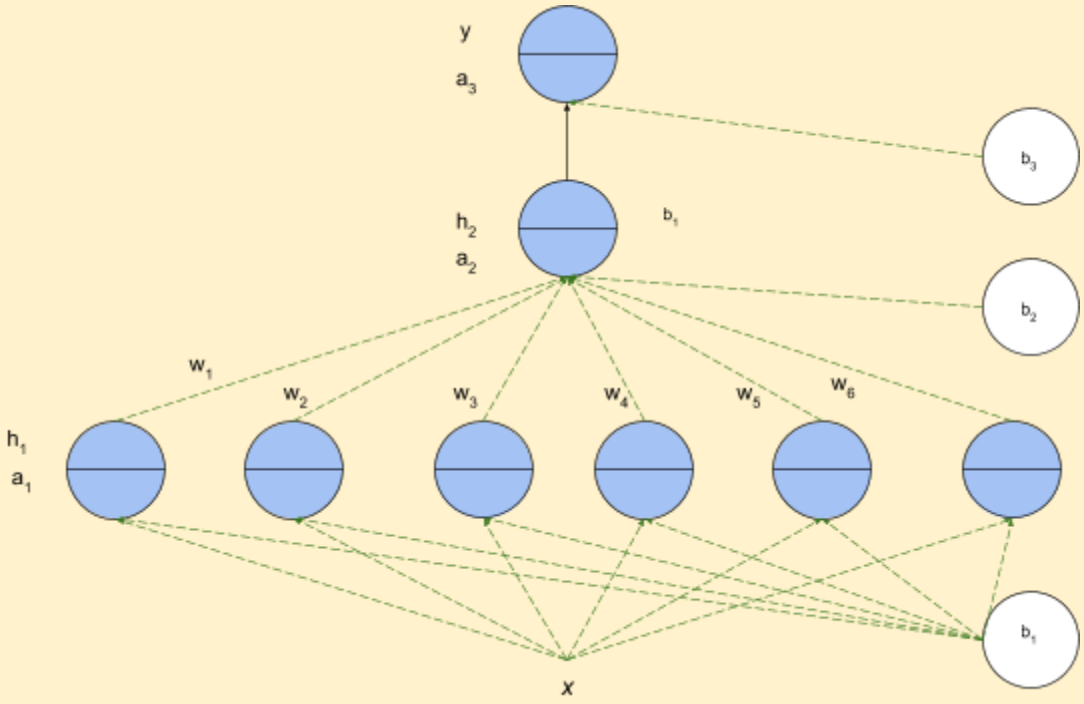
PadhAI: Backpropagation - the light math version

One Fourth Labs

Takeaways and what next?

What have we learned so far and what more do we need to learn?

1. For calculating w_{132}, the paths are almost identical to w_{131} except for the last step
2. This is applicable for the weights w_{133} and w_{134} as well
3. In the next slot (math-heavy), we will learn how to re-use a lot of the computations when calculating new weights.



4. No matter how complex the function, we can always compute the derivative w.r.t any variable using the chain rule.
5. We can reuse a lot of work by starting backwards and computing simpler elements in the chain

PadhAI: Backpropagation - the full version

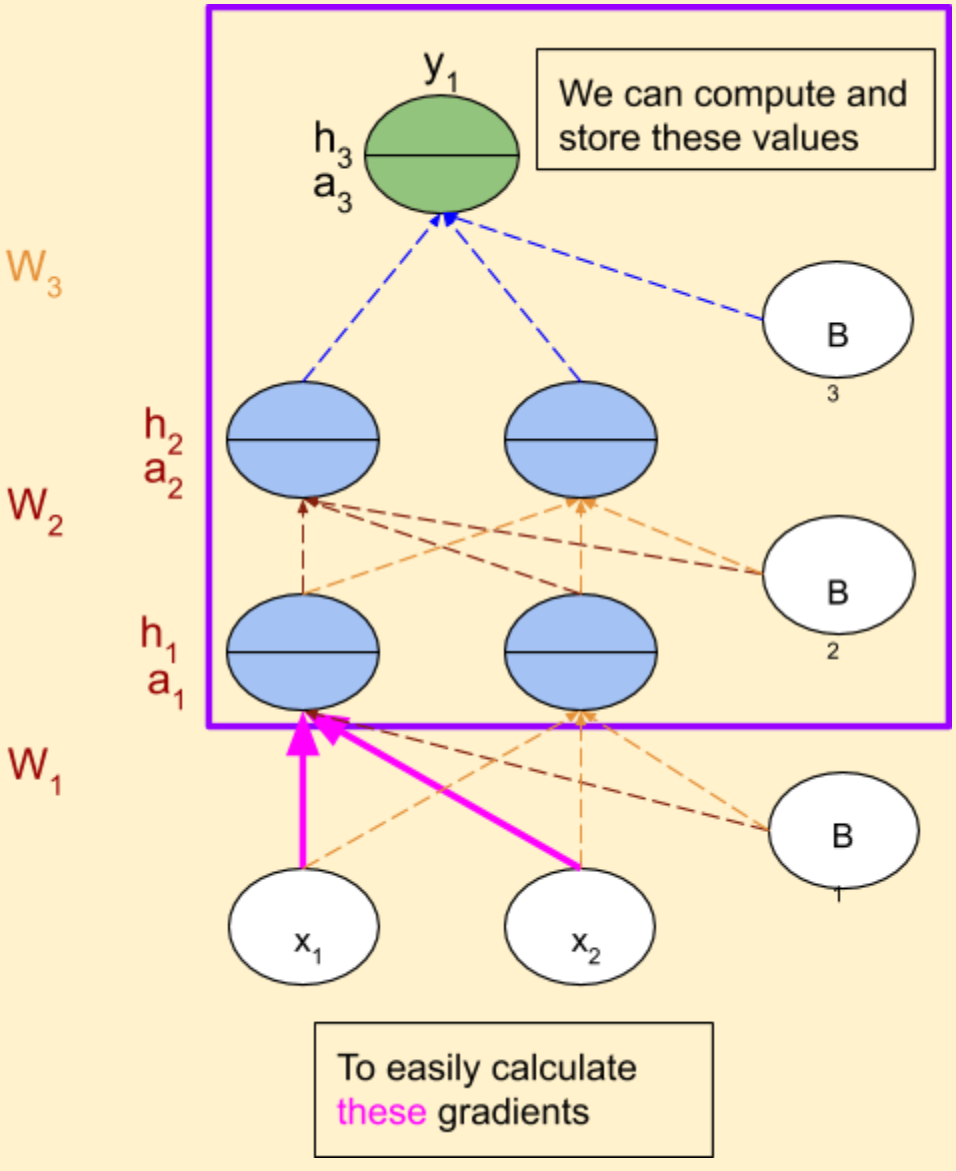
One Fourth Labs

Backpropagation (Math-heavy/Vectorized)

Setting the context

How does this differ from the previous section

1. We've looked at two different levels of complexity to the backpropagation algorithm so far
   a. No-math: A simple forward pass with no gradient calculation
   b. Light-math: Gradient calculation for each weight using chain rule
2. Now, with the Heavy-math version, our objective is to identify common calculations between different weights and re-use them to make our work simpler

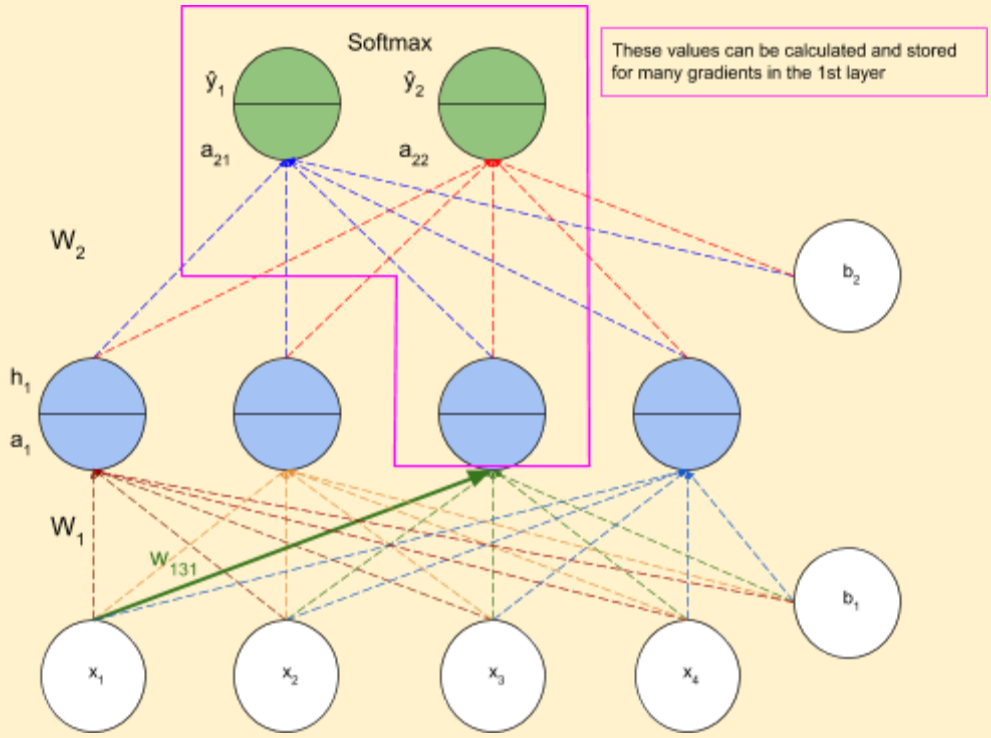


PadhAI: Backpropagation - the full version

One Fourth Labs

---

3. Let us consider the example from the light-math backpropagation chapter



4. Consider dw_{131} or $\frac{\partial L}{\partial w_{131}}$
   a. Here, we are certain about using the highlighted values for gradient computation. So we can pre-calculate and store them
   b. In the outermost layer

$$\frac{\partial L}{\partial a_2} = \begin{bmatrix} \frac{\partial L}{\partial a_{21}} \\ \\ \frac{\partial L}{\partial a_{22}} \end{bmatrix}$$

5. Similarly, storing the values of $\frac{\partial h_1}{\partial a_2}$ will prove useful down the line

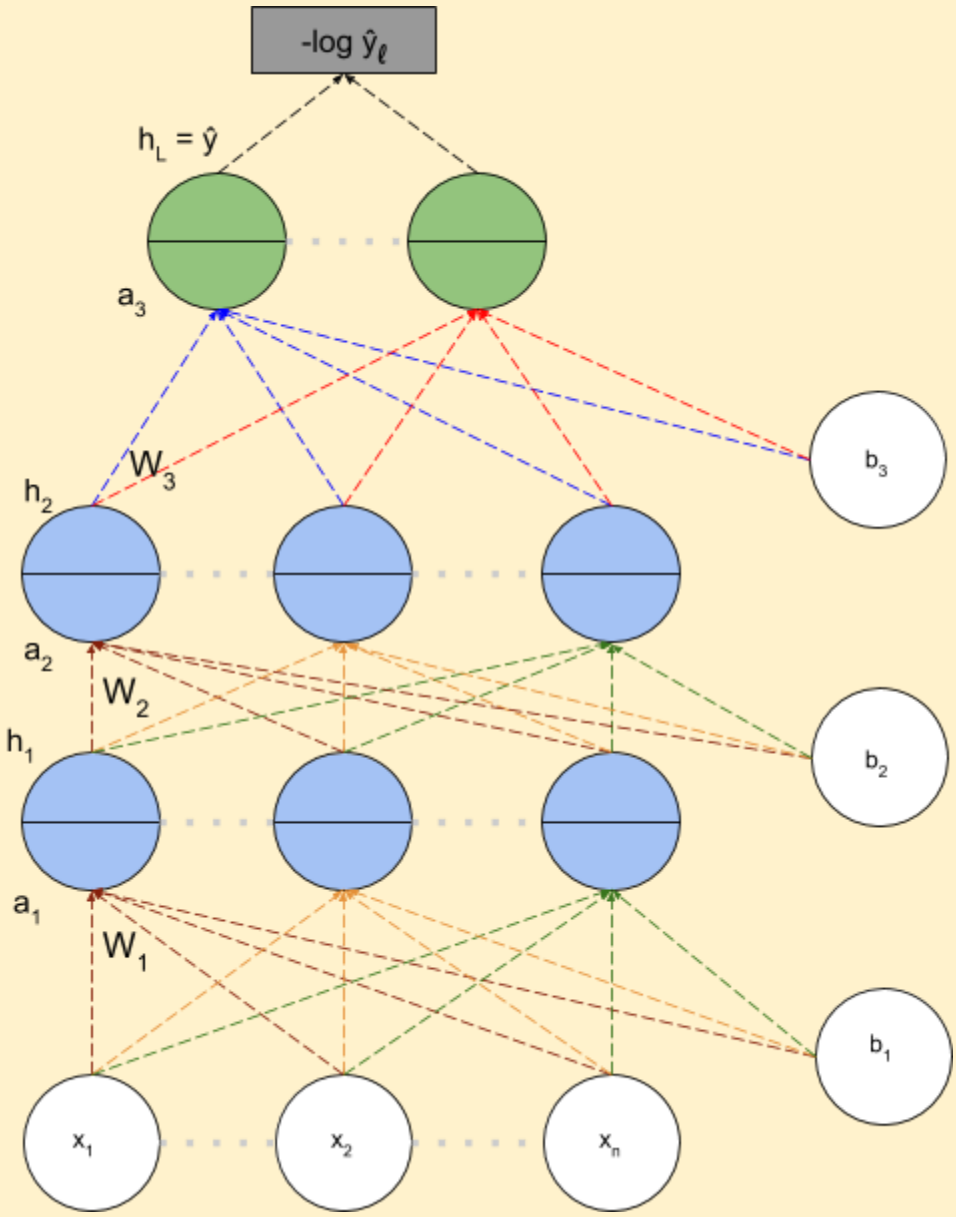
PadhAI: Backpropagation - the full version

One Fourth Labs

Intuition behind backpropagation

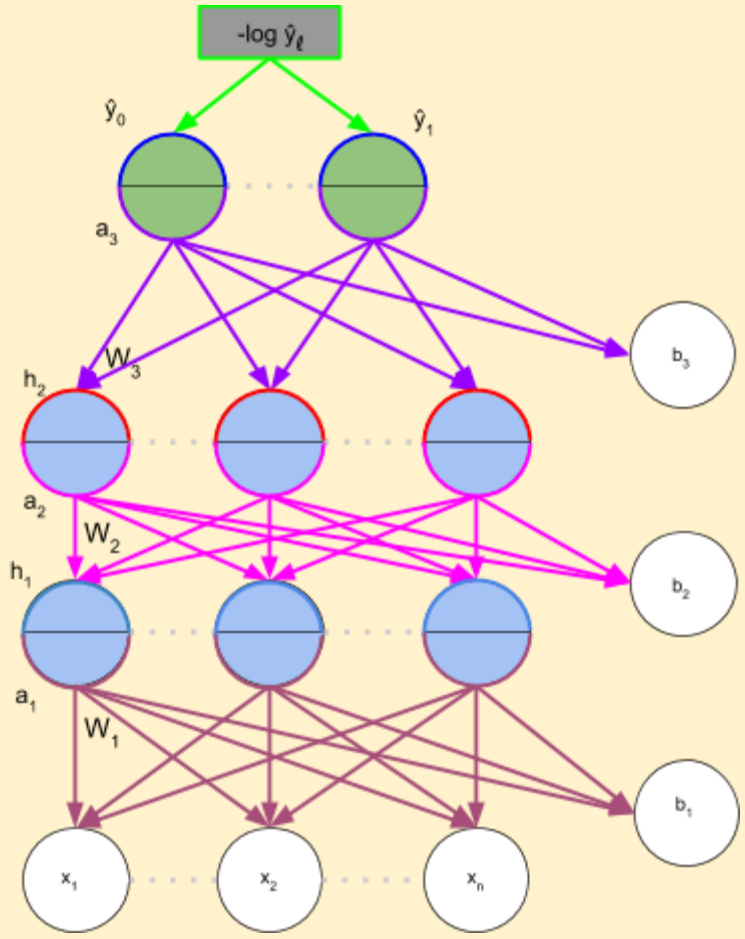
Let us look at an intuitive explanation of backpropagation

1. Let's have a quick calculus recap
   a. $f(x) = x^2$, $Derivative: \frac{df(x)}{dx}$
   b. $f(x) = x^2 + y^2$, $Partial\ Derivatives: \frac{\partial f(x)}{\partial x}\ and\ \frac{\partial f(x)}{\partial y}$ a
   c. $f(\theta) = [x, y]$, $Gradient\ \nabla_\theta = [\frac{\partial f(x)}{\partial x}, \frac{\partial f(x)}{\partial y}]$
2. Consider the following sample network



3. Now, let us assume our network has undergone some training, so we have a Loss value in hand.
4. W_1, W_2, W_3, b_1, b_2, b_3 have been updated and reflect accordingly in the Loss function

5. Now, we want to scrutinise how each parameter is responsible for the loss. So we move backwards from the Loss in a step-by-step manner



6. Stepwise calculation (while personifying the neurons and layers 😄)
   a. Step 1: The loss talks to the output layer, saying "You better take responsibility for the poor output!"
   b. Step 2: The output activation layer says, "Hey, I'm simply applying the softmax function to the input given to me by the output preactivation layer"
   c. Step 3: The output preactivation layer says, "I take responsibility for my part, but I am only as good as the hidden layer and the weights below me." After all $f(x) = \hat{y} = O(W_L h_{L-1} + b_L)$
   d. Step 4: The parameters W_3 and b_3 say "It is our mistake, **please update our values**"
   e. Step 5: However, the second hidden activation layer says "Hey, I'm simply applying the sigmoid function to the input given to me by the second hidden preactivation layer"
   f. Step 6: The second hidden preactivation layer says, "I am only as good as the hidden layer and weights below me"
   g. Step 7: The parameters W_2 and b_3 say "It is our mistake, **please update our values**"
   h. Step 8: However, the first hidden activation layer says "Hey, I'm simply applying the sigmoid function to the input given to me by the first hidden preactivation layer"
   i. Step 9: The first hidden preactivation layer says, "I am only as good as the weights below me, we cannot blame the input layer."
   j. Step 10: The last set of parameters W_1 and b_1 say "It is our mistake, please update us"

7. Thus, to arrive at the derivative of the Loss function w.r.t any of the weights, we must proceed downwards from the top. We cannot simply calculate it without knowing the preceding values.
8. Our roadmap for the rest of the module
   a. To calculate the desired gradient, we need to compute
   b. Gradient w.r.t output units
   c. Gradient w.r.t hidden units
   d. Gradient w.r.t weights and biases
   e.

$$\frac{\partial L(\theta)}{\partial W_{111}} = \frac{\partial L(\theta)}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a_3} \frac{\partial a_3}{\partial h_2} \frac{\partial h_2}{\partial a_2} \frac{\partial a_2}{\partial h_1} \frac{\partial h_1}{\partial a_1} \frac{\partial a_3}{\partial W_{111}}$$

Talk to the weight directly	Talk to the output layer	Talk to the previous hidden layer	Talk to the previous hidden layer	Talk to the weights
		works for any number of output layers		

   f. Our aim is to do these calculations for any of the possible weights using notation i, j, k instead of numbers
   g. For the rest of this exercise, our focus is on *Cross Entropy loss* and *Softmax* output.
9. To reduce the tediousness of applying the chain rule each time to get the desired gradient, we will look to re-use common values and pathways to more efficiently calculate any gradient.

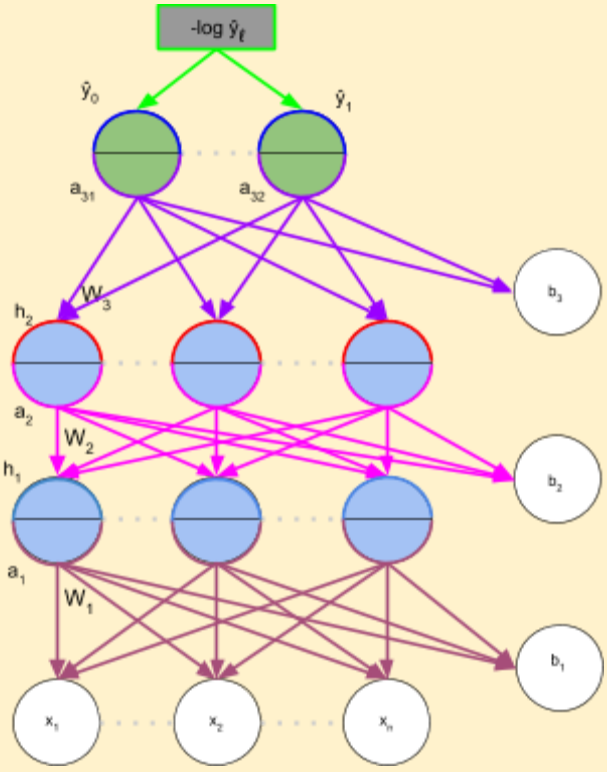
PadhAI: Backpropagation - the full version

One Fourth Labs

Understanding the dimensions of gradients

What are we interested in?

1. Consider the backpropagation illustration from the previous section



2. What we are interested in is a $\frac{\partial L(\theta)}{\partial a_{Li}} = \frac{\partial(-\log \hat{y}_l)}{\partial a_{Li}}$ (where true output y = 1, L = Layer number, *l* is the index of the correct class-label for the given input, and i is the neuron number)
3. We know that $\frac{\partial L(\theta)}{\partial a_{Li}}$ is dependent on a_{31} and a_{32}
4. Therefore, the derivative at the output layer

$$\nabla_{a_3} L(\theta) = \begin{bmatrix} \frac{\partial L}{\partial a_{31}} \\ \\ \frac{\partial L}{\partial a_{32}} \end{bmatrix}$$

5. In the above gradient, L = 3 and i ∈ {1, 2}
6. Henceforth, we can use these notations in place of numbers to simplify gradient calculation for all possible gradients.

PadhAI: Backpropagation - the full version

One Fourth Labs

Computing derivatives w.r.t Output Layer

Part 1

The first derivative in the chain

1. What we are actually interested in is:
   a. $\frac{\partial L(\theta)}{\partial a_{Li}} = \frac{\partial(-\log \hat{y}_l)}{\partial a_{Li}}$
   b. Where L = layer number, i = neuron (from 1 to k), l = index of correct output
   c. Here, we use the cross entropy loss function
   d. In the output layer L, assume we have neurons a_{L1}, a_{L2} ... a_{Lk}
   e. The output layer L involves applying the softmax function the all the neurons
   f. $\hat{y}_l = \frac{e^{a_{Ll}}}{\Sigma_i e^{a_{Li}}}$ again, (l refers to the index of the correct output neuron)
   g. Thus, $\hat{y}_l$ depends on all the neurons' outputs as they all appear in the denominator, thereby making the derivative non-zero for all the output neurons
2. $\frac{\partial L(\theta)}{\partial a_{Li}} = \frac{\partial(-\log \hat{y}_l)}{\partial a_{Li}} = \frac{\partial(-\log \hat{y}_l)}{\partial \hat{y}_l} \frac{\partial \hat{y}_l}{\partial a_{Li}}$
3. From the previous points, we know that $\hat{y}_l$ depends on a_{Li}
4. The first part of the derivative is fairly straightforward (of the form $\frac{\partial \log x}{\partial x}$)
5. $\frac{\partial(-\log \hat{y}_l)}{\partial \hat{y}_l} = \frac{-1}{\hat{y}_l}$

PadhAI: Backpropagation - the full version

One Fourth Labs

Computing derivatives w.r.t Output Layer
Part 2

1. Continuing from where we left off $\frac{\partial L(\theta)}{\partial a_{Li}} = \frac{-1}{\hat{y}_l} \frac{\partial \hat{y}_l}{\partial a_{Li}}$
2. Here, we know that $\hat{y}_l = \frac{e^{a_{Ll}}}{\Sigma_i a^{Li}}$ (taking the l-th entry of the softmax function applied to vector a_L)
3. $\frac{\partial L(\theta)}{\partial a_{Li}} = \frac{-1}{\hat{y}_l} \frac{\partial\, softmax(a_L)_l}{\partial a_{Li}}$ where $a_L = [a_{L1}, a_{L2} \dots a_{Lk}]$
   a. Where $softmax(a_L) = [\frac{e^{a_{L1}}}{\Sigma_i e^{a_{Li}}}, \frac{e^{a_{L2}}}{\Sigma_i e^{a_{Li}}} \dots \frac{e^{a_{Lk}}}{\Sigma_i e^{a_{Li}}}]$
   b. Selecting the l-th entry would give us the value $softmax(a_L)_l = \frac{exp(a_L)_l}{\Sigma_i\, exp(a_L)i}$
4. $\frac{\partial L(\theta)}{\partial a_{Li}} = \frac{-1}{\hat{y}_l} \frac{\partial}{\partial a_{Li}} \frac{exp(a_L)_l}{\Sigma_{i'}\, exp(a_L)_{i'}}$
   a. This is of the form $\frac{g(x)}{h(x)}$ $which\ gives\ derivatives$ $\frac{\partial \frac{g(x)}{h(x)}}{\partial x} = \frac{\partial g(x)}{\partial x}\frac{1}{h(x)} - \frac{g(x)}{h(x)^2}\frac{\partial h(x)}{\partial x}$
   b. Here $g(x) = exp(a_L)_l\ and\ h(x) = \Sigma_{i'}\, exp(a_L)_{i'}$
   c. Substitute the values and expand the formula
5. $\frac{\partial L(\theta)}{\partial a_{Li}} = \frac{-1}{\hat{y}_l}\left(\frac{\frac{\partial}{\partial a_{Li}} exp(a_L)_l}{\Sigma_{i'}\, exp(a_L)_{i'}} - \frac{exp(a_L)_l\ (\frac{\partial}{\partial a_{Li}} \Sigma_{i'}\, exp(a_L)_{i'})}{(\Sigma_{i'}\, exp(a_L)_{i'})^2}\right)$
   a. Here, consider $\frac{\partial}{\partial a_{Li}} exp(a_L)_l$, this value is 0 for all values of i : 0 to k except for when i = *l*
   b. Thus, we use an indicator variable $1_{(l=i)}\ \exp(a_L)_l$ to denote that all other values except i=*l* resolve to 0
   c. Now consider $\frac{\partial}{\partial a_{Li}} \Sigma_{i'}\, exp(a_L)_{i'}$, here i' ranges from 1 to k. When taking the derivative, only the index i=i' remains, which is simply a derivative of an exponent.
6. $\frac{\partial L(\theta)}{\partial a_{Li}} = \frac{-1}{\hat{y}_l}\left(\frac{1_{(l=i)}\ \exp(a_L)_l}{\Sigma_{i'}\, exp(a_L)_{i'}} - \frac{exp(a_L)_l}{\Sigma_{i'}\, exp(a_L)_{i'}} \frac{exp(a_L)_i}{\Sigma_{i'}\, exp(a_L)_{i'}}\right)$
   a. This is can be rewritten in terms of the softmax function for the different variables
7. $\frac{\partial L(\theta)}{\partial a_{Li}} = \frac{-1}{\hat{y}_l}(1_{(l=i)} softmax(a_L)_l - softmax(a_L)_l softmax(a_L)_i$
   a. We know that the Softmax function is ŷ, so we rewrite it.
8. $\frac{\partial L(\theta)}{\partial a_{Li}} = \frac{-1}{\hat{y}_l}(1_{(l=i)} \hat{y}_l - \hat{y}_l \hat{y}_i)$
9. After cancellation $\frac{\partial L(\theta)}{\partial a_{Li}} = -(1_{(l=i)} - \hat{y}_i)$

PadhAI: Backpropagation - the full version

One Fourth Labs

Computing derivatives w.r.t Output Layer
Part 3

1. So far, we have derived the partial derivative with respect to the *i*-th element of layer a_L
   a. $\frac{\partial L(\theta)}{\partial a_{Li}} = -(1_{(l=i)} - \hat{y}_i)$
2. We can now write the gradient w.r.t the vector a_L
3. As we saw earlier, $a_L = [a_{L1}, a_{L2} \dots a_{Lk}]$
4. Going by the indicator variable in step 1, it resolves to 0 for all values of *i* except for *i* = *l*
5. Let us assume a scenario where k = 4, and *l* = 2
   a. $\frac{\partial L(\theta)}{\partial a_{L1}} = -(0 - \hat{y}_i)$
   b. $\frac{\partial L(\theta)}{\partial a_{L2}} = -(1 - \hat{y}_i)$
   c. $\frac{\partial L(\theta)}{\partial a_{L3}} = -(0 - \hat{y}_i)$
   d. $\frac{\partial L(\theta)}{\partial a_{L4}} = -(0 - \hat{y}_i)$
   e. Here, the indicator variable values as a vector would be
6. The gradient w.r.t a_L is $\nabla_{a_L} =$

$$\nabla_{a_L} = \begin{bmatrix} \frac{\partial L(\theta)}{\partial a_{L1}} \\ . \\ . \\ . \\ \frac{\partial L(\theta)}{\partial a_{Lk}} \end{bmatrix}$$

$$\nabla_{a_L} = \begin{bmatrix} -(1_{(l=1)} - \hat{y}_i) \\ \\ \\ . \\ . \\ . \\ -(1_{(l=k)} - \hat{y}_i) \end{bmatrix}$$

7. The above can be seen as a difference of two vectors, [0, 1, 0, … 0_k] and $\hat{y}$
8. The first vector is essentially the one hot representation of the true output e(*l*): $-(e(l) - \hat{y}_i)$
9. In reality, this is simply the difference between the true distribution y and the predicted distribution $\hat{y}$
10. $\nabla_{a_L} L(\theta) = -(y - \hat{y}_i)$

PadhAI: Backpropagation - the full version

One Fourth Labs

Quick recap of the story so far

What have we covered up till this point?

1. Our roadmap for this module
   a. To calculate the desired gradient, we need to compute
   b. Gradient w.r.t output units
   c. Gradient w.r.t hidden units
   d. Gradient w.r.t weights and biases
   e.

$$\frac{\partial L(\theta)}{\partial W_{111}} = \frac{\partial L(\theta)}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a_3} \frac{\partial a_3}{\partial h_2} \frac{\partial h_2}{\partial a_2} \frac{\partial a_2}{\partial h_1} \frac{\partial h_1}{\partial a_1} \frac{\partial a_3}{\partial W_{111}}$$

Talk to the weight directly	Talk to the output layer	Talk to the previous hidden layer	Talk to the previous hidden layer	Talk to the weights
		works for any number of output layers		

   f. For the rest of this exercise, our focus is on *Cross Entropy loss* and *Softmax* output.
2. Here, what the sections highlighted in green are what we have covered so far, i.e. the derivative with respect to the last layer a_L
3. The gradient was calculated to be $\nabla_{a_L} L(\theta) = \frac{\partial L(\theta)}{\partial a_{Li}} = -(y - \hat{y}_i)$

PadhAI: Backpropagation - the full version

One Fourth Labs

Computing derivatives w.r.t Hidden Layers

Part 1

The derivatives corresponding to the hidden layers

1. What we are interested in is
   a. $\frac{\partial L(\theta)}{\partial h_{ij}} = \sum_{m=1}^{k} \frac{\partial L(\theta)}{\partial a_{i+1,m}} \frac{\partial a_{i+1,m}}{\partial h_{ij}}$
   b. This formula is the summation of all the paths that lead from the concerned neuron to the loss function
   c. Here, *i* = layer number, *m* = neuron number for a, *j* = neuron number for h
   d. From the previous section, we already know how to compute $\frac{\partial L(\theta)}{\partial a_{i+1,m}}$ so we need to only focus on $\frac{\partial a_{i+1,m}}{\partial h_{ij}}$
   e. However, when we compute the derivative of the neuron $a_{i+1,m}$ w.r.t $h_{i,j}$ we are left with the weight component $W_{i+1,m,j}$
   f. This refers to the weight component between the output neuron($a_{i+1,m}$) and input neuron ($h_{i,j}$)
2. Thus we have $\frac{\partial L(\theta)}{\partial h_{ij}} = \sum_{m=1}^{k} \frac{\partial L(\theta)}{\partial a_{i+1,m}} W_{i+1,m,j}$
3. Now consider these two vectors
   a.

$$\nabla_{a_{i+1}} L(\theta) = \begin{bmatrix} \frac{\partial L(\theta)}{\partial a_{i+1,1}} \\ . \\ . \\ . \\ \frac{\partial L(\theta)}{\partial a_{i+1,k}} \end{bmatrix}$$

$$W_{i+1,.,j} = \begin{bmatrix} W_{i+1,1,j} \\ \\ . \\ . \\ . \\ W_{i+1,k,j} \end{bmatrix}$$

   b. Here, $\nabla_{a_{i+1}} L(\theta)$ refers to the gradient vector of the loss function w.r.t to all output neurons from $a_{i+1,1}$ to $a_{i+1,k}$
   c. And $W_{i+1,.,j}$ refers to all rows of the j-th column of the W_{i+1} matrix, ie a vector.
4. The dot product of these two vectors is $(W_{i+1,.,j})^T \nabla_{a_{i+1}} L(\theta) = \sum_{m=1}^{k} \frac{\partial L(\theta)}{\partial a_{i+1,m}} W_{i+1,m,j}$

5. Here, the RHS is the same as the value from step 2. Therefore, the derivative of the loss function with respect to the hidden layers is the dot-product between the gradient of loss w.r.t output layer and the corresponding weights.

PadhAI: Backpropagation - the full version

One Fourth Labs

Computing derivatives w.r.t Hidden Layers
Part 2

1. We have $\frac{\partial L(\theta)}{\partial h_{ij}} = (W_{i+1,\cdot,j})^T \nabla_{a_{i+1}} L(\theta)$
   a. This is with respect to one neuron
   b. We would like to speed up this computation by solving all the derivatives in one go
2. We can now write the gradient w.r.t h_i
   a.

$$\nabla_{h_i} L(\theta) = \begin{bmatrix} \frac{\partial L(\theta)}{\partial a_{h_{i1}}} \\ \vdots \\ \frac{\partial L(\theta)}{\partial h_{in}} \end{bmatrix}$$

$$\nabla_{h_i} L(\theta) = \begin{bmatrix} (W_{i+1,\cdot,1})^T \nabla_{a_{i+1}} L(\theta) \\ \vdots \\ (W_{i+1,\cdot,n})^T \nabla_{a_{i+1}} L(\theta) \end{bmatrix}$$

   b. Can be written more compactly as $(W_{i+1})^T \nabla_{a_{i+1}} L(\theta)$
3. Thus, the formula for gradient of loss function for the last hidden layer before the output layer is given by $\nabla_{h_i} L(\theta) = (W_{i+1})^T \nabla_{a_{i+1}} L(\theta)$
4. This calculates the gradient w.r.t all neurons of layer *i*. It uses simple matrix-vector multiplication to achieve this.
5. Now, we have seen a special case applied to the last hidden layer. We must figure out how to make this formula applicable for any generic hidden layer.

PadhAI: Backpropagation - the full version

One Fourth Labs

Computing derivatives w.r.t Hidden Layers
Part 3

1. Consider the next layer a_i
   a. $\frac{\partial L(\theta)}{\partial a_{ij}} = \frac{\partial L(\theta)}{\partial h_{ij}} \frac{\partial h_{ij}}{\partial a_{ij}}$
   b. The first derivative is what we computed in part 2
   c. We need to compute the second derivative $\frac{\partial h_{ij}}{\partial a_{ij}}$
   d. We know that h_{ij} is simply the application of an activation function (sigmoid, tanh etc) to a_{ij}
   e. So it can be rewritten as $\frac{\partial h_{ij}}{\partial a_{ij}} = g'(a_{ij})$ where $h_{ij} = g(a_{ij})$ and $g'(a_{ij})$ is its derivative
2. $\frac{\partial L(\theta)}{\partial a_{ij}} = \frac{\partial L(\theta)}{\partial h_{ij}} g'(a_{ij})$
3. The full gradient can be written as
   a.

$$\nabla_{a_i} L(\theta) = \begin{bmatrix} \frac{\partial L(\theta)}{\partial h_{i1}} g'(a_{i1}) \\ \vdots \\ \frac{\partial L(\theta)}{\partial h_{in}} g'(a_{in}) \end{bmatrix}$$

   b. This vector is the element-wise product of two vectors $\nabla_{h_i} L(\theta)$ and [...,g'(a_{ik}),...] (which is a vector of derivations of the activation function w.r.t the pre-activation layer. They are both vectors of n-terms
4. Thus $\nabla_{a_i} L(\theta) = \nabla_{h_i} L(\theta) \odot [..., g'(a_{ik}), ...]$ ($\odot$ refers to element-wise multiplication)
5. This formula can be applied to any of the hidden layers