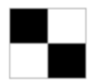
Gradient Descent

Logistic Regression Revisited Given a 2 x 2 grid where each cell aij can take on one of two colors c1 and c2 , find a function that can identify the following diagonal pattern:

That is, find f such that

$$f\left(\begin{array}{|c|c|}\hline a_{00} & a_{01} \\ \hline a_{10} & a_{11} \\ \hline\end{array}\right) = \begin{cases} ✔ \text{ if } \begin{array}{|c|c|}\hline a_{00} & a_{01} \\ \hline a_{10} & a_{11} \\ \hline\end{array} = \begin{array}{|c|c|}\hline c_1 & c_2 \\ \hline c_2 & c_1 \\ \hline\end{array} = \\ ✘ \text{ otherwise} \end{cases}$$



We can define: ✔ = 1 and ✗ = 0

We can assign weights to each cell

w_1	w_2
w_3	w_4

$$\begin{array}{|c|c|}\hline w_1 & w_2 \\ \hline w_3 & w_4 \\ \hline\end{array} \otimes \begin{array}{|c|c|}\hline a_{00} & a_{01} \\ \hline a_{10} & a_{11} \\ \hline\end{array} = w_1a_{00}$$

$$\begin{array}{|c|c|}\hline w_1 & w_2 \\ \hline w_3 & w_4 \\ \hline\end{array} \otimes \begin{array}{|c|c|}\hline a_{00} & a_{01} \\ \hline a_{10} & a_{11} \\ \hline\end{array} = w_1a_{00} + w_2a_{01}$$

$$\begin{array}{|c|c|}\hline w_1 & w_2 \\ \hline w_3 & w_4 \\ \hline\end{array} \otimes \begin{array}{|c|c|}\hline a_{00} & a_{01} \\ \hline a_{10} & a_{11} \\ \hline\end{array} = w_1a_{00} + w_2a_{01} + w_3a_{10}$$

$$\begin{array}{|c|c|}\hline w_1 & w_2 \\ \hline w_3 & w_4 \\ \hline\end{array} \otimes \begin{array}{|c|c|}\hline a_{00} & a_{01} \\ \hline a_{10} & a_{11} \\ \hline\end{array} = w_1a_{00} + w_2a_{01} + w_3a_{10} + w_4a_{11}$$

Equivalently we can decide to move the value b to the left of the equation in order for the weighted sum to reveal a diagonal pattern at 0: w1 a00 + w2 a01 + w3 a10 + w4 a11 + b = 0 if diagonal pattern found We could then find a function **σ** to apply to the result of this sum in order to make predictions {0, 1}:

$$\boldsymbol{\sigma}(w_1a_{00} + w_2a_{01} + w_3a_{10} + w_4a_{11} + b) = 1 \text{ if } w_1a_{00} + w_2a_{01} + w_3a_{10} + w_4a_{11} + b = 0 \text{ else } 0$$

Suppose we relax our definition of diagonal by having a continuum of colors [c1 , c2]. This means there will be a continuum of values for our weighted sum to take when a diagonal pattern is found:

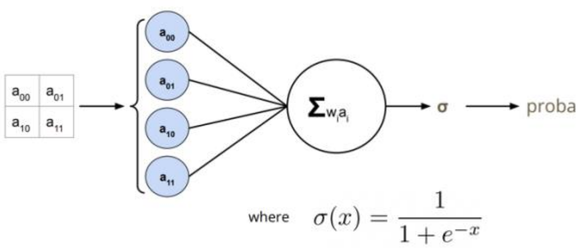
$$w_1a_{00} + w_2a_{01} + w_3a_{10} + w_4a_{11} + b > 0 \text{ if diagonal}$$

We would like our function to adapt to this vagueness of specification / definition by reflecting an uncertainty in prediction (i.e. predicting probabilities of being diagonal)

$$\boldsymbol{\sigma}(w_1a_{00} + w_2a_{01} + w_3a_{10} + w_4a_{11} + b) > 0.5 \text{ then diagonal}$$

When **σ** is the logistic (also called sigmoid) function, this is Logistic Regression. So for each cell we're looking to learn a weight wi that makes **σ** larger for diagonal patterns. The bias term b lets us account for systemic dimming or brightening of cells (i.e. when the data is not normalized).

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



Recall that logistic regression is looking for weights and a bias that maximizes the probability of having seen the data we saw:

$$\max \prod_{i=1}^{n} P(y_i = 1|x_i)$$
$$= \min -\frac{1}{n}\sum_{i=1}^{n}\left[yi \log(\sigma(-w^T x_i + b)) + (1-y_i)\log(1-\sigma(-w^T x_i + b))\right]$$
$$= \min \mathrm{Cost}(w, b)$$

Gradient Descent (intuition):
There is no closed form solution to finding the extrema of this cost function. We can however use an iterative process by which we increment w and b gradually toward some minimum (most likely local). Goal: find a sequence of wi 's (and b's) that converge toward a minimum.

Intuitively the best nudge should be in the direction of the largest rate of change (steepness) of the function. Rate of change -> think derivatives

$$\nabla f(x) = f'(x)$$

Intuitively, the rate of change of a multi-dimensional function should be a combination of the rate change in each dimension. For a 3-dimensional function, the rate of change would be:

$$\nabla f(x, y, z) = \frac{\partial f}{\partial x}\vec{i} + \frac{\partial f}{\partial y}\vec{j} + \frac{\partial f}{\partial z}\vec{k}$$

$f(x) = 3x^2 - 2y$
Without even computing derivatives we can see that changes in x create more positive change in f than changes in y.
$\nabla f = 6xi - 2j$ This is the gradient of f and can be evaluated at any point (x, y) in the space.

However, the gradient expresses the instantaneous rate of change. At p, ∇f p is the steepest but the highest value of f will depend on how many units we step in that direction. If we step too many units away, the instantaneous change in f is no longer representative of what values f will take.

Given a "smooth" function f for which there exists no closed form solution for finding its maximum, we can find a local maximum through the following steps:
1. Define a step size $\boldsymbol{\alpha}$ (tuning parameter)
2. Initialize p to be random
3. $p_{new} = \boldsymbol{\alpha}\, \nabla f_p + p$
4. p p_{new}
5. Repeat 3 & 4 until $p \sim p_{new}$

Notes about $\boldsymbol{\alpha}$:
- If $\boldsymbol{\alpha}$ is too large, GD may overshoot the maximum, take a long time to or never be able to converge
- If $\boldsymbol{\alpha}$ is too small, GD may take too long to converge

We need to compute ∇Cost(w, b):

$$\nabla \text{Cost}(w, b) = \left[\frac{\partial}{\partial w}\text{Cost}, \frac{\partial}{\partial b}\text{Cost}\right]$$

$$\frac{\partial}{\partial w}\text{Cost} = \frac{1}{n}\sum_{i=1}^{n} x_i(y_i - \sigma(-w^T x_i + b))$$

$$\frac{\partial}{\partial b}\text{Cost} = \frac{1}{n}\sum_{i=1}^{n} \sigma(-w^T x_i + b) - y_i$$

1. Start with random w and b: $w = [0\ 0\ 0\ 0]^T$, $b = 0$ Note: $\boldsymbol{\sigma}(0) = 0.5$
2. Compute the Cost(w, b) Cost([0 0 0 0]T , 0) = -1 log($\boldsymbol{\sigma}$(0)) = -log(0.5)
3. Compute the gradient ∇Cost at (w, b)

$$\frac{\partial}{\partial w}\text{Cost} = \frac{1}{1}\sum_{i=1}^{1}\begin{bmatrix}0\\1\\1\\0\end{bmatrix}(1-\sigma(0)) = \begin{bmatrix}0\\1/2\\1/2\\0\end{bmatrix}$$

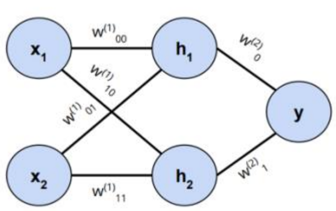
4. Adjust w & b by taking $\boldsymbol{\alpha}$ steps in the direction of $-\nabla\text{Cost}_{(w,\ b)}$
5. Compute the updated Cost

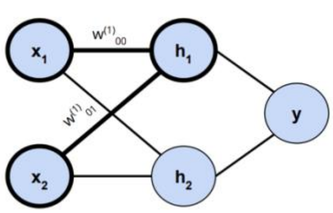
$$\text{Cost}(\begin{bmatrix}0\\-\alpha/2\\-\alpha/2\\0\end{bmatrix}, \frac{\alpha}{2}) = -\log(\sigma(\alpha + \frac{1}{2}))$$

Recall the Cost is computed for the entire dataset. This has some limitations:
1. It's expensive to run
2. The result we get depends only on the initial starting point

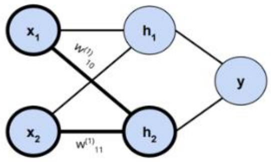
Note:
The magnitude of ∇f_p depends on p. A p gets closer to the min / max, the size of ∇f_p decreases. This also means that points p that contain more "information" have larger gradients. So the order with which this process is exposed to examples matters.

Neural Networks:
We need to define:
1. How input flows through the network to get the output (forward propagation)
2. How the weights and biases gets updated (Backpropagation)

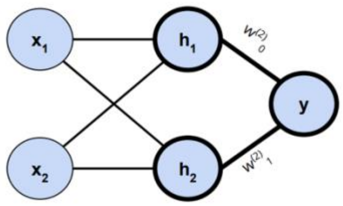




$$h_1 = \boldsymbol{\sigma}(w^{(1)}_{00}x_1 + w^{(1)}_{01}x_2 + b^{(1)}_{1})$$



$h_2 = \boldsymbol{\sigma}(w^{(1)}_{10}\, x_1 + w^{(1)}_{11}\, x_2 + b^{(1)}_2)$



$y = \boldsymbol{\sigma}(w^{(2)}_0\, h_1 + w^{(2)}_1\, h_2 + b^{(2)}_1)$

Using matrix notation:

$$\begin{bmatrix} h_1 \\ h_2 \end{bmatrix} = \sigma\left(\begin{bmatrix} w_{00}^{(1)} & w_{01}^{(1)} \\ w_{10}^{(1)} & w_{11}^{(1)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \end{bmatrix}\right)$$

$$y = \sigma(\begin{bmatrix} w_{00}^{(2)} \\ w_{01}^{(2)} \end{bmatrix}^T \begin{bmatrix} h_1 \\ h_2 \end{bmatrix} + b^{(2)})$$

If we don't, we just end up with normal logistic regression on x_1 and x_2.

$h_1 = w^{(1)}_{00}\, x_1 + w^{(1)}_{01}\, x_2 + b^{(1)}_1$

$h_2 = w^{(1)}_{10}\, x_1 + w^{(1)}_{11}\, x_2 + b^{(1)}_2$

Then

$y = \boldsymbol{\sigma}(w^{(2)}_0\, h_1 + w^{(2)}_1\, h_2 + b^{(2)}_1)$

$\quad = \boldsymbol{\sigma}(w^{(2)}_0(w^{(1)}_{00}\, x_1 + w^{(1)}_{01}\, x_2 + b^{(1)}_1) + w^{(2)}_1(w^{(1)}_{10}\, x_1 + w^{(1)}_{11}\, x_2 + b^{(1)}_2) + b^{(2)}_1)$

$\quad = \boldsymbol{\sigma}(w_1\, x_1 + w_2\, x_2 + b_2)$

Neural Networks - BackPropagation

Using the chain rule:

$$\frac{\partial C}{\partial W^{(2)}} = \frac{\partial C}{\partial u^{(2)}} \frac{\partial u^{(2)}}{\partial W^{(2)}} \quad \text{where} \quad u^{(2)} = W^{(2)} h + b^{(2)}$$

$$= \frac{\partial C}{\partial u^{(2)}} \cdot h = \frac{1}{n} \sum_{i=1}^{n} h(y_i - \sigma(u^{(2)}))$$

$h = \boldsymbol{\sigma}(W^{(1)}\, X + b^{(1)})$

$$\frac{\partial C}{\partial b^{(2)}} = \frac{\partial C}{\partial u^{(2)}} \frac{\partial u^{(2)}}{\partial b^{(2)}} = \frac{1}{n} \sum_{i=1}^{n} y_i - \sigma(u^{(2)})$$

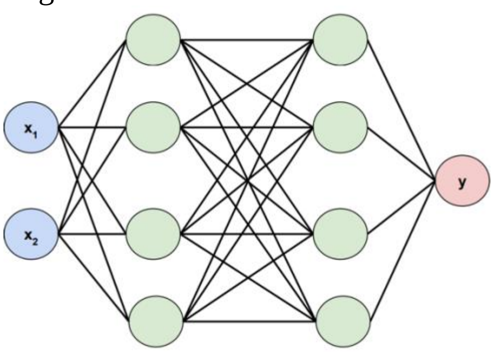
So we can update W(2) and b(2) as follows:

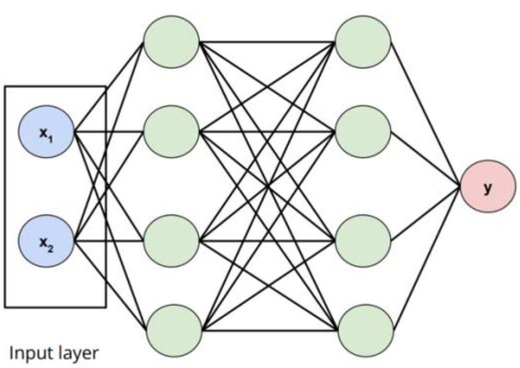
$$\begin{bmatrix} W^{(2)}_{new} \\ \\ b^{(2)}_{new} \end{bmatrix} = -\alpha \begin{bmatrix} \frac{\partial C}{\partial W^{(2)}} \\ \\ \frac{\partial C}{\partial b^{(2)}} \end{bmatrix} + \begin{bmatrix} W^{(2)} \\ \\ b^{(2)} \end{bmatrix}$$

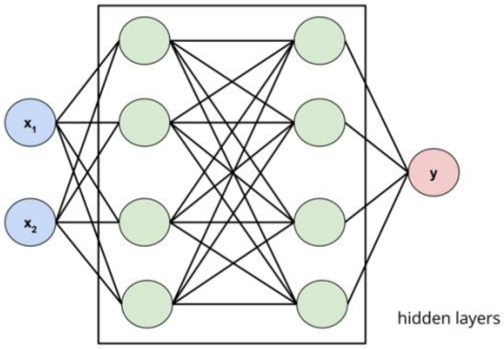
Important Note:

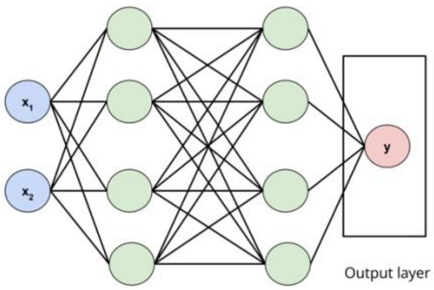
> Important Note:
>
> $$\frac{\partial C}{\partial W^{(1)}} = \frac{\partial C}{\partial h} \cdot \frac{\partial h}{\partial W^{(1)}} = \frac{\partial C}{\partial h} \cdot \frac{\partial h}{\partial u^{(1)}} \cdot \frac{\partial u^{(1)}}{\partial W^{(1)}} \quad \text{where} \quad u^{(1)} = w^{(1)}x + b^{(1)}$$
>
> $$= \frac{\partial C}{\partial u^{(2)}} \cdot \frac{\partial u^{(2)}}{\partial h} \cdot \frac{\partial h}{\partial u^{(1)}} \cdot \frac{\partial u^{(1)}}{\partial W^{(1)}} = \frac{\partial C}{\partial u^{(2)}} \cdot W^{(2)} \cdot \sigma'(u^{(1)}) \cdot x$$
>
> Depends on both data and weights
> Initializing all weights to zero then is not a good idea

In general:









Neural Networks Can do both Classification and Regression

Neural Networks - Tuning Parameters
1. Step size **α**
2. Number of BackPropagation iterations
3. Batch Size
4. Number of hidden layers
5. Size of each hidden layer
6. Activation function used in each layer
7. Cost function
8. Regularization (to avoid overfitting)

Neural Networks - Convolutional Neural Networks
Creating such a filter allows us to:
1. Reduce the number of weights
2. Capture features all over the image
The process of applying a filter (or kernel) is called a convolution

Recurrent Neural Networks
Handling sequences of input.
Intuition: What a word is / might be in a sentence is easier to figure out if you know the words around it.
Applications:
1. Predicting the next word
2. Translation

3. Speech Recognition
4. Video Tagging