3. Initialisation

3.1. 2-layer NN $(W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]})$

3.2. L-layer NN → np.random.randn(layer_dims[l], layer_dims[l-1])*

→ np.zeros((layer_dims[l], 1)) → Does (()) give the D.S shape?

↳ They create a tuple. It's passed to the func as a single argument

4. For. prop. module

4.1. Linear forward $Z^{[l]} = W^{[l]}A^{[l-1]} + b^{[l]}$ $(A^{[0]} = X)$

4.2. Linear-Activation Forward Sigmoid, ReLU

A, activation_cache = sigmoid(Z)

A, activation_cache = relu(Z)

$A^{[l]} = g(Z^{[l]}) = g(W^{[l]}A^{[l-1]} + b^{[l]})$

4.3. L-Layer Model Function that replicates the previous one (linear_activation_forward w/ RELU) L-1 times, then follows that w/ one linear_activation_forward w/ SIGMOID

Now we've got for. prop. w/ input X, output $A^{[L]}$; using $A^{[L]}$ we can compute ↑ cost

5. Cost function Implement for. & backprop.

$$J = -\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}\log\left(a^{[L](i)}\right) + (1-y^{(i)})\log\left(1-a^{[L](i)}\right)\right)$$

Note: np.multiply: multiply arguments element-wise

6. Backprop. module Calc. gradient of loss func. w.r.t params.

- LINEAR backward
- LINEAR → ACTIVATION backward
- [LINEAR → RELU] × (L-1) → LINEAR → SIGMOID backward (whole model)

6.1. Linear backward

$$dW^{[l]} = \frac{\partial L}{\partial W^{[l]}} = \frac{1}{m}dZ^{[l]}A^{[l-1]T}$$

$$db^{[l]} = \frac{\partial L}{\partial b^{[l]}} = \frac{1}{m}\sum_{i=1}^{m}dZ^{[l](i)}$$

$$dA^{[l-1]} = \frac{\partial L}{\partial A^{[l-1]}} = W^{[l]T}dZ^{[l]}$$

Diff. betw. np.dot & np.multiply?

np.squeeze: remove 1D entries from the shape of an array

6.2. Linear-activation backward Merge linear_backward & linear_activation_backward

- dZ = sigmoid_backward(dA, activation_cache)
- dZ = relu_backward(dA, activation_cache)

$dZ^{[l]} = dA^{[l]} * g'(Z^{[l]})$ ← activation