PadhAI: 6 Jars of Sigmoid Neuron

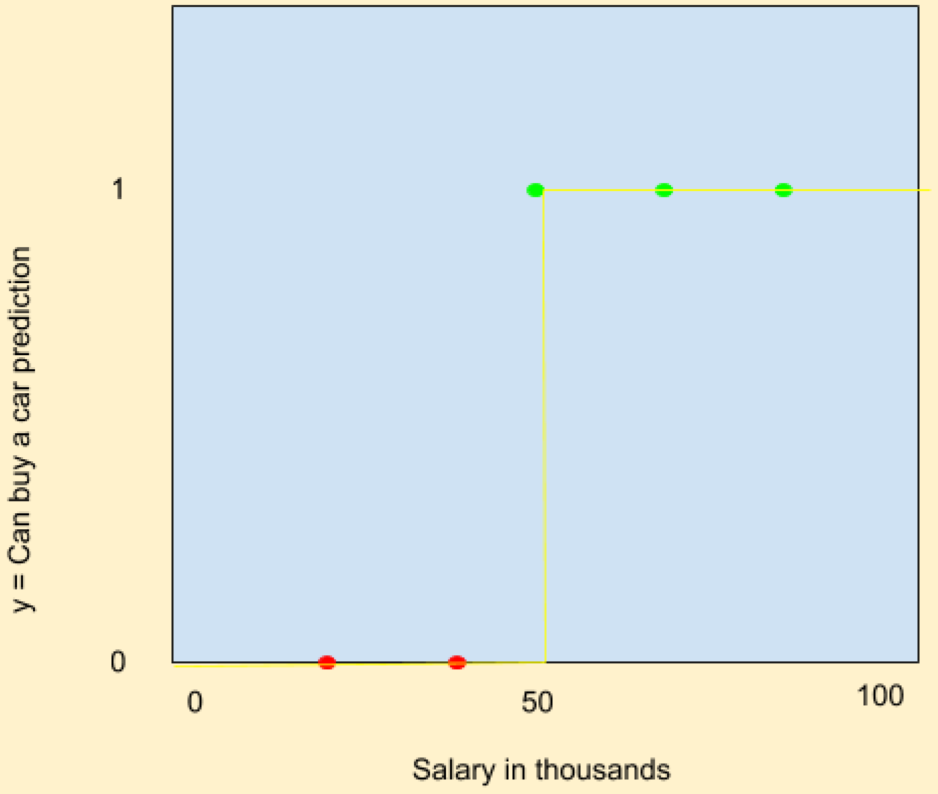
One Fourth Labs

Limitations of Perceptron

1. Perceptron model: $y = \Sigma^{n}_{i=1} w_i x_i >= b$
2. Consider the following dataset

Salary in thousands	Can buy a car?
20	0
30	0
50	1
60	1
70	1

3. Plotting the perceptron results



4.
5. The function looks like a step, it has a value(50) beyond which the curve suddenly changes orientation
6. So it divides the input space into two halves with negative on one side and positive on one side
7. This case reproduces in higher dimensions, 2D, 3D etc.
8. It cannot be applied to non-linearly separable data.

9. The function is harsh at the boundary. For eg: 49.9 would be 0 and 50.1 would be 1. In practical real-life scenarios, a much smoother boundary is more applicable.
10. What is the road ahead?
    a. Data: Real inputs 😃
    b. Task: Regression/Classification, Real output 😃
    c. Model: Smooth at boundaries, Non-linear(😃 and ☹ because it's not a very advanced non-linear model)
    d. Loss: $\Sigma_i(y_i - \hat{y}_i)^2$ 😃
    e. Learning: A more generic Learning Algorithm 😃
    f. Evaluation: Accuracy, Root-mean-squared-error

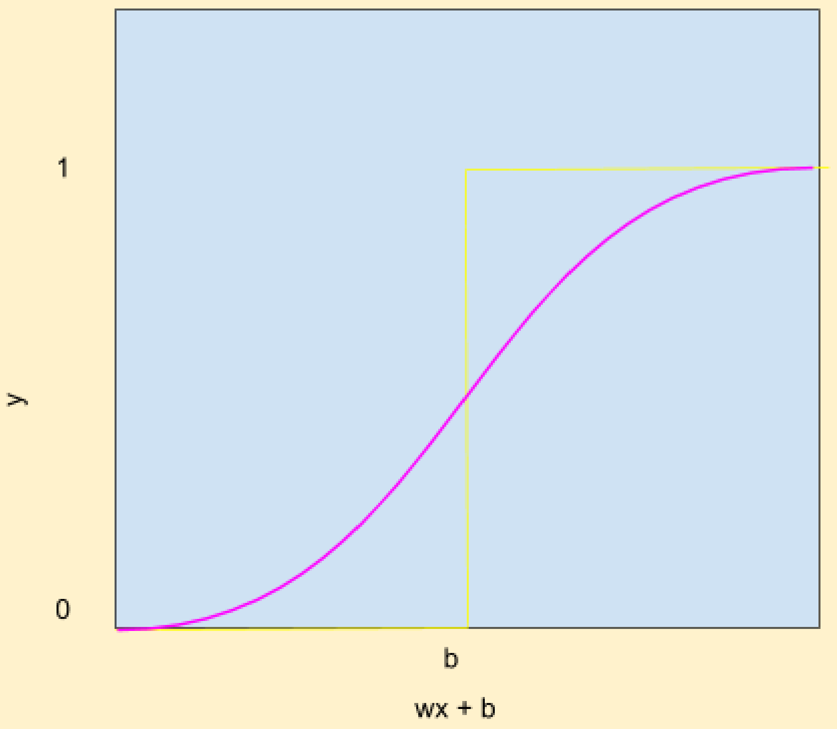
Sigmoid Neuron

Sigmoid Model

Model Part 1

Can we have a smoother (not-so-harsh) function?

1. The sigmoid function is provides a smoother, s-shaped curve as opposed to a stepped line.



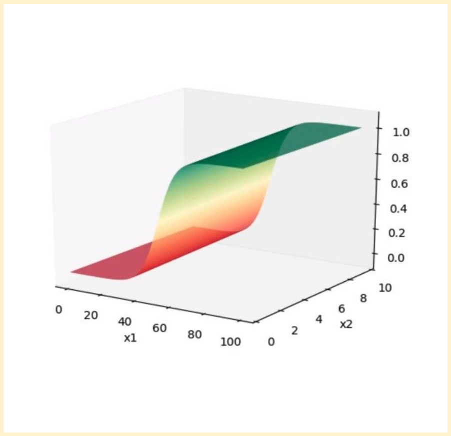
2.
3. The function is defined as $y = 1/(1 + \exp(-(\sum_i w_i x_i + b))$, where i = 1 to n
4. Substituting different values for $(w^T x + b)$ and y, we will be able to trace out a curve similar to the one drawn above
   a.

Sigmoid Neuron

Sigmoid Model

Model Part 2

What happens when we have two inputs?

1. Consider y = 1/(1 + exp(-($w_1x_1 + w_2x_2$ + b))
2. 

3. Substituting different values of w and x would result in the shape as seen in the curve above
4. Whenever $w_1x_1 + w_2x_2 + b = 0$, then y = 0.5

Sigmoid Neuron

Sigmoid Model

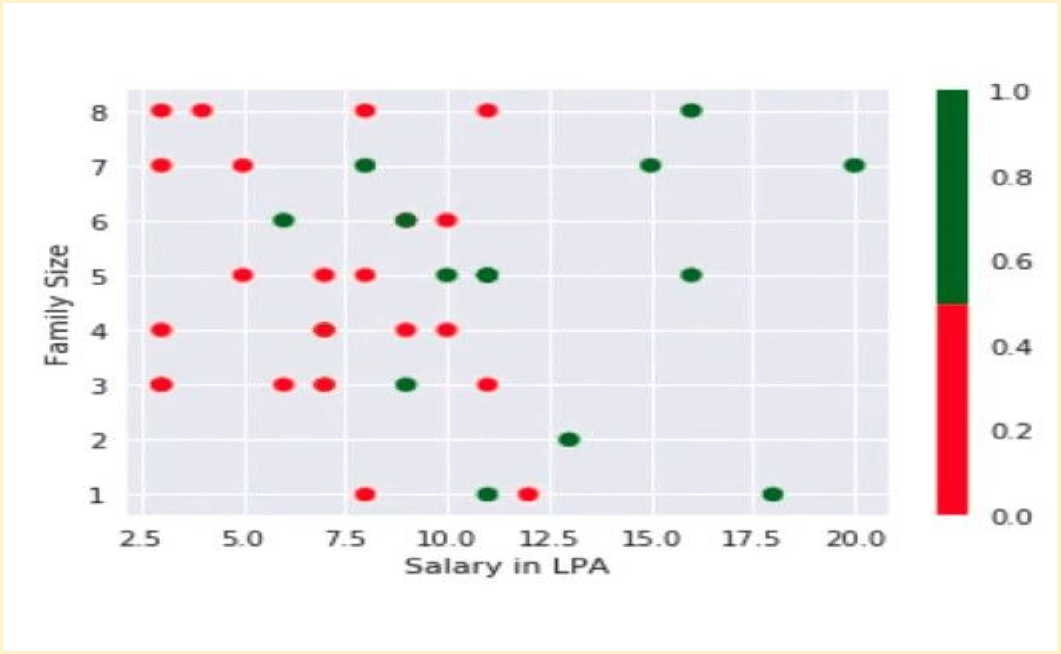
Model Part 3

How does this help when the data is not linearly separable

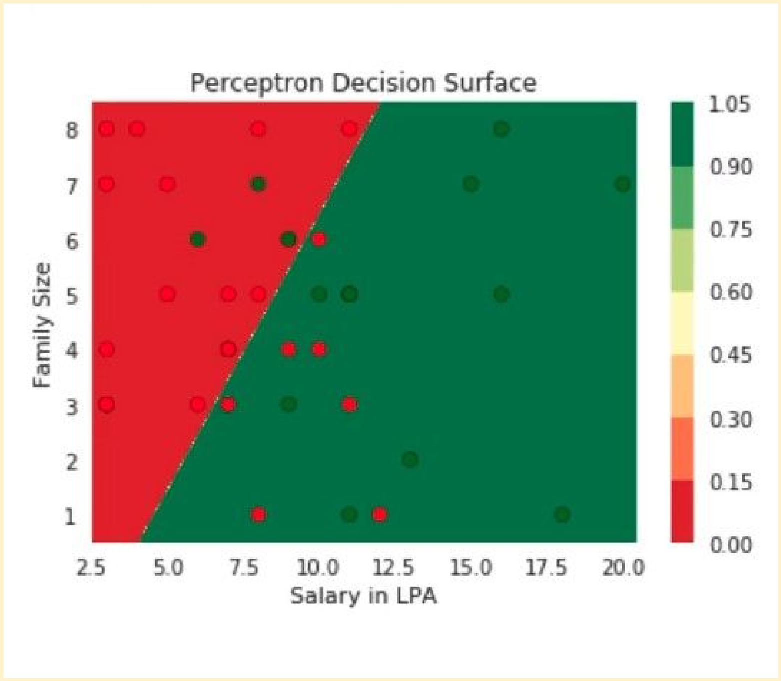
1. $y = 1/(1 + \exp(-(w^Tx + b))$
2. Consider the following dataset

	Salary in LPA	Family Size	Buys Car?
0	11	8	1
1	20	7	1
2	4	8	0
3	8	7	0
4	11	5	1

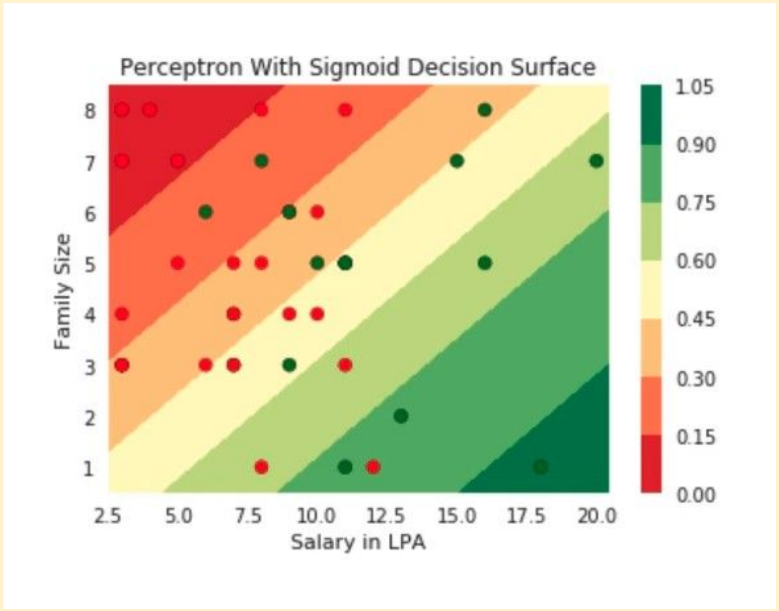
3. The dataset is visualised



4. Decision Boundary: Perceptron



5. Decision boundary: Perceptron with sigmoid. (Not optimised to separate outputs efficiently)



6. Here even the sigmoid function doesn't effectively separate the outputs.
7. We must play around with different values of w and b to find the best fit
8. This can be done with the learning algorithm

Sigmoid Neuron

Sigmoid Model

Model Part 4

How does the function behave if we change w and b

1. **w**: (controls the slope)
   a. Negative w, negative slope, mirrored s-shape, becomes more harsh(vertical/less smooth) the more negative it goes
   b. Positive w, positive slope, normal s-shape, becomes more harsh(vertical/less smooth) the more positive it goes
2. **b**: (controls the midpoint)
   a. $y = 1/(1 + \exp(-(wx + b)) = ½$ (for $w=1.00$, $b = -5$)
   b. $\exp(-(wx + b)) = 1$
   c. $wx + b = 0$
   d. $x = -b/w$ (As b becomes more -ve, boundary moves more to the right +ve, and vice versa)

Sigmoid Data and Tasks

What kind of data and tasks can Sigmoid Neuron process

1. Here, the Sigmoid neuron can process data similar to the Perceptron, the difference being the output is real valued, from 0 to 1.
2. This allows us to perform regression: Where we predict y as a continuous value, being some function applied to x,
3. $\hat{y} = f(x)$, where f() is the sigmoid function in this case
4. Here is a sample, similar to perceptron except for real values output y.

	phone 1	phone 2	phone 3	phone 4	phone 5	phone 6	phone 7	phone 8	phone 9
Launch (within 6 months) x_1	0	1	1	0	0	1	0	1	1
Weight (g) x_2	151	180	160	205	162	182	138	185	170
Screen Size (< 5.9in) x_3	5.8	6.18	5.84	6.2	5.9	6.26	4.7	6.41	5.5
Dual sim x_4	1	1	0	0	0	1	0	1	0
Internal mem(>= 64gb, 4gb ram) x_5	1	1	1	1	1	1	1	1	1
NFC x_6	0	1	1	0	1	0	1	1	1
Radio x_7	1	0	0	1	1	1	0	0	0
Battery (mAh) x_8	3060	3500	3060	5000	3000	4000	1960	3700	3260
Price? (k) x_9	15k	32k	25k	18k	14k	12k	35k	42k	44k
Liked (y)	0.6	0.31	0.55	0.23	0.8	0.75	0.16	0.59	0.40

Sigmoid Loss Function

What is the loss function used for this model

1. Here the squared-error function is used **loss =** $\Sigma_i (y_i - \hat{y}_i)^2$. Another useful loss function is called the cross entropy function.
2. Consider the following data

x_1	x_2	y	ŷ
1	1	0.5	0.6
2	1	0.8	0.7
1	2	0.2	0.2
2	2	0.9	0.5

3. Loss = $\Sigma^4_i (y_i - \hat{y}_i)^2 = 0.18$
4. This also works if y is boolean valued

x_1	x_2	y	ŷ
1	1	1	0.6
2	1	1	0.7
1	2	0	0.2
2	2	0	0.5

5. Loss = $(1 - 0.6)^2 + (1 - 0.7)^2 + (0 - 0.2)^2 + (0 - 0.5)^2$
6. The interesting thing to note here is that in sigmoid neuron, each individual points contribute differently to the overall loss. Some points are more correct than others and some are more wrong than others.
7. Whereas in Perceptron, it was either right or wrong, no degrees of correctness or wrongness.

Learning Algorithm (Gradient Descent)

Intro to Learning Algorithm

Learning Objective: In sigmoid neuron or any other model, we have parameters that influence the way the output is predicted (ie, the way the curve is drawn). Changing these parameters changes the curve. The objective of a learning algorithm is to determine the values for these parameters such that the overall loss of the model over the training data is minimized.

Steps (for sigmoid neuron)

1. Initialize w and b (random initialization)
2. Iterate over the data
    a. Update w and b for every iteration
3. End when satisfied (ie, loss is minimized)

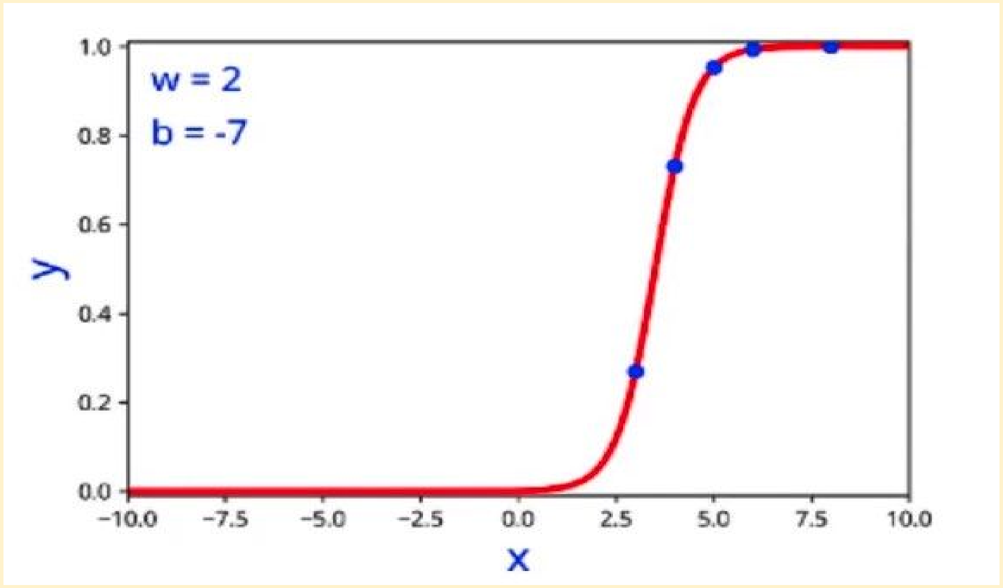
Learning by guessing

Can we try to estimate w,b by using some guess work?

1. Steps:
   a. **Initialise:** w,b to 0
   b. **Iterate over Data:** guess_and_update(x_i)
      i. $w = w + \Delta w$
      ii. $b = b + \Delta b$
      iii. Here, Δw and Δb are the amounts we change w and b, by pure guess-work. We need to design a function to replace the guess-work.
   c. **Till satisfied**
2. Consider the following dataset

I/P	O/P
2	0.047
3	0.268
4	0.73
5	0.952
8	0.999

3. Manually change the slope w and the midpoint b till it looks to fit the data, then perform fine-tuning to match the training examples as closely as possible



4. We have guessed, by trial and error and found that w=2 and b=-7 fits the training data best.
5. This is only possible in lower dimensions, 1D or 2D, and becomes much harder as more features are involved.

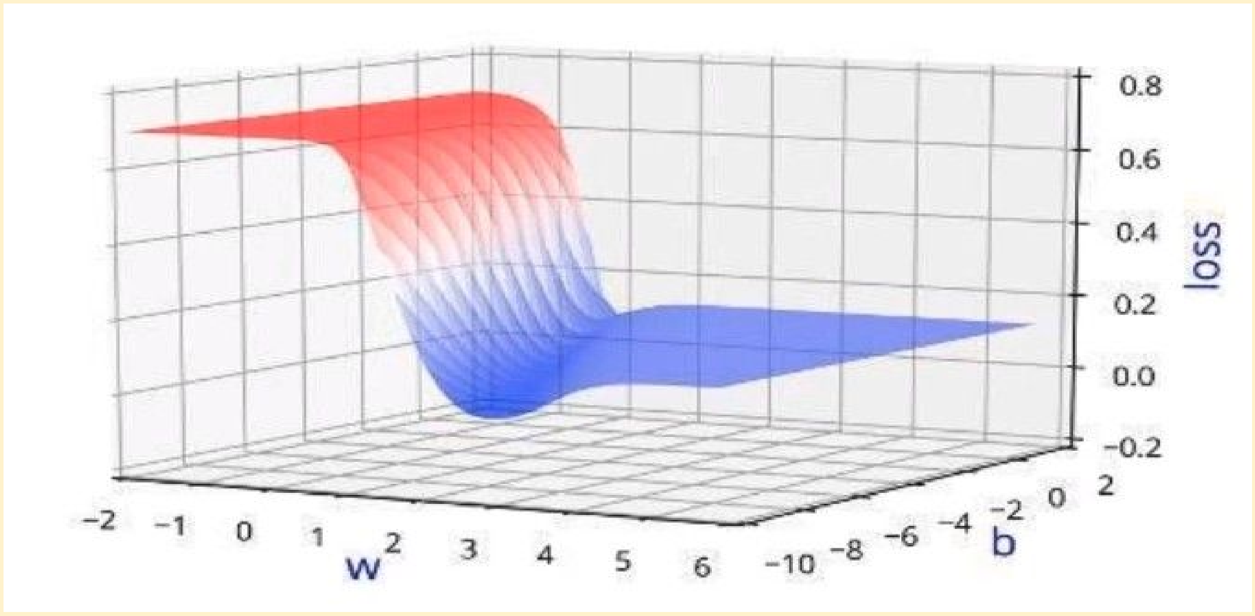
Error Surfaces for learning

Can we connect this to the loss function?

1. So far, we have iteratively modified w and b till we reached the values which yielded minimum loss
2. However, instead of a smooth descent from initial value to the minimum, the loss fluctuates each iteration.
3. For eg, for the previously used dataset, loss over each iteration was

Iteration	w	b	loss	Increase/decrease
1	0	0	0.1609	-
2	1	0	0.1064	decrease
3	2	0	0.1210	increase
4	3	0	0.1217	increase
5	3	-2	0.1215	decrease
6	3	-9	0.0209	decrease
7	2	-9	0.0696	increase
...final	2	-7	0	decrease

4. Now, this erratic fluctuation is undesirable. We need to use an algorithm that ensures that the loss decreases on every iteration or at the very least, doesn't increase.
5. The following image shows the plot of the loss wrt w and b. The lowest point refers to the minimum loss which corresponds to the ideal parameters of w(2) and b(-7).



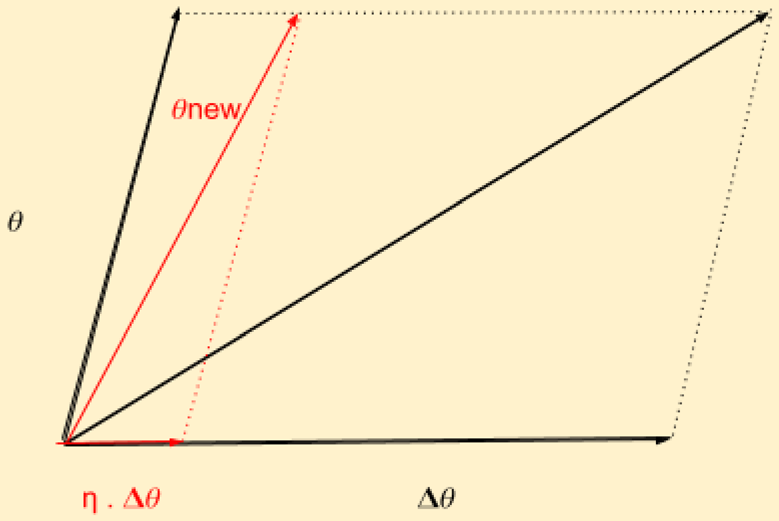
PadhAI: 6 Jars of Sigmoid Neuron

One Fourth Labs

Mathematical setup for the learning algorithm

What is our aim now?

1. Instead of guessing Δw and Δb, we need a principled way of changing w and b based on the loss function.
2. First, let's formulate this more mathematically
   a. $\theta = [w, b]$ (Theta is a vector containing the values of w and b)
   b. $\Delta\theta = [\Delta w, \Delta b]$ ($\Delta\theta$ is the change vector, the value we change w and b by)
   c. $\theta = \theta + \eta\Delta\theta$ (Where η is the learning rate, which allows for small changes in θ)



   d. We need to compute $\Delta\theta$ such that Loss(θ_{new}) < Loss(θ_{old})

The math-free version of the learning algorithm

What does the algorithm look like now

1. The learning algorithm
   a. **Initialise:** w, b
   b. **Iterate over data**
      i. Compute ŷ
      ii. Compute Loss(w,b)
      iii. $w_{t+1} = w_t + \eta \Delta w_t$
      iv. $b_{t+1} = b_t + \eta \Delta b_t$
   c. **Till satisfied**
2. Where
   a. model: $\hat{y} = 1/(1 + \exp(-(wx + b))$
   b. $\text{Loss}(w,b) = \Sigma_i (y_i - \hat{y}_i)^2$
   c. Δw and Δb are the partial derivatives of Loss(w,b) with respect to w and b respectively.
3. Frameworks like Pytorch and Tensorflow can automatically implement the learning algorithm and return the ideal values of parameters w and b

Introducing Taylor Series

Can we get the answer from some basic mathematics.

1. Our aim is
   a. w ⇒ w + η Δw
   b. Loss(w) > Loss(w + η Δw)
2. Taylor Series: $f(x + \Delta x) = f(x) + \frac{\{f'(x)\}}{\{1!\}}\Delta x + \frac{\{f''(x)\}}{\{2!\}}(\Delta x^2) + \frac{\{f'''(x)\}}{\{3!\}}(\Delta x^3) + \ldots$
3. Here, f(x + Δx) is Loss(w + η Δw) and f(x) is Loss(w)
4. We need to find Δx such that everything after f(x) sums to a negative value, ie, lowering the overall value of f(x + Δx)

More intuitions about Taylor Series

Can we get the answer from some basic mathematics?

1. The real aim is:
   a. $w \Rightarrow w + \eta\Delta w$
   b. $b \Rightarrow b + \eta\Delta b$
   c. $Loss(w) > Loss(w + \eta\Delta w)$
   d. $Loss(b) > Loss(b + \eta\Delta b)$
   e. $Loss(w, b) > Loss(w + \eta\Delta w, b + \eta\Delta b)$
   f. $Loss(\theta) > Loss(\theta + \eta\Delta\theta)$ (where $\theta = [w, b]$)

2. Vectorized Taylor Series: $L(\theta + \eta u) = L(\theta) + \eta * u^T \nabla_\theta L(\theta) + \frac{\eta^2}{2!} * u^T \nabla^2 L(\theta) u + \frac{\eta^3}{3!} * \ldots$
3. Where, $u = \Delta\theta$
4. Here, we know that in practice, η is very small ie (0.001) etc
5. So η^2 ,η^3… all end up being negligible, so remove those corresponding terms
6. New Vectorized Taylor Series: $L(\theta + \eta u) \approx L(\theta) + \eta * u^T \nabla_\theta L(\theta)$
   a. Here, ∇_θ refers to Gradient w.r.t θ and it consists of the partial derivatives of $L(\theta)$ w.r.t w and b, stacked up into a vector
   b. $L(\theta + \eta u) \in \mathbb{R}$
   c. $L(\theta) \in \mathbb{R}$
   d. $\eta \in \mathbb{R}$ $\frac{\partial Loss}{\partial w}$ $\frac{\partial Loss}{\partial b}$
   e. $u^T(\nabla_\theta L(\theta))$ = Dot product of $\Delta\theta$ transposed and the partial derivative vector ∇_θ ($\in \mathbb{R}$)

$$\begin{bmatrix} [\Delta w & \Delta b] \end{bmatrix} \quad \begin{bmatrix} \frac{\partial Loss}{\partial w} \\ \frac{\partial Loss}{\partial b} \end{bmatrix}$$

Deriving the Gradient Descent Update rule

How does Taylor series help us arrive at the right answer?

1. For ease of notation, let $\Delta\theta$ = u
2. Then from Taylor series,we have:
   a. $L(\theta + \eta u) = L(\theta) + \eta * u^T \nabla_\theta L(\theta)$
   b. Rearranging: $L(\theta + \eta u) - L(\theta) = \eta * u^T \nabla_\theta L(\theta)$
   c. Note,that the move ηu would only be favourable if
      i. $L(\theta + \eta u) - L(\theta) < 0$ (i.e. if the new loss is less than the previous loss)
      ii. This implies $u^T \nabla_\theta L(\theta) < 0$
   d. Now we have $u^T \nabla_\theta L(\theta) < 0$
      i. Let β be the angle between u and $\nabla_\theta L(\theta)$, then we know that,
      ii. $-1 \leq cos(\beta) = \frac{u^T \nabla_\theta L(\theta)}{||u|| * ||\nabla_\theta L(\theta)||} \leq 1$
      iii. Multiply throughout by k = $||u|| * ||\nabla_\theta L(\theta)||$
      iv. This gives us $-k \leq u^T \nabla_\theta L(\theta) \leq k$
   e. Thus, $L(\theta + \eta u) - L(\theta) = u^T \nabla_\theta L(\theta) = k * cos(\beta)$ will be most negative when $cos(\beta) = -1$, i.e. when β is 180°
3. Gradient Descent Rule
   a. The direction u that we intend to move in should be at 180° w.r.t, the gradient
   b. In other words, move in a direction opposite to the gradient
4. Parameter Update Rule
   a. $w_{t+1} = w_t - \eta \Delta w_t$
   b. $b_{t+1} = b_t + \eta \Delta b_t$
   c. Where $\Delta w_t = \frac{\partial L(w,b)}{\partial w}$ $at\ w = w_t\ ,\ b = b_t$
   d. Where $\Delta b_t = \frac{\partial L(w,b)}{\partial b}$ $at\ w = w_t\ ,\ b = b_t$

The complete learning algorithm

How does the algorithm look like now?

1. The algorithm
   a. **Initialise:** w, b randomly
   b. **Iterate over data**
      i. Compute ŷ
      ii. Compute L(w,b)
      iii. $w_{t+1} = w_t - \eta \Delta w_t$
      iv. $b_{t+1} = b_t + \eta \Delta b_t$
      v. Pytorch/Tensorflow have functions to compute $\frac{\delta l}{\delta w}$ *and* $\frac{\delta l}{\delta b}$
   c. **Till satisfied**
      i. Number of epochs is reached (ie 1000 passes/epochs)
      ii. Continue till Loss < ε (some defined value)
      iii. Continue till $\text{Loss}(w,b)_{t+1} \approx \text{Loss}(w,b)_t$

Computing Partial Derivatives

How do I compute Δw and Δb

1. Consider the following example
2. Loss = $\frac{1}{5}\Sigma^{5}_{i=1}(f(x_i) - y_i)^2$ (where f(x_i) refers to the sigmoid function)
3. $\Delta w = \frac{\partial L}{\partial w} = \frac{1}{5}\Sigma^{5}_{i=1}\frac{\partial}{\partial w}(f(x_i) - y_i)^2$
4. Let's consider only one term in this sum
5. $\nabla w = \frac{\partial}{\partial w}[\frac{1}{2} * (f(x) - y)^2]$ (where ∇w refers to the gradient/partial derivative of L(w,b) w.r.t w)
6. Using chain rule, we expand it
    a. $\nabla w = \frac{1}{2} * [2 * (f(x) - y) * \frac{\partial}{\partial w}(f(x) - y)]$
    b. $\nabla w = (f(x) - y) * \frac{\partial}{\partial w}(f(x))$
    c. $\nabla w = (f(x) - y) * \frac{\partial}{\partial w}(\frac{1}{1 + e^{-(wx+b)}})$, Let's look into the derivative of the sigmoid function in detail
        i. $\frac{\partial}{\partial w}(\frac{1}{1 + e^{-(wx+b)}})$
        ii. $\frac{-1}{(1 + e^{-(wx+b)})^2}\frac{\partial}{\partial w}(e^{-(wx+b)})$
        iii. $\frac{-1}{(1 + e^{-(wx+b)})^2} * (e^{-(wx+b)})\frac{\partial}{\partial w}(-(wx + b))$
        iv. $\frac{-1}{(1 + e^{-(wx+b)})^2} * (e^{-(wx+b)}) * (-x)$
        v. $\frac{1}{(1 + e^{-(wx+b)}} * \frac{(e^{-(wx+b)})}{(1 + e^{-(wx+b)})} * (x)$
        vi. $f(x) * (1 - f(x)) * x$
    d. Therefore, $\nabla w = (f(x) - y) * f(x) * (1 - f(x)) * x$
7. For each of the 5 points
    a. $\Delta w = \frac{1}{5}\Sigma^{5}_{i=1}(f(x_i) - y_i) * f(x_i) * (1 - f(x_i)) * x_i$
    b. Similarly $\Delta b = \frac{1}{5}\Sigma^{5}_{i=1}(f(x_i) - y_i) * f(x_i) * (1 - f(x_i))$

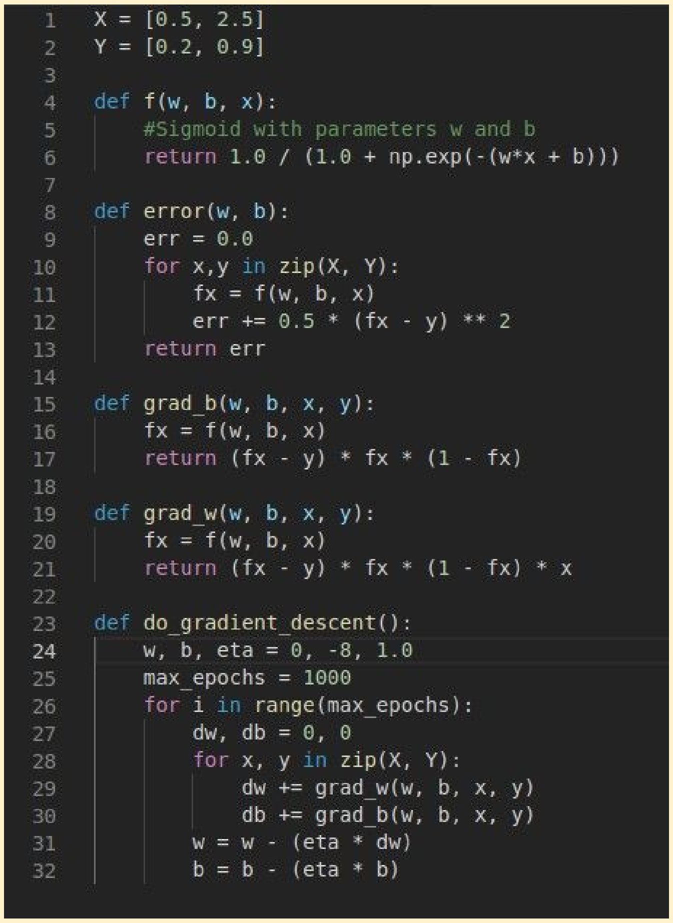
PadhAI: 6 Jars of Sigmoid Neuron

One Fourth Labs

Writing the code

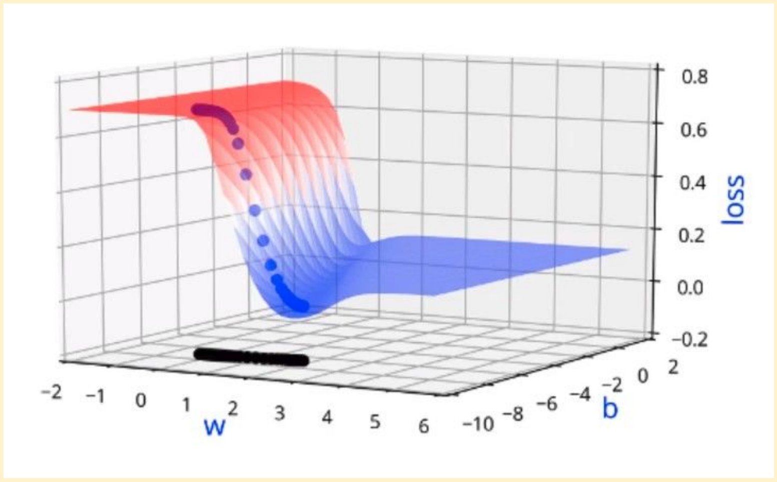
How do we implement this in Python

1. Here is the Python code for Gradient descent



```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

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):
    err = 0.0
    for x,y in zip(X, Y):
        fx = f(w, b, x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w, b, x, y):
    fx = f(w, b, x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w, b, x, y):
    fx = f(w, b, x)
    return (fx - y) * fx * (1 - fx) * 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. This is how the algorithm works



PadhAI: 6 Jars of Sigmoid Neuron

One Fourth Labs

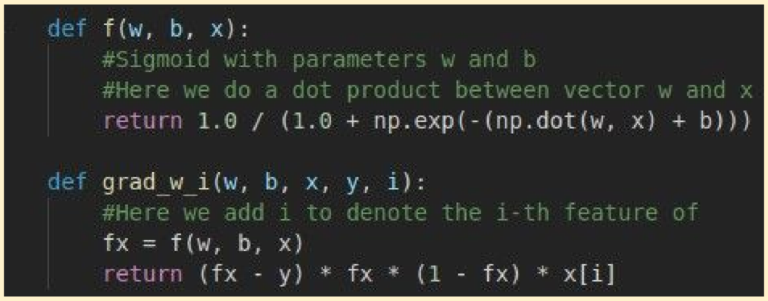
Dealing with more than 2 parameters

What happens when we have more than 2 parameters

1. Consider the following dataset

ER_visits	Narcotics	Pain	TotalVisits
0	2	6	11
1	1	4	25
0	0	5	10
1	3	5	7

2. Then,
   a. $z = \Sigma^{n}_{i=1} w_i x_i$
   b. Or z = (w_1 * ER_visits) + (w_2 * Narcotics) + (w_3 * Pain) + (w_4 * TotalVisits) + b
   c. $\hat{y} = \frac{1}{1+e^{-z}}$
3. So the algorithm is as follows
   a. **Initialise:** w_1, w_2, w_3, w_4 and b randomly
   b. **Iterate over data**
      i. Compute $\hat{y}$
      ii. Compute L(w,b)
      iii. $w_1 = w_1 - \eta \Delta w_1$
      iv. $w_2 = w_2 - \eta \Delta w_2$
      v. $w_3 = w_3 - \eta \Delta w_3$
      vi. $w_4 = w_4 - \eta \Delta w_4$
      vii. $b = b + \eta \Delta b$
      viii. Where $\Delta w_j = \Sigma^{m}_{i=1} (\hat{y} - y) * (\hat{y}) * (1 - \hat{y}) * x_{ij}$
   c. **Till satisfied**
      i. Number of epochs is reached (ie 1000 passes/epochs)
      ii. Continue till Loss < ε (some defined value)
      iii. Continue till $\text{Loss}(w,b)_{t+1} \approx \text{Loss}(w,b)_t$
   d. A few of the functions from the code also change, namely



```
def f(w, b, x):
    #Sigmoid with parameters w and b
    #Here we do a dot product between vector w and x
    return 1.0 / (1.0 + np.exp(-(np.dot(w, x) + b)))

def grad_w_i(w, b, x, y, i):
    #Here we add i to denote the i-th feature of
    fx = f(w, b, x)
    return (fx - y) * fx * (1 - fx) * x[i]
```

   e. The function do_grad_descent also changes, but we will figure it out in the practical implementation

Sigmoid Evaluation

How do you check the performance

1. Consider the following test data

	phone1	phone2	phone3	phone4
Launch (within 6 months) x_1	1	0	0	1
Weight (g) x_2	0.2	0.73	0.6	0.8
Screen Size (< 5.9in) x_3	0.2	0.7	0.8	0.9
Dual sim x_4	0	1	0	0
Internal mem(>= 64gb, 4gb ram) x_5	1	0	0	0
NFC x_6	0	0	1	0
Radio x_7	1	1	1	0
Battery (mAh) x_8	0.83	0.96	0.9	0.2
Price? (k) x_9	0.34	0.4	0.6	0.1
Liked (y)	0.17	0.67	0.9	0.3
Predicted($\hat{y}$)	0.24	0.67	0.9	0.3

2. Calculate the Root Mean Square Error
3. $RMSE = \sqrt{\frac{1}{n}\Sigma^{n}{}_{i=1}(y - \hat{y})^2}$
4. Here, RMSE = 0.311, the smaller the better
5. For classification problems, set a threshold ε , such that
    a. $(y|\,\hat{y} < \varepsilon) = 0$
    b. $(y|\,\hat{y} >= \varepsilon) = 1$

Summary

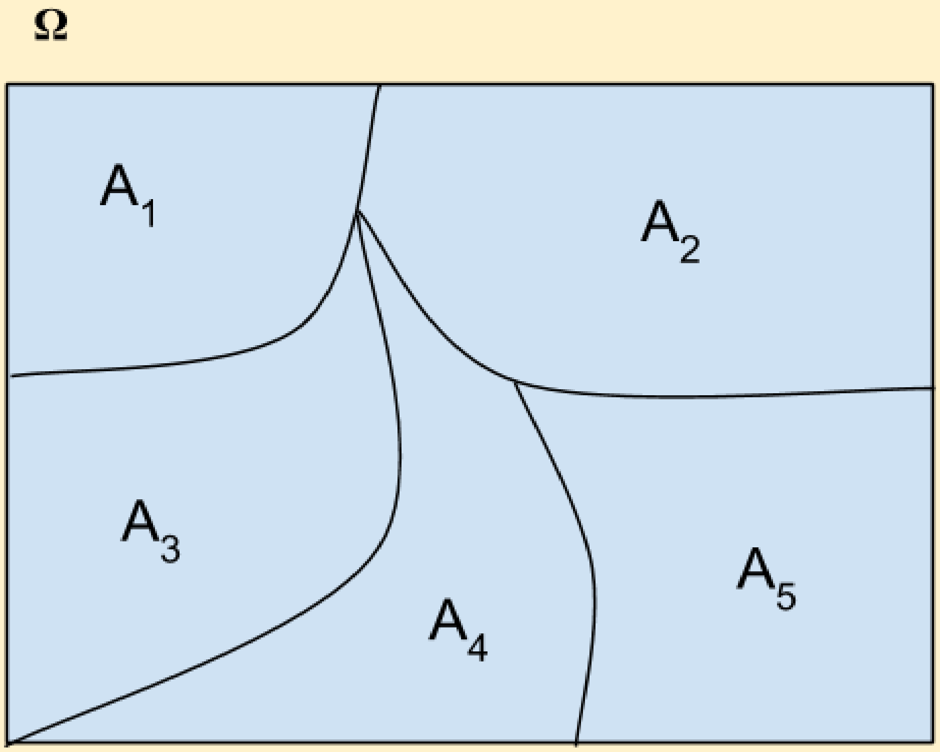
Let’s compare MP Neuron, Perceptron and Sigmoid Neuron

	Data	Task	Model	Loss	Learning	Evaluation
MP Neuron	{0,1}	Binary Classification	$g(x) = \Sigma^{n}_{i=1} x_i$ y = 1 if g(x) >= b y = 0 otherwise	Loss = $\Sigma_i (y_i \,!= \hat{y}_i)$	Brute Force Search	Accuracy
Perceptron	Real Inputs	Binary Classification	y = 1 if $\Sigma^{n}_{i=1} w_i x_i$ >= b y = 0 otherwise	Loss = $\Sigma_i (y_i - \hat{y}_i)^2$	Perceptron Learning Algorithm	Accuracy
Sigmoid	Real Inputs	Classification/ Regression	$y = \frac{1}{1 + e^{-(w^T x + b)}}$	Loss = $\Sigma_i (y_i - \hat{y}_i)^2$	Gradient Descent	Accuracy/RMSE

Basics of Probability Theory

What are the axioms of Probability

1. Consider the following sample space

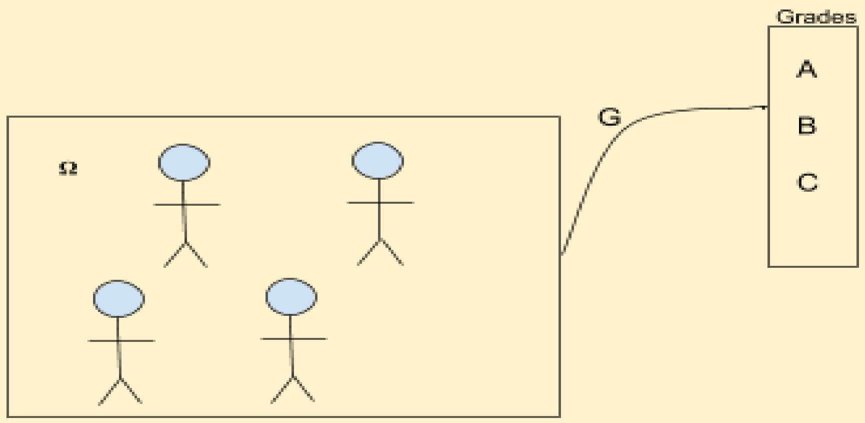


2. For any event A,
   a. 0 <= P(A) <= 1
3. If A_1, A_2,…A_n are disjoint events, ie $A_i \cap A_j = \phi \quad \forall (!i) = j$
   a. $P(\cup A_i) = \Sigma_i P(A_i)$
   b. The probability of the union of all the events is equal to the sum of the individual probabilities of those events
   c. $P(\cup A_i) = P(A_1) + P(A_2) + P(A_3) + P(A_4) + P(A_5)$
4. If $\mathbf{\Omega}$ is the universal set containing all the events, then
   a. $P(\Omega) = 1$

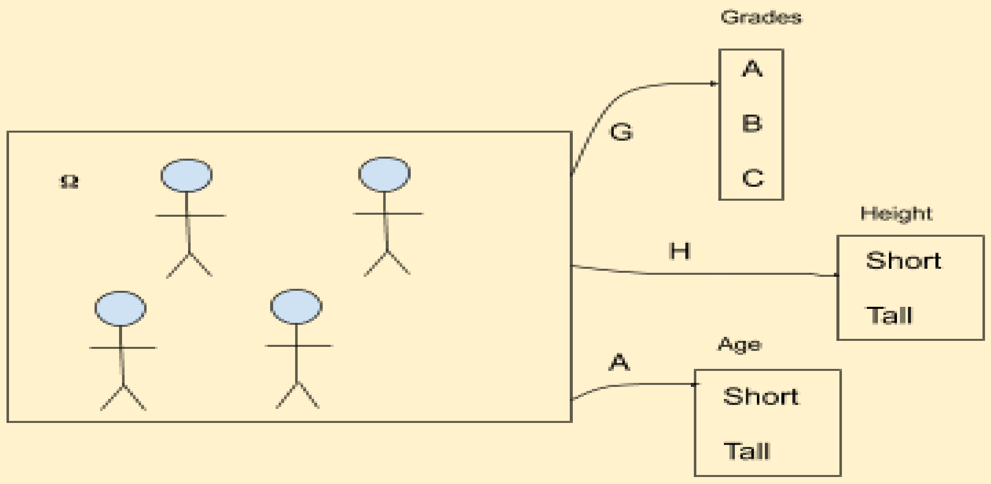
Random Variable Intuition

What is a Random Variable (intuition)

1. Suppose a student gets one of 3 possible grades in a course: A, B, C
2. One way of interpreting this is that there are 3 possible events here.
    a. For eg, to find P(A) we take $\frac{No.\ of\ students\ with\ A\ grade}{Total\ No.\ of\ students}$
3. Another way of looking at this is that there is a random variable G which maps each student to one of the 3 possible values



4. Here, the random variable G is treated more like a function that serves to map a student to a grade
5. And we are interested in $P(G = g)$ where $g \in \{A, B, C\}$
6. The benefit of this is that we can use multiple random variables on the same set to map to different outcomes



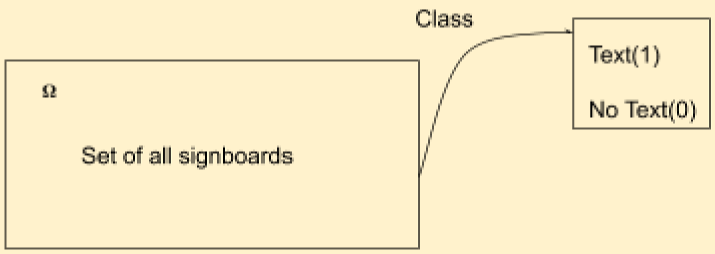
PadhAI: Probability Theory

One Fourth Labs

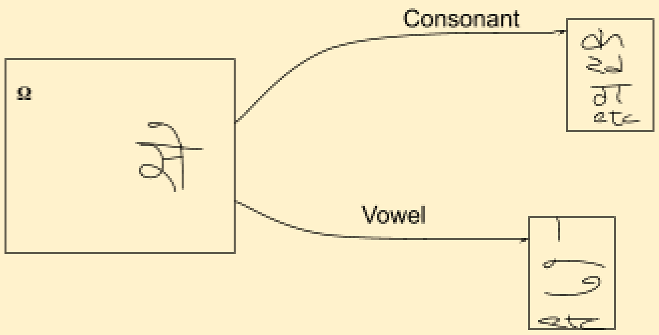
Random Variable Formal Definition

What is a random variable (formal definition)

1. A random variable is a function which maps each outcome in Ω to a value
2. In the previous example, G (or f_{grade}) maps each student in Ω to a value: A, B or C
3. The event Grade=A is a shorthand for the event
    a. $\{\omega \in \Omega : f_{grade} = A\}$
    b. In other words, All the elements such that when you apply f_{grade} the answer is A
    c. Grade is a random variable
    d. P(grade = A) = $\frac{\{\omega \in \Omega : f_{grade} = A\}}{Total\ number\ of\ students}$
    e. In the context of our example



4. This also applies to multiclass classification
    a. Mapping one Letter to its respecting vowel, and consonant.



5. Here, it would be P(Consonant=स) and P(Vowel = ै)

Random Variable Continuous and Discrete

What are continuous and discrete random variables

1. A random variable can either take a continuous values/Real values (ie, weight, height)
2. Or discrete values(ie, Grade, Nationality)
3. For the scope of this course, we will mostly be dealing with discrete random variables. Ie, P(Vowels), P(Consonants) which all draw from a fixed set of discrete values

Probability Distribution

What is a marginal distribution?

1. Consider a random variable G for grades

G	**P(G=g)**
A	0.1
B	0.2
C	0.7

2. The above table represents the marginal distribution over G
   a. $(G = g) \quad \forall g \in A, B, C$
3. i.e. The probability of every possible value that the random variable can take (sums to 1)
4. We denote this marginal distribution compactly by P(G)

True and Predicted Distribution

What are true and predicted distributions

1. Consider the above example

G	P(G=g) (y)	(ŷ)
A	0.1	0.2
B	0.2	0.3
C	0.7	0.5

2. Here, y refers to the true distribution, or the actual probabilities for each value of G
3. And ŷ is the predicted distribution, or what we estimate the probabilities to be based on our observations
4. To measure the degree of correctness of our predictions, we can use a loss function.
5. However, Squared-error function might not be appropriate as it doesn't factor in some of the basic assumption of probability theory, ie P(G) >= 0 and <= 0, etc
6. So, we must select a different loss function that is more rooted in probability theory (Cross Entropy)

Certain Events

Events with 100% probability

1. We need something better than the squared error loss
2. Consider the scenario of a random variable X that maps to the winner in a tournament of 4 teams: A, B, C, D
3. We stop watching after the semi-finals, so we are unaware of the outcome, but in truth, team A has won, thus it is a certain event, with probabilities (P(A) = 1, P(B) = 0, P(C) = 0, P(D) = 0).

X	P(X=x) True distribution, unknown to us.	$\hat{Y}$ Predicted by us
A	1 (Certain event)	0.6
B	0	0.2
C	0	0.15
D	0	0.15

4. Before the tournament's completion, based on the point we have watched till(Semi-finals), we can predict the probabilities of each team's chance at victory (P(A) = 0.6, P(B) = 0.2, P(C) = 0.15, P(D) = 0.15)

Why do we care about Distributions

Let us put it into the context of our final project

1. Consider the signboard with the text ‘**Mumbai**’. Now our classifier is analysing the text character by character, and a random variable char maps the character to one of the 26 possible characters in the english language
2. For the first character **M,** we know the True distribution intuitively.

char	**Y = P(char=c)** **The certain event/True distribution**	**Ŷ** **Obtained from model**
a	0	0.01
b	0	0.01
...	0...	0.01...
m	1	0.7
...	...0...	...0.01...
z	...0	...0.01

3. We compute the difference between the True and Predicted distributions using squared-error loss or some other loss function. From this, it is clear why we use distributions in the scope of our learning.

Expectation

What is the expectation of a distribution

1. Let us consider the random variable X that maps to the winning team amongst the 4 teams: A, B, C, D
2. P(X = x) represents the probability of team x winning where $x \in \{A, B, C, D\}$
3. Consider G(X=x), the gain associated with each of the teams if they win, where $x \in \{A, B, C, D\}$
4. Now, the expectation E(x) is given by $\Sigma_{i \in \{A,B,C,D\}} P(X = i) * G(X = i)$
5. Consider the following data

X	P(X = x)	G(X = x)
A	0.4	10000
B	0.2	2000
C	0.1	-8000
D	0.3	5000

6. Therefore, E(X) = $(0.4 * 10000) + (0.2 * 2000) + (0.1 * -8000) + (0.3 * 5000)$ = 5100

Information Content

What is Information content?

1. Consider the Random variable SR which maps to the direction in which the sun rises: East, West, North & South.
   a. Now, we are told that P(SR=East) is 1.
   b. Here, this is almost a blatantly obvious truth, thus we can say that the Information Gained here is very low.
2. Consider another Random variable ST, which maps to whether there is going to be a storm today: Yes, No.
   a. Now, we are told that P(ST=Yes) = 1
   b. Here, the information gained is very high as this is a rather surprising(low probability) event
   c. We can almost say that $Information\ Content \propto Surprise$
   d. Or in other words $Information\ Content \propto \frac{1}{P(X=Surprise)}$
   e. Thus, it can be inferred that the information content is a function of the probability of the event
   f. $IC(P(X=S))$ Where IC is information content
3. Now, consider two separate events
   a. X maps to which cricket team won the match: A, B, C, D
   b. Y maps to the state of a light switch: On, Off
   c. Now we are told that Team B won the match AND the light switch is On
   d. The total Information gained is $IC(X=B \cap Y=On) = IC(X=B) + IC(Y=On)$
4. Combining the points from above, we have
   a. $IC(P(X=S))$ (Information Content is a function of probability)
   b. $IC(P(X \cap Y)) = IC(P(X)) + IC(P(Y))$ (From the previous example)
   c. From probability theory, if P(X) and P(Y) are disjoint, then $(P(X \cap Y)) = P(X) \, . \, P(Y)$
   d. Therefore $IC(P(X).P(Y)) = IC(P(X)) + IC(P(Y))$
   e. Therefore we need a family of function that satisfy $f(a.b) = f(a) + f(b)$
   f. The log functions satisfy this $log(a.b) = log(a) + log(b)$
5. Now we can write the IC function as follows
   a. $IC(X=A) = log(\frac{1}{P(X=A)})$
   b. $IC(X=A) = log(1) - log(P(X=A))$
   c. $IC(X=A) = -\ log_2 P(X=A)$ (All the logs use base 2)

PadhAI: Information Theory and Cross Entropy

One Fourth Labs

Entropy

What is Entropy

1. First, a quick recap of the concepts we've studied so far

Random Variable: X	**Probability Distribution: P(X=?)**	**Information Content: IC(X=?)**	**Expectation E(Gain)**
A	P(X=A)	$-\log_2 P(X=A)$	$\Sigma_{i\in\{A,B,C,D\}} P(X=i) * Gain(X=i)$
B	P(X=B)	$-\log_2 P(X=B)$	
C	P(X=C)	$-\log_2 P(X=C)$	
D	P(X=D)	$-\log_2 P(X=D)$	

2. Based on these four concepts, we can talk about Entropy
3. Entropy H(X) is the Expected Information Content of a Random Variable
4. $H(X) = \ -\Sigma_{i\in\{A,B,C,D\}} P(X=i) * log_2 P(X=i)$
5. Basically, substitute Gain for Information Content in the

Relation to Number of Bits

Relation between number of bits and entropy

1. Consider the Entropy equation from the previous section using shorthand P_i for P(X=i)
2. $H(X) = -\Sigma_{i \in \{A,B,C,D\}} P_i * log\, P_i$
3. Suppose there is a message X that you want to transfer that can take 4 values: A, B, C, D
4. For 4 values, we would use 2 Bits to transfer each message

Random Variable: X	2 Bit version	Probability Distribution: P(X=?)	Information Content: IC(X=?)
A	00	1/4	$-log_2 2^2 = 2$ (ie $log_a a^n = n$)
B	01	1/4	$-log_2 2^2 = 2$
C	10	1/4	$-log_2 2^2 = 2$
D	11	1/4	$-log_2 2^2 = 2$

5. Now we can make the connection that the number of bits required to transfer a message is equal to the information content of that message
6. Consider another message X with 8 values: A, B, C, D, E, F, G, H

Random Variable: X	3 Bit version	Probability Distribution: P(X=?)	Information Content: IC(X=?)
A	000	1/8	$-log_2 2^3 = 3$ (ie $log_a a^n = n$)
B	001	1/8	$-log_2 2^3 = 3$
C	010	1/8	$-log_2 2^3 = 3$
D	100	1/8	$-log_2 2^3 = 3$
E	011	1/8	$-log_2 2^3 = 3$
F	101	1/8	$-log_2 2^3 = 3$
G	110	1/8	$-log_2 2^3 = 3$
H	111	1/8	$-log_2 2^3 = 3$

7. While sending a continuous stream of messages, we would be interested in minimizing the stream of bits that we send
8. Consider the same 4 valued example but with a different distribution

Random Variable: X	Probability Distribution: P(X=?)	Information Content: IC(X=?)
A	1/2 (High prob)	$-log_2 2^1 = 1$ (ie $log_a a^n = n$)
B	1/4 (Medium prob)	$-log_2 2^2 = 2$
C	1/8 (Low prob)	$-log_2 2^3 = 3$
D	1/8 (Low prob)	$-log_2 2^3 = 3$

9. This situation is considered favourable only if the average number of bits is less that the value it takes for an equally distributed set of values
10. The average is calculated using Entropy $H(X) = -\Sigma_{i \in \{A,B,C,D\}} P_i * log\, P_i$
11. Average/Entropy $= \frac{1}{2}(1) + \frac{1}{4}(2) + \frac{1}{8}(3) + \frac{1}{8}(3) = 1.75\ which\ is < 2$
12. Thus, the Entropy gives us the ideal number of bits that should be used to transmit the message

KL- Divergence and Cross Entropy

How we deal with true and predicted distributions

1. Consider the following data:

X	True Distribution: y	True IC(X)	Predicted Distribution: y	Predicted IC(X)
A	y_1	$-\log y_1$	$\hat{y}_1$	$-\log\hat{y}_1$
B	y_2	$-\log y_2$	$\hat{y}_2$	$-\log\hat{y}_2$
C	y_3	$-\log y_3$	$\hat{y}_3$	$-\log\hat{y}_3$
D	y_4	$-\log y_4$	$\hat{y}_4$	$-\log\hat{y}_4$

2. Initially, we do not know the values of the True distribution and thereby the True Information Content
3. Hence, we generate a Predicted distribution and use that to compute the predicted information content.
4. But, the actual message will come from the True distribution y.
5. So therefore, the No. of bits will **not be** $-\Sigma\hat{y}_i \, log\hat{y}_i$ but **instead** $-\Sigma y_i \, log\hat{y}_i$
6. This is because the value associated with each of these messages comes from the predicted distribution $-log\hat{y}_i$ but the messages themselves comes from the True distribution y
7. Now, we have formed the basis to talk about KL-Divergence:
   a. $H_y = -\Sigma y_i \, log y_i$ is called the entropy
   b. $H_{y,\hat{y}} = -\Sigma y_i \, log\hat{y}_i$ is called the cross entropy
   c. Now we want to find the difference/distance between the predicted case and the true case, using something more efficient than the squared error
   d. So y||ŷ = $H_{y,\hat{y}} - H_y$
   e. y||ŷ = $-\Sigma y_i \, log\hat{y}_i + \Sigma y_i \, log y_i$
   f. This is called the KL-Divergence
8. Thus, we now have **KLD(y||ŷ) =** $-\Sigma y_i \, log\hat{y}_i + \Sigma y_i \, log y_i$
9. Now, we have a way of computing the difference between the two distributions.

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

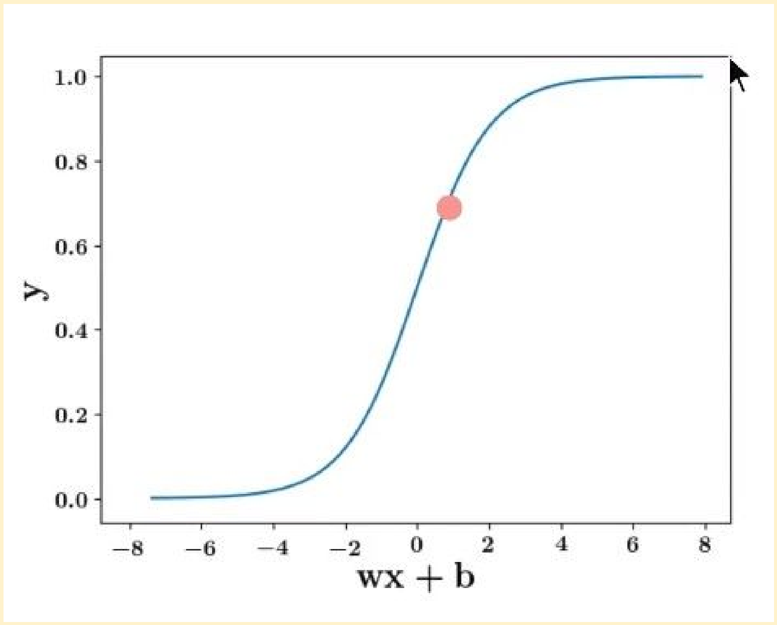
    d. Previously, we were using **Squared-error Loss** = $\Sigma_i(y_i - \hat{y}_i)^2$
    e. Now, we have a better metric, one that is grounded in probability theory (**KL- Divergence**)
    f. KLD(y||ŷ) = $-\Sigma y_i\, log\hat{y}_i + \Sigma y_i\, log y_i$
    g. We aim to minimize loss by KLD with respect 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 $min(-\Sigma_i y_i log\hat{y}_i)\quad here\ i\ \in T,\ NT$
    j. $Cross\ Entropy\ Loss =\ -1 * log(0.7)\ -\ 0 * log(0.3)$
    k. The second term 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

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

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 $Loss(w,b)_{t+1} \approx Loss(w,b)_t$

Computing Partial Derivatives With Cross Entropy Loss

How do we compute **Δ**w and **Δ**b

1. Loss Function $L(\theta) = -[(1-y)log(1-\hat{y}) + ylog(\hat{y})]$
2. Consider **Δ**w 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. $\frac{\partial L(\theta)}{\partial w} = \frac{y\hat{-}y}{(1-\hat{y})\hat{y}}$
   c. The second part: $\frac{\partial \hat{y}}{\partial w} = \frac{\partial}{\partial w}\frac{1}{1+e^{-(w.x+b)}}$
      i. This is the exact same as from the Squared Error Loss
      ii. $\frac{\partial}{\partial w}(\frac{1}{1+e^{-(wx+b)}})$
      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 **Δ**w and **Δ**b into the learning algorithm to optimise the parameters w and b.

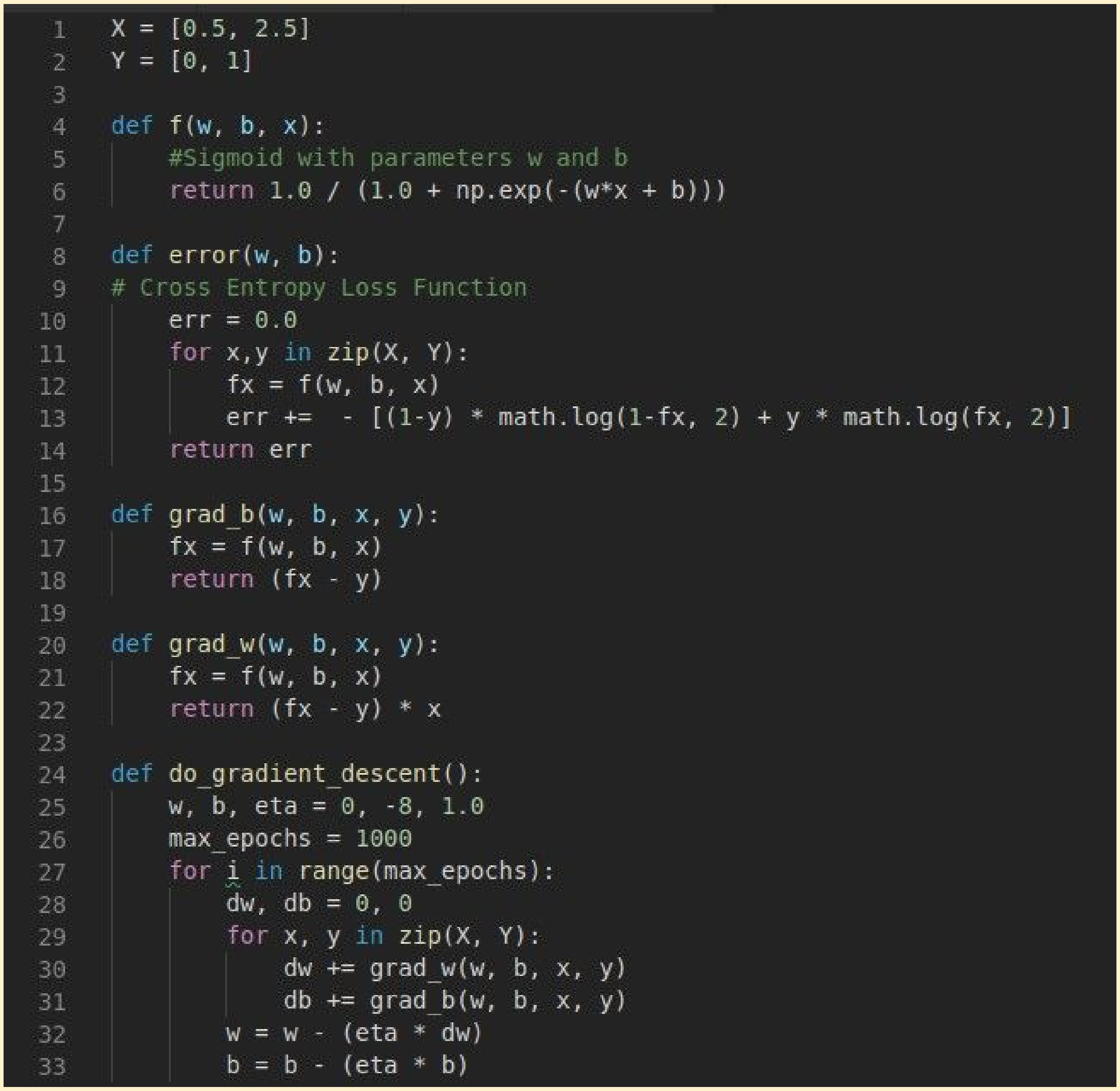
PadhAI: Information Theory and Cross Entropy

One Fourth Labs

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.