Neural Networks Part III

Back Propagation Part II

Derivatives for Output Layer

Intermediate Summary:

For a given training example $\mathbf{x}_k$, perform forward substitution to get activities $\mathbf{z}^\ell$ and activations $\mathbf{a}^\ell$ on each layer.

Then, to get partial derivatives w.r.t. weights and biases connected to output layer

1. Compute $\delta^L = \nabla_{\mathbf{a}^L} C \odot g'(\mathbf{z}^L)$
2. Compute $\dfrac{\partial C}{\partial W^{L-1}} = \delta^L (\mathbf{a}^{L-1})^T$ and $\dfrac{\partial C}{\partial \mathbf{b}^{L-1}} = \delta^L$

OK, that wasn't so bad! We found simple, vectorized formulas for derivatives connected to the output layer.

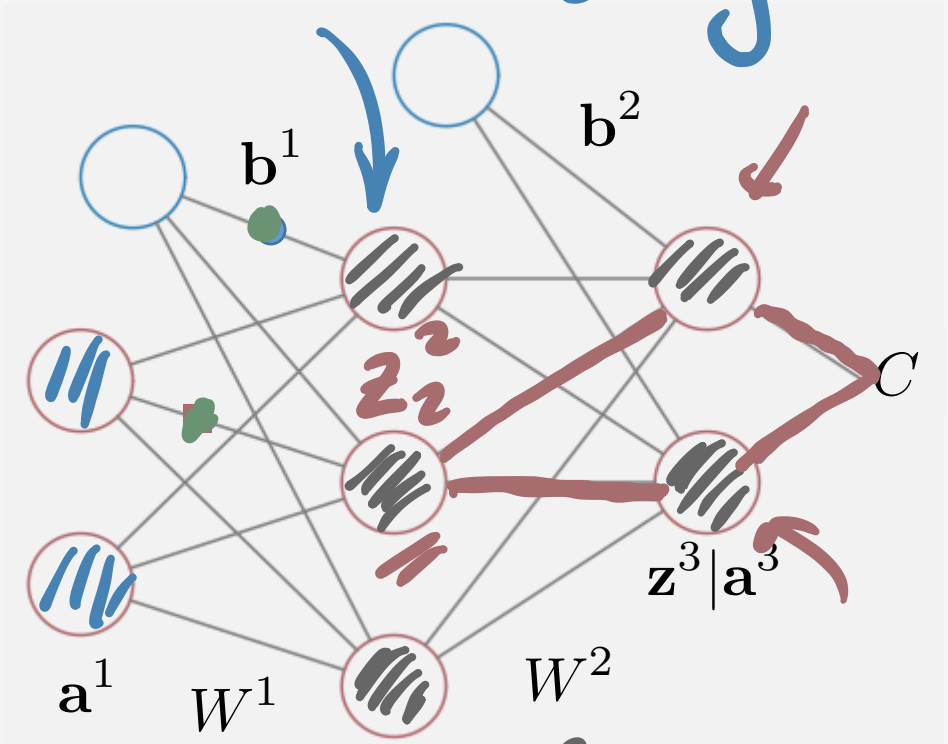
Problem: How do we get derivatives w.r.t. weights and biases earlier in network?

Back Propagation

$z^2 = W^1 a^1 + b^1$

$a^2 = g(z^2)$

SGD: $w^l = w^l - \eta \frac{\partial C}{\partial w^l}$



Formulas for W^2 and $\mathbf{b}^2$ were nice b/c we knew δ^3

Seems like formulas for W^1 and $\mathbf{b}^1$ would be nice if we knew δ^2

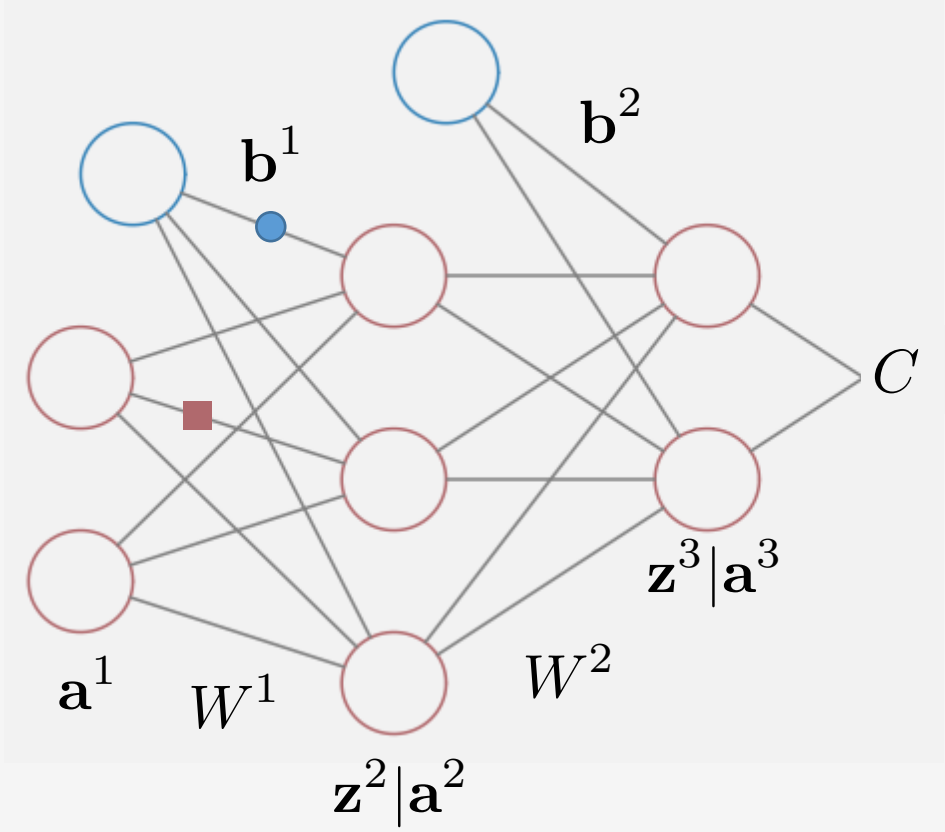
But relationship b/w C and $\mathbf{z}^2$ is super complicated

Multiple passes through activation functions :(

$\frac{\partial C}{\partial w^2} = \delta^2 a^1 g'(z^2)$

$\delta^l = \frac{\partial C}{\partial z^l}$

Back Propagation



Formulas for W^2 and $\mathbf{b}^2$ were nice b/c we knew δ^3

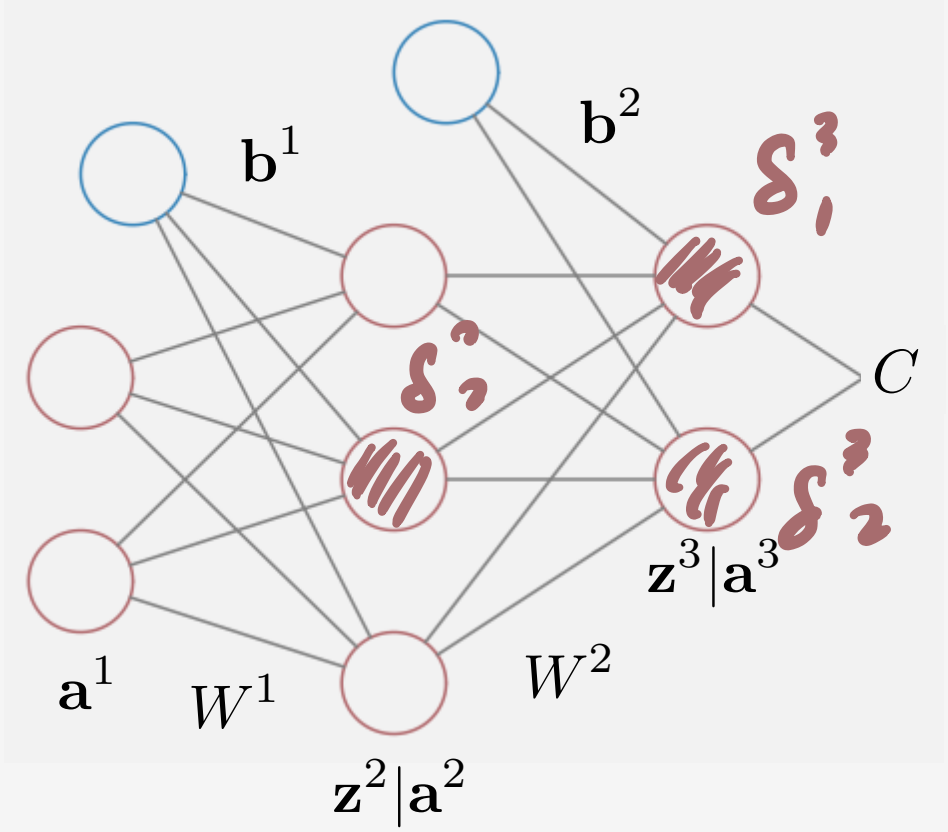
Seems like formulas for W^1 and $\mathbf{b}^1$ would be nice if we knew δ^2

But relationship b/w C and $\mathbf{z}^2$ is super complicated

Multiple passes through activation functions :(

But it's OK! There's a nice-ish process for **propagating** $\delta's$ **backwards** through layers!

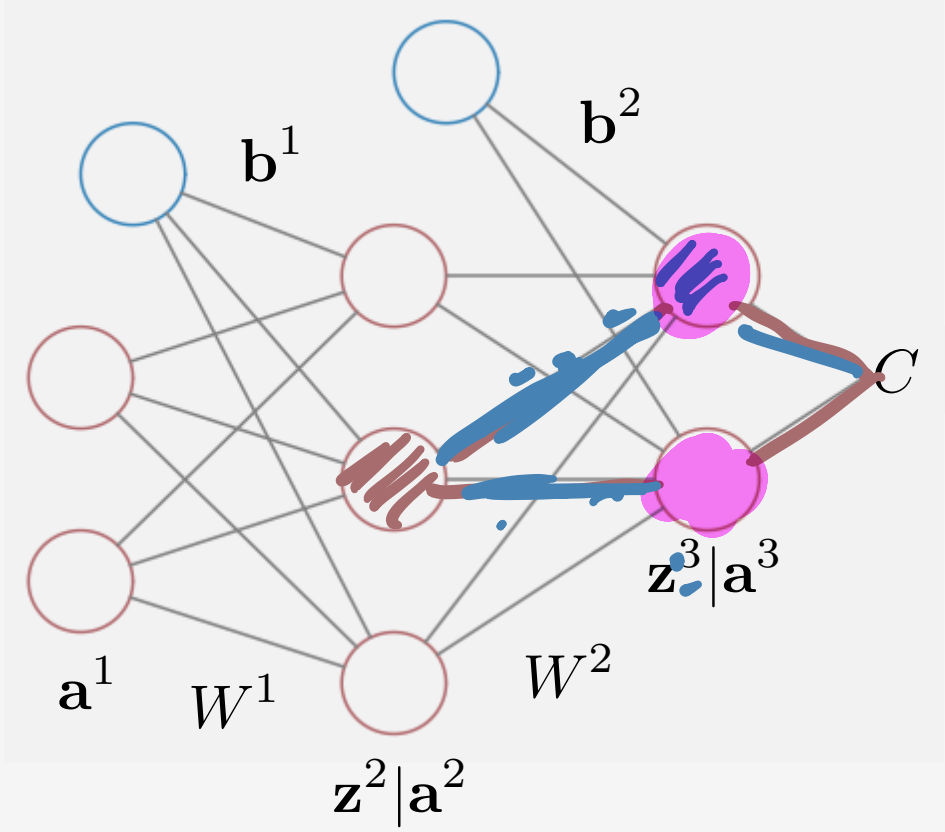
Back Propagation



Goal: Use $\delta's$ on Layer 3 to compute $\delta's$ on Layer 2

Notice that e.g. δ_2^2 depends on both δ_1^3 and δ_2^3

Back Propagation



Goal: Use $\delta's$ on Layer 3 to compute $\delta's$ on Layer 2

Notice that e.g. δ_2^2 depends on both δ_1^3 and δ_2^3

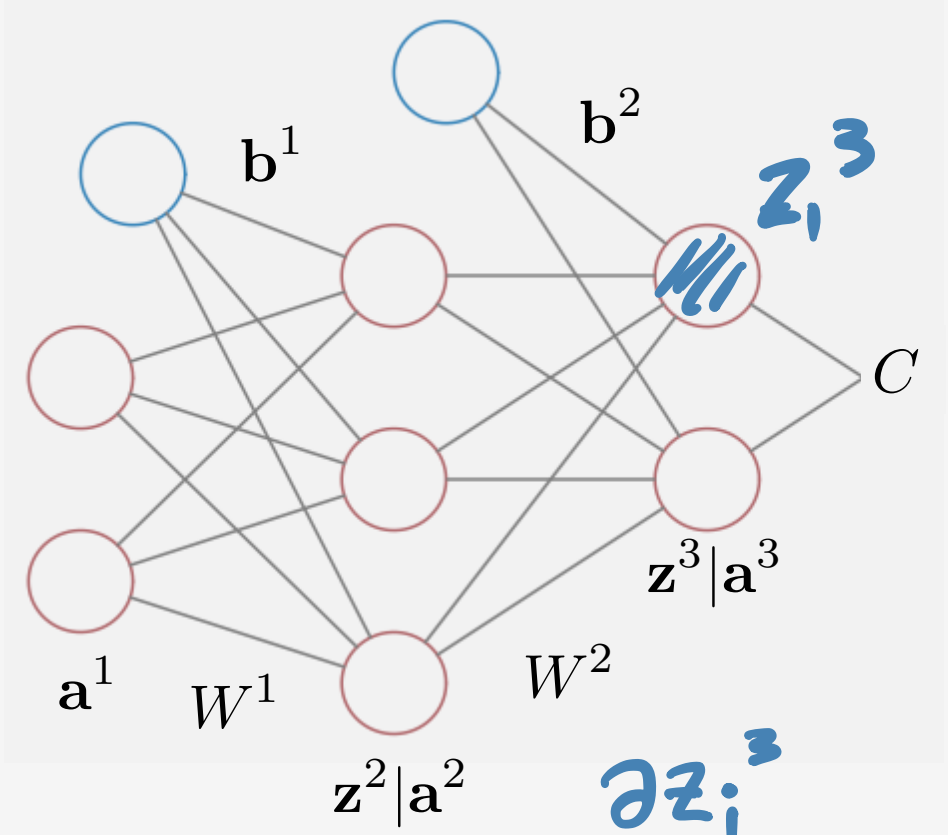
By the **Adult Chain Rule**:

$$\delta_2^2 = \frac{\partial C}{\partial z_2^2} = \frac{\partial C}{\partial z_1^3}\frac{\partial z_1^3}{\partial z_2^2} + \frac{\partial C}{\partial z_2^3}\frac{\partial z_2^3}{\partial z_2^2}$$

$$= \delta_1^3 \frac{\partial z_1^3}{\partial z_2^2} + \delta_2^3 \frac{\partial z_2^3}{\partial z_2^2}$$

To finish this, we just need to compute derivatives of Layer 3 activities w.r.t. Layer 2 activities

Back Propagation



$$\delta_2^2 = \delta_1^3 \frac{\partial z_1^3}{\partial z_2^2} + \delta_2^3 \frac{\partial z_2^3}{\partial z_2^2}$$

To finish this, we just need to compute derivatives of Layer 3 activities w.r.t. Layer 2 activities

Recall: $\mathbf{z}^3 = W^2\mathbf{a}^2 + \mathbf{b}^2$, the i^{th} entry of which is

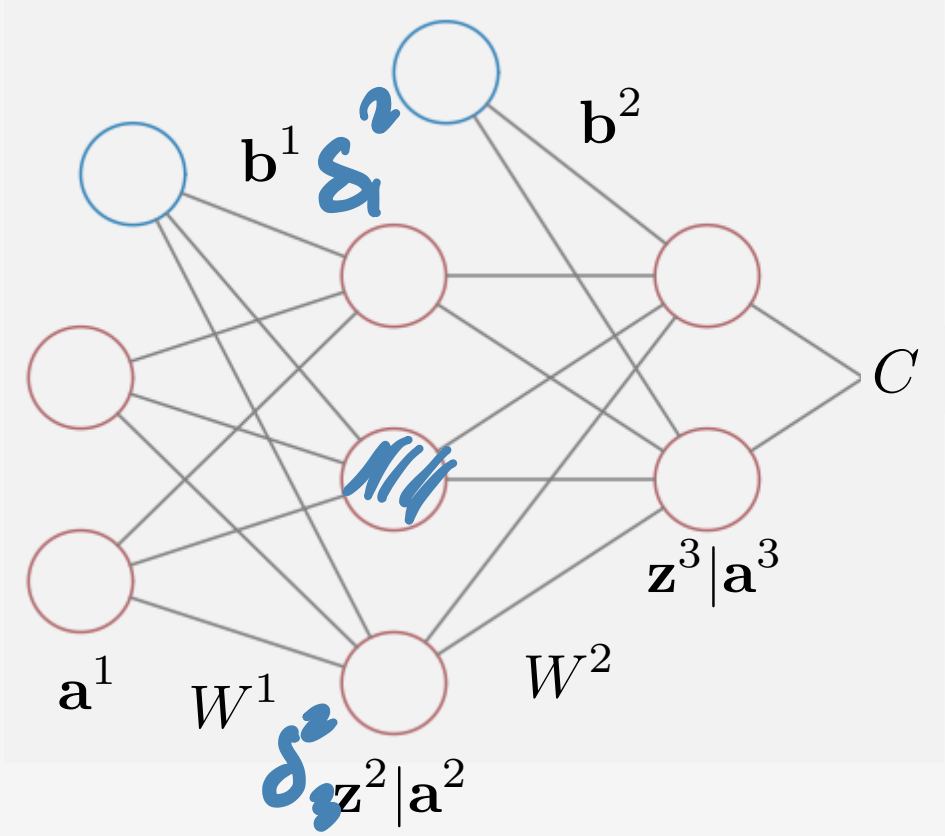
$$z_i^3 = w_{i1}^2 a_1^2 + w_{i2}^2 a_2^2 + w_{i3}^2 a_3^2 + b_i^2$$

But the activations are just activities run through g

$$z_i^3 = w_{i1}^2 g(z_1^2) + w_{i2}^2 g(z_2^2) + w_{i3}^2 g(z_3^2) + b_i^2$$

$$\frac{\partial z_i^3}{\partial z_2^2} = w_{i2} g'(z_2^2)$$

Back Propagation



$$\delta_2^2 = \delta_1^3 \frac{\partial z_1^3}{\partial z_2^2} + \delta_2^3 \frac{\partial z_2^3}{\partial z_2^2}$$

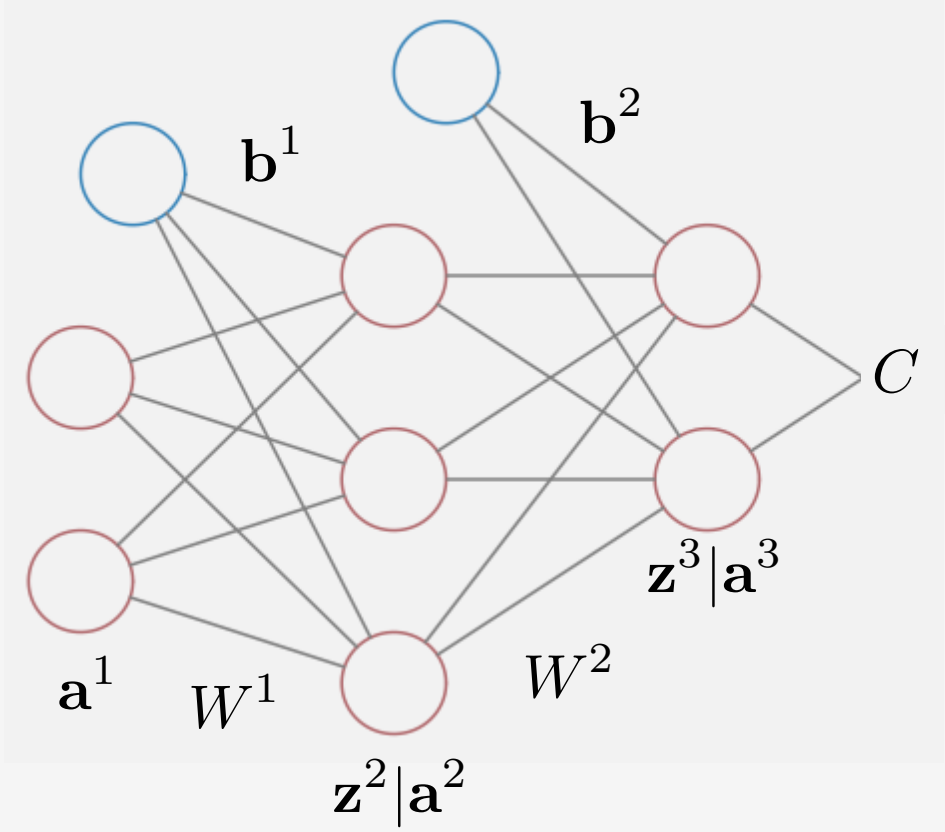
To finish this, we just need to compute derivatives of Layer 3 activities w.r.t. Layer 2 activities

$$z_i^3 = w_{i1}^2 g(z_1^2) + w_{i2}^2 g(z_2^2) + w_{i3}^2 g(z_3^2) + b_i^2$$

$$\Rightarrow \quad \frac{\partial z_1^3}{\partial z_2^2} = w_{12}^2 g'(z_2^2), \quad \frac{\partial z_2^3}{\partial z_2^2} = w_{22}^2 g'(z_2^2)$$

So finally... $\delta_2^2 = \delta_1^3 w_{12}^2 g'(z_2^2) + \delta_2^3 w_{22}^2 g'(z_2^2)$

Back Propagation



Doing this for all three δ_i^2's yields something nice...

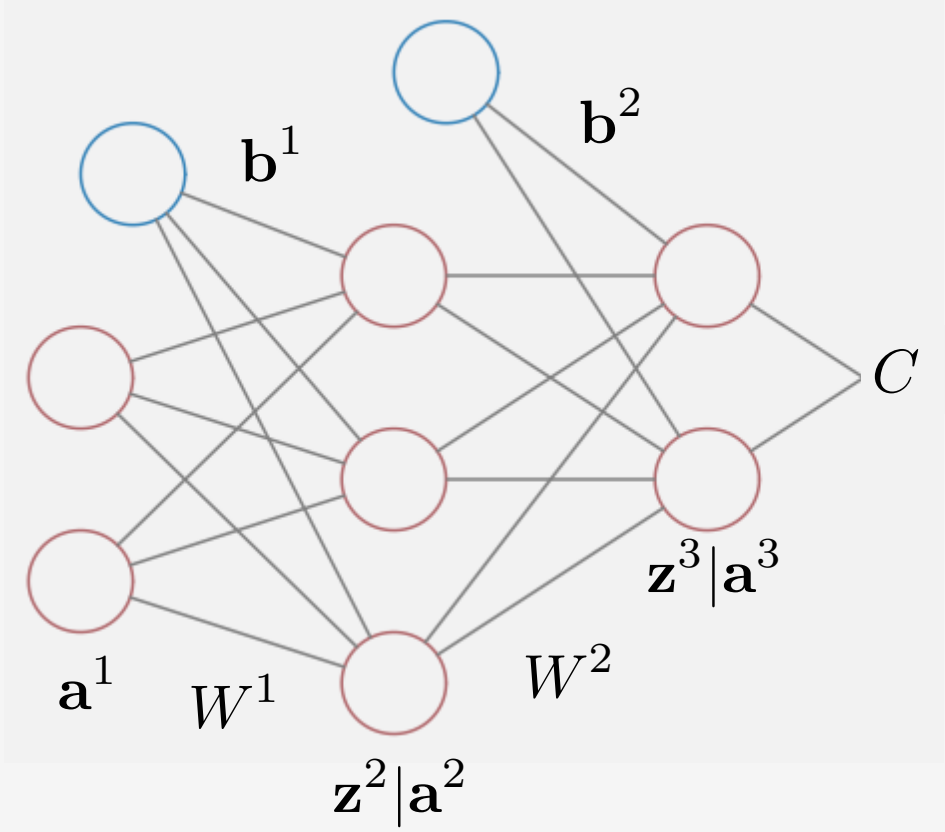
$$\delta_1^2 = \delta_1^3\, w_{11}^2\, g'(z_1^2) + \delta_2^3\, w_{21}^2\, g'(z_1^2)$$

$$\delta_2^2 = \delta_1^3\, w_{12}^2\, g'(z_2^2) + \delta_2^3\, w_{22}^2\, g'(z_2^2)$$

$$\delta_3^2 = \delta_1^3\, w_{13}^2\, g'(z_2^2) + \delta_2^3\, w_{23}^2\, g'(z_3^2)$$

Notice that each row of the system gets multiplied by $g'(z_i^2)$ so let's factor those out.

Back Propagation



Doing this for all three δ_i^2's yields something nice...

$$\delta_1^2 = (\delta_1^3\, w_{11}^2 + \delta_2^3\, w_{21}^2) \cdot g'(z_1^2)$$

$$\delta_2^2 = (\delta_1^3\, w_{12}^2 + \delta_2^3\, w_{22}^2) \cdot g'(z_2^2)$$

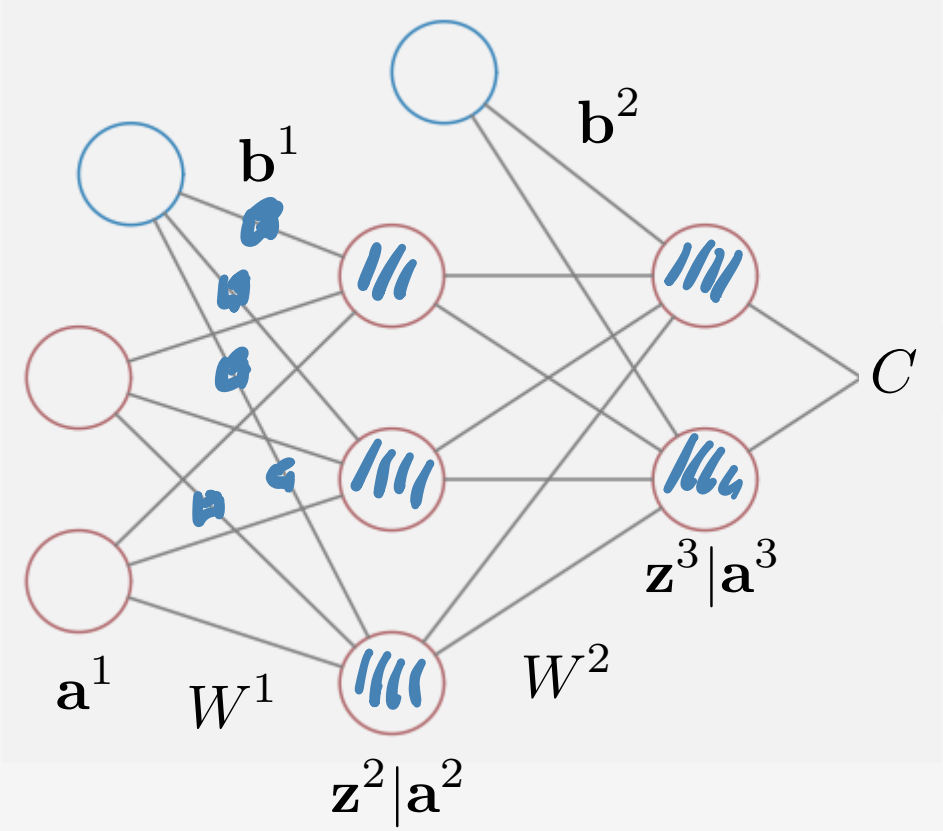
$$\delta_3^2 = (\delta_1^3\, w_{13}^2 + \delta_2^3\, w_{23}^2) \cdot g'(z_3^2)$$

Remember that W^2 and δ^3 look as follows

$$W^2 = \begin{bmatrix} w_{11}^2 & w_{12}^2 & w_{13}^2 \\ w_{21}^2 & w_{22}^2 & w_{23}^2 \end{bmatrix} \qquad \delta^3 = \begin{bmatrix} \delta_1^3 \\ \delta_2^3 \end{bmatrix}$$

Do you see these lurking above?

Back Propagation



Doing this for all three δ_i^2's yields something nice...

$$\delta_1^2 = (\delta_1^3\, w_{11}^2 + \delta_2^3\, w_{21}^2) \cdot g'(z_1^2)$$

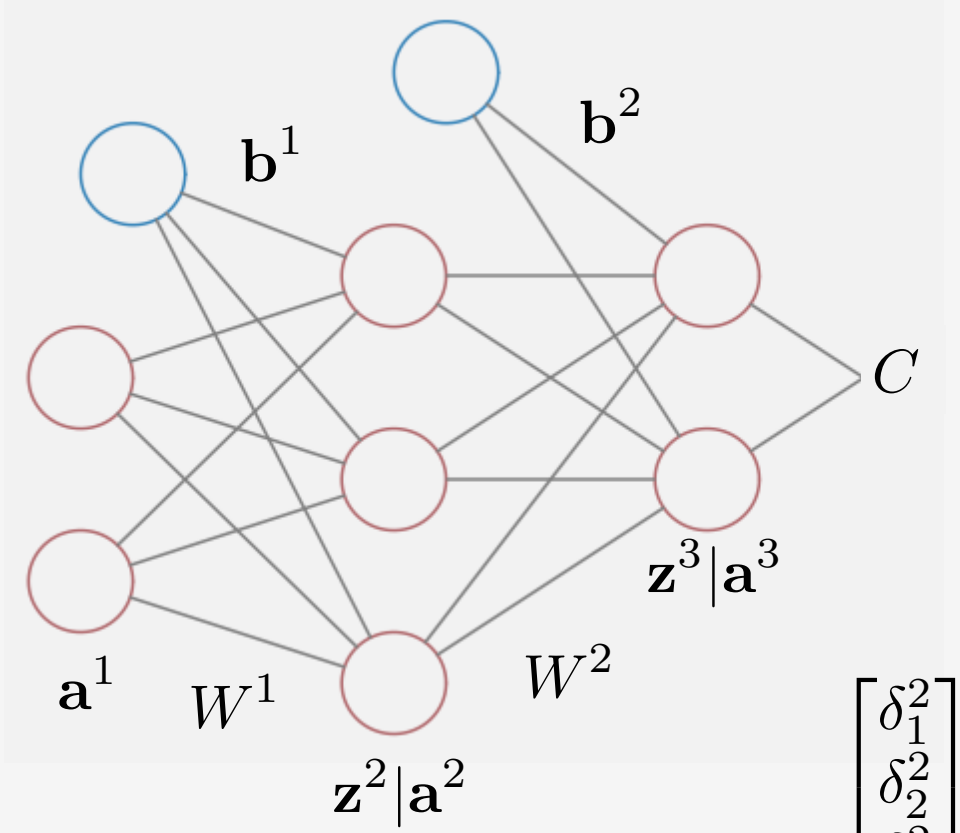
$$\delta_2^2 = (\delta_1^3\, w_{12}^2 + \delta_2^3\, w_{22}^2) \cdot g'(z_2^2)$$

$$\delta_3^2 = (\delta_1^3\, w_{13}^2 + \delta_2^3\, w_{23}^2) \cdot g'(z_3^2)$$

Does this help?

$$(W^2)^T = \begin{bmatrix} w_{11}^2 & w_{21}^2 \\ w_{12}^2 & w_{22}^2 \\ w_{13}^2 & w_{23}^2 \end{bmatrix} \qquad \delta^3 = \begin{bmatrix} \delta_1^3 \\ \delta_2^3 \end{bmatrix}$$

Back Propagation



Doing this for all three δ_i^2's yields something nice...

$$\delta_1^2 = (\delta_1^3\ w_{11}^2 + \delta_2^3\ w_{21}^2) \cdot g'(z_1^2)$$

$$\delta_2^2 = (\delta_1^3\ w_{12}^2 + \delta_2^3\ w_{22}^2) \cdot g'(z_2^2)$$

$$\delta_3^2 = (\delta_1^3\ w_{13}^2 + \delta_2^3\ w_{23}^2) \cdot g'(z_3^2)$$

How about now?

$$\begin{bmatrix} \delta_1^2 \\ \delta_2^2 \\ \delta_3^2 \end{bmatrix} = \begin{bmatrix} w_{11}^2 & w_{21}^2 \\ w_{12}^2 & w_{22}^2 \\ w_{13}^2 & w_{23}^2 \end{bmatrix} \begin{bmatrix} \delta_1^3 \\ \delta_2^3 \end{bmatrix} \odot \begin{bmatrix} g'(z_1^2) \\ g'(z_2^2) \\ g'(z_3^2) \end{bmatrix} = (W^2)^T \delta^3 \odot g'(\mathbf{z}^2)$$

Back Propagation

OK Great! With δ^3 we can easily(-ish) back propagate to get δ^2

Once we know δ^2 we can easily compute derivatives of C w.r.t. W^1 and $\mathbf{b}^1$ just like we did with W^2 and $\mathbf{b}^2$

1. Compute $\delta^2 = (W^2)^T \delta^3 \odot g'(\mathbf{z}^2)$
2. Compute $\frac{\partial C}{\partial W^1} = \delta^2 (\mathbf{a}^1)^T$ and $\frac{\partial C}{\partial \mathbf{b}^1} = \delta^2$

We've now worked this out for a network with one hidden layer

But nothing we've done assumed anything about the number of layers, so we can use this procedure for any number of of layers

SGD with Back Propagation

FORWARD PROP 1 TRAINING EXAMPLES

For a general L-layer network we have the following procedure

$\delta^L = \nabla_{\mathbf{a}^L} C \odot g'(\mathbf{z}^L)$ # Compute δ's on output layer

For $\ell = L-1, \ldots, 1$:

$\frac{\partial C}{\partial W^\ell} = \delta^{\ell+1}(\mathbf{a}^\ell)^T$ # Compute weight derivatives

$\frac{\partial C}{\partial \mathbf{b}^\ell} = \delta^{\ell+1}$ # Compute bias derivatives

$\delta^\ell = (W^\ell)^T \delta^{\ell+1} \odot g'(\mathbf{z}^\ell)$ # Back-Prop. δ's to next layer back

Perform SGD update: $W^\ell \leftarrow W^\ell - \eta \frac{\partial C}{\partial W^\ell}, \quad \mathbf{b}^\ell \leftarrow \mathbf{b}^\ell - \eta \frac{\partial C}{\partial \mathbf{b}^\ell}$ for $\ell = 1, 2, \ldots, L-1$

Training a Feed-Forward Neural Net

Given initial guess for weights and biases:

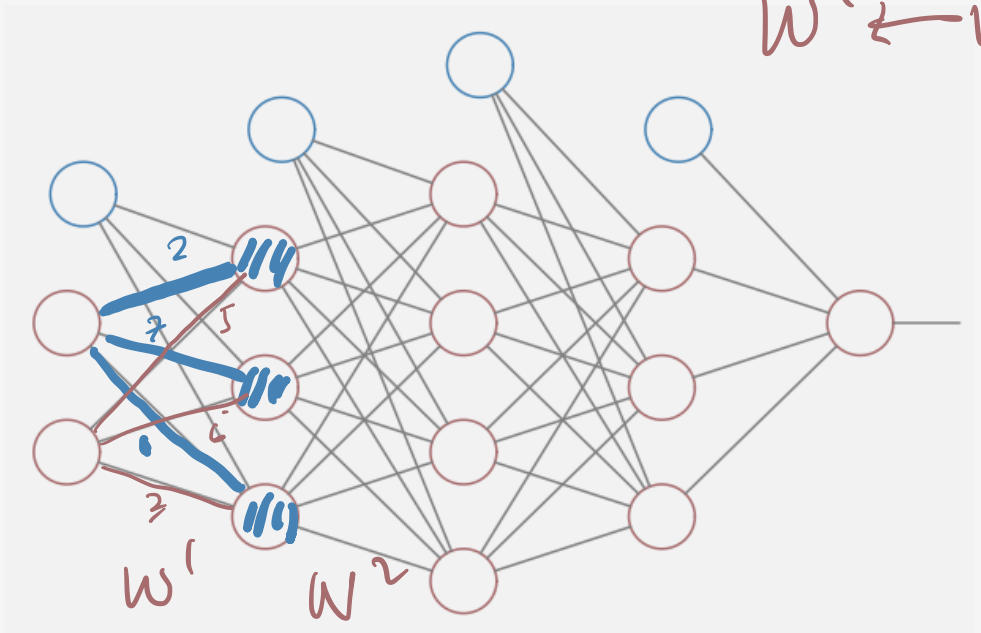
Loop over each training example in random order

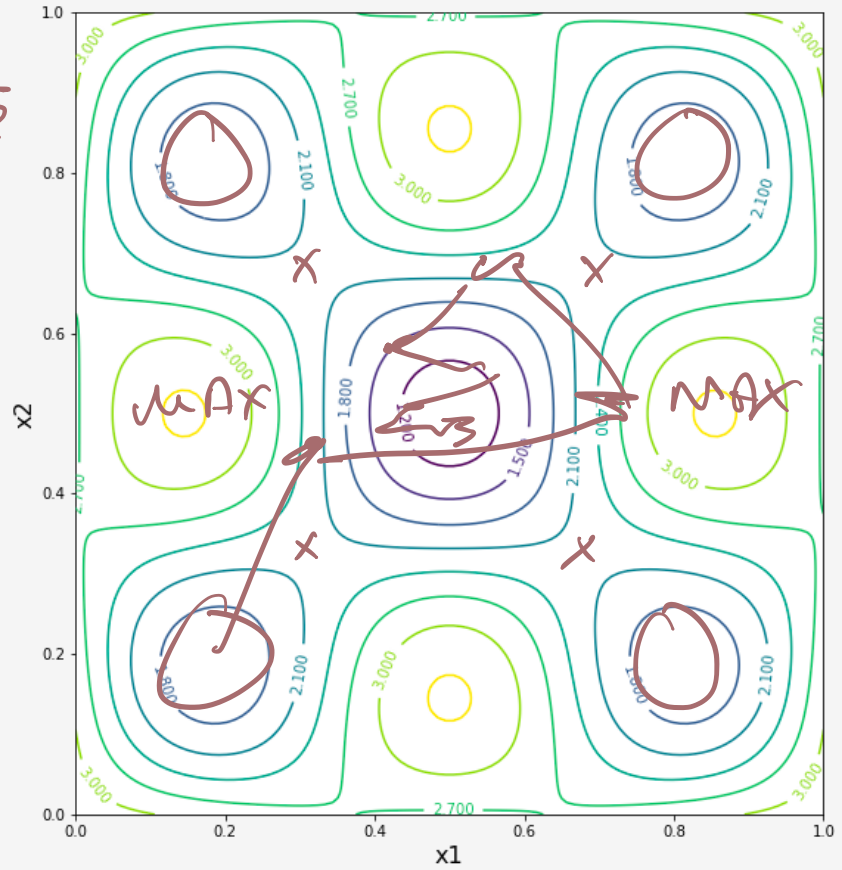
1. Forward Propagate to get activations on each layer
2. Back Propagate to get derivatives
3. Update weights and biases via Stochastic Gradient Descent
4. Rinse and Repeat

Training a Feed-Forward Neural Net

Should we be worried about local minima?

$$w^l \leftarrow w^l - \eta \frac{\partial C}{\partial w^l}$$



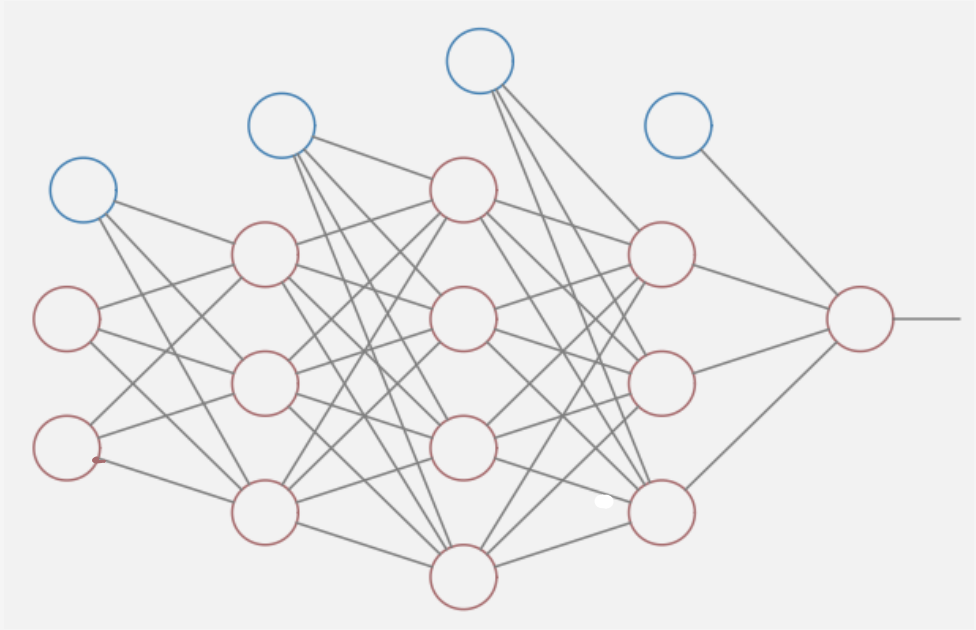


Small networks may have bad local minima

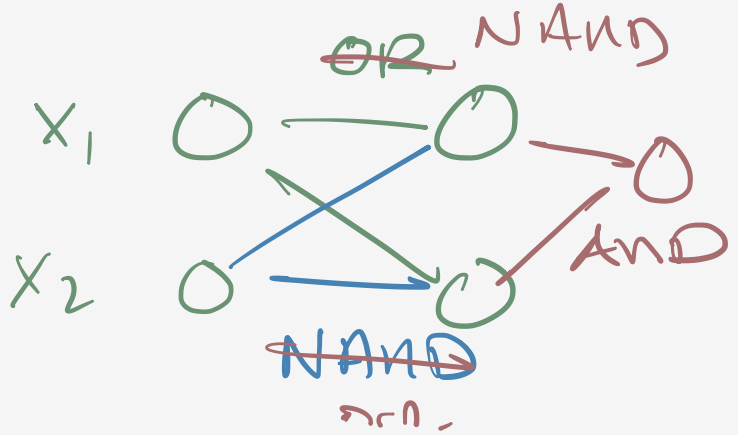
Saddle points are more of a problem in training though

Training a Feed-Forward Neural Net

How should we initialize weights and biases?



XOR = X_1 or X_2 (AND)
not (X_1 AND X_2)



Initializing with all zeros (or all the same value) causes locking

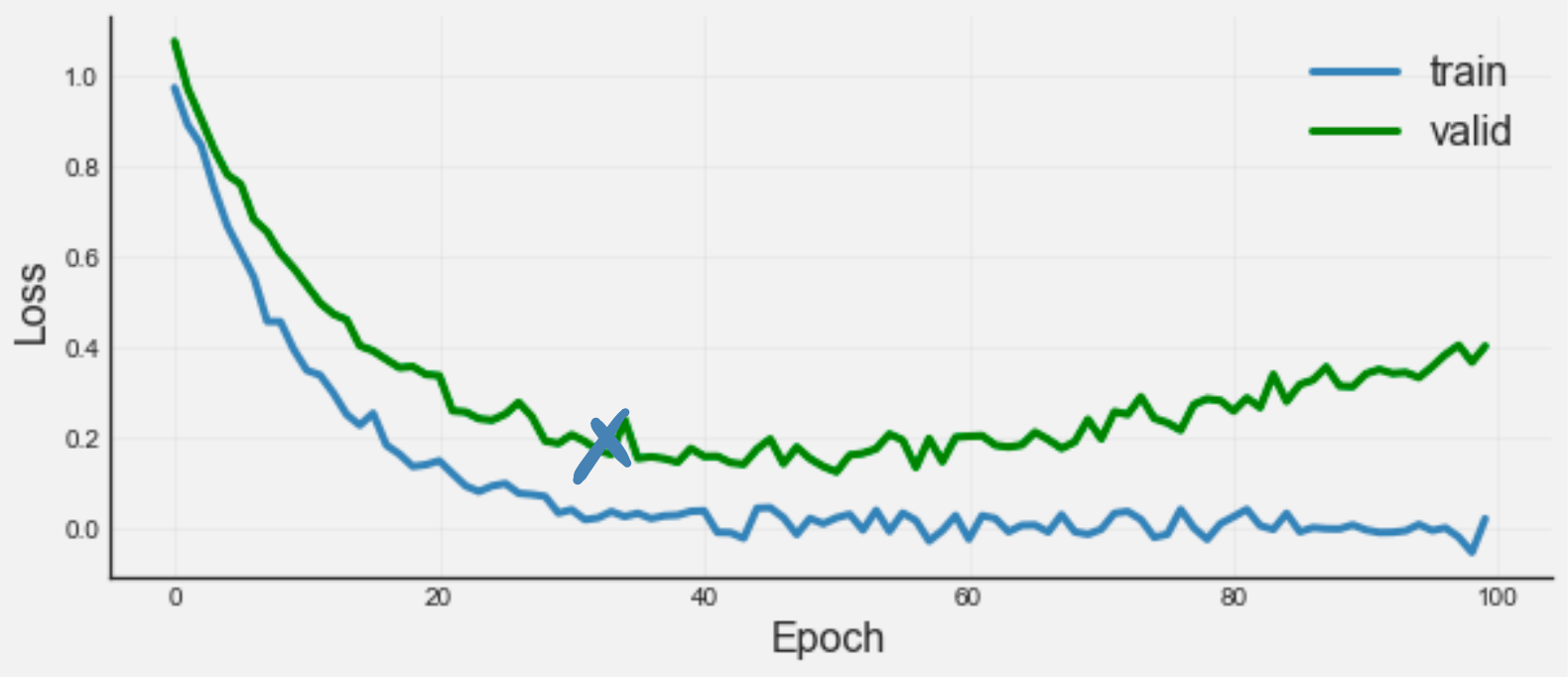
Better to initialize with small normal random values

Training a Feed-Forward Neural Net

When do we stop? In general, the same way we do in any SGD training

Monitor loss function on training and validation data

Stop when validation loss stops improving



Training a Feed-Forward Neural Net

Should we do some form of **Regularization**?

Definitely in some form

Can do the usual ℓ_2-Regularization by modifying the loss function

$$C_\lambda = C + \frac{\lambda}{2}\sum_w w^2$$

Other common forms of Regularization

- Early stopping
- Dropout

Training a Feed-Forward Neural Net

Should we do some form of **Regularization**?

Definitely in some form

Can do the usual ℓ_2-Regularization by modifying the loss function

$$C_\lambda = C + \frac{\lambda}{2}\sum_w w^2$$

Other common forms of Regularization

- Early stopping
- Dropout

Mini-Batch Training

Better to update parameters based on multiple training examples

Strikes balance between updating frequently and using all data that fits in cache

Loop over b training examples in random order

1. Forward Prop / Back Prop each example to get derivatives
2. Update weights and biases via SGD based on averaged gradients

$$W^\ell \leftarrow W^\ell - \frac{\eta}{b} \sum_j \frac{\partial C(\mathbf{x}_j)}{\partial W^\ell}, \qquad \mathbf{b}^\ell \leftarrow \mathbf{b}^\ell - \frac{\eta}{b} \sum_j \frac{\partial C(\mathbf{x}_j)}{\partial \mathbf{b}^\ell}$$

3. Rinse and Repeat

Numerical Gradient Checking

Remember that we hardcode symbolic derivatives of Loss and activation functions

Sometimes we screw this up. **Very very important** to run tests to make sure.

Consider a loss function C which is a function of all weights and biases in network

We can estimate derivative of C w.r.t. particular parameter in the network by

$$\frac{\partial C}{\partial w_i} \approx \frac{C(w_1, \ldots, w_i + \epsilon, \ldots, w_N) - C(w_1, \ldots, w_i - \epsilon, \ldots, w_N)}{2\epsilon}$$

Choose random training example. Perturb single weight. Forward prop to evaluate Loss

Compute difference between two Loss values and compare to deriv from Back Prop

$$\frac{df}{dx}(ex-3)$$

$$\frac{f(3.1) - f(2.9)}{.2}$$

$$\frac{f(3+\epsilon) - f(3-\epsilon)}{2\epsilon}$$