PadhAI Week 3: Sigmoid Neuron & Cross Entropy

by Manick Vennimalai

3.6: Sigmoid Neuron and Cross Entropy

3.6.1: Sigmoid Neuron and Cross Entropy

How does it all tie up to the Sigmoid Neuron

1. Consider the Example:
   a. A signboard with the text **Mumbai**
   b. A random variable X which maps the signboard to: Text, No-Text
   c. The distributions are as follows

X	**y (We don't know initially)**	**ŷ (Predicted using sigmoid)**
T	1	0.7
NT	0	0.3

   d. Previously, we were using **Squared-error Loss** = $\Sigma_i(y_i - \hat{y}_i)^2$
   e. Now, we have a better metric, one that is grounded in probability theory (**KL- Divergence**)
   f. KLD(y||ŷ) = $-\Sigma y_i \, log\hat{y}_i + \Sigma y_i \, log y_i$
   g. We aim to minimize loss by KLD with respect to the parameters w, b
   h. From KLD equation, we can see that y_i doesn't depend on w, b. So therefore, we are really only trying to minimize the first term, i.e. the cross-entropy
   i. So in practice, we can treat the second term as a constant, and the equation would really be $min(-\Sigma_i y_i log\hat{y}_i) \;\; here\ i \;\in T,\ NT$
   j. $Cross\ Entropy\ Loss = \; -1 * log(0.7) \; - \; 0 * log(0.3)$
   k. The second terms cancels out and we are left with $-log(0.7)$ which is the same as $-log\hat{y} \;\; (for\ the\ true\ case)$
   l. It can be called $-log\ \hat{y}_c$ where c can take the value 0 or 1 which correspond to NT and T

PadhAI Week 3: Sigmoid Neuron & Cross Entropy

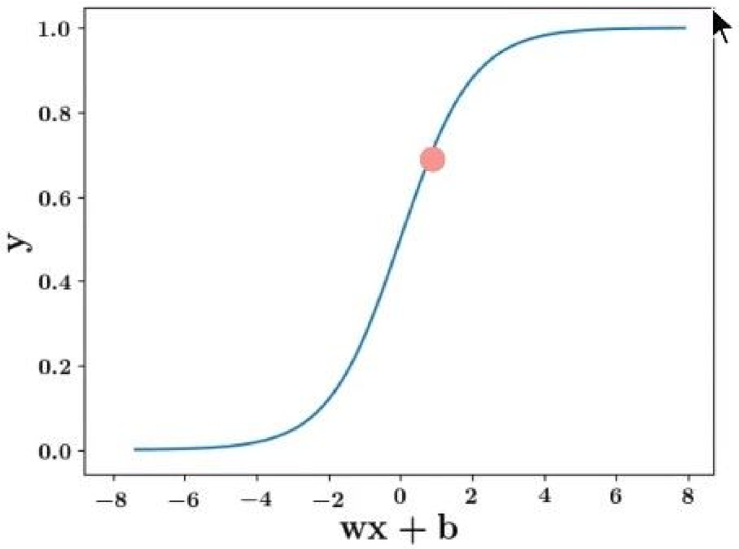
by Manick Vennimalai

3.6.2: Using Cross Entropy With Sigmoid Neuron

What does the cross entropy loss function look like

1. Consider an example in the scope of our final project
2. Look at the following signboard

3. $x = image, \quad y = [0, 1]$ (True distribution, where 1 corresponds to Text)
4. $\hat{y} = \frac{1}{1+e^{(-(w.x+b))}}$
5. This corresponds to $\hat{y}$ =0.7



6. Thus, the predicted distribution is $\tilde{y} = [0.3,\ 0.7]$ (where 0.7 corresponds to Text)
7. The Loss function is $L(\theta) = -\Sigma_i y_i log\tilde{y}_i$ where $i \in \{0, 1\}$
8. $L(\theta) = -((y_o\, log\, \tilde{y}_o) + (y_1 log\, \tilde{y}_1))$
9. $L(\theta) = -((y_o\, log\, (1 - \tilde{y}_1)) + (y_1 log\, \tilde{y}_1))$ (from probability axioms, $y_0 = 1 - y_1$)

10. Consider two examples side by side

Training Data	Image	$L(\theta) = -((y_o \, log\,(1-\tilde{y}_1)) + (y_1 log\,))$	Loss function

y = [0, 1]
ŷ = 0.7
ỹ = [0.3, 0.7]
(Text)
		$L(\theta) = -(0 * log(0.3)) + (1 * log\,(0.7)))$
		$L(\theta) = -log(0.7)$
			$L(\theta) = -log(\hat{y})$
			
			When true output is 1

y = [1, 0]
ŷ = 0.2
ỹ = [0.8, 0.2]
(No-Text)
		$L(\theta) = -(1 * log(0.8)) + (0 * log\,(0.2)))$
		$L(\theta) = -log(0.8)$
			$L(\theta) = -log(1-\hat{y})$
			
			When true output is 0

11. The Loss function can be expressed as follows
    a. $L(\theta) = -log(\hat{y})$ if y = 1
    b. $L(\theta) = -log(1-\hat{y})$ if y = 0
    c. Combining them and removing the if conditions:
    d. $L(\theta) = -[(1-y)log(1-\hat{y}) + ylog(\hat{y})]$
        i. When y = 1, the first term becomes 0
        ii. When y = 0, the second term becomes 0

3.6.3: Learning Algorithm for Cross Entropy Function

What is a more simplified way of writing the cross entropy loss function

1. From the previous step, we have $L(\theta) = -[(1-y)log(1-\hat{y}) + ylog(\hat{y})]$
2. Cross entropy loss only makes sense for classification problems
3. The rest of the procedure is the same as the sigmoid neuron, except we use Cross-Entropy to minimize the loss and choose the best parameters w & b
4. **Initialise:** w, b randomly
5. **Iterate over data**
    a. Compute ŷ
    b. Compute L(w,b)   (Where L is the cross-entropy loss function)
    c. $w_{t+1} = w_t - \eta \Delta w_t$
    d. $b_{t+1} = b_t + \eta \Delta b_t$
    e. Pytorch/Tensorflow have functions to compute $\frac{\delta l}{\delta w}$ *and* $\frac{\delta l}{\delta b}$
6. **Till satisfied**
    a. Number of epochs is reached (ie 1000 passes/epochs)
    b. Continue till Loss < ε (some defined value)
    c. Continue till $Loss(w,b)_{t+1} \approx Loss(w,b)_t$

3.6.4: Computing Partial Derivatives With Cross Entropy Loss

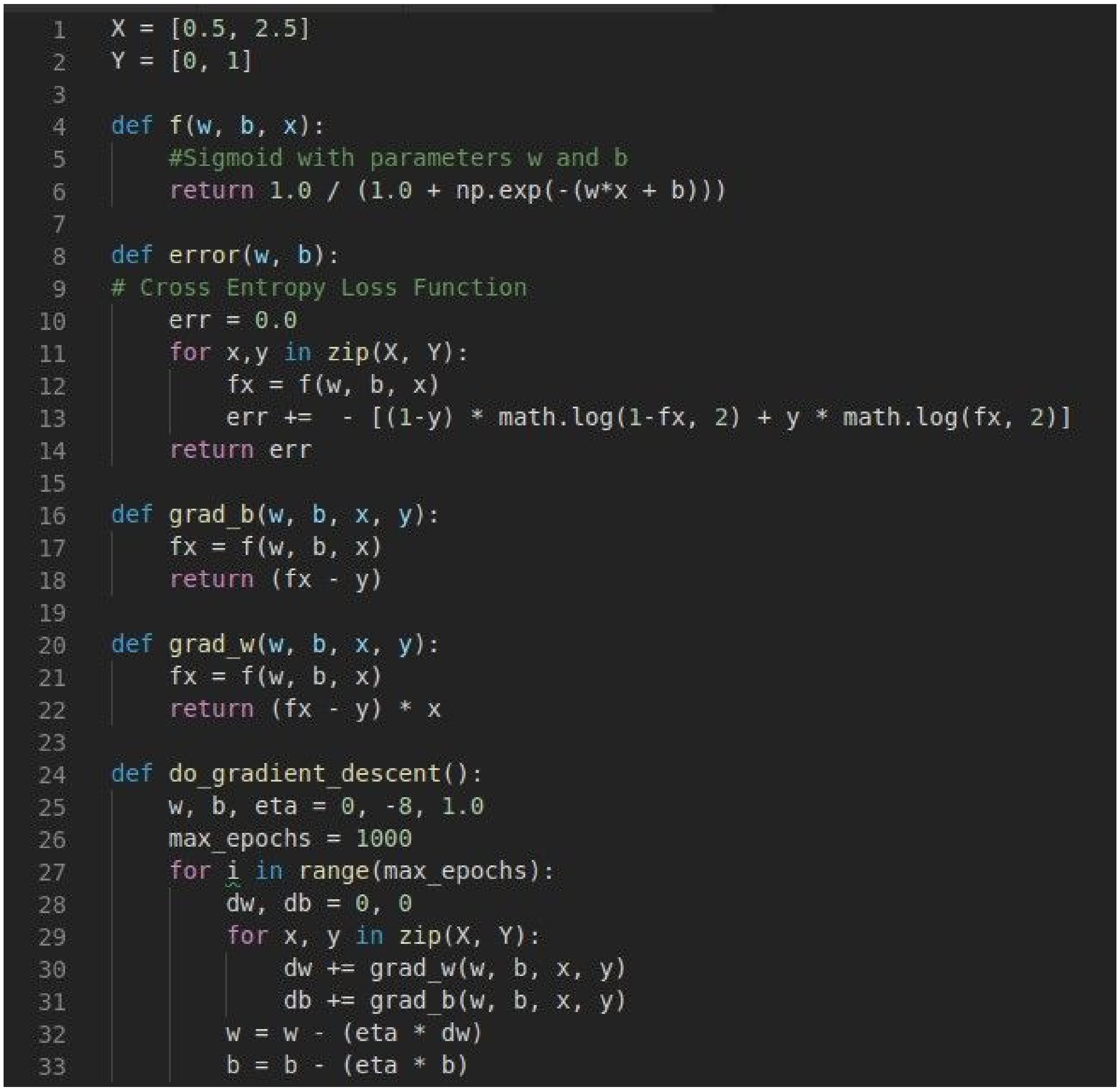
How do we compute Δw and Δb

1. Loss Function $L(\theta) = -[(1-y)log(1-\hat{y}) + ylog(\hat{y})]$
2. Consider Δw for 1 training example
   a. $\Delta w = \frac{\partial L(\theta)}{\partial w} = \frac{\partial L(\theta)}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w}$
   b. The first part: $\frac{\partial L(\theta)}{\partial w} = \frac{\partial}{\partial \hat{y}}\{-(1-y)log(1-\hat{y}) - ylog(\hat{y})\}$
      i. $\frac{\partial L(\theta)}{\partial w} = (-)(-1)\frac{(1-y)}{(1-\hat{y})} - \frac{y}{\hat{y}}$
      ii. $\frac{\partial L(\theta)}{\partial w} = \frac{\hat{y}(1-y) - y(1-\hat{y})}{(1-\hat{y})\hat{y}}$
      iii. a $\frac{\partial L(\theta)}{\partial w} = \frac{\hat{y}-y}{(1-\hat{y})\hat{y}}$ a
   c. The second part: $\frac{\partial \hat{y}}{\partial w} = \frac{\partial}{\partial w} \frac{1}{1+e^{-(wx+b)}}$
      i. This is the exact same as from the Squared Error Loss
      ii. $\frac{\partial}{\partial w}\left(\frac{1}{1+e^{-(wx+b)}}\right)$
      iii. $\frac{-1}{(1+e^{-(wx+b)})^2} \frac{\partial}{\partial w}(e^{-(wx+b)})$
      iv. $\frac{-1}{(1+e^{-(wx+b)})^2} * (e^{-(wx+b)}) \frac{\partial}{\partial w}(-(wx + b))$
      v. $\frac{-1}{(1+e^{-(wx+b)})^2} * (e^{-(wx+b)}) * (-x)$
      vi. $\frac{1}{(1+e^{-(wx+b)})} * \frac{(e^{-(wx+b)})}{(1+e^{-(wx+b)})} * (x)$
      vii. $\hat{y} * (1-\hat{y}) * x$
   d. The final derivative is the first part multiplied with the second part
   e. $\Delta w = (\hat{y} - y) * x$
   f. $|||^{ly}\ \Delta b = (\hat{y} - y)$
3. We then plug the values of Δw and Δb into the the learning algorithm to optimise the parameters w and b.

3.6.5: Code for Cross Entropy Loss Function

What are the changes we need to make in the code

1. Here is the Python code for Gradient Descent with Cross Entropy Loss function



```python
X = [0.5, 2.5]
Y = [0, 1]

def f(w, b, x):
    #Sigmoid with parameters w and b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error(w, b):
# Cross Entropy Loss Function
    err = 0.0
    for x,y in zip(X, Y):
        fx = f(w, b, x)
        err +=  - [(1-y) * math.log(1-fx, 2) + y * math.log(fx, 2)]
    return err

def grad_b(w, b, x, y):
    fx = f(w, b, x)
    return (fx - y)

def grad_w(w, b, x, y):
    fx = f(w, b, x)
    return (fx - y) * x

def do_gradient_descent():
    w, b, eta = 0, -8, 1.0
    max_epochs = 1000
    for i in range(max_epochs):
        dw, db = 0, 0
        for x, y in zip(X, Y):
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - (eta * dw)
        b = b - (eta * b)
```

2. Here, the functions f(w, b, x), grad_b(w,b,x,y) and grad_w(w,b,x,y) have been changed to suit the Cross entropy loss function.

Fin