5 Multilayer Perceptron

Deep Network - Simplest version

limitations of linear models is that linear assump implies monotonicity asum

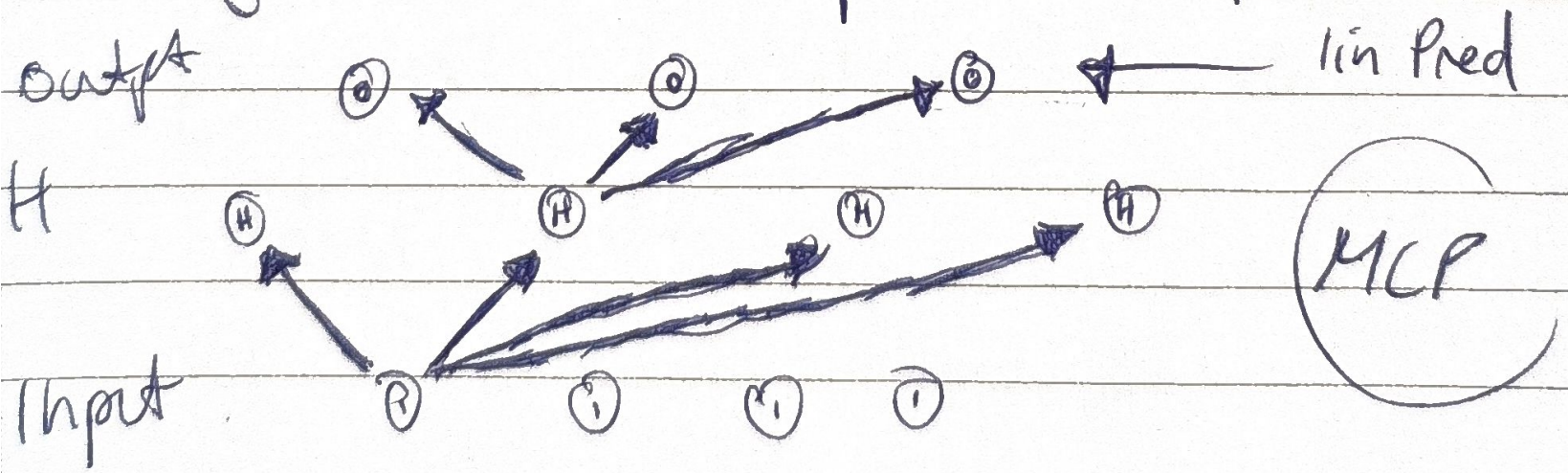
= increase in x must always increase output, & visa for decrease

- often this is only true upto a point
- or varys at diff points in the data

NN uses a hidden layer to learn joint relationship between dimensions

Hidden layers →

= layers between inputs & outputs

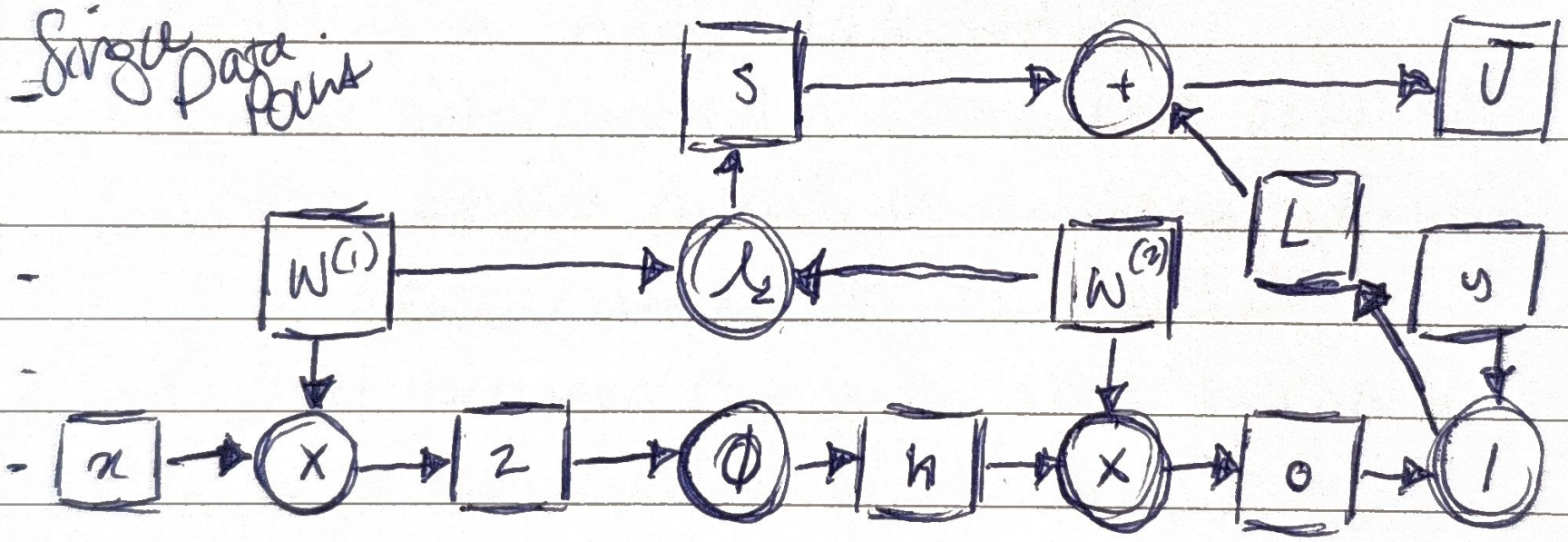


Forward Pass / Propigation

Note this section is "incomplete" as there are later steps (Backpropigation) that come back and up date the model

that said, forward pass calculates the output of a network given its inputs, layers, weights & transforms

Single Data Point



English Explanation or other side

Forward Prop – written Explanation (3)

- We have a data point x
- We want to determin the weights $W^{(1)}$ for a linear model given $z = W^{(1)}x$ (no bias)
- These weights are obtained by minimizing an objective function using gradient descent
- using the weight/Model we calc an intermediary point z $\quad x \longrightarrow z$
- These new data points are then non-linearly transformed using $\phi(z)$
- Resulting a new dataset ϕ
- An new linear model is constructed $o = W^{(2)}H$ w/ the weights calced w/ the same optimizer
- An a new Dataset o is created
- The diff between o & y is used to create the loss ~~func~~ value $L = l(o, y)$
- To prevent overfit, a reg term is calced which penalized large weights $S = \frac{\lambda}{2}\|W^{(1)}\|_F^2 + \|W^{(2)}\|_F^2$
- final objective function given by $J = L + S$
- later we will minimize/optimize this (J) using Backpropigation

forward Prop Equations

intermediate value = $z = W^{(1)} x$

hidden activ funcval = $h = \phi(z)$

output layer value = $o = W^{(2)} h$

loss func = $l = l(o, y)$

reg Value = $s = \frac{\lambda}{2}(\|W^1\|_F^2 + \|W_F^1\|)$

Where λ = learning hyperparta

Objective func = $L + S$

Backpropagation

= the method of calc the gradient of neural network parameters

Traverses the network in reverse order, from output layer to input layer, according to the calculus chain rule

Algo store intermiade varcabls partial Derivatives while calcing the gradient w/ respect to some paras

- Assum funcs: $Y = f(x)$ & $Z = g(Y)$
- where input/outputs ~~are~~ (x, Y, Z) are tensors

using chain rules, compute Derivatie of Z (Finout) w/ respect to X (first input)

$$\underbrace{Z}_{z} = g(\underbrace{f(\overbrace{x}^{x})}_{y}) \quad \frac{\partial Z}{\partial X} = \text{Prod}\left(\frac{\partial Z}{\partial Y}, \frac{\partial Y}{\partial X}\right)$$

Recall, chain rules
- if Y depends on u which depends on x
- Chain rules lets us calc Derive dy w/ respect to x
- $dy/dx = (dy/du) * (du/dx)$
- how small change in x affects y

With vectors, we use partial derivatives to capture how each component of one vector affects another. Usem MatM to comb

Chain rule for tensors:
- tensors are MultiDimension arrays
  Matrix = 2-D tensor
- instead of MM uses tensor contrael to comb PD's
- Notation becomes v complex but concept of the chain rule remains

Prod(·)encapsulates the notation we will use to chain calcs

Recall the paras of a simple Network are given by $W^{(1)}$ & $W^{(2)}$

Obj of Backpropagation is to calculate the gradients $\partial J / \partial W^{(1)}$ & $\partial J / \partial W^{(2)}$

To accomplish this, we apply chain rule and calculate

In turn, the gradient of each intermed variable & parameter

the order of calc is opposite to forward prop

Step 1 calc grads of $J = L + S$ w/ respect to loss func & reg term

$$\frac{\partial J}{\partial L} = 1 \quad \& \quad \frac{\partial J}{\partial S} = 1$$

$X \rightarrow h \rightarrow O$ (2)

Step 2 Compute gradient of obj func w/ respect to variable of the output layer O

$$\partial J / \partial o = P_{roduct}\left(\frac{\partial J}{\partial L}, \frac{\partial L}{\partial o}\right)$$

cached in Step 1 = 1

Step 3 calc gradient of reg term w/ respect to paras (weights)

Note; this step is done for comp^{n} easy & reusability. Keras doesn't have X (data) term so easy to use

$$\frac{\partial S}{\partial w^{(1)}} = \lambda w^{(1)} \qquad \frac{\partial S}{\partial w^{(2)}} = \lambda w^{(2)}$$

↓ used for h

↓ used for O

$X \rightarrow h \rightarrow O$

$h = f(x)$ $o = g(h)$ $L = l(o, y)$ $s = w^1 \; w^2$ (4)

(2) (3)

$J = L + S$

(1)

Step 4 calc gradient of object w/ respect to $W^{(2)}$ (or whatever n is closest to output o)

$$\frac{\partial J}{\partial W^{(2)}} = \text{Prod}\left(\frac{\partial J}{\partial o}, \frac{\partial o}{\partial W^{(2)}}\right) + \text{Prod}\left(\frac{\partial J}{\partial s}, \frac{\partial s}{\partial W^2}\right)$$

$$= \frac{\partial J}{\partial o} h^T + \lambda W^{(2)}$$

recall W^2 connected direct to $o = g(n)$ to is easy to work w/

Step 5 Obtain gradient w/ resp to $W^{(1)}$ need to continue down network as $W^{(1)}_{ij}$ connected to h // Note we need $\frac{\partial J}{\partial h}$ first

Gradient w/ respect to hidden layer

$\frac{\partial J}{\partial h}$ two parts = how much J change w/ res to o

how much o chgs w/ resp to h

$$\frac{\partial J}{\partial h} = \text{Prod}\left(\frac{\partial J}{\partial o}, \frac{\partial o}{\partial h}\right) = W^{(2)} + \frac{\partial J}{\partial o}$$

Step 6 need to get from h to z which was the activ func

Transform was element wise - hence need to back element wise use ⊙

$$\frac{\partial U}{\partial h} = \text{Prod}\left(\frac{\partial J}{\partial h}, \frac{\partial h}{\partial z}\right) = \frac{\partial J}{\partial h} \odot \phi'(z)$$

Step 7 now we can finally obtain gradient $\frac{\partial U}{\partial W^1}$

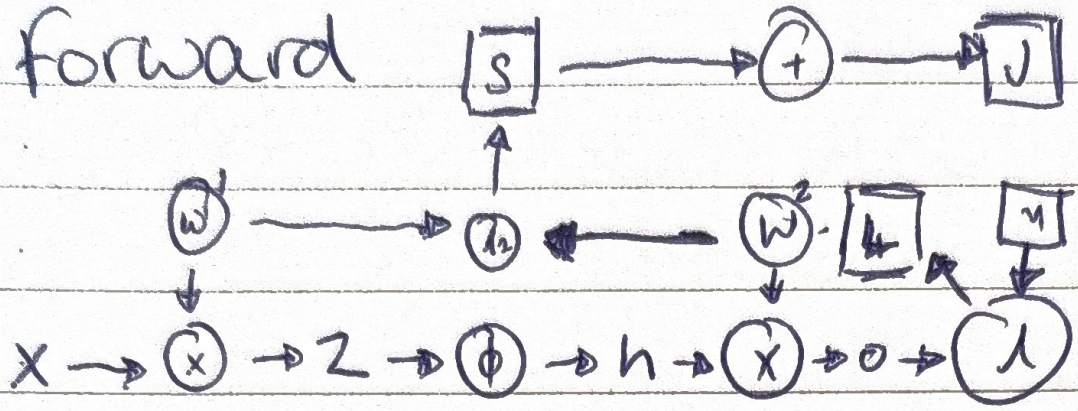
$$\frac{\partial J}{\partial W^{(1)}} = \text{Prod}\left(\frac{\partial U}{\partial z}, \frac{\partial z}{\partial W^{(1)}}\right) + \left(\frac{\partial U}{\partial S}, \frac{\partial S}{\partial W^{(1)}}\right)$$

$$= \frac{\partial J}{\partial z} x^T + \lambda W^{(1)}$$

Graphs

x n o

forward



Backward

1. $J = L + S$ $\frac{\partial J}{\partial}$ = gradient of deg w/ RS loss + Reg
2. $\frac{\partial J}{\partial o}$ – Diff loss func to get o
3. Reg term diff weights
4. $\frac{\partial J}{\partial w^2}$ Diff w^2 term
5. $\frac{\partial J}{\partial h}$ Diff from o to h
6. Undo activ func for $\frac{\partial J}{\partial z}$
7. $\frac{\partial J}{\partial w^1}$ final Diff w^1 term starts from 2

Training NN

When training, Forward & Back-ward propagation depend on eachoth

FP traverses graph & computes vars
BP Picks up order & optimizes weights Based on the opt obj func

Note, there is a very crucial point, missed in the previous note

forward propagation makes no optimization of the layers weights. these are set randomly

BP does all the optimization FP is merely for calcing the layers outputs, L2, loss, grads & obj based on the current weights

f = calc

B = learn

NN is an iterative process. the FP -> BP -> FP takes place many times.

each iteration is called an epoch

within the epochs the train set is often split into batches & the FP/BP steps performed for each batch

epoches decided by user (hyperpara)