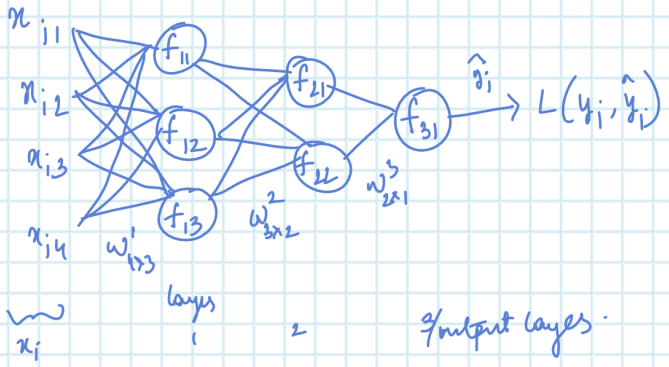
1.8 Traing an MLP (Chain Rule) :-

$D = \{x_i, y_i\}$ $x_i \in \mathbb{R}^4$, $y_i \in \mathbb{R}$ (example), loss is squared loss in regression.



x_i

→ By the end we need to determine $w^1_{4\times3}$, $w^2_{3\times2}$, $w^3_{2\times1}$ = 20 weights (this is the training in MLP)

step ① :-

$$L = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 + \text{reg}$$ → L_2 reg equivalent would be $\sum_{i,j,k} \left(\|w^k_{ij}\|\right)^2$

squared loss

L_1 reg equivalent would be $\sum_{i,j,k} \left(|w^k_{ij}|\right)$

$$L_i = (y_i - \hat{y}_i)^2$$

$$L = \sum_{i=1}^{n} L_i + \text{reg}$$

step ② :- optimization problem :-

$$\min_{w^1, w^2, w^3} L$$ → squared loss & regularization.

$$= \min_{w^k_{ij}} L$$

step ③ :- SGD (or) GD

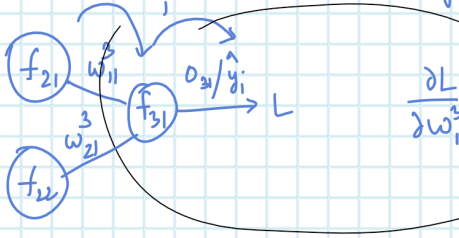
(a) initialization of w^k_{ij} randomly (there are lot of techniques to initialize these)

(b) $(w^k_{ij})_{new} = (w^k_{ij})_{old} - \eta \left(\frac{\partial L}{\partial w^k_{ij}}\right)$ → most important part

η → learing rate

(c) perform upates till convergence.

ex :- we are trying to compute $\frac{\partial L}{\partial w^3_{11}}$. ($w^3_{11}$ directly impacts $O_{31}/\hat{y}_i$ which impacts L)

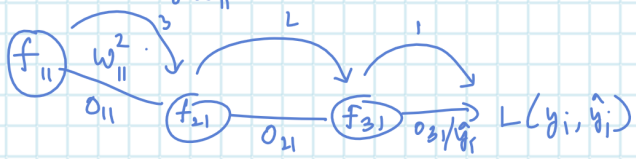


$$\frac{\partial L}{\partial w^3_{11}} = \frac{\partial L}{\partial O_{31}} \cdot \frac{\partial O_{31}}{\partial w^3_{11}}$$

$$\frac{\partial L}{\partial w^3_{21}} = \frac{\partial L}{\partial O_{31}} \cdot \frac{\partial O_{31}}{\partial w^3_{21}}$$

chain rule. w_3 is tuned now.

→ Now we calculate $\frac{\partial L}{\partial w_{11}^2}$. w_{11}^2 impacts O_{11}, O_{21} & $O_{31}/\hat{y}_i$

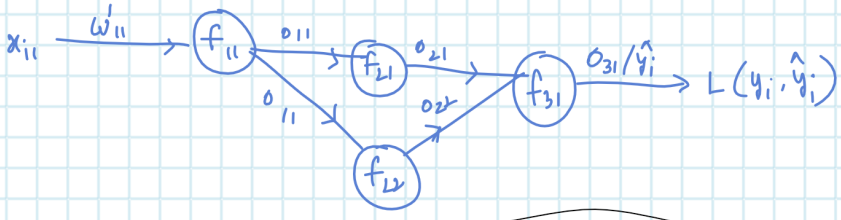


$$\Rightarrow \frac{\partial L}{\partial w_{11}^2} = \frac{\partial L}{\partial O_{31}} \cdot \frac{\partial O_{31}}{\partial O_{21}} \cdot \frac{\partial O_{21}}{\partial w_{11}^2}$$

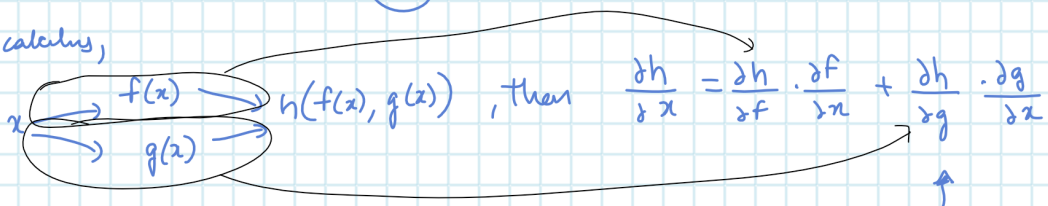
Similarly

$$\frac{\partial L}{\partial w_{21}^2} = \frac{\partial L}{\partial O_{31}} \cdot \frac{\partial O_{31}}{\partial O_{21}} \cdot \frac{\partial O_{21}}{\partial w_{21}^2}$$

→ Calculating $\frac{\partial L}{\partial w_{11}^1}$



In calculus,



, then $\frac{\partial h}{\partial x} = \frac{\partial h}{\partial f} \cdot \frac{\partial f}{\partial x} + \frac{\partial h}{\partial g} \cdot \frac{\partial g}{\partial x}$

We have

$x \rightarrow f(x) \rightarrow h(f(x), g(x)) \rightarrow K(x)$

$x \rightarrow g(x) \rightarrow$

$$\frac{\partial K}{\partial x} = \frac{\partial K}{\partial h}\left(\frac{\partial h}{\partial x}\right)$$

$$\Rightarrow \frac{\partial K}{\partial x} = \frac{\partial h}{\partial K}\left[\frac{\partial h}{\partial f} \cdot \frac{\partial f}{\partial x} + \frac{\partial h}{\partial g} \cdot \frac{\partial g}{\partial x}\right]$$

$$\rightarrow \frac{\partial L}{\partial w_{11}^1} = \frac{\partial L}{\partial O_{31}} \cdot \frac{\partial O_{31}}{\partial w_{11}^1}$$

$$= \frac{\partial L}{\partial O_{31}} * \left[\frac{\partial O_{31}}{\partial O_{21}} \cdot \frac{\partial O_{21}}{\partial O_{11}} \cdot \frac{\partial O_{11}}{\partial w_{11}^1} + \frac{\partial O_{31}}{\partial O_{22}} \cdot \frac{\partial O_{22}}{\partial O_{11}} \cdot \frac{\partial O_{11}}{\partial w_{11}^1}\right]$$

Training an MLP: Memoization (1.9):-

→ As seen in Dynamic Programming

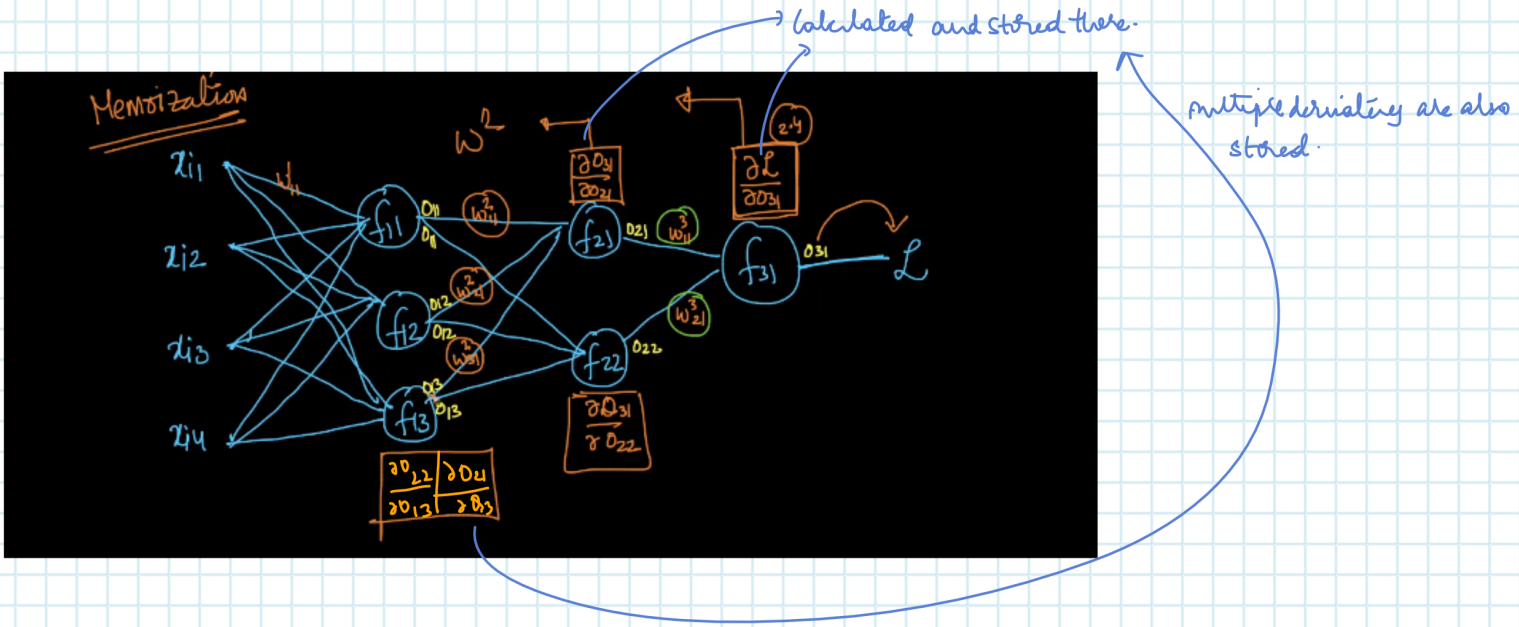
→ $\frac{\partial L}{\partial w_{11}^3} = \frac{\partial L}{\partial O_{31}} \cdot \frac{\partial O_{31}}{\partial w_{11}^3}$

$\frac{\partial L}{\partial w_{21}^3} = \frac{\partial L}{\partial O_{31}} \cdot \frac{\partial O_{31}}{\partial w_{21}^3}$

→ same - core idea is to compute and reuse.

→ if there is any operations performed repeatedly, compute it once & reuse it.

→ takes slightly more memory though.



→ Chain Rule + Memoization = Backpropagation, with some optimizations.

Backpropagation (1.10):-

Given $D = \{x_i, y_i\}$

step ① :- Initialize w_{ij}^{k}

step ② :- for each x_i in D, pass x_i forward through the network. The end result would be $\hat{y}_i \rightarrow L(y_i, \hat{y}_i)$
This step is called forward propagation.
Then compute the $L(\hat{y}_i, y_i)$
Compute all derivatives using chain rule and memorization.
Update weights from end of the network to the start → $(w_{11}^{3})_{new} = (w_{11}^{3})_{old} - \eta\left(\frac{dL}{dw_{11}^{3}}\right)$

Backward propagation. We are using the loss to update the weights.

$$\vdots$$
$$(w_{11}^{2})_{new} = \dots$$
$$\vdots$$
$$(w_{11}^{1})_{new} = \dots$$

step ③ :- Repeat step ② till convergence. ie, $(w_{ij}^{k})_{new} \approx (w_{ij}^{k})_{old}$

epoch = Passed all the datapoints to the neural network once. We do multiple epochs in Real World.

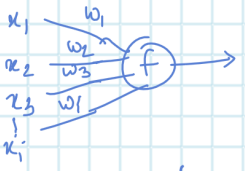
Backpropagation = Multi epoch training methodology that uses chain rule & memorization to update weights.

→ Backpropagation works iff activation functions are differentiable. If it is easily differentiable, then training of NN can be sped up.

→ Instead of sending single point in backprop we can send a subset $S \subseteq D$, (Similar to mini batch SGD)
$|S|$ = mini batch size - (typically powers of 2 like 64, 128, ...)
keeping all datapoints in RAM & computing gradients is hard.
∴ Mini batch SGD is most popular.

Activation functions (1.11) :-

→ In 1980 & 1990's, sigmoid and tanh were the most popular.



let $z = \sum_{i=1}^{n} w_i x_i = w^T x$

if f is sigmoid, then $f(z) = \frac{e^z}{1+e^z} = \frac{1}{1+e^{-z}}$ (This is also used in LR squashing)

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

$$\frac{d\sigma}{dz} = \sigma(z)(1-\sigma(z))$$

Derivative of $\sigma(z)$ is expressed in terms of $\sigma(z)$.
⇒ Can be helpful in backpropagation.

→ $a = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$

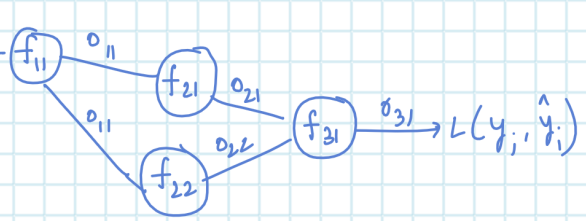
$$\frac{d\tanh}{dz} = 1 - \tanh^2(z)$$

→ ReLU are more popular activation functions these days.

Vanishing Gradient Problem :-

→ Most imp reason NN lost hype back in 90's

→



$$\frac{\partial L}{\partial w'_{11}} = \frac{\partial L}{\partial o_{31}} \left[\frac{\partial o_{31}}{\partial o_{21}} \cdot \frac{\partial o_{21}}{\partial o_{11}} \cdot \frac{\partial o_{11}}{\partial w'_{11}} + \frac{\partial o_{31}}{\partial o_{22}} \cdot \frac{\partial o_{22}}{\partial o_{11}} \cdot \frac{\partial o_{11}}{\partial w'_{11}} \right]$$

$0 \le \quad < 1$

$$\frac{\partial o_{31}}{\partial o_{21}} = \frac{\partial f_{31}}{\partial o_{21}} \quad \text{since } o_{31} = f(\text{something})$$

All f's are sigmoids and lie b/w 0 & 1 and the derivatives are being multiplied. Results in much smaller value.

As a result, in

$$(w'_{11})_{new} = (w'_{11})_{old} - \eta\left(\frac{\partial L}{\partial w'_{11}}\right)$$

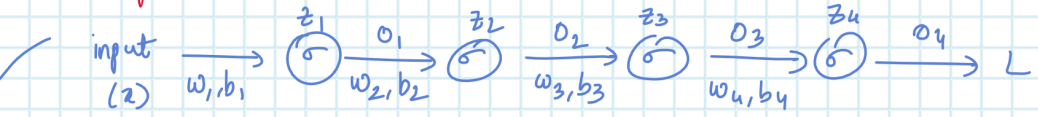
$(w'_{11})_{new}$ & $(w'_{11})_{old}$ will be very close.

This problem is called vanishing gradient.

$\frac{\partial L}{\partial w_{ij}^{k}}$ is small → vanishing gradient

$\frac{\partial L}{\partial w_{ij}^{k}}$ is very large → exploding gradient (happens when activation function's result > 1)

Exploding Gradient using Sigmoid function:-



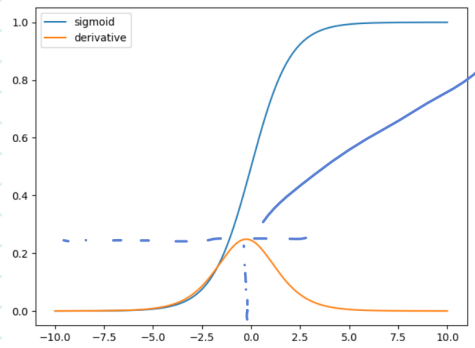
$o_1 = \sigma(z_1)$ $\quad o_2 = \sigma(z_2)$ $\quad o_3 = \sigma(z_3)$

$z_1 = w_1 x + b_1$ $\quad z_2 = w_2 o_1 + b_2$ $\quad z_3 = w_3 o_2 + b_2$

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{o_4} \cdot \frac{\partial o_4}{\partial o_3} \cdot \frac{\partial o_3}{\partial o_2} \cdot \frac{\partial o_2}{\partial o_1} \cdot \frac{\partial o_1}{\partial w_1}$$

if activation func such that each > 1, it leads to exploding gradient.

But sigmoid max is (0.25)



take $\frac{\partial o_2}{\partial o_1} = \frac{\partial o_2}{\partial z_2} \cdot \frac{\partial z_2}{\partial o_1}$

$= \frac{\partial \sigma(z)}{\partial z_2} \cdot \frac{\partial (w_2 o_1 + b_2)}{\partial o_1}$

$= \sigma'(z_2) * w_2$ where $\sigma'(z) = \frac{\partial \sigma(z)}{\partial z}$

w_2: weights are initialized randomly. If it is a large enough value, it leads to exploding gradient.

Bias Variance Tradeoff (1.13):-

→ # layers ↑ → more weights → higher chance of overfit → high variance.

1 layer → aka log reg → higher chance of underfit → high bias.

→ MLP generally tends to have high variance since no one chooses one layer.

→ to reduce variance we use regularization term along with loss in optimization

$$L = \sum_{i=1}^{n} \text{loss}(y, \hat{y}_i) + \sum_{i,j,k} (w_{ij}^k)^2$$

$$= \sum_{i=1}^{n} \text{loss} + \lambda L_2$$

larger $\lambda \Rightarrow$ lesser overfit

using L_1 reg leads to sparsity (connections b/w neurons of various layers dont exist)