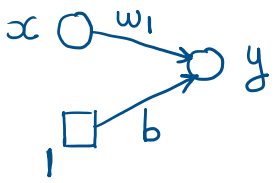
Gradient Descent

Friday, July 21, 2023 3:18 PM

X	y
0.553482	6.660447
0.484087	6.452262
0.047556	5.142669
0.677248	7.031744
0.997598	7.992793
0.367771	6.103314
0.532385	6.597154
0.293828	5.881483
0.282263	5.846789
0.258933	5.776799

$y = b + w_1 x$

b, w_1 : weights



$w_1 = 0.1$

$b = -0.5$

e.g $f(x,y) = x^2 + y^3$

$\frac{\partial f}{\partial x} = 2x$ $\quad \frac{\partial y^3}{\partial x} = 0$

$\frac{\partial f}{\partial y} = 3y^2$ $\quad \frac{dy}{dx}$: rate of change of y w.r.t x

Steps :-

1) Weights are initialized at random bet^n $(-1, 1)$

2) The predictions are calculated & error is measured & so is the error function $J()$

3) If errfunction < tolerance $^{(0.00)}$ then existing weights are taken as best weights o.w. the weights are updated as :-

$$\text{new } w_1 = \text{old}_{w_1} - \frac{\partial J}{\partial w_1} \eta$$

$$\text{new } b = \text{old}_{b} - \frac{\partial J}{\partial b} \eta$$

η : learning rate $\in (0,1)$

4) Continue (2) & (3) till err funct < tolerance

X_1 X_2

Fat	Salt	Acceptance
0.2	0.9	like
0.1	0.1	dislike
0.2	0.4	dislike
0.2	0.5	dislike
0.4	0.5	like
0.3	0.8	like

$w_1 = 0.1$

$b = -0.5$

$w_2 = 0.4$

Steps :-

1) Weights are initialized at random bet^n $(-1, 1)$

2) The predictions are calculated & error is measured & so is the error function $J()$ log loss

3) If errfunction < tolerance $^{(0.00)}$ then existing weights are taken as best weights o.w. the weights are updated as :-

$$\text{new } w_1 = \text{old}_{w_1} - \frac{\partial J}{\partial w_1} \eta$$

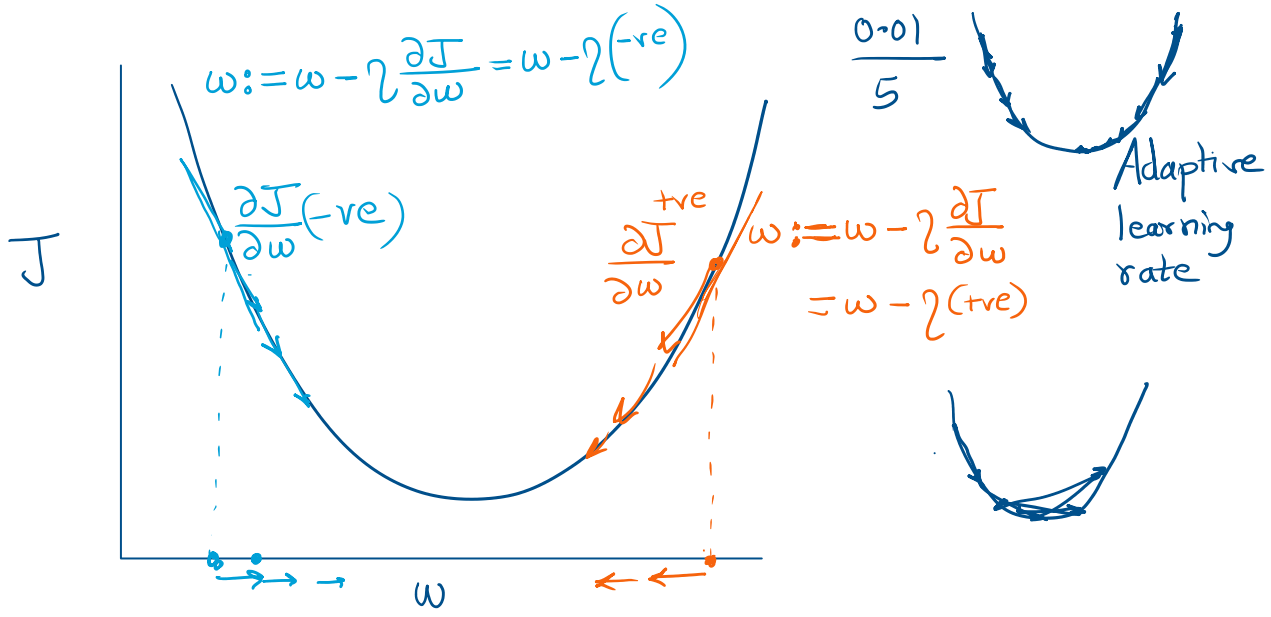
$$\text{new } b = \text{old}_{b} - \frac{\partial J}{\partial b} \eta$$

$b + w_1 x_1 + w_2 x_2$

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

$$b = b - \eta \frac{\partial J}{\partial b}$$

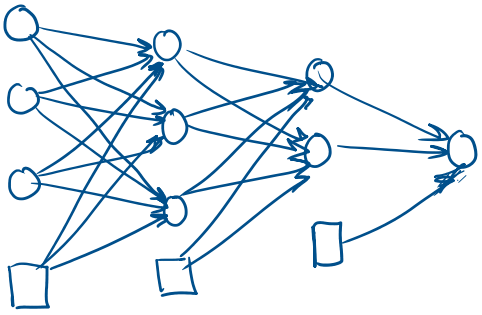
4) Continue (2) & (3) till err funct < tolerance



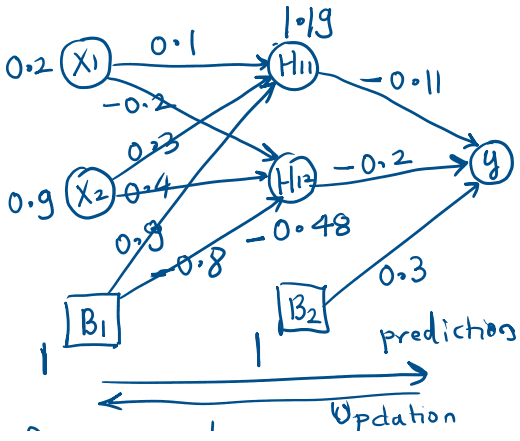
MLP / Fully connected layers

$X_1\ X_2\ X_3$

y



Fat	Salt	Acceptance	
0.2	0.9	like	1
0.1	0.1	dislike	0
0.2	0.4	dislike	0
0.2	0.5	dislike	0
0.4	0.5	like	1
0.3	0.8	like	1

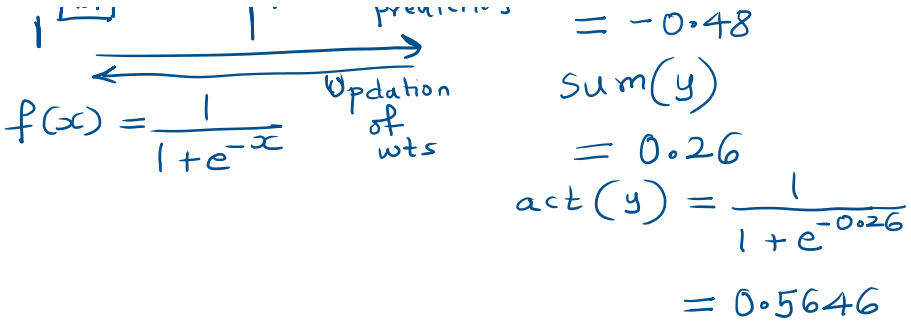


Sum(H_{11})
$= 0.2(0.1) + 0.9(0.3) + 1(0.9)$
$= 1.19$

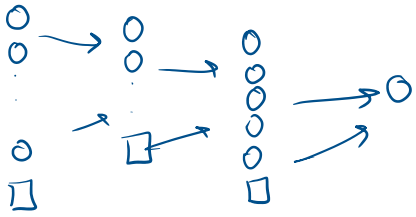
Sum(H_{12})
$= 0.2(-0.2) + 0.9(0.4) + 1(-0.8)$
$= -0.48$

Sum(y)

0.4	0.5	like	1
0.3	0.8	like	1



mlp = MLPClassifier(hidden_layer_sizes=(15,5),
random_state=23)



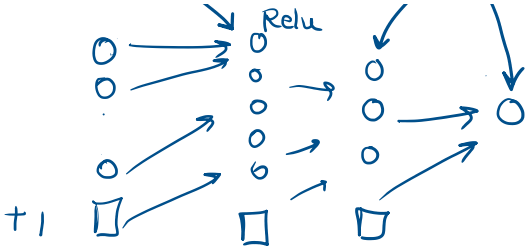
```
def train(model, inputs, outputs, learning_rate):
  current_loss = loss(model(inputs),outputs)
  current_loss.backward() # Calculates the gradient
  model.w = torch.sub(input=w,other=w.grad,alpha=learning_rate)
  model.b = torch.sub(input=b,other=b.grad,alpha=learning_rate)
  return current_loss;
```

Prediction
MSError
$\frac{\partial J}{\partial w}$ $\frac{\partial J}{\partial b}$
$w := w - \eta \frac{\partial J}{\partial w}$
$b := b - \eta \frac{\partial J}{\partial b}$

```
model = Model()

list_w = []
list_b = []
losses = []
for epoch in range(25):
  list_w.append(model.w.detach().numpy())
  list_b.append(model.b.detach().numpy())
  curr_loss = train(model, x, y, learning_rate=0.1)
  losses.append(curr_loss)
  print('Epoch %2d: w=%1.2f b=%1.2f, loss=%2.5f' % (epoch,model.w.detach().numpy(),model.b.detach().numpy(),curr_los
```

```
torch.manual_seed(23)
# Create a model
model = nn.Sequential(nn.Linear(in_features=X_scl_trn.shape[1], out_features=5),
                      nn.ReLU(),
                      nn.Linear(5, 3),
                      nn.ReLU(),
                      nn.Linear(3,1),
                      nn.Sigmoid())
```




```
for epoch in np.arange(0,1000):
        # Forward pass: Compute predicted y by passing x to the model
        y_pred_prob = model(X_torch.float()) ⟶ Prediction

        # Compute and print loss
        loss = criterion(y_pred_prob, y_torch.float())
        if epoch%100 == 0:
           print('epoch: ', epoch+1,' loss: ', loss.item())

        # Zero gradients, perform a backward pass, and update the weights.
        optimizer.zero_grad() ⟶

        # perform a backward pass (backpropagation)
        loss.backward()

        # Update the parameters
        optimizer.step()
#print('epoch: ', epoch+1,' loss: ', loss.item())
```