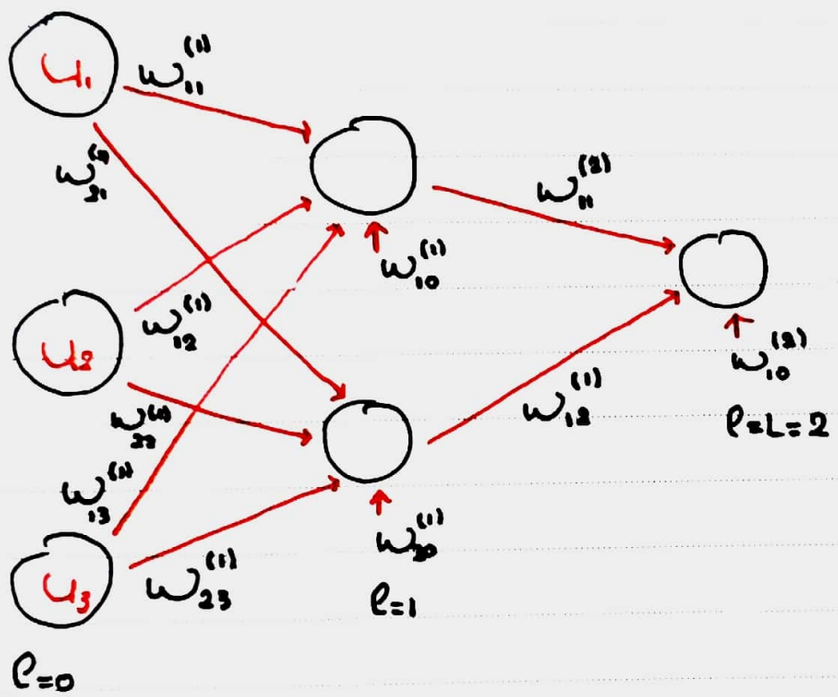
# NN Sheet 5

## 1) Consider

- Assume familiarity with lecture notation
- Drop (m) which denotes current example




- Recall $W_{ij}^{(\ell)}$ (going to (i) layer; going to (neuron); coming from (neuron))

→ Can make a $W^{(\ell)}_{N(\ell)\times N(\ell-1)}$ matrix for each layer $\ell=1,2$

- let $W_{0}^{(\ell)}_{N(\ell)\times 1} = \begin{pmatrix} w_{10}^{(\ell)} \\ w_{20}^{(\ell)} \\ \vdots \end{pmatrix}$

- Recall that the $i$th neuron in the $\ell$th layer has output

$$Y_i^{(\ell)} = f(X_i^{(\ell)})$$

where

$$X_i^{(\ell)} = \sum_{j=1} (w_{ij}^{(\ell)} \cdot Y_j^{(\ell-1)}) + w_{i0}$$

Hence, by matrix multiplication if we let $\underline{Y}^{(\ell)} = \begin{pmatrix} Y_1^{(\ell)} \\ Y_2^{(\ell)} \\ \vdots \end{pmatrix}$ (and similarly for $\underline{X}^{(\ell)}$) then

- $\underline{X}^{(\ell)} = W^{(\ell)} \underline{Y}^{(\ell-1)} + W_0^{(\ell)}$
- $\underline{Y}^{(\ell)} = f(\underline{X}^{(\ell)})$

- where $f$ is applied element wise

a) Equations of Feed Forward Propagation (i.e., X and Y for each layer)

$\ell = 0$)

$$\left.\begin{aligned} Y_1^{(0)} &= U_0 \\ Y_2^{(0)} &= U_1 \\ Y_3^{(0)} &= U_2 \end{aligned}\right\} \quad \underline{Y}^{(0)} = \underline{U}$$

$\ell = 1$)

$$X_1^{(1)} = \sum_{j=1}^{3} (\omega_{1j}^{(1)} \cdot U_j) + \omega_{10}^{(1)} \qquad (Y_j^{(0)})$$

$$Y_1^{(1)} = F(X_1^{(1)})$$

$$X_2^{(1)} = \sum_{j=1}^{3} (\omega_{2j}^{(1)} \cdot U_j) + \omega_{20}^{(1)}$$

$$Y_2^{(1)} = F(X_2^{(1)})$$

$$\left.\begin{aligned} \underline{X}^{(1)} &= W^{(1)} \underline{U} + W_0^{(1)} \qquad (\underline{Y}^{(0)}) \\ \underline{Y}^{(1)} &= F(\underline{X}^{(1)}) \end{aligned}\right\}$$

$\ell = 2$)

$$X_1^{(2)} = \sum_{j=1}^{2} (\omega_{1j}^{(2)} \cdot Y_j^{(1)}) + \omega_{10}^{(2)}$$

$$Y_1^{(2)} = F(X_1^{(2)})$$

$$\left.\begin{aligned} \underline{X}^{(2)} &= W^{(2)} \underline{Y}^{(1)} + W_0^{(2)} \\ \underline{Y}^{(2)} &= F(\underline{X}^{(2)}) \end{aligned}\right\}$$

- On the left are the equations in index notation (by the definition of the neuron) and on the right are the equivalent vector representation for the whole layer (as was discussed last page).

- Note that the activation function is given as

$$P(x) = \frac{1}{1+e^{-x}} \quad \text{(Sigmoid Function)}$$

b) Derive the loss for binary classification

- The neural network's output models $P(y=1|\theta)$ (network parameters)
- Hence, $y$ follows the Bernoulli distribution

$$P(y|\theta) = (P(y|\theta))^{y} \cdot (1 - P(y|\theta))^{1-y}$$

(when $y=1$ · when $y=0$)

- The Probability of observing the dataset is

$$\prod_{i=1}^{M} P(y_i|\theta)$$

- Maximizing this is equal to maximizing $\sum_{i=1}^{M} \log P(y_i|\theta)$

- To find $\theta$ that maximizes such Probability,

maximize

$$\sum_{i=1}^{M} \ln P(y_i|\theta) = \sum_{i=1}^{M} y_i \ln(P(y_i=1|\theta)) + (1-y_i)\ln(1-P(y_i=1|\theta))$$

which is equivalent to minimizing

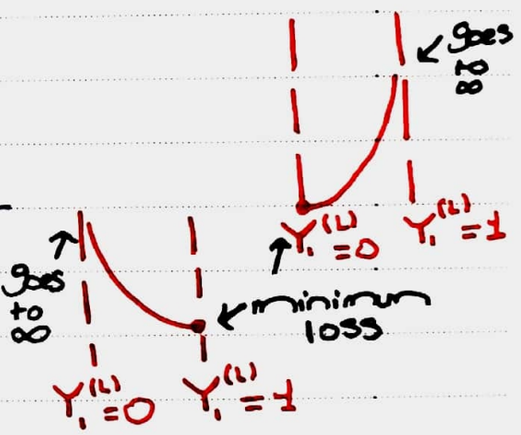
$$J = -\frac{1}{M}\sum_{i=1}^{M} y_i \ln(P(y_i=1|\theta)) + (1-y_i)\ln(1-P(y_i=1|\theta))$$

To yield the Parameters $\theta$ for which the observed data is most Probable.

- Known as binary cross entropy and can be justified using two ways

1- Consider $J_m = -y_m \ln Y_1^{(L)} - (1-y_m)\ln(1-Y_1^{(L)})$

| | |
|---|---|
| $y_m = 0$ | $-\ln(1-Y_1^{(L)})$ |
| $y_m = 1$ | $-\ln(Y_1^{(L)})$ |




- The loss is hence 0 for correct classification and gets higher the more incorrect it is

2- The cross entropy between 2 Probability distributions P and q is given by (q relative to P)

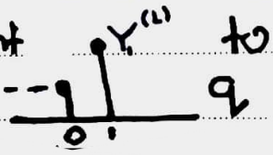
$$H(P,q) = -\sum_{x \in X} P(x)\log(q(x))$$

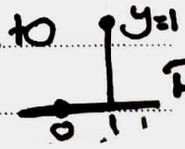
- gives entropy when $q(x) = P(x)$

- Measures the similarity between two Probability distributions (becomes entropy (min.) when they're the same)

For each example
- Minimize total cross entropy

Here we want $Y_1^{(L)}$ to be as close as possible to



c) Derive the equations of backPropagation

Recall,

$$J = \sum_{m=1}^{M} \underbrace{-y_m \ln Y_1^{(L)} - (1-y_m)\ln(1-Y_1^{(L)})}_{J_m}$$

$$X_1^{(2)} = \sum_{j=1}^{2} (w_{1j}^{(2)} \cdot Y_j^{(1)}) + w_{10}^{(2)}$$

$$Y_1^{(2)} = P(X_1^{(2)})$$

} last layer $\ell = 2 = L$

$$X_i^{(1)} = \sum_{j=1}^{3} (w_{ij}^{(1)} \underbrace{U_j}_{Y_j^{(0)}}) + w_{i0}^{(1)}$$

$$Y_i^{(1)} = P(X_i^{(1)})$$

} $\ell = 1$, $i = 1, 2$

$$Y_i^{(0)} = U_i$$

} $\ell = 0$, $i = 1, 2, 3$

→ Let's Find $\frac{\partial J_m}{\partial w_{1j}^{(2)}}$

• $\frac{\partial J}{\partial w} = \sum_{m=1}^{M} \frac{\partial J_m}{\partial w}$

$$\frac{\partial J_m}{\partial w_{1j}^{(2)}} = \frac{\partial J_m}{\partial Y_1^{(2)}} \cdot \frac{\partial Y_1^{(2)}}{\partial X_1^{(2)}} \cdot \frac{\partial X_1^{(2)}}{\partial w_{1j}^{(2)}}$$

① ② ③

① $$\frac{\partial J_m}{\partial Y_1^{(2)}} = -y_m \cdot \frac{1}{Y_1^{(2)}} - (1-y_m) \cdot \frac{-1}{1-Y_1^{(2)}} = \begin{cases} \frac{1}{1-Y_1^{(2)}} & y_m = 0 \\ \frac{-1}{Y_1^{(2)}} & y_m = 1 \end{cases}$$

② $$\frac{\partial Y_1^{(2)}}{\partial X_1^{(2)}} = P'(X_1^{(2)})$$

Since $P(x) = \frac{1}{1+e^{-x}}$ then $P'(x) = -(1+e^{-x})^{-2} \cdot -e^{-x}$

$$= \frac{e^{-x}}{1+e^{-x}} \cdot \frac{1}{1+e^{-x}}$$

$$= \frac{1}{1+e^{-x}} \cdot \left( \frac{1+e^{-x}}{1+e^{-x}} - \frac{1}{1+e^{-x}} \right)$$

$$= P(x)\,(1 - P(x))$$

Thus,

$$\frac{\partial Y_1^{(2)}}{\partial X_1^{(2)}} = Y_1^{(2)}\,(1 - Y_1^{(2)})$$

③ $$\frac{\partial X_1^{(2)}}{\partial w_{1J}^{(2)}} = \frac{\partial}{\partial w_{1J}^{(2)}} \left( \sum_{j=1}^{2} (w_{1j}^{(2)} Y_j^{(1)}) + w_{10}^{(2)} \right)$$

$$= \frac{\partial}{\partial w_{1J}^{(2)}} \left( w_{1J}^{(2)} Y_J^{(1)} \right) = Y_J^{(1)}$$

$J = 1, 2$

$Y_0^{(1)} = 1$

Thus,

$$\frac{\partial J_m}{\partial w_{1J}^{(2)}} = \left( -\frac{y_m}{Y_1^{(2)}} + \frac{(1-y_m)}{1-Y_1^{(2)}} \right) \cdot Y_1^{(2)}\,(1-Y_1^{(2)}) \cdot Y_J^{(1)}$$

$$= \left( -y_m\,(1-Y_1^{(2)}) + (1-y_m)\,Y_1^{(2)} \right) \cdot Y_J^{(1)}$$

$$= \begin{cases} Y_1^{(2)} \cdot Y_J^{(1)} & y_m = 0 \\ -Y_J^{(1)} \cdot (1 - Y_1^{(2)}) & y_m = 1 \end{cases}$$

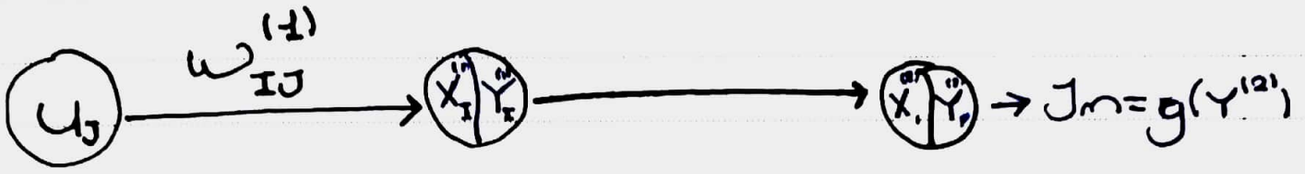
I only later noticed that we can just write this as $Y_J^{(1)} (Y_1^{(2)} - y(m))$

. In vector form:

exclude for $J=0$ (bias)

$$\partial J_m / \partial W^{(2)} = \left( \frac{\partial J_m}{\partial w_{11}^{(2)}} \; \frac{\partial J_m}{\partial w_{12}^{(2)}} \right) = \left( Y^{(1)} \right)^t \cdot \begin{cases} Y_1^{(2)} & y_m = 0 \\ -(1 - Y_1^{(2)}) & y_m = 1 \end{cases}$$

→ Now find $\frac{\partial J_m}{\partial w_{IJ}^{(1)}}$




$$\frac{\partial J_m}{\partial w_{IJ}^{(1)}} = \frac{\partial J_m}{\partial Y_1^{(2)}} \cdot \frac{\partial Y_1^{(2)}}{\partial X_1^{(2)}} \cdot \frac{\partial X_1^{(2)}}{\partial Y_I^{(1)}} \cdot \frac{\partial Y_I^{(1)}}{\partial X_I^{(1)}} \cdot \frac{\partial X_I^{(1)}}{\partial w_{IJ}^{(1)}}$$

$$\underbrace{-y_m(1-Y_1^{(2)}) + (1-y_m)Y_1^{(2)}}$$

(Calculated Previously)

$$\frac{\partial X_1^{(2)}}{\partial Y_I^{(1)}} = \frac{\partial}{\partial Y_I^{(1)}} \sum_{j=1}^{2} (w_{1j}^{(2)} \cdot Y_j^{(1)}) + w_{10}^{(2)}$$

$$= \frac{\partial}{\partial Y_I^{(1)}} (w_{1I}^{(2)} \cdot Y_I^{(1)} + w_{10}^{(2)}) = w_{1I}^{(2)}$$

• regardless of J (bias)

$$\frac{\partial Y_I^{(1)}}{\partial X_I^{(1)}} = Y_I^{(1)}(1-Y_I^{(1)})$$

$$\frac{\partial X_I^{(1)}}{\partial w_{IJ}^{(1)}} = \frac{\partial}{\partial w_{IJ}^{(1)}} \sum_{j=1}^{3} (w_{Ij}^{(1)} \cdot u_j) + w_{I0}^{(1)} = \frac{\partial}{\partial w_{IJ}^{(1)}} w_{IJ}^{(1)} \cdot u_J$$

$$= u_J \quad \text{(or 1 for } J=0\text{)}$$

$$\frac{\partial J_m}{\partial w_{IJ}^{(1)}} = \begin{cases} Y_1^{(2)} \cdot w_{1I}^{(2)} \cdot Y_I^{(1)}(1-Y_I^{(1)}) \cdot u_J & y_m = 0 \\ (Y_1^{(2)} - 1) \cdot w_{1I}^{(2)} \cdot Y_I^{(1)}(1-Y_I^{(1)}) \cdot u_J & y_m = 1 \end{cases}$$

→ which if we let ⊙ denote element-wise vector multiplication gives in vector notation:

$$\frac{\partial J_n}{\partial W^{(1)}} = \begin{cases} Y_1^{(2)} \cdot ((W^{(2)})^t \odot \underline{Y}^{(1)} \odot (\underline{1} - \underline{Y}^{(1)}))\, \underline{U}^t & y_n = 0 \\ (Y_1^{(2)} - 1) \cdot ((W^{(2)})^t \odot \underline{Y}^{(1)} \odot (\underline{1} - \underline{Y}^{(1)}))\, \underline{U}^t & y_n = 1 \end{cases}$$

We earlier had

$$\frac{\partial J_n}{\partial W^{(2)}} = \begin{cases} Y_1^{(2)} \cdot (Y^{(1)})^t & y_n = 0 \\ (Y_1 - 1)^{(2)} \cdot (Y^{(1)})^t & y_n = 1 \end{cases}$$

- ignore last term for bias vector
- exclude last term for bias

→ These are the equations of backward propagation of the network.

- Top of 1st of last 3 pages includes the forward propagation equations.

d) Suppose an input $\underline{U} = \begin{pmatrix} 3 \\ 0.2 \\ -0.1 \end{pmatrix}$ what's the output)

- $\ell = 0$) $\underline{Y}^{(0)} = \underline{U} = \begin{pmatrix} 3 \\ 0.2 \\ -0.1 \end{pmatrix}$
- $\ell = 1$) $\underline{X}^{(1)} = W^{(1)} \underline{U} + W_0^{(1)} \begin{pmatrix} 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0.31 \\ -0.28 \end{pmatrix}$

$$\begin{pmatrix} 0.1 & 0.2 & 0.3 \\ -0.1 & 0 & -0.2 \end{pmatrix}$$

$$\underline{Y}^{(1)} = P(\underline{X}^{(1)}) = \begin{pmatrix} 0.5769 \\ 0.43045 \end{pmatrix}$$

$P(z) = (1+e^{-z})^{-1}$

f) $\ell = 2$)

$$\underline{X}^{(2)} = W^{(2)} \underline{Y}^{(1)} + W_0^{(2)} = 0.02929$$

$$(0.2 \quad -0.2) \begin{pmatrix} 0.5769 \\ 0.43045 \end{pmatrix} \quad 0$$

$$\underline{Y}^{(2)} = P(\underline{X}^{(2)}) = 0.50732$$

← Network's output.

e) $0.50732 < 0.6$ and hence, class 0 will be chosen.

f) $J_m = -y_m \ln Y^{(2)} - (1-y_m)\ln(1-Y^{(2)})$ $\cdot y_m = 0$

$$= -\ln(1-Y^{(2)}) = -\ln(1-0.50732)$$

$$= 0.7078954$$

g)

$$\frac{\partial J_m}{\partial W^{(2)}} = Y_1^{(2)} \cdot (\underline{Y}^{(1)})^t = 0.50732\,(0.5769 \quad 0.43045)$$

$$= (0.29267 \quad 0.218376)$$

$$\frac{\partial J_m}{\partial W_0^{(2)}} = Y_1^{(2)} = 0.50732$$

$$\frac{\partial J_m}{\partial W^{(1)}} = Y_1^{(2)} ((W^{(2)})^t \odot \underline{Y}^{(1)} \odot (\underline{1} - \underline{Y}^{(1)})) \underline{u}^t$$

$$= 0.50732 \left( \begin{pmatrix} 0.2 \\ -0.2 \end{pmatrix} \odot \begin{pmatrix} 0.2441 \\ 0.24516 \end{pmatrix} \right) (3 \quad 0.2 \quad -0.1)$$

$$= \begin{pmatrix} 0.02476736 \\ -0.0248749 \end{pmatrix} (3 \quad 0.2 \quad -0.1)$$

$$= \begin{pmatrix} 0.0743 & 4.953\times10^{-3} & -2.4767\times10^{-3} \\ -0.0746 & -4.974\times10^{-3} & 2.48749\times10^{-3} \end{pmatrix}$$

$$\frac{\partial J_m}{\partial W_0^{(1)}} = Y_1^{(2)}\left((W^{(2)})^t \odot \underline{Y}^{(1)} \odot (\underline{1} - \underline{Y}^{(1)})\right) = \begin{pmatrix} 0.024767 \\ -0.0248749 \end{pmatrix}$$

. Continued last Page

h)

$$W_{(new)}^{(2)} = W_{(old)}^{(2)} - \lambda \frac{\partial J_m}{\partial W_{(old)}^{(2)}}$$

$W_{(old)}^{(2)} \rightarrow (0.2 \quad -0.2)$, $\lambda \rightarrow 0.05$, $\frac{\partial J_m}{\partial W_{(old)}^{(2)}} \rightarrow (0.29267 \quad 0.218376)$

$$= (0.1853665 \quad -0.2109188)$$

$$W_{(new)}^{(2)} = W_{(old)}^{(2)} - \lambda \frac{\partial J_m}{\partial W_{(old)}^{(2)}} = -0.025366$$

$W_{(old)}^{(2)} \rightarrow 0$, $\lambda \rightarrow 0.05$, $\frac{\partial J_m}{\partial W_{(old)}^{(2)}} \rightarrow 0.50732$

$$W_{(new)}^{(1)} = W_{(old)}^{(1)} - \lambda \frac{\partial J_m}{\partial W_{(old)}^{(1)}}$$

$W_{(old)}^{(1)} \rightarrow \begin{pmatrix} 0.1 & 0.2 & 0.3 \\ -0.1 & 0 & -0.2 \end{pmatrix}$, $\lambda \rightarrow 0.05$, $\frac{\partial J_m}{\partial W_{(old)}^{(1)}} \rightarrow \begin{pmatrix} 0.0743 & 4.953\times10^{-3} & -2.4\ldots \\ -0.0746 & -4.974\times10^{-3} & 2.4\ldots \end{pmatrix}$

$$= \begin{pmatrix} 0.096285 & 0.199752 & 0.300124 \\ -0.09627 & 2.487\times10^{-4} & -0.2 \end{pmatrix}$$

$$W_{\cdot(new)}^{(1)} = W_{\cdot(old)}^{(1)} - \lambda \cdot \frac{\partial J_m}{\partial W_{\cdot(old)}^{(1)}}$$

$$\begin{pmatrix} 0 \\ 0 \end{pmatrix} \quad 0.05 \quad \begin{pmatrix} 0.024767 \\ -0.024874 \end{pmatrix}$$

$$= \begin{pmatrix} -1.23835 \times 10^{-3} \\ -1.243745 \times 10^{-3} \end{pmatrix}$$

2) If it was solving a regression Problem where the output can take any value then the activation function at the output layer must be linear i.e. $F(x) = x$, $F'(x) = 1$

- FeedForward Equations

$$\underline{Y}^{(0)} = \underline{U}$$

$$\underline{X}^{(1)} = W^{(1)} \underline{U} + W_0^{(1)}$$

$$\underline{Y}^{(1)} = F(\underline{X}^{(1)})$$

- Hidden layer activations needn't change

$$\underline{X}^{(2)} = W^{(2)} \underline{Y}^{(1)} + W_0^{(2)}$$

$$\underline{Y}^{(2)} = \underline{X}^{(2)}$$

← Only Change in FeedForward

* The loss function should change as well to a regression loss:

$$J_m = \frac{1}{2}(y_m - Y^{(2)})^2, \quad J = \frac{1}{M} \sum_{m=1}^{M} J_m \quad \text{(MSE)}$$

# • Backward Propagation Equations

$$\rightarrow \frac{\partial J_m}{\partial w_{ij}^{(2)}} = \frac{\partial J_m}{\partial Y_i^{(2)}} \cdot \frac{\partial Y_i^{(2)}}{\partial X_i^{(2)}} \cdot \frac{\partial X_i^{(2)}}{\partial w_{ij}^{(2)}}$$

$\frac{\partial J_m}{\partial Y_i^{(2)}} \rightarrow (Y^{(2)} - y_m)$ ; $\frac{\partial Y_i^{(2)}}{\partial X_i^{(2)}} \rightarrow 1$ ; $\frac{\partial X_i^{(2)}}{\partial w_{ij}^{(2)}} \rightarrow Y_j^{(1)}$

→ If you look closely — $\frac{\partial J_m}{\partial Y_i^{(2)}} \cdot \frac{\partial Y_i^{(2)}}{\partial X_i^{(2)}}$ didn't really change

$$= (Y^{(2)} - y_m) \underline{Y_j^{(1)}}$$ ← exclude for bias (w.)

$$\frac{\partial J_m}{\partial W^{(2)}} = (Y^{(2)} - y_m)(\underline{Y}^{(1)})^t$$

$$\rightarrow \frac{\partial J_m}{\partial W_{IJ}^{(1)}} = \underbrace{\frac{\partial J_m}{\partial Y_i^{(2)}} \cdot \frac{\partial Y_i^{(2)}}{\partial X_i^{(2)}}}_{(Y^{(2)} - y_m)} \cdot \frac{\partial X_i^{(2)}}{\partial Y_I^{(1)}} \cdot \frac{\partial Y_I^{(1)}}{\partial X_I^{(1)}} \cdot \frac{\partial X_I^{(1)}}{\partial w_{IJ}^{(1)}}$$

$\frac{\partial X_i^{(2)}}{\partial Y_I^{(1)}} \rightarrow w_{iI}^{(2)}$ ; $\frac{\partial Y_I^{(1)}}{\partial X_I^{(1)}} \rightarrow Y_I^{(1)}(1 - Y_I^{(1)})$ ; $\frac{\partial X_I^{(1)}}{\partial w_{IJ}^{(1)}} \rightarrow U_J$

$$= (Y^{(2)} - y_m)\left((W^{(2)})^t \odot \underline{Y} \odot (\underline{1} - \underline{Y})\right)\underline{U}^t$$ ← exclude for bias

* So in essence, the backPropagation equations won't change!

Sigmoid + Binary cross entropy loss ↔ Linear + MSE loss

# 3) What would've changed for multi-class classification?

→ The output layer should involve K neurons rather than 1 (one for each class)

→ The labels in the dataset are now K x 1 vectors if the ith index = 1 then the example belongs to ith class

- The activations used in output layer should be Softmax

$$Y_i^{(L)} = \frac{e^{X_i^{(L)}}}{\sum_{j=1}^{K} e^{X_j^{(L)}}}$$

- Looks like we'll have to revisit chainrule

- The loss function should generalize from binary cross entropy to multi-class cross entropy

$$J_m = -\sum_{i=1}^{K} y_{m,i} \ln(Y_i^{(2)})$$

* Changes in FeedForward Pass

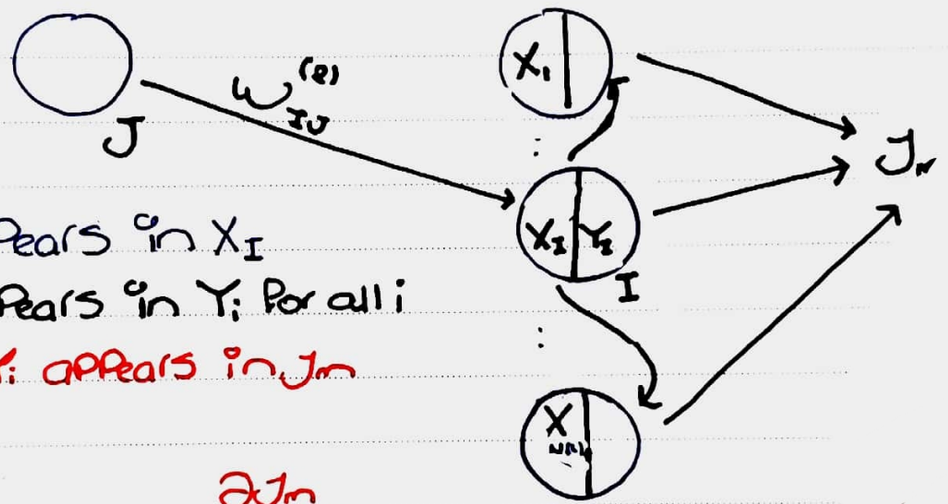
$$\underline{X}^{(2)} = W_{K\times 2}^{(2)} \underline{Y}^{(1)} + W_{0(K\times 1)}^{(2)} \qquad \left(X_i^{(2)} = \sum_{j=1}^{2} w_{ij}^{(2)} Y_j^{(1)} + w_{i0}^{(2)}\right)$$

$$\underline{Y}^{(2)} = e^{\underline{X}^{(2)}} \cdot \frac{1}{\sum_{j=1}^{K} e^{X_j^{(2)}}} \qquad \left(Y_i^{(2)} = \frac{e^{X_i^{(2)}}}{\sum_{j=1}^{K} e^{X_j^{(2)}}}\right)$$

# * Changes in backward Pass

→ First want $\frac{\partial J_m}{\partial w_{IJ}^{(2)}}$

Observe




- $w_{IJ}^{(2)}$ appears in $X_I$
- which appears in $Y_i$ for all $i$
- and each $Y_i$ appears in $J_m$

$$\frac{\partial J_m}{\partial w_{IJ}^{(2)}} = \Big(\sum_{i=1}^{K} \frac{\partial J_m}{\partial Y_i^{(2)}} \cdot \frac{\partial Y_i^{(2)}}{\partial X_I^{(2)}}\Big) \cdot \frac{\partial X_I}{\partial w_{IJ}^{(2)}}$$

$\frac{\partial J_m}{\partial X_I^{(2)}}$

$$\frac{\partial}{\partial w_{IJ}^{(2)}} \sum_{j=1}^{Q} w_{Ij}^{(2)} Y_j^{(1)} + w_{I0}^{(2)} = Y_J^{(1)} \quad (\text{or } 1 \text{ for } J=0)$$

$\frac{\partial J_m}{\partial X_I^{(2)}}$

$$\sum_{\substack{i=1 \\ i \neq I}}^{K} \underset{③}{\frac{\partial J_m}{\partial Y_i^{(2)}}} \cdot \underset{④}{\frac{\partial Y_i^{(2)}}{\partial X_I^{(2)}}} + \underset{①}{\frac{\partial J_m}{\partial Y_I^{(2)}}} \cdot \underset{②}{\frac{\partial Y_I^{(2)}}{\partial X_I^{(2)}}}$$

① $$\frac{\partial J_m}{\partial Y_I^{(2)}} = \frac{\partial}{\partial Y_I^{(2)}}\Big(-\sum_{i=1}^{K} y_{m,i} \ln(Y_i^{(2)})\Big) = \frac{\partial}{\partial Y_I^{(2)}} -y_{m,I} \ln(Y_I^{(2)}) = \frac{-y_{m,I}}{Y_I^{(2)}}$$

② $$\frac{\partial Y_I^{(2)}}{\partial X_I^{(2)}} = \frac{\partial}{\partial X_I^{(2)}}\left(\frac{e^{X_I^{(2)}}}{\sum_{i=1}^{K} e^{X_i^{(2)}}}\right)$$

$$= \frac{e^{X_I^{(2)}}}{\sum_{i=1}^{K} e^{X_i^{(2)}}}\left(\frac{e^{X_I^{(2)}}}{e^{X_I^{(2)}}} - \frac{e^{X_I^{(2)}}}{\sum_{i=1}^{K} e^{X_i^{(2)}}}\right)$$

• Take log then differentiate

$$= Y_I^{(2)}(1 - Y_I^{(2)})$$

③ $$\frac{\partial J_m}{\partial Y_i^{(2)}}_{i \neq I} = \frac{-y_{m,i}}{Y_i^{(2)}}$$

④ $$\frac{\partial Y_i^{(2)}}{\partial X_I^{(2)}}_{i \neq I} = \frac{\partial}{\partial X_I^{(2)}}\left(\frac{e^{X_i^{(2)}}}{\sum_{i=1}^{K} e^{X_i^{(2)}}}\right) = \frac{e^{X_i^{(2)}}}{\sum_{j=1}^{K} e^{X_j^{(2)}}}\left(\frac{0}{e^{X_i^{(2)}}} - \frac{e^{X_I^{(2)}}}{\sum_{j=1}^{K} e^{X_j^{(2)}}}\right)$$

$$= -Y_i^{(2)} \cdot Y_I^{(2)}$$

Now

$$\frac{\partial J_m}{\partial w_{IJ}^{(2)}} = \sum_{\substack{i=1 \\ i \neq I}}^{K} \frac{\partial J_m}{\partial Y_i^{(2)}} \cdot \frac{\partial Y_i^{(2)}}{\partial X_I^{(2)}} + \frac{\partial J_m}{\partial Y_I^{(2)}} \cdot \frac{\partial Y_I^{(2)}}{\partial X_I^{(2)}}$$

$$= \sum_{\substack{i=1 \\ i \neq I}}^{K} -\frac{y_{m,i}}{Y_i^{(2)}} \cdot -Y_i^{(2)} \cdot Y_I^{(2)} + -\frac{y_{m,I}}{Y_I^{(2)}} \cdot Y_I^{(2)}(1 - Y_I^{(2)})$$

$$= Y_I^{(2)} \sum_{\substack{i=1 \\ i \neq I}}^{K} y_{m,i} + y_{m,I}(Y_I^{(2)} - 1)$$

$$= Y_I^{(2)} \underbrace{\sum_{i=1}^{K} y_{m,i}}_{1 \text{ as } y_m \text{ is one-hot vector}} - y_{m,I} = Y_I^{(2)} - y_{m,I}$$

Thus we have

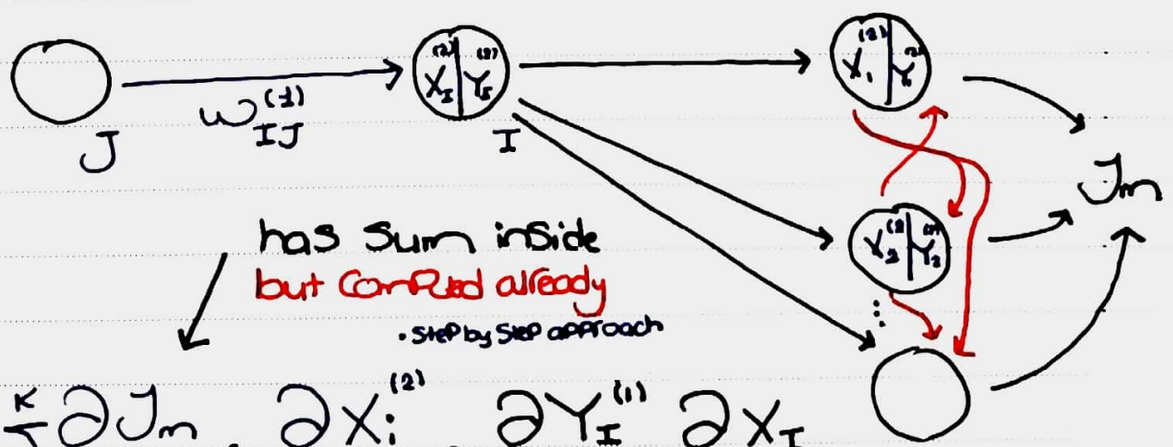
$$\frac{\partial J_m}{\partial w_{IJ}^{(2)}} = (Y_I^{(2)} - y_{m,I}) \cdot Y_J^{(1)}$$

. last term excluded for bias.

$$\frac{\partial J_m}{\partial W^{(2)}} = (\underline{Y}^{(2)} - \underline{y_m})(\underline{Y}^{(1)})^t$$

* which is Pretty much the SAME EXPRESSION that we had before (with the 1st term being generalized to a vector)

. Now For $\frac{\partial J_m}{\partial w_{IJ}^{(1)}}$




has Sum inside but Confused already

. step by step approach

$$\frac{\partial J_m}{\partial w_{IJ}^{(2)}} = \sum_{i=1}^{K} \frac{\partial J_m}{\partial X_i^{(2)}} \cdot \frac{\partial X_i^{(2)}}{\partial Y_I^{(1)}} \cdot \frac{\partial Y_I^{(1)}}{\partial X_I^{(1)}} \cdot \frac{\partial X_I}{\partial w_{IJ}^{(1)}} \rightarrow U_J$$

just like before

$$(Y_i^{(2)} - y_{m,i}) \qquad \frac{\partial \sum_{j=1}^{2} w_{ij}^{(2)} Y_j^{(1)} + w_{i0}^{(2)}}{\partial Y_I^{(1)}} = w_{iI}^{(2)} \qquad Y_I^{(1)}(1 - Y_I^{(1)})$$

$$= \sum_{i=1}^{K} (Y_i^{(2)} - y_{m,i}) w_{iI}^{(2)} \cdot Y_I^{(1)}(1 - Y_I^{(1)}) \cdot U_J$$

exclude for bias

In vector notation — let's call this $\delta_I^{(2)}$

$$\frac{\partial J_m}{\partial W_{IJ}^{(2)}} = \sum_{i=1}^{K} (Y_i^{(2)} - y_{m,i}) W_{iI}^{(2)} \cdot (Y_I^{(1)}(1 - Y_I^{(1)})) \cdot U_J$$

- Dot Product between Ith Column in Weight matrix and $(\underline{Y}^{(2)} - \underline{y}_m)$ vector
- Then multiply by Ith element here

- Now by definition of Outer Product $(A_{ij} = a_i b_j = \underline{a}\,\underline{b}^t)$

$$\rightarrow \frac{\partial J_m}{\partial W_{IJ}^{(2)}} = \underline{\delta}^{(2)} \cdot \underline{U}^t \qquad \underline{\delta}^{(2)} = \begin{pmatrix} \delta_1^{(2)} \\ \delta_2^{(2)} \\ \vdots \end{pmatrix}$$

$$\underline{\delta}^{(2)} = \left( (\underline{Y}^{(2)} - \underline{y}_m)^t_{1\times K} \underline{W}^{(2)}_{K\times 2} \right)^t_{2\times 1} \odot \underline{Y}^{(1)}(\underline{1} - \underline{Y}^{(1)})$$

(need it a col. vector)

- For the Ith element in $\delta$, we dot Product $(\underline{Y}^{(2)} - \underline{y}_m)$ with the Ith Column in W

✱ Observe

If $\underline{C}$ is a $\ell \times 1$ vector and $\underline{a}$ is a $K \times 1$ vector and $B$ is a $K \times \ell$ matrix

- Let $C_i = \sum_{j=1}^{K} a_j B_{ji} = a^T B_{:,i} = B_{:,i}^T a$

then $\underline{C} = B^T \underline{a}$ (for $i=1$, $i=2$)

- Back Prop. Eqns:

$$\partial J_m / \partial W^{(2)} = (\underline{Y}^{(2)} - \underline{y}_m)(Y^{(1)})^t$$

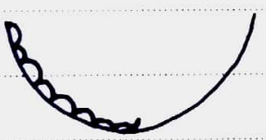
- exclude last term for bias

$$\partial J_m / \partial W^{(1)} = ((W^{(2)})^t (\underline{Y}^{(2)} - \underline{y}_m) \odot \underline{Y}^{(1)}(1 - \underline{Y}^{(1)})) \cdot U^t$$

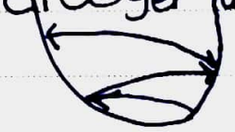
4) The learning rate signifies the step size used in each iteration of gradient descent to reach the minimum

- Changing $\eta$ affects convergence in the training process

$\eta$ too low → very slow convergence



$\eta$ too large → oscillations & possibly divergence



- Hence, the larger $\eta$ is the faster is possibly the convergence but the more prone we are to overshooting/divergence.

5) Mini-batch gradient descents divides the training data into a set of batches and performs a weight update batch by batch (the loss function considers one batch at a time).

Mini-batch GD with

Stochastic Gradient Descent → Set Batch Size = 1
Full Batch Gradient Descent → Set Batch Size = N

→ faster convergence & less memory foot print among other desirable properties.

6) Mention four types of activation function

1-

$$f(x) = Relu(x) = \max(0, x)$$

$$f'(x) = \begin{cases} 1 & x > 0 \\ 0 & x < 0 \end{cases}$$

→ Simple and easily computed (both $f(x)$ & $f'(x)$)

→ Can cause the "dying relu" problem

- If the inputs to a Neuron are -ve for all training data then the derivative will be zero & the weights will never be updated

  * leaky relu is a solution.

$$f(x) = \begin{cases} x & x > 0 \\ ax & x < 0 \end{cases}$$

→ Practical results show fast learning (due to const. $f'$) and a popular choice when dealing with images.

2-

$$f(x) = x \qquad f'(x) = 1$$

. linear activation

→ Used in the output layer in regression tasks (Predict continuous quantity)

→ Simple & efficient but the neural network becomes linear if used in hidden layers (not a universal approximator).

3-

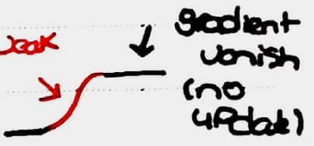
$$f(x) = \frac{1}{1+e^{-x}}$$

#Sigmoid

$$f'(x) = f(x)(1-f(x))$$

. as demonstrated earlier

. Used in the output layer when the classification task is binary




. Slow updates (weak gradients)
. Suffers from the vanishing gradient problem
. exponential operations

4-

$$f(x) = \tanh(x)$$

$$f'(x) = \text{Sech}^2(x)$$

. Pretty much scaled Sigmoid $(2\sigma(2x)-1)$ such that the gradients are stronger (faster convergence) but some disadvantages.

5-

$$f(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{N(L)} e^{x_j}}$$

$$f_{x_i}(x_i) = f(x_i)(1-f(x_i))$$

. as demonstrated earlier

$$f_{x_j}(x_i) = -f(x_i).f(x_j)$$

. Used in the output layer when the task is multi-class classification.
. Can cause numerical instability, not so efficient...
→ Solved by log Softmax

# 7) Gradient Descent HyperParameters

- Depending on the Iterative Scheme the hyperParameters can range from just the learning rate (and batch Size, #epochs) to include as well $\beta_1$, $\beta_2$ decay factors, $\epsilon$, ...

→ Manual hyperParameter tuning / rule of thumb values or better → random search with a coarse to fine scheme with appropriate ranges for each Parameter. [NN Lec. 11, 6]

# 8) Training Set  Validation Set  Test Set

- to train the model
- To tune hyperParameters
- Estimate Prediction error for model Selection
- As a final assesment to model Perf. (before deployment)

- Good Practice
  60% - 20% - 20% split
  unless dataset is gigantic then (~$10^8$)
  98% - 1% - 1%

9)

| | Overfitting | Underfitting |
|---|---|---|
| Training data Performance | High | low |
| Testing data Performance | low | low |

- Sources of error are bias & variance

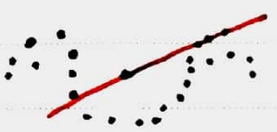
* High Variance (modelling training noise)
* High Bias (model with erroneous assumptions)

Ex.

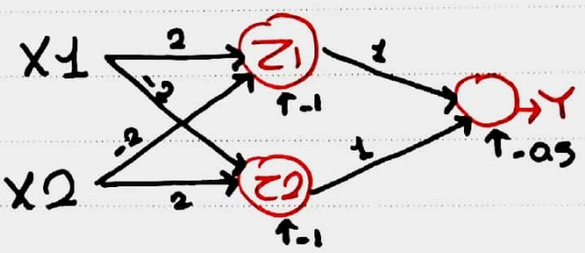
- e.g. Fitting an nth degree Polynomial on n+1 Points (dataset)

~~When the real~~ (instead of )

- Fitting a linear model on highly non linear data



10)




- Binary Input & Output

| X1 | X2 | Z1 $U(2X_1-2X_2-1)$ | Z2 $U(2X_2-2X_1-1)$ | Y $U(Z_1+Z_2-\frac{1}{2})$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 0 |

→ The function is clearly XOR

* If you look closely
  - → Y Performs OR
  - → Z1 Performs $X1\,\overline{X2}$
  - → Z2 Performs $\overline{X1}\,X2$

11)

$$Y = X1.X2 + \overline{X1}.\overline{X2}$$

- Need Z1, Z2 that implement the two terms

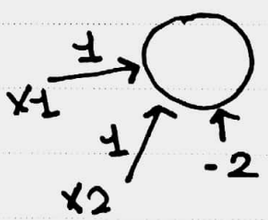
- Observe the following
  → Suppose we wanted to implement
  $Z = X1\,\overline{X2}\,X3\,X4\,\overline{X5}$
  then clearly
  $X1 - X2 + X3 + X4 - X5 + C$

no. of non-inverted

is equal to zero only when $C = 3$ $(Z = 1)$
and $X1 = X3 = X4 = 1$, $X2 = X5 = 0$. For
any other assignment, it's less than 0. $(Z = 0)$

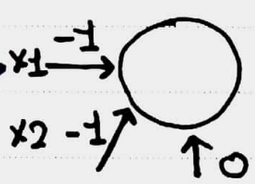
Hence,

$Z_1 = X1\,X2 \longleftrightarrow$




$X1 + X2 \geq 2$ iff & only if $X1 = X2 = 1$

$Z_2 = \overline{X1}\,\overline{X2} \longleftrightarrow$




$-X1 - X2 \geq 0$ iff & only if $X1 = X2 = 0$

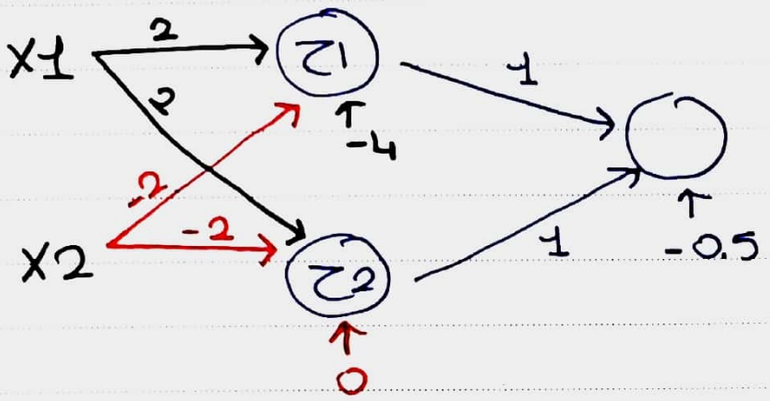
* This would always work (universal and) but you could've rather thought about

$$Z_1 = X1X2 \longleftrightarrow aX1 + bX2 + C \geq 0$$

what a, b, C make this only hold for $X1 = X2 = 1$

- Notice that $U(x) = U(ax)$, can hence always multiply all weights into a neuron by weight. (we will x2)

XNOR:




12.

Classic Gradient Descent

$$w_{t+1} = w_t - \eta \frac{\partial J}{\partial w_t}$$

← Stochastic, minibatch or Full depending on how many examples we sum over.

Gradient Descent with momentum

$$w_{t+1} = w_t - \eta \cdot m_t$$

where

$$m_t = \beta m_{t-1} + (1-\beta) \frac{\partial J}{\partial w_t}$$

RMS Prop

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{V_t + \epsilon}} \cdot \frac{\partial J}{\partial w_t}$$

where

$$V_t = \beta V_{t-1} + (1-\beta) \left(\frac{\partial J}{\partial w_t}\right)^2$$

ADAM

$$w_{t+1} = w_t - \eta \frac{m_t}{\sqrt{v_t} + \epsilon}$$

where

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) \frac{\partial J}{\partial w_t}$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) \left(\frac{\partial J}{\partial w_t}\right)^2$$

13.

→ Gradient descent with momentum performs the update based on an exponential weighted moving average of the current & previous gradients rather than just the current gradient.

- Smooths out oscillations as an average of gradients with emphasis on recent ones is a good estimate of the intended direction.

14. [NN lec. 11, 9:11] (The two types of regularization) (besides actually making the network smaller, training with more data, less features, ...)

15)

1- Learning Rate → Momentum Param. → Hidden layer Nodes → Mini-batch Size

→ # Hidden layers → Learning Rate Decay → ADAM Parameters

2- Softmax → Outputs a Probability distribution
→ Only Multi-class classification (output layer)

3- # Hidden layers ↑↑ →
- May be subject to Overfitting
- Takes more time / Slower Convergence
- Network learns more complex functions & features*

4- ReLU recommended for Images

5- Siamese Networks → Match & Compare two Inputs

6- RNN → Has SKIP connections from Output to Input layer

7- Autoencoder → Generate features from Input Images and represent them in compressed low dim. Space

8- Subject to overfitting → More training data

↓ DropOut Regularization

↳ L2 Regularization

9- Time Series = Trend + Seasonality + Random Noise

10- 1 Hidden layer will be enough for any boolean expression (can be written as Sum of Products)
OR (Output) AND (Hidden)