4-11-2021 Tuesday

week # 11 lecture # 14

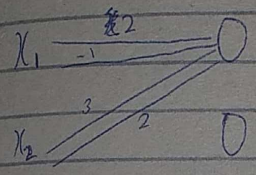
Data Processing.

Activation Function:

① Maxout function:

$$\max(w_1 x_1 + b_1, w_2 x_2 + b_2)$$

eg

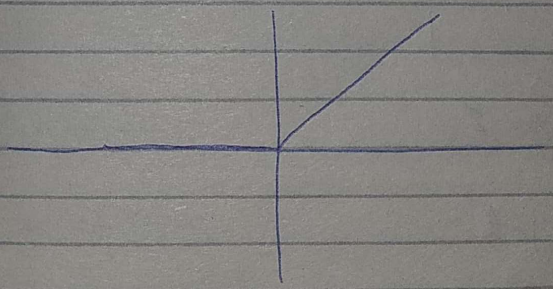


for each input there are two weights

a) weights are doubled, memory is required more

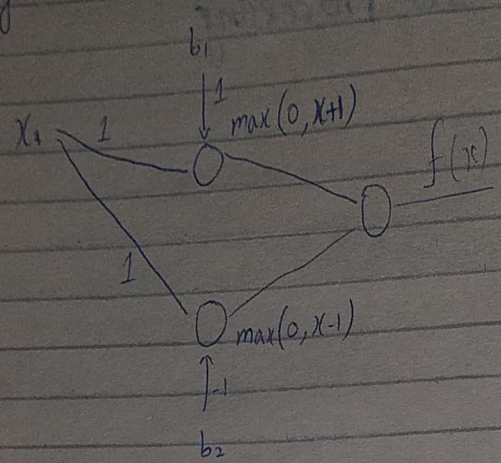
Training time is increased.

3) ReLU Function:



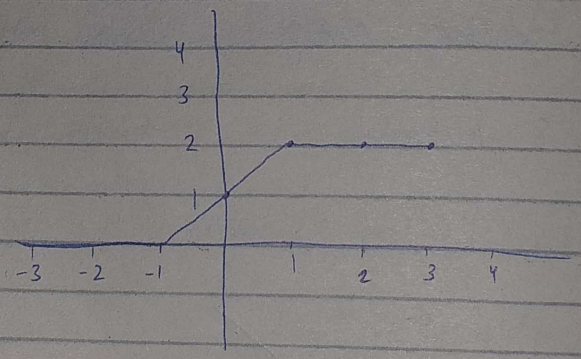
how does this introduce non-linearity in our NN?

e.g



So, $f(x) = \max(0, x+1) - \max(0, x-1)$

Now if we plot this eq graph,



if $x = 1$, $y = 2$

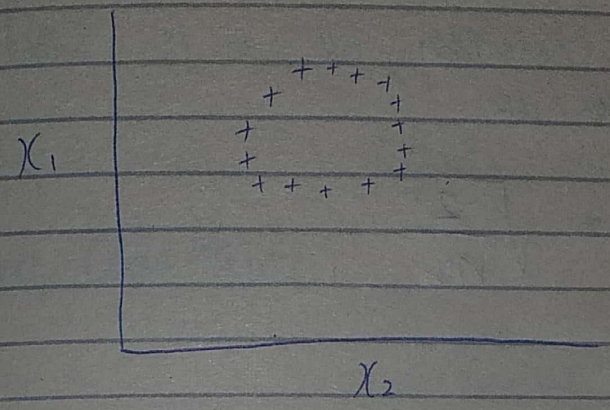
if $x = 0$, $y = 1$

if $x = 2$, $y = 2$

if $x=-1$, $y=0$

if $x=-2$, $y=0$

Another example:



X_1 & X_2 2 features

for this type of Dataset, a circle equation meaning not linear function would be better for this Dataset that's why we use ReLU function. ReLU supports more complex function learning.

weights Initialization:

2) Glorot Initialization:

Instead of dividing by $\sqrt{n}$ we do.

$W = np.random.randn(n, D) * \sqrt{\frac{2}{n_{in} + n_{out}}}$

n_{in} = input no. of neurons

n_{out} = output no. of neurons

Researchers tell us this below one

For ReLU, we use

$$* \sqrt{\frac{2}{n_{in}}}$$

or divide by

$$/ \sqrt{\frac{n_{in}}{2}} \qquad n_{in} = \text{no. of inputs}$$

above & below are same just rearranged

To overcome overfitting:-

① we did L_1 & L_2 Regularization

So our loss formula was

½ added bcz to cancel 2 from gradients calculation

$$L = \sum_{i=1}^{n} L_i + \boxed{\frac{1}{2}} \lambda \Sigma |w^2|$$

L_2 Regularization weights are diffused

$$L = \sum_{i=1}^{n} L_i + \lambda \Sigma |w|$$

L_1 Regularization weights are sparse

If we apply both, then it is called Elastic Net Regularization.

$$L = \sum_{i=1}^{n} L_i + \lambda_1 \{\Sigma |w| + \frac{1}{2} \lambda_2 \Sigma |w^2|$$

Sometimes people set different lamba's values for different weights.

Use L_2 Regularization & Dataset should be large.

② Max Norm Constraint:-

$|W| < C$, C = some Constant

when $|W|$ becomes greater than C, we set that weight value to C

usually its value is 3.0, 2.0

we are penalizing our weight values.

③ Increase Dataset size to overcome overfitting.

④ Drop out:-

To ensure that our whole NN neurons are contributing to our training, we don't have neurons that are contributing very little & others are contributing a lot.

x_1

x_2

x_3

x_4

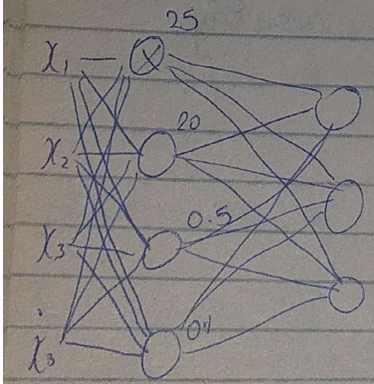
dw_1

$W = W - \alpha\, dw_1$

Mini batch gradient Descent

So, concept is for different batches we disable different neurons, we randomly disable them for one batch then ~~the~~ Inactive neurons would send 0 output to other neurons in forward pass.

Then what it does is other neurons which had smaller contribution before now ~~get~~ will be updated to their contribute more. & weights are updated more.



Its Code:

```
def forward(X):
    prob = probability = 0.5
    h1 = np.maximum(0, W1.dot(X))
    d = np.random.rand(h1.shape) < prob
    h1 = h1 * d
    h2 = np.maximum(0, W2.dot(h1))
    h3 = np.maximum(0, W3.dot(h2))
    out =
```

So if rand have returned value

1	0.3	< 0.5	✓
1	0.4	< 0.5	✓
0	0.7	< 0.5	×
0	0.8	< 0.5	×

Mask = d

So first 2 neurons will be active, last 2 neurons will be inactive & their prev output was Multiplied by Mask where there is 1 it will be passed forward, here where there is 0 it will not be passed forward.

h1 = h1 * d

h2 = np.maximum(0, W2.dot(h1))

At test time, no neurons would Inactive, output Preprocessing

keep_Probability was 0.5

is required because in training output was 50% bcz 50% neurons was Inactive,

So in test time we also want output to be same as in train time. So In test time we multiply output of neural Network layers with keep_Probability then our output would be 50% If all neurons were active, then our output would be doubled.

```
h1 = np.maximum(0, w1.dot(X)) * keep_Probability
```

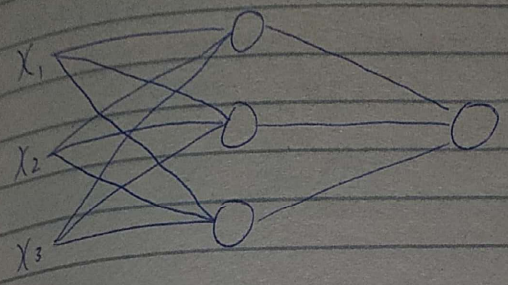
Problem:-

When we are multiplying Prob with every layer output, then our test time increases. if our N.N is big

Solution:- 5) Inverted Dropout.

In train time, we divide outputs of layers with Probability. So we don't have to do anything in test time.

eg:-



Dropout code:-

```
def Forward(X):                    //Test time code
    keep_prob = 0.5
    h1 = np.maximum(0, W1.dot(X))
    d1 = np.random.rand(h1.shape) < keep_prob
    h1 = h1 * d1
    out = np.maximum(0, W2.dot(h1))
    return out

def Test(X):
    keep_prob = 0.5
```

h1 = np.maximum(0, W1.dot(X)) * keep_probability

out = np.maximum(0, W2.dot(h1)) * keep_probability

return out

Inverted Dropout Code:

def Test(X):

h1 = np.maximum(0, W1.dot(X))

out = np.maximum(0, W2.dot(h1))

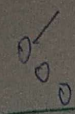
return out

def Forward(X):

keep_prob = 0.5

h1 = np.maximum(0, W1.dot(X))

d1 = np.random.rand(h1.shape) < keep_prob



h1 = h1 * (d1 / keep_prob)

out = np.maximum(0, W2.dot(h1))

return out

Libraries for ML/DL:-

1. Sklearn
2. Pytorch
3. Keras
4. Tensorflow

before next class check these libraries.