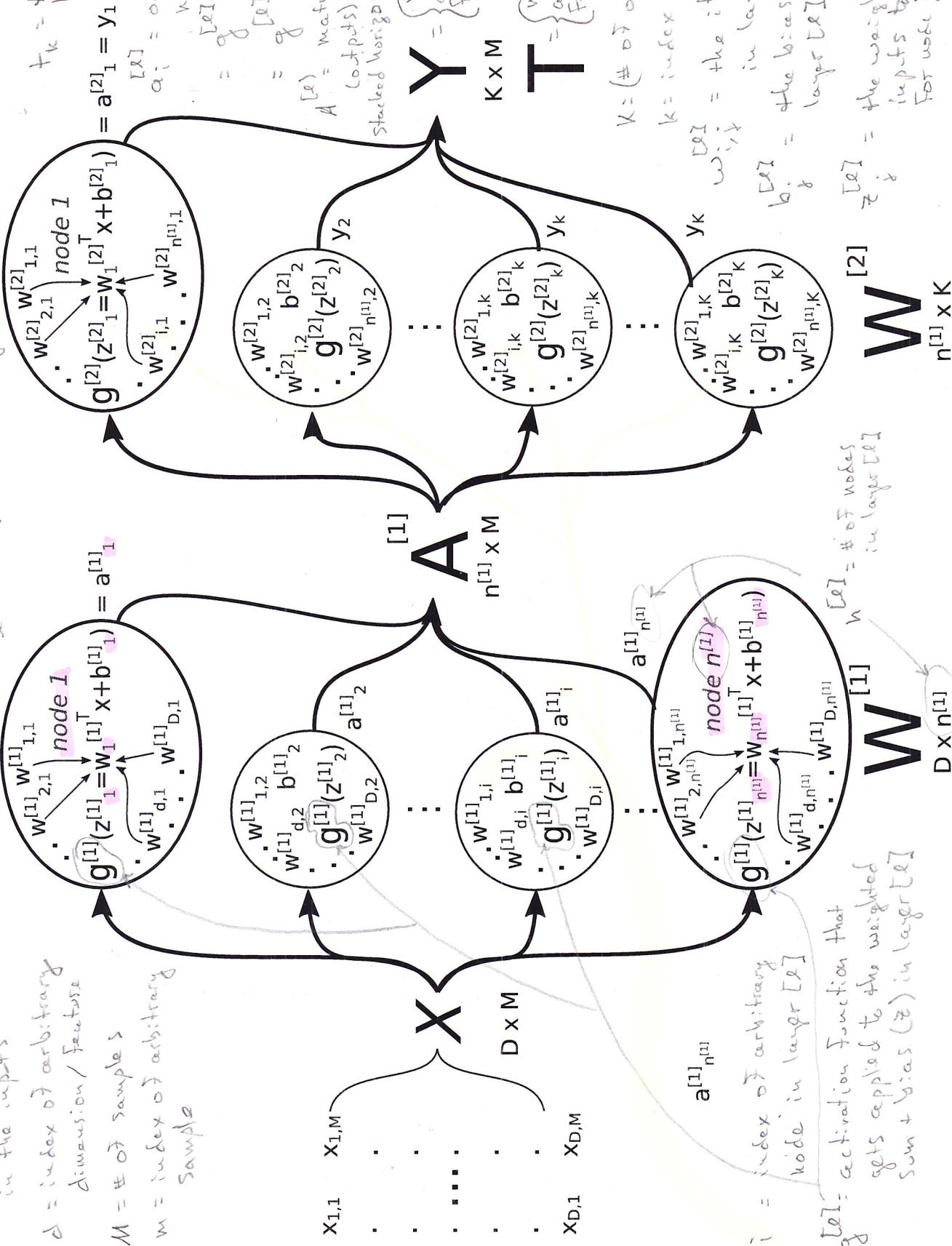


$D = \#$  of dimensions / features  
in the inputs

$d = \#$  of arbitrary  
dimension / feature  
 $M = \#$  of samples  
 $m = \#$  of arbitrary  
Sample

$L = \#$  layers in the network  
 $\ell = \#$  index of arbitrary layer

$y_k = \text{estimate for the } k^{\text{th}}$   
 $t_k = \text{target value for the } k^{\text{th}}$   
 $\text{output/response}$



2-Layer, with Single Hidden Layer Feedforward Neural Network (M samples in columns stacked horizontally)

(1)

11

## Derivation of Error Backpropagation - Regression

Start with the end in mind - Let's say that we have a fully-trained regression knowing all our weights & biases. How we compute estimates for our  $\hat{y}$  values given an input vector  $x^{(1)}$ ?

Inputs - Each sample  $m$  of our input is a  $D$ -dimensional column vector:

(1)

$x_m = \begin{bmatrix} x_{1,m} \\ x_{2,m} \\ \vdots \\ x_{D,m} \end{bmatrix}$  If we have  $M$  samples and stack each sample horizontally, we get

a  $D \times M$  matrix that looks like this:

$$(2) X = \begin{bmatrix} x_{1,1} & x_{1,M} \\ x_{2,1} & x_{2,M} \\ \vdots & \vdots \\ x_{D,1} & x_{D,M} \end{bmatrix}$$

We can call each column vector  $x_m$  to describe an arbitrary sample  $m$

Hidden-Layer - In our fully-connected network, each input is sent to each node in the first hidden-layer (the only hidden-layer in our 2-layer example). Within each node, the inputs are multiplied by the weights and then summed with a bias term and then a non-linear transform is applied to this weighted sum:

(3)

$$z_{i,m}^{[1]} = \left( \sum_{d=1}^D w_{di}^{[1]T} x_{d,m} \right) + b_i^{[1]}$$

The superscript  $[1]$  we use here just denotes the layer we are computing from.

(4)

$$a_{i,m}^{[1]} = g^{[1]}(z_{i,m}^{[1]})$$

The index  $i$  refers to the node. The  $m$  refers to the sample as described above.

(1) This is called doing a "forward pass" thru the network

The  $z_{i,m}^{[l]}$  values for an arbitrary layer  $l$  can be collected as a vector and written as:

$$(5) \quad z_m^{[l]} = \begin{bmatrix} z_{1,m}^{[l]} \\ z_{2,m}^{[l]} \\ \vdots \\ z_{n,m}^{[l]} \end{bmatrix}$$

where  $n^{[l]}$  is the number of nodes in layer  $l$ : a hyperparameter

If we stack the vectors generated by each sample horizontally, like we did for  $X$  we'd create the following  $n^{[l]} \times M$  matrix:

$$(6) \quad Z^{[l]} = \begin{bmatrix} z_{1,1}^{[l]} & z_{1,2}^{[l]} & \dots & z_{1,n}^{[l]} & z_{1,1}^{[l]} \\ z_{2,1}^{[l]} & z_{2,2}^{[l]} & \dots & z_{2,n}^{[l]} & z_{2,1}^{[l]} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ z_{n,1}^{[l]} & z_{n,2}^{[l]} & \dots & z_{n,n}^{[l]} & z_{n,1}^{[l]} \end{bmatrix}$$

To get the outputs from layer  $l$ , we apply an activation function element-wise to (6) which we'll write as:

$$(7) \quad A^{[l]} = g^{[l]}(Z^{[l]}) = g^{[l]} \left( \begin{bmatrix} z_{1,1}^{[l]} & \dots & z_{1,n}^{[l]} & \dots & z_{1,n}^{[l]} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ z_{n,1}^{[l]} & \dots & z_{n,n}^{[l]} & \dots & z_{n,n}^{[l]} \end{bmatrix} \right)$$

$\xleftarrow{\quad n^{[l]} \quad}$

In general, for  $L$ -layer, the output for each layer can be written as:

$$(8) \quad A^{[l]} = g^{[l]}(W^{[l]}A^{[l-1]} + B^{[l]})$$

where  $g^{[l]}$  is the activation function applied to layer  $l$ : another hyperparameter

So to do a complete forward pass, we use our input matrix  $X$  to compute  $Z^{[1]}$  by computing the output of each node in layer  $l=1$  using ③, stacking them vertically into column vectors as shown in ⑤, then stacking the column vectors horizontally to form the matrix shown in ⑥.

With  $Z^{[1]}$ , we apply  $g^{[1]}$  element-wise to get the activations  $A^{[1]}$  as shown in ⑦.

If we had an  $L$ -layer network, we'd continue to compute activations for each layer using

⑧ until we get to our last layer  $A^{[l=L]}$  where  $L = \text{number of layers in the network}$ .

Since this is last layer  $A^{[L]} = Y$  and we're done.

Since we only have 2 layers, we only need to compute:

$$X \xrightarrow{g^{[1]}} A^{[1]} \xrightarrow{g^{[2]}} Y$$

In our 2-layer regression net,  $g^{[2]}$  is just the identity function, so  $Z^{[2]} = Y$ . When we modify our net for classification, we'll use a soft-max function for  $g^{[2]}$ , but we'll save those details for later.

So to summary, to compute the estimates for  $Y$  with our 2-layer net we need to compute:

$$⑩ Y = W^{[2]^T} A^{[1]} + B^{[2]} = W^{[2]^T} \left( g^{[1]}(W^{[1]^T} X + B^{[1]}) \right) + B^{[2]}$$

We didn't talk about those  $b_i^{[l]}$  values or those  $B^{[l]}$  vectors. These are called "bias" values and are just another set of parameters we need to estimate along with the weights.

Now we know how to compute outputs from a given set of (weights & biases). Next, we need to get good values for these. We'll use gradient descent to find a set of weights which minimizes an error function. For regression, the mean square error (MSE) works well. This is defined as:

$$(11) \quad E = \frac{1}{2} \sum_{m=1}^M \sum_{k=1}^K (y_m(x_m) - \hat{y}_m)^2$$

Why the  $\frac{1}{2}$ ? Well, it's just a convenience and minimizing  $\frac{1}{2}E$  is the same as minimizing  $E$ , so we'll use it.

The way gradient descent works is by the following steps:

- 1) Initialize all our weights & biases to random values
- 2) Do a forward pass with these weights to compute outputs  $\hat{Y}$
- 3) Then update the weights using equation (12) until  $E$  in (11) is sufficiently minimized

$$(12) \quad w_{p,q}^{[l+1]} = w_{p,q}^{[l]} - \rho \frac{\partial E}{\partial w_{p,q}^{[l]}} \quad \text{where } p \text{ is the } p^{\text{th}} \text{ weight of node } q \text{ in layer } l$$

For layer 1 (hidden)

$$(13) \quad w_{d,i}^{[l+1]} = w_{d,i}^{[l]} - \rho_i \frac{\partial E}{\partial w_{d,i}^{[l]}} \quad \begin{aligned} \text{where } d \text{ is the } d^{\text{th}} \\ \text{dimension of input } X \text{ and} \\ i \text{ is the } i^{\text{th}} \text{ node in layer } l \end{aligned}$$

For layer 2 (output), (12) becomes

$$(14) \quad w_{i,k}^{[2]} = w_{i,k}^{[2]} - p_2 \frac{\partial E}{\partial w_{i,k}^{[2]}} \quad \text{where } i \text{ is the } i^{\text{th}} \text{ weight for node } k \text{ in layer 2}$$

The values  $p_1$  &  $p_2$  are referred to as the learning rate for layers 1 & 2 respectively.

These 2 values are also hyperparameters.

So our next job is to figure out the gradient terms in (13) & (14). We'll start with the one in (14):

Let's define error for a single sample as:

$$(15) \quad E_m = \frac{1}{2} \sum_{k=1}^K (y_k(x_m) - t_{km})^2 \quad \text{which allows us to rewrite (14) as:}$$

$$(16) \quad E = \frac{1}{M} \sum_{m=1}^M E_m \quad \text{which allows us to rewrite (14) as:}$$

$$(17) \quad w_{i,k}^{[2]} = w_{i,k}^{[2]} - p_2 \frac{1}{M} \frac{1}{K} \sum_{m=1}^M \frac{\partial E_m}{\partial w_{i,k}^{[2]}}$$

$$(18) \quad \frac{\partial E_m}{\partial w_{i,k}^{[2]}} = -(y_k(x_m) - t_{km}) \frac{\partial y_k(x_m)}{\partial w_{i,k}^{[2]}} \quad \text{Notice how the } \frac{1}{2} \text{ goes away...}$$

$$y_k(x_m) = w_k^{[2] T} a_m^{[1]} + b_k^{[2]}$$

$$(19) \quad = w_{1,k}^{[2]} a_1^{[1]} + w_{2,k}^{[2]} a_2^{[1]} + \dots + w_{i,k}^{[2]} a_i^{[1]} + \dots + w_{n,k}^{[2]} a_{n,k}^{[1]} + b_k^{[2]}$$

Looking closely at ⑨, we see that only one term is not considered a constant wrt. LHS of ⑧:

$$⑩ \frac{\partial y(x_m)}{\partial w_{i,k}} = a_i^{[2]} \quad \therefore ⑧ \text{ becomes:}$$

$$⑪ \frac{\partial E_m}{\partial w_{i,k}} = (y_i(x_m) - t_{k,m}) a_i^{[2]}$$

Now for the derivative in ⑬:

$$⑫ \frac{\partial E_m}{\partial w_{d,i}} = - \sum_{k=1}^K (y_k(x_m) - t_{k,m}) \boxed{\frac{\partial y_k(x_m)}{\partial w_{d,i}}}$$

(Chain rule to the rescue for this term)

$$⑬ \frac{\partial y_k(x_m)}{\partial w_{d,i}} = \underbrace{\frac{\partial y_k(x_m)}{\partial a_i^{[1]}}}_{\text{From ⑨, this}} \underbrace{\frac{\partial a_i^{[1]}}{\partial w_{d,i}}}_{\text{derivative is:}}$$

$$⑭ \frac{\partial y_k(x_m)}{\partial a_i^{[1]}} = w_{i,k}^{[2]} \quad \text{Apply the Chain Rule again to evaluate this term:}$$

$$⑮ \frac{\partial a_i^{[1]}}{\partial w_{d,i}} = \underbrace{\frac{\partial a_i^{[1]}}{\partial z_{i,m}^{[0]}}}_{\text{Using ④ to evaluate}} \underbrace{\frac{\partial z_{i,m}^{[0]}}{\partial w_{d,i}}}_{\text{this term:}}$$

$$⑯ \frac{\partial a_i^{[1]}}{\partial z_{i,m}^{[0]}} = g^{[1]}(z_{i,m}^{[0]}) \quad \text{To evaluate this expression, we'll need to select } g^{[1]}. \text{ We'll leave it general for now.}$$

7

Using (5) to evaluate the other term:

$$\frac{\partial z_{i,m}^{[1]}}{\partial w_{d,i}^{[1]}} = \sum_{j=1}^D \left( w_{1,i}^{[1]T} x_{1,m} + w_{2,i}^{[1]T} x_{2,m} + \dots + w_{d,i}^{[1]T} x_{d,m} \right)$$

$$(25) \quad \frac{\partial z_{i,m}^{[1]}}{\partial w_{d,i}^{[1]}} = x_{d,m} \quad \text{Putting it all together:}$$

$$(26) \quad \frac{\partial E_m}{\partial w_{d,i}^{[1]}} = \sum_{k=1}^K (y_k(x_m) - t_{k,m}) w_{i,k}^{[2]} g'(z_{i,m}^{[1]}) x_{d,m}$$

From (16), we can write:

$$(27) \quad \boxed{\frac{\partial E}{\partial w_{d,i}^{[1]}} = \frac{1}{M} \frac{1}{K} \sum_{m=1}^M \sum_{k=1}^K (y_k(x_m) - t_{k,m}) w_{i,k}^{[2]} g'(z_{i,m}^{[1]}) x_{d,m}}$$

Similarly, we can write for the layer 2 weights:

$$(28) \quad \boxed{\frac{\partial E}{\partial w_{i,k}^{[2]}} = \frac{1}{M} \frac{1}{K} \sum_{m=1}^M (y_k(x_m) - t_{k,m}) q_i^{[2]}}$$

The next step is to get expressions for the bias terms. After deriving the 2 bias expressions, we'll put these into equations (13), (14) and two more for the bias terms which we still need to derive. The resulting 4 update equations are all we need to implement gradient descent, but we'll still want to do one final step: convert these update rules to matrix expressions.