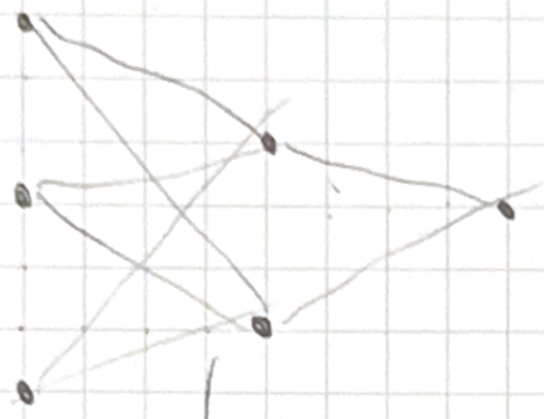
Forward Propagation

each layer is a matrix $W = \begin{bmatrix} - w_1 - \\ - w_2 - \end{bmatrix}$ ← we do this because it is the same as $W^T x$. no need for transpose now

- each row in the matrix represents the weights for a neuron in the next layer
- the length of a row (the number of columns) is the dimensions of 1 input vector (example)

each example / activation map X } equal

- each column is an example
- the length of a col (the number of rows) is the dimensions of 1 input vector (example)



$W = [\quad]$ $V = [\quad]$

2×3 1×2

1 input

$x = \begin{bmatrix} \\ \\ \end{bmatrix}$

3×1

$q_1 = Wx$

$[2 \times 3][3 \times 1]$

2×1

$q_2 = Vq_1$

$= [1 \times 2][2 \times 1]$

$= 1 \times 1$

h inputs/examples

$x = \begin{bmatrix} & & \\ & & \\ & & \end{bmatrix}$

$3 \times h$

$q_1 = Wx$

$[2 \times 3][3 \times h] = [2 \times h]$

$2 \times h$

$q_2 = Vq_1$

$= [1 \times 2][2 \times h] = [1 \times h]$

$= 1 \times h$

each column is the output of 1 example

Forward Propagation

$z_1 = W_1 x + b_1$
$a_1 = \tanh(z_1)$

$z_2 = W_2 a_1 + b_2$
$a_2 = \sigma(z_2)$ sigmoid

$\hat{y} = a_2$

Binary Cross Entropy loss function (Binary)

$$J_{BCE} = -\frac{1}{n} \sum_{i=1}^{n} y_i \ln(a_2) + (1 - y_i) \ln(1 - a_2)$$

Cross Entropy Loss function (multi class)

$$J = -\frac{1}{n} \sum_{i=1}^{n} \sum_{k=1}^{K} y_{ki} \ln(\hat{y}_{ki})$$

↑ one hot / indicator matrix

If you have _, then:
- high bias, then
  - deeper network to learn patterns
  - change architecture that might capture more
- high variance, then
  - prevent overfitting and improve generalization
  - regularization
  - more data

Info
- Binary Classification | Sigmoid
- Multi-Class Classification | Softmax
- Regularization and dropout reduce overfitting

Issues
- long training time and lots of data used
- sensitive to initialization
- objective is non-convex, many local optima
- hyper parameter tuning
- mismatched train/dev/test. Ensure they have same distribution

Vanishing Gradients
- in deep networks, the gradients in the lower layers are very small
- lots of multiplication $\hat{y} = W_{[L]} W_{[L-1]} \cdots W_2 W_1 x$

Reduce Vanishing Gradients
- random weights * $\sqrt{\frac{2}{n-1}}$

Batch Normalization
- input have zero mean and unit variance

Mini batch Gradient Descent
- make batches of training set
- update weights at the end of each batch, rather than at the end
- batches fit data in CPU/GPU

Gradient Descent Momentum
- update on gradient and previous direction

learning rate decay
- slowly reduce learning rate

Chain Rule

$z = f(y)$
$y = g(x)$

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \cdot \frac{\partial y}{\partial x}$$

$z = 2y + 1$
$y = x^2$

$$\frac{\partial z}{\partial x} = \frac{\partial}{\partial y}(2y+1) \cdot \frac{\partial}{\partial x}(x^2)$$
$$= 2 \cdot 2x$$
$$= 4x$$

$z = 2y + 1$
$= 2x^2$

$$\frac{\partial z}{\partial x} = 4x$$

equal from chain rule

$z = y_1^2 + y_2^2$

$y_1 = x_1 + x_2^2$
$y_2 = x_2 + x_3^2$

$z = y_1(x_1, x_2)^2 + y_2(x_2, x_3)^2$

$$\nabla_x z = \begin{bmatrix} \frac{\partial}{\partial x_1}[z] \\ \frac{\partial}{\partial x_2}[z] \\ \frac{\partial}{\partial x_3}[z] \end{bmatrix}$$

$$\frac{\partial z}{\partial x_1} = \frac{\partial z}{\partial y_1} \cdot \frac{\partial y_1}{\partial x_1} = (2 \cdot y_1) \cdot (1) = 2y_1$$

$$\frac{\partial z}{\partial x_2} = \frac{\partial z}{\partial y_1} \cdot \frac{\partial y_1}{\partial x_2} + \frac{\partial z}{\partial y_2} \cdot \frac{\partial y_2}{\partial x_2} = (2 \cdot y_1)(2x_2) + (2 \cdot y_2)(1)$$
$$= 4y_1 x_2 + 2y_2$$

$$\frac{\partial z}{\partial x_3} = \frac{\partial z}{\partial y_2} \cdot \frac{\partial y_2}{\partial x_3} = (2 \cdot y_2) \cdot (2x_3) = 4y_2 x_3$$

$$\nabla_x z = \begin{bmatrix} 2y_1 \\ 4y_1 x_2 + 2y_2 \\ 4y_2 x_3 \end{bmatrix}$$