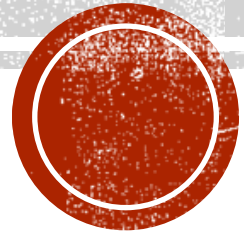
BACKPROPAGATION

Dr. Brian Mc Ginley

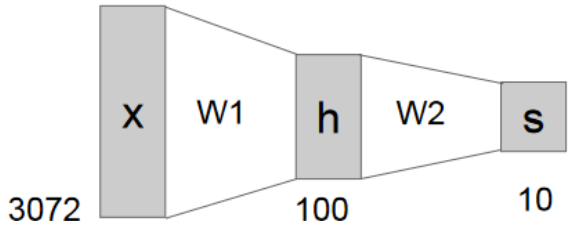


NEURAL NETWORK

- (Before) Linear function:

$$f = W.X$$

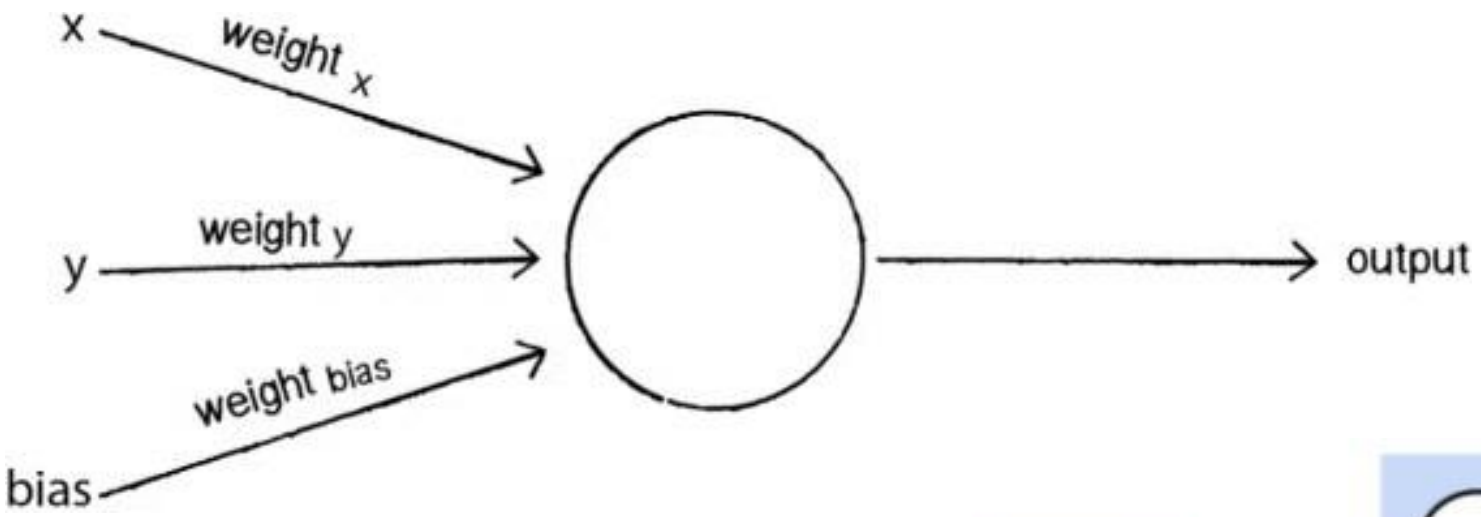
- 2-layer Neural Network (with ReLU activation function)

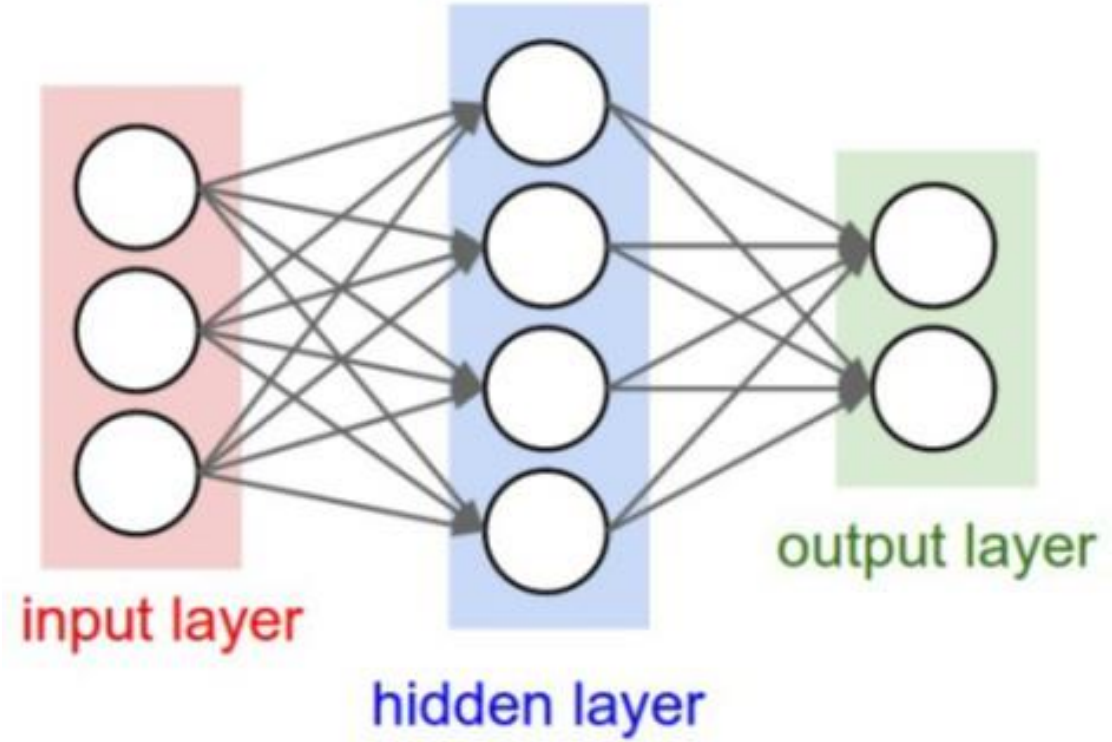


$$f = W_2.\max(0, W_1.X)$$

- 3-layer Neural Network (with ReLU activation function)

$$f = W_3.\max(W_2.\max(0, W_1X))$$


x
weight x
y
weight y
output
weight bias
bias


input layer
hidden layer
output layer

HISTORY

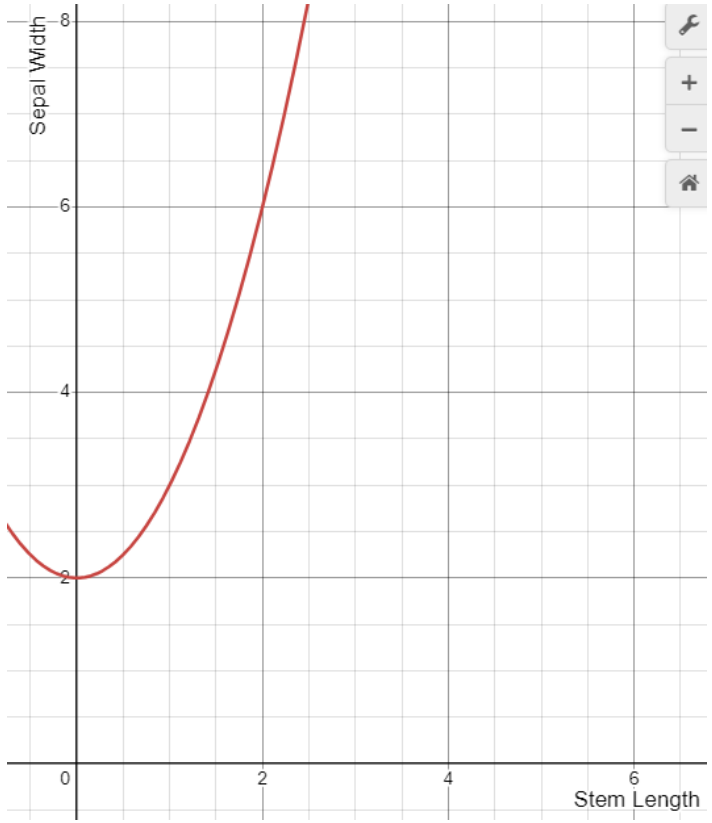
- 1970 - Backpropagation (for training multiple layers) was discovered by Finnish Masters student Seppo Linnainmaa.
  - Algorithm not discovered again until 1980s by Rumelhart (1982) and Hinton (1986) – kickstarting new research in the field
- Based on gradient descent and application of chain rule (calculus)
- Note: It's not biologically plausible. The hardware in the brain isn't suited to performing backpropagation

USES

- Used to train neural networks including:
  - Natural Language Processing/Deep LLMs such as ChatGPT etc.
  - Speech Processing
  - Object detection computer vision networks such as YOLO from Ultralytics
  - Fine-Tuning Pre-trained Models
    - Can be used to fine-tune pre-trained models to adapt them to new tasks/datasets.
    - Transfer Learning
      - Earlier layers in models which are capturing good general features are "frozen" – which means the layer's weights are no longer updated
      - Sometimes a new output layer is added & trained to perform the new task - E.g.
        - Taking a pre-trained image classifier like ResNet of VGG and fine tuning the final layers
        - Taking a pre-trained speech recognition classifier like Wav2Vec2 and fine tuning for a new accent/language.

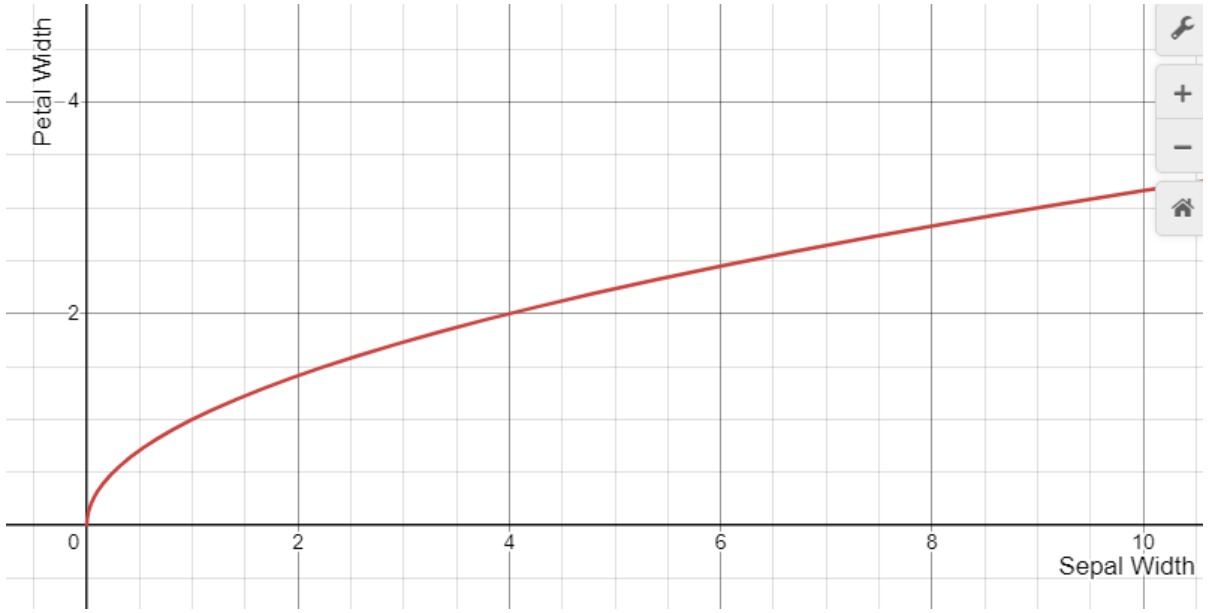
A CHAIN RULE RECAP

- Let S = Sepal Width, L = Stem Length
- For an Iris flower imagine:
  - $S = L^2 + 2$
  - So, sepal length is dependent on the stem length of the flower



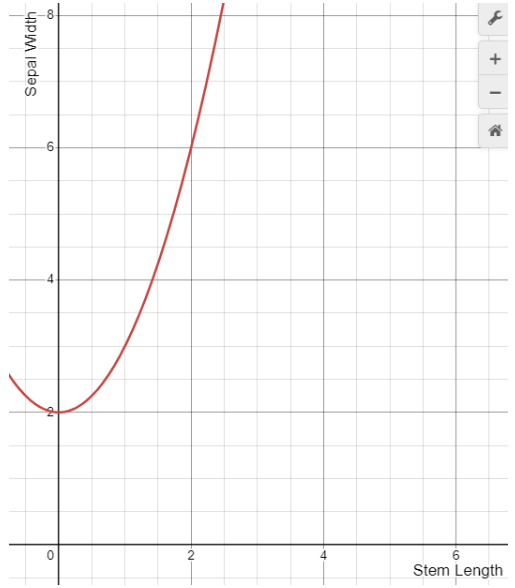
A CHAIN RULE RECAP

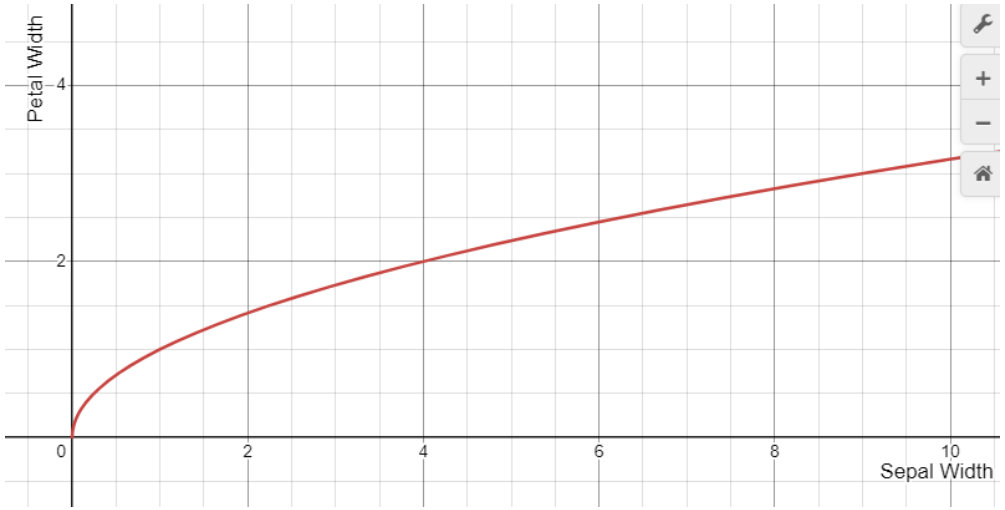
- Let S = Sepal Width, L = Stem Length
- For an Iris flower imagine:
  - $S = L^2 + 2$
  - So, sepal width is dependent on the stem length of the flower
- Next, imagine that:
  - Petal Width (P) is $P = \sqrt{S}$
  - So, petal width is dependent on the sepal width of the flower



A CHAIN RULE RECAP

- Let S = Sepal Width, L = Stem Length
- For an Iris flower imagine:
  - $S = L^2 + 2$
  - So, sepal width is dependent on the stem length of the flower
- Next, imagine that:
  - Petal Width (P) is $P = \sqrt{S}$
  - So, petal width is dependent on the sepal width of the flower
- If we wanted to see how Petal Width changes with respect to varying Stem Length, we need to find the derivative





A CHAIN RULE RECAP

- So, for us to see how Petal Width changes with respect to varying Stem Length, we need to find the derivative

$$\frac{dP}{dL} = \frac{dP}{dS} * \frac{dS}{dL}$$

- Recall: $P = \sqrt{S} = S^{1/2}$ so $\frac{dP}{dS} = \frac{1}{2}S^{-1/2} = \frac{1}{2\sqrt{S}}$
- Recall: $S = L^2 + 2$ so $\frac{dS}{dL} = 2L$
- So, $\frac{dP}{dL} = \frac{dP}{dS} * \frac{dS}{dL} = \frac{1}{2\sqrt{S}} * 2L = \frac{2L}{2\sqrt{L^2+2}} = \frac{L}{\sqrt{L^2+2}}$

A CHAIN RULE RECAP

- Now, mostly in the real world, you don't get 2 separate equations like $P = \sqrt{S}$ and $S = L^2 + 2$, you just get one like below and you have to find the derivative:

$$P = \sqrt{L^2 + 2}$$

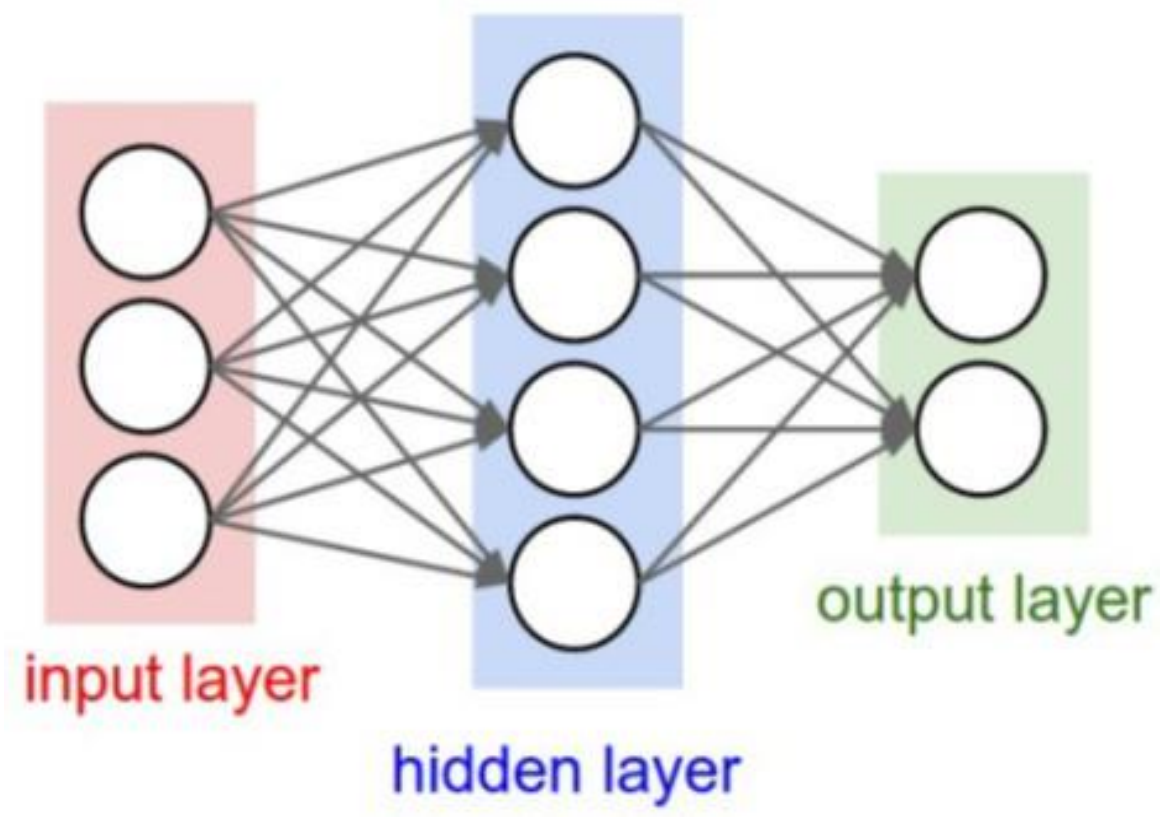
- Here, you use brackets to re-write the equation as $P = \sqrt{(I)}\ \ where\ I = L^2 + 2$
- And you perform the following equation

$$\frac{dP}{dL} = \frac{dP}{dI} * \frac{dI}{dL} = \frac{1}{2\sqrt{I}} * 2L = \frac{2L}{2\sqrt{L^2 + 2}} = \frac{L}{\sqrt{L^2 + 2}}$$

HOW DOES THIS APPLY TO NNS

- When we looked at Gradient Descent before with Linear Regression, we had
- $L = \frac{1}{2m}\sum_{i=1}^{m}(y^i - w_1 x^i)^2$ and $\frac{dL}{dw} = \frac{1}{m}\sum_{i=1}^{m}(-x^i)(y^i - w_1 x^i)$

- Now we have a much more complex network
  - Lots more weights and inter-dependencies



BACKPROPAGATION – THE STEPS

1. Initialise the network (some random values to begin)
2. Forward Pass:
   - Input training data
   - Compute the output of each neuron layer by layer
   - Continue until the final layer produces the predicted output.
3. Calculate the loss – with metric of choice
4. Backward Pass (Gradient Calculation)
   - Compute the gradient of the loss function with respect to the weights and biases for the final layer
   - Propagate the error back through the layers (using chain rule)
5. Update Weights and Biases
6. Repeat

STEP 1: INITIALISING THE NN

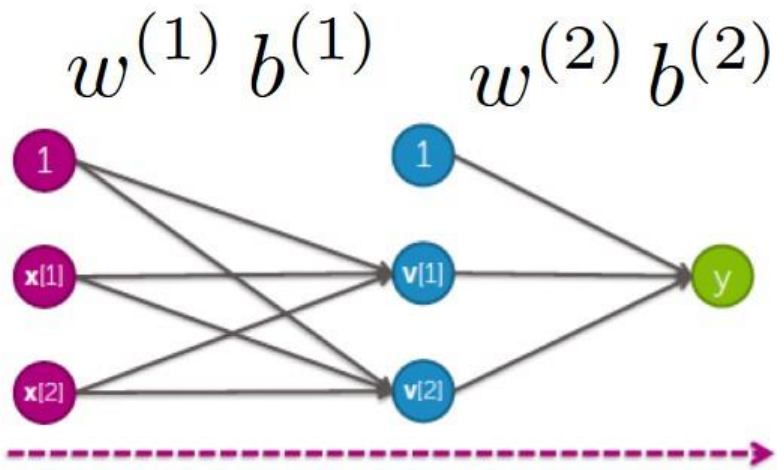
- Set all weights randomly from a Normal Distribution
- Set all biases to zero
- Then work out the outputs and the error

STEP 2: FORWARD PASS

For fixed weights, forming predictions is easy!

Compute values left to right

1. Inputs: **x**[1],...,**x**[d]
2. Hidden: **v**[1],...,**v**[d]
3. Output: y




```
v = activation(np.dot(w1,x) + b1)
yhat = np.dot(w2,v) + b2
```

STEP 3: CALCULATE THE LOSS

- Calculate the loss using MSE or more commonly with Neural Networks
- Often with NNs, we use Cross Entropy Loss

$$CrossEntropyLoss = -\sum_{c=1}^{M} ObservedProb_c * \log(PredictedProb_c)$$

- Where M is the number of output classes

STEP 4: BACKWARD PASS (GRADIENT DESCENT)

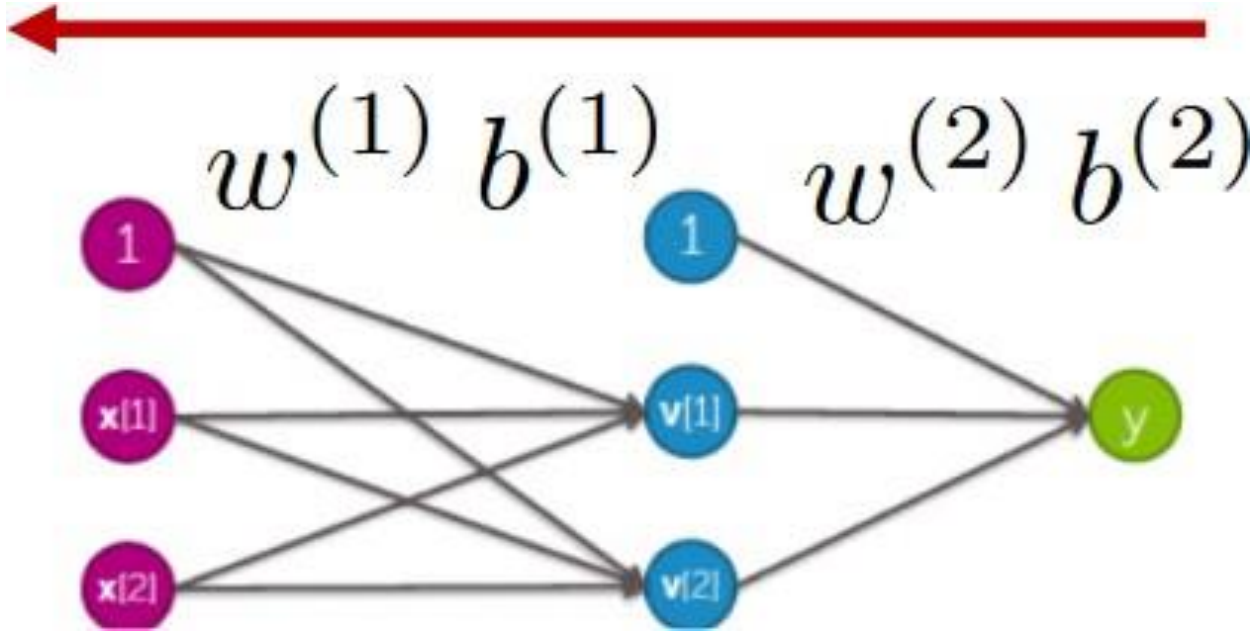
- We've seen in the various machine learning methods that we can use Gradient descent to descend the loss function.
- To do this we needed to determine the derivative of the loss function with respect to each weight, and make small moves (α) in a downward direction.
- This was achievable with relatively simple models like logistic regression or SVM but what do we do with complex multi layer perceptron models like neural networks?
- There are many complex designs, sometimes incorporating millions of weights, must we determine the derivative of the loss function to each?
- Well, the answer to this is Yes and No.
- No, we don't have to do it on paper but Yes, we do have to do it.

STEP 4: BACKWARD PASS (GRADIENT DESCENT)

- Like the methods we used for the less complex machine learning method, we must come up with an analytic gradient of the loss function with respect to every weight in the network.
- Neural Networks will be very large
- There is no really hope of writing down the gradient formula by hand for all parameters
- This sounded impossible, but thanks to an algorithm called back propagation it can be achieved by a computer algorithm.

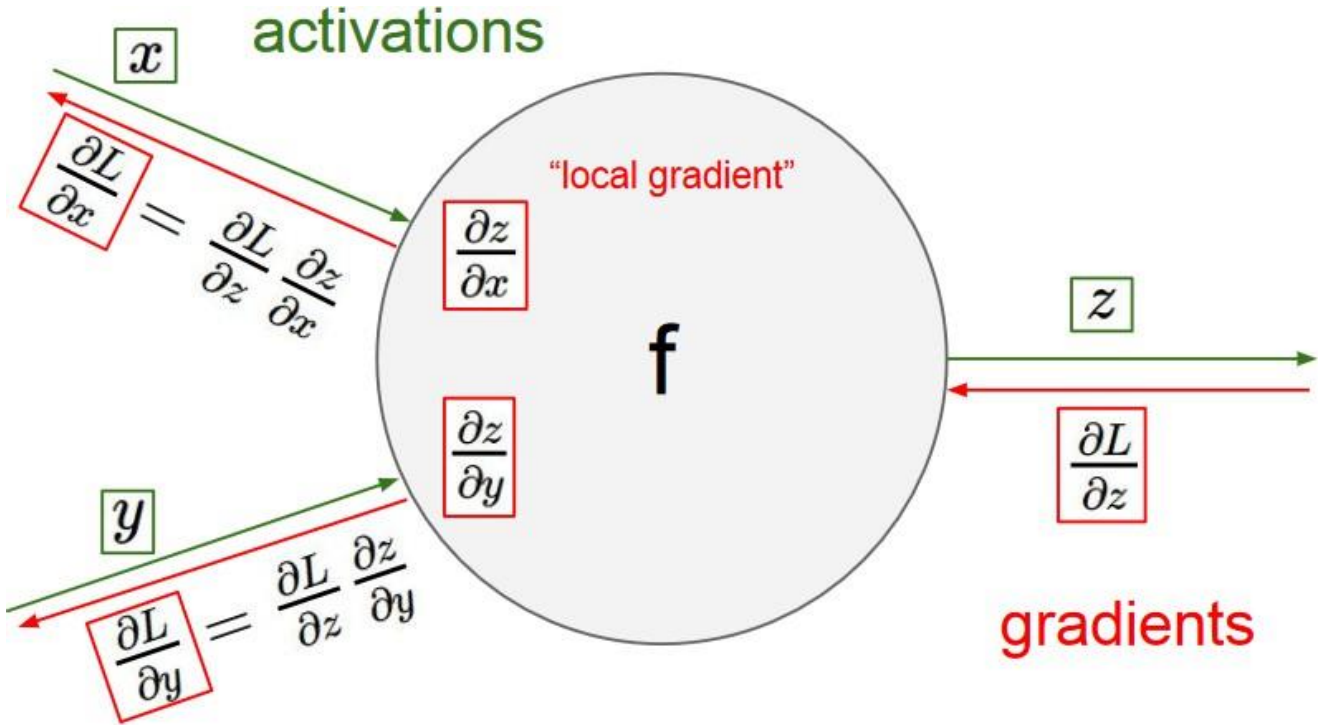
STEP 4: BACKPROPAGATION

- https://developers-dot-devsite-v2-prod.appspot.com/machine-learning/ crash-course/backprop-scroll



STEP 4: BACKPROPAGATION

- **Backpropagation** will be a recursive application of the ”chain rule” along a graph to compute the gradients of all inputs/parameters/intermediates



STEP 4: BACKPROPAGATION SUMMARY

- Neural nets will be very large: no hope of writing down gradient formula by hand for all parameters
- Backpropagation = recursive application of the chain rule along a computational graph to compute the gradients of all inputs/parameters/intermediates
- Implementations maintain a graph structure, where the nodes implement the forward()/backward() API.
- Forward: compute result of an operation and save any intermediates needed for gradient computation in memory
- Backward: apply the chain rule to compute the gradient of the loss function with respect to the inputs

STEP 5 & 6: UPDATE WEIGHTS AND BIASES AND REPEAT

- Adjust weights and biases according to the gradients

$$W = W - \alpha * \frac{dL}{dW} \qquad b = b - \alpha * \frac{dL}{db}$$

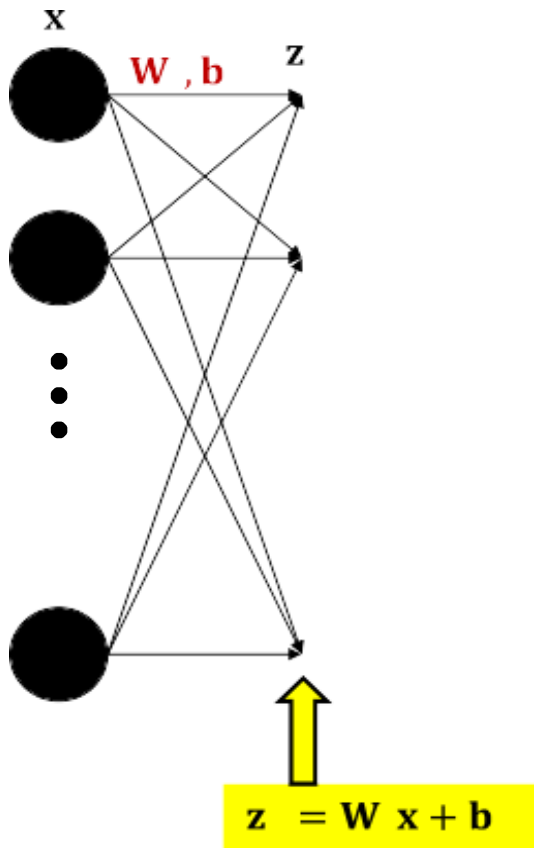
- Where α is the learning rate.

$$\frac{\partial L}{\partial w_0}, \frac{\partial L}{\partial w_1}, \frac{\partial L}{\partial w_2}, \ldots, \frac{\partial L}{\partial w_m}$$

- Repeat over multiple epochs until the loss converges or meets some performance criteria.

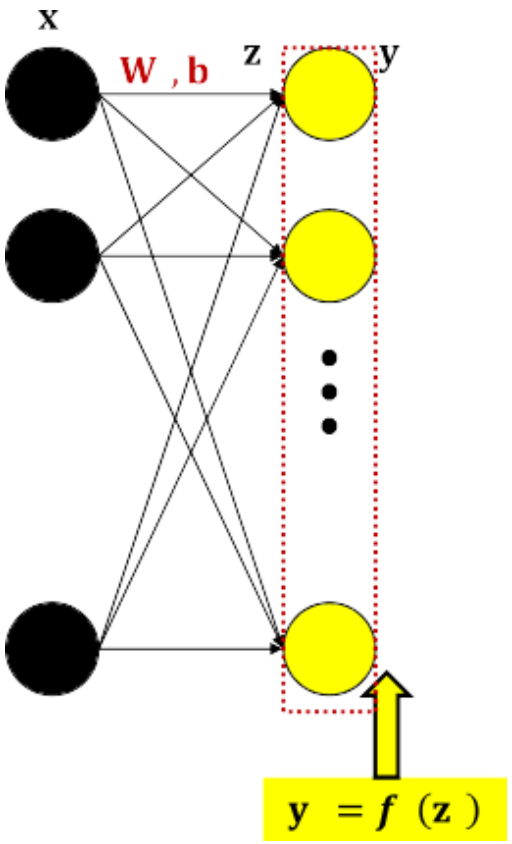
QUICK RECAP OF BACKPROP: FORWARD PASS

- Forward pass: Compute output and all intermediate variables in the network, for the input (X)



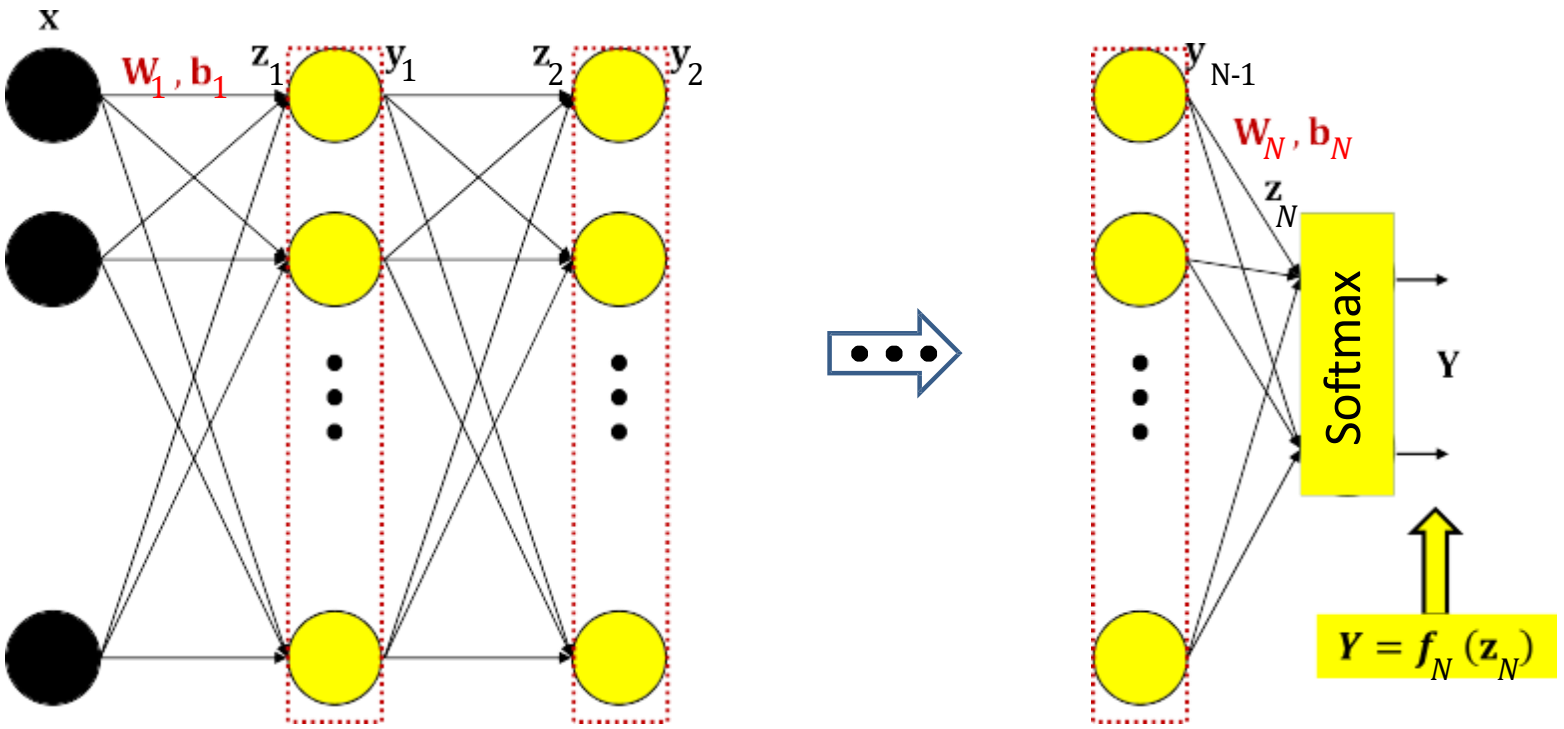
QUICK RECAP OF BACKPROP: FORWARD PASS

- Forward pass: Compute output and all intermediate variables in the network, for the input (X)



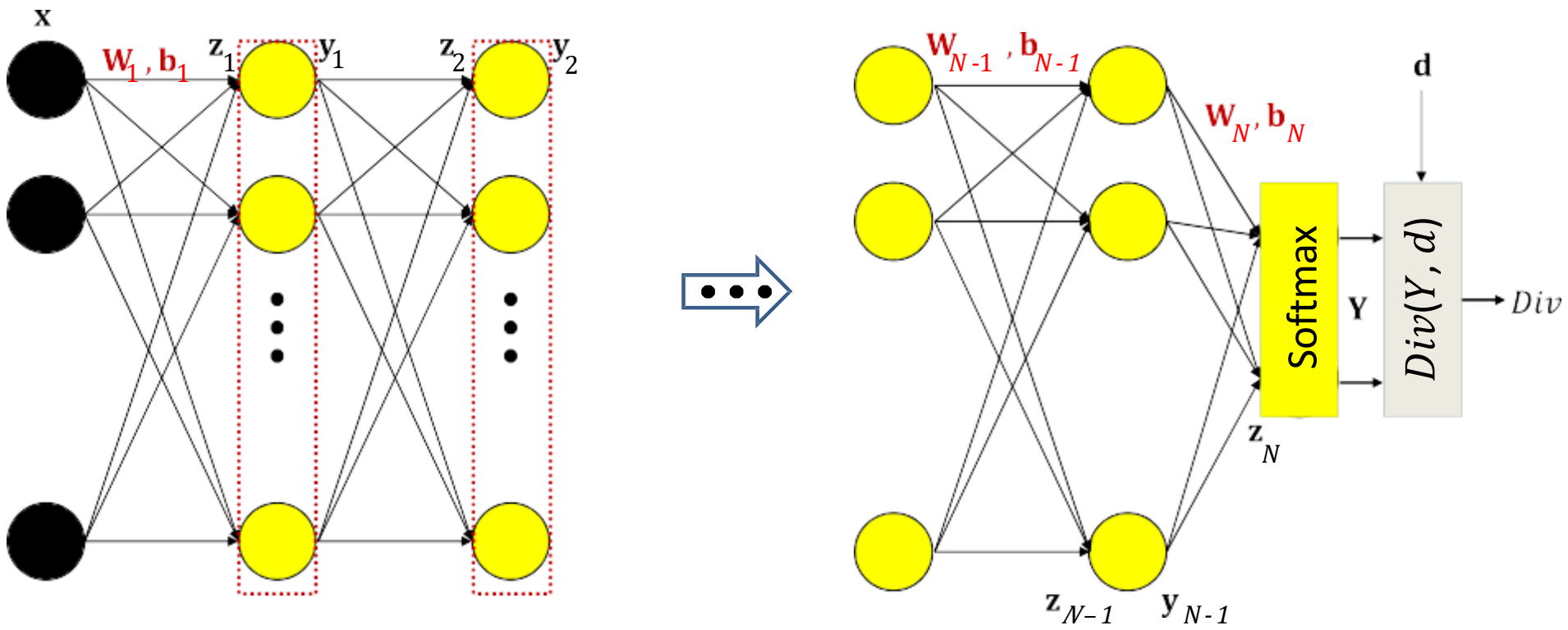
QUICK RECAP OF BACKPROP: FORWARD PASS

- Forward pass: Compute output and all intermediate variables in the network, for the input (X)



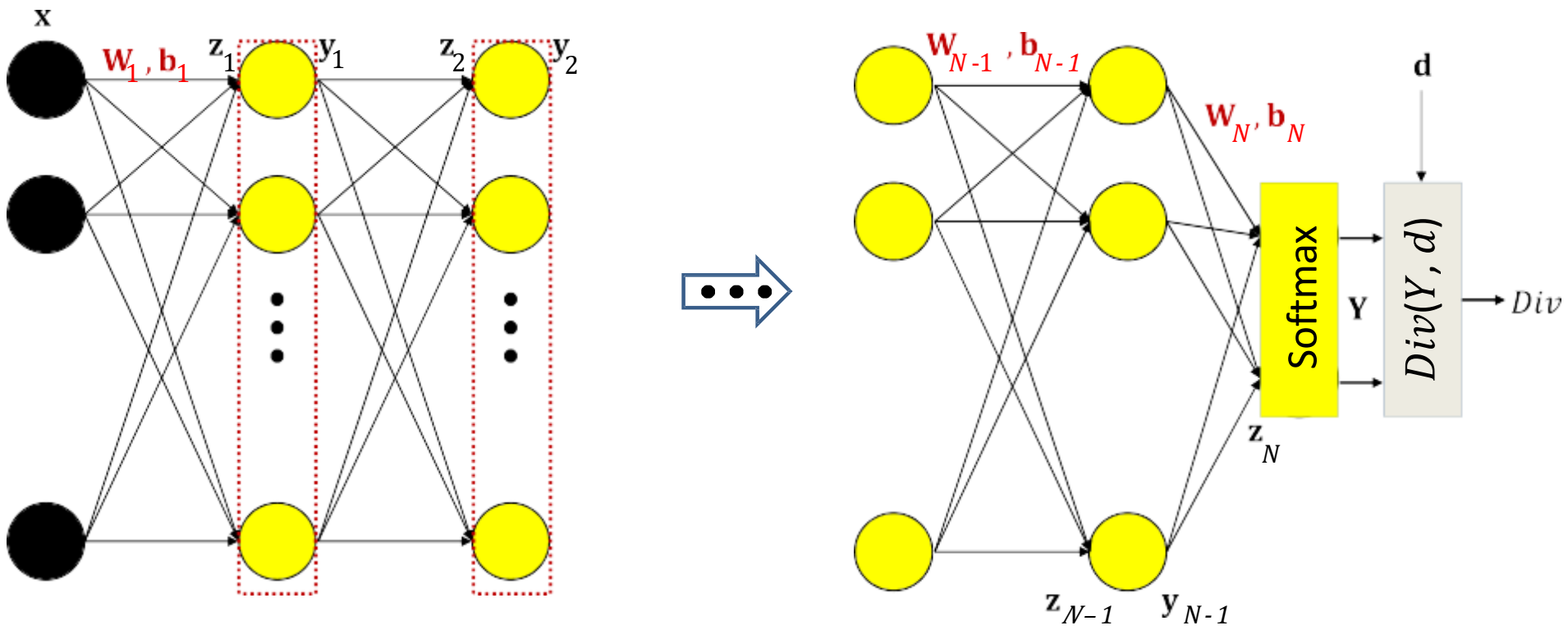
QUICK RECAP OF BACKPROP: FORWARD PASS

- Forward pass: Compute output and all intermediate variables in the network, for the input (X)



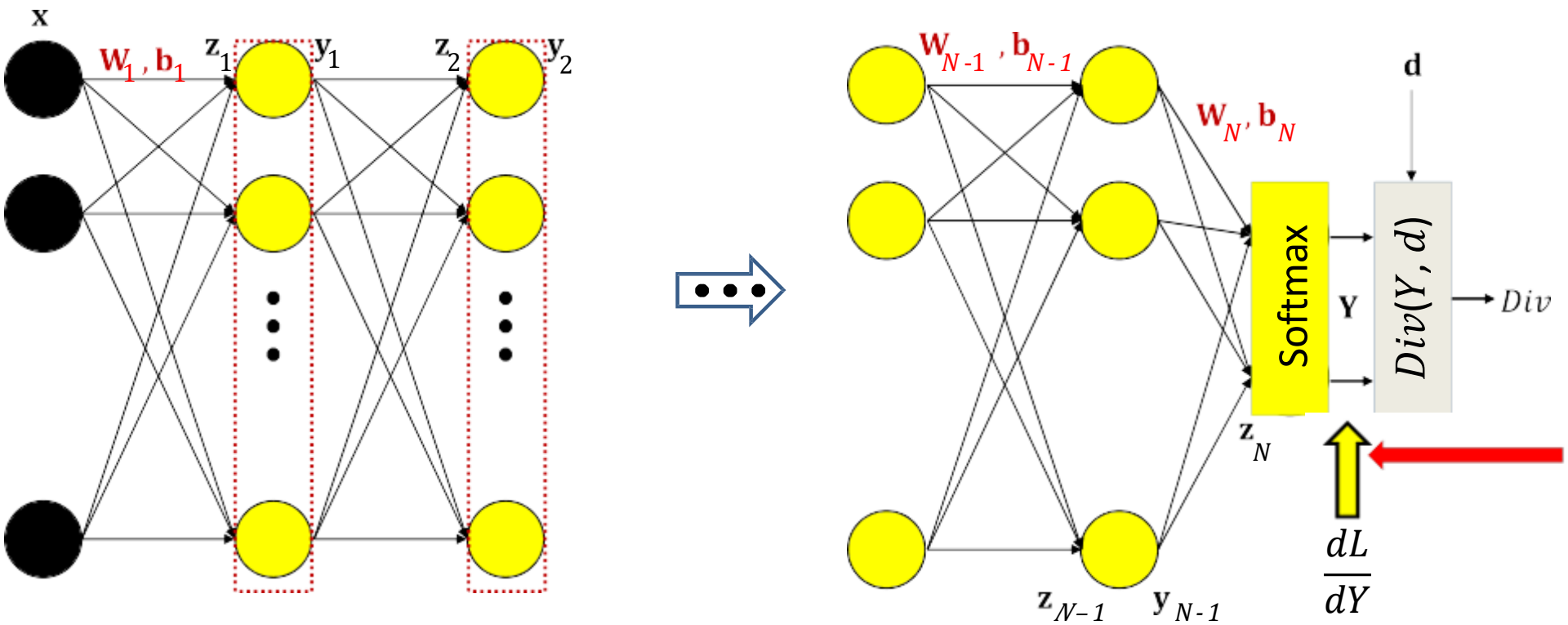
QUICK RECAP: BACKPROPAGATION

- Now work your way backward through the net to compute the derivative w.r.t each intermediate variable and each weight/bias



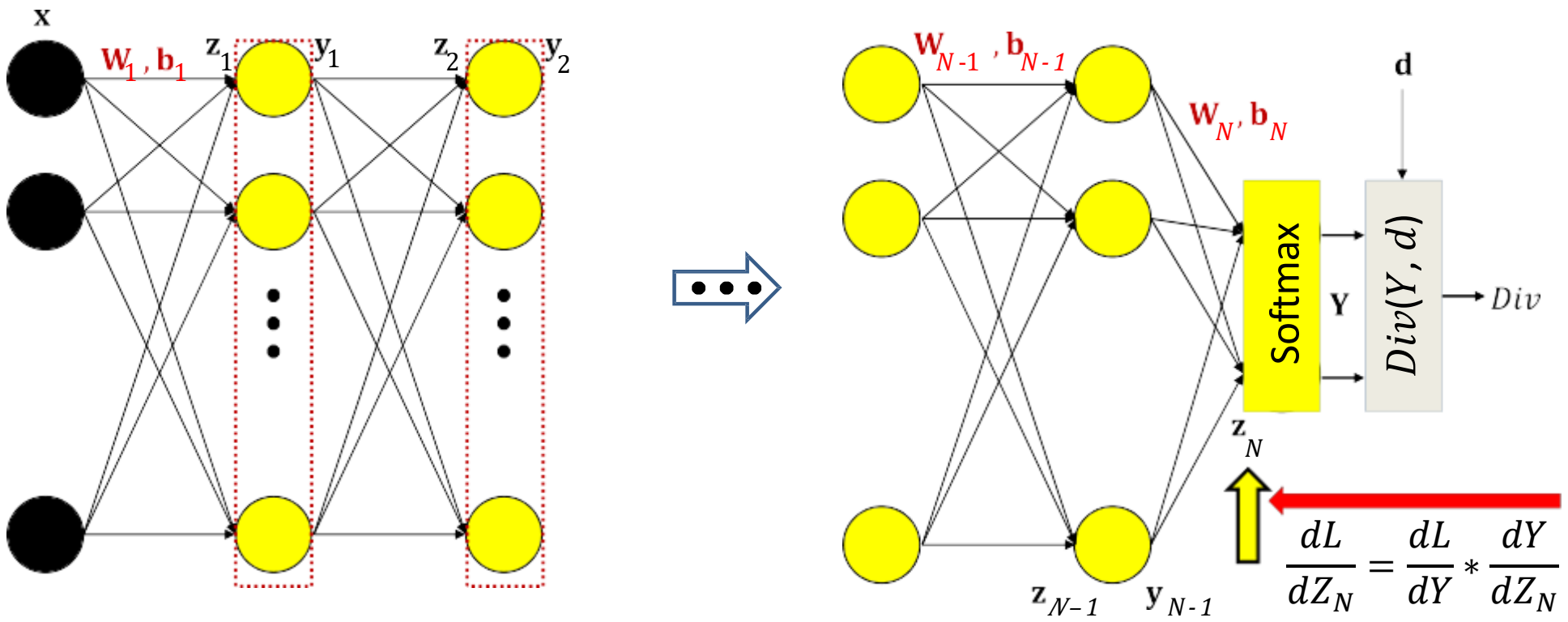
QUICK RECAP: BACKPROPAGATION

- Go backwards to the next step using the chain rule.



QUICK RECAP: BACKPROPAGATION

- Keep going backwards through the network using the chain rule until you've calculated the gradient for each parameter. Then do all your updates



QUICK RECAP: BACKPROPAGATION

- First compute the gradient of the divergence with respect to Y. The actual gradient depends on the divergence function.