Use of Python

• Both GPU and CPU has parallelization instructions and many Python built-in functions supports that.

• Use built-in functions for matrix (or vector) operations and avoid using for loops in your code whenever you can!
Cost function and GD implementation

Remember the loss for single sample:

\[
\frac{\partial L(a, y)}{\partial w_1} = x_1(a - y) \\
\frac{\partial L(a, y)}{\partial w_2} = x_2(a - y) \\
\frac{\partial L(a, y)}{\partial b} = a - y
\]

Cost Function:

\[
J(w, b) = \frac{1}{m} \sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)})
\]

Final derivatives to be used:

\[
\frac{\partial J(w, b)}{\partial w_1} = \frac{1}{m} \sum_{i=1}^{m} \frac{\partial L(a^{(i)}, y^{(i)})}{\partial w_1} \\
\frac{\partial J(w, b)}{\partial w_2} = \frac{1}{m} \sum_{i=1}^{m} \frac{\partial L(a^{(i)}, y^{(i)})}{\partial w_2} \\
\frac{\partial J(w, b)}{\partial b} = \frac{1}{m} \sum_{i=1}^{m} \frac{\partial L(a^{(i)}, y^{(i)})}{\partial b}
\]

Implement all that:

\[
J = 0; \ dw_1 = 0; \ dw_2 = 0; \ db = 0; \ \alpha = 0.00001
\]

For i=1 to m

\[
z^{(i)} = w^T x^{(i)} + b \\
a^{(i)} = \sigma(z^{(i)}) \\
J += -[y^{(i)} \log a^{(i)} + (1 - y^{(i)}) \log(1 - a^{(i)})] \\
dz^{(i)} = a^{(i)} - y^{(i)} \\
dw_1 += x_1^{(i)} dz^{(i)} \\
dw_2 += x_2^{(i)} dz^{(i)} \\
wb += dz^{(i)}
\]

\[
J = J/m; \ dw_1 = dw_1/m; \\
dw_2 = dw_2/m; \ db = db/m \\
w_1 := w_1 - \alpha dw_1 \\
w_2 := w_2 - \alpha dw_2 \\
b := b - \alpha db
\]
Let's Re-implement the Logistic Regression

Remember:

\[ X_{nxm} = \begin{bmatrix} X_1 \ldots X_m \end{bmatrix} \]
\[ Y = \begin{bmatrix} y_1, y_2, \ldots, y_m \end{bmatrix} \]

(Training) Data

\[ Z = \begin{bmatrix} Z_1^T \ldots Z_m^T \end{bmatrix} = w^T X + [b, b, \ldots, b] \]
\[ A = \begin{bmatrix} a_1^T \ldots a_m^T \end{bmatrix} = \sigma(Z) \]

Implementation for \( Z \):

\[ Z = \text{np. dot}(w, T, X) + b \]

Note that in this particular example, the output is a vector. Therefore, the matrix \( Z \) (thus, the matrix \( A \)) becomes a vector.
A Better Implementation in Python

### One iteration of gradient descent

\[ J = 0; \ dw_1 = 0; \ dw_2 = 0; \ db = 0 \]

For \( i = 1 \) to \( m \)

\[
\begin{align*}
  z_i^{(t)} &= w^T x_i^{(t)} + b \\
  a_i^{(t)} &= \sigma(z_i^{(t)}) \\
  J &= -[y_i^{(t)} \log a_i^{(t)} + (1 - y_i^{(t)}) \log (1 - a_i^{(t)})] \\
  dz_i^{(t)} &= a_i^{(t)} - y_i^{(t)} \\
  dw_1 &= x_1^{(t)} dz_i^{(t)} \\
  dw_2 &= x_2^{(t)} dz_i^{(t)} \\
  db &= dz_i^{(t)} \\
  J &= J/m; \ dw_1 = dw_1/m; \ dw_2 = dw_2/m; \ db = db/m \\
  w_1 &= w_1 - \alpha dw_1 \\
  w_2 &= w_2 - \alpha dw_2 \\
  b &= b - \alpha db
\end{align*}
\]

### One iteration of gradient descent

\[
\begin{align*}
  Z &= w^T X + b \\
  A &= \sigma(Z) \\
  dZ &= A - Y \\
  dW &= (1/m)X dZ^T \\
  db &= (1/m) \text{np.sum}(dZ) \\
  w &= w - \alpha dw \\
  b &= b - \alpha db
\end{align*}
\]
Neural “Networks”
Neural Networks

• So far, we have seen only a “single neuron” (with two models)!
  • Linear model
  • Logistic regression model
• The performance of a single unit (single neuron) is limited!
• Higher performance can be achieved by forming a network of multiple neurons.
A Neuron vs. A Neural Network

\[ z^{[1]} = W^{[1]}x + b^{[1]} \]
\[ a^{[1]} = \sigma(z^{[1]}) \]
\[ z^{[2]} = W^{[2]}a^{[1]} + b^{[2]} \]
\[ a^{[2]} = \sigma(z^{[2]}) \]
\[ \mathcal{L}(a^{[2]}, y) \]

\[ z = w^T x + b \]
\[ a = \sigma(z) \]
\[ \mathcal{L}(a, y) \]
A 2-layer FC-NN Example:

Each layer has its own weights and bias values. So… the k\textsuperscript{th} layer would have: \( W^{[k]} \) and \( b^{[k]} \)

\[ a^{[1]} = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \end{bmatrix}_{3x1} \]

\[ \hat{y} = a^{[2]} = \begin{bmatrix} a_1^{[2]} \end{bmatrix}_{1x1} = \begin{bmatrix} \hat{y}_1 \end{bmatrix}_{1x1} \]

The Weight Matrix for the entire layer 1:

\[ W^{[1]} = \begin{bmatrix} w_1^1 & w_1^2 & w_1^3 & w_1^4 & w_1^5 \\ w_2^1 & w_2^2 & w_2^3 & w_2^4 & w_2^5 \\ w_3^1 & w_3^2 & w_3^3 & w_3^4 & w_3^5 \end{bmatrix}^{[1]}_{3x5} \]

\( 1\text{st} \) unit in L1 (\( w_1^{[1]T} \))

\( 2\text{nd} \) unit in L1 (\( w_2^{[1]T} \))

\( 3\text{rd} \) unit in L1 (\( w_3^{[1]T} \))

\( \mathbf{W}^{[1]} \) is a (3x5) matrix, \( b^{[1]} \) is a (3x1) vector

\( \mathbf{W}^{[2]} \) is a (1x3) matrix (i.e., a row vector), \( b^{[2]} \) is a (1x1) vector

\( x = a^{[0]} \)

Input Layer (L0)

Hidden Layer (L1)

Output Layer (L2)

\( \hat{y} = a^{[2]} \)

\( \hat{y} \)

(Reads: the weight vector of the 1st unit at the first layer)
What if I have more than 2 classes?

• In logistic regression we assumed that we had only two classes: Class 0 & Class 1.
• What if I have more than two classes?
• One typical approach is: using **one vs. all approach**.
  • (where, for example, you first consider Class 0 as one class and the combination of all the other classes as the “other” class. Then you consider Class 1 as one class and the combination of all the other classes as the “other class”,...)

• Another approach might be using **Softmax Regression** instead of logistic regression.
  • Define a new cost function and derive all the weight update rules according to that cost function.
Softmax Regression

• Remember Logistic Regression: We had only two classes (Class 0 and Class 1, i.e., $y^{(i)} \in \{0, 1\}$).

• Softmax Regression is the case where we have $K$ classes ($K > 2$) such that: $y^{(i)} \in \{1, \ldots, K\}$.

• Sigmoid function is no longer being used.

• Now the output can take $K$ different values rather than just two.

$$a_i = \hat{y}_i = g(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

Softmax Activation Function

$$\mathcal{L}(a, y) = - \sum_{j=1}^{K} y_j \log(a_j)$$

Softmax Loss Function

$$J(w, b) = \frac{1}{m} \sum_{j=1}^{m} \mathcal{L}(a^{(j)}, y^{(j)})$$

Softmax Cost Function
Lets have a look at an example:

\[ n: \text{total number of features} = 6 \]
A Fully Connected (FC) Neural Network (6 inputs, 1 output)

\[ x_1, x_2, x_3, x_4, x_5, x_6 \rightarrow \hat{y} \]

n: total number of features = 6
A Fully Connected (FC) Neural Network (6 inputs, 2 outputs)

- A "unit" (a neuron)

- $x_1, x_2, x_3, x_4, x_5, x_6$: total number of features = 6

- $\hat{y}_1, \hat{y}_2$: outputs
A 2-layer FC-NN Example:

\[ z^{[1]} = W^{[1]} x + b^{[1]} = \begin{pmatrix} w_1^1 & w_2^1 & w_3^1 & w_4^1 & w_5^1 \\ w_1^2 & w_2^2 & w_3^2 & w_4^2 & w_5^2 \\ w_1^3 & w_2^3 & w_3^3 & w_4^3 & w_5^3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} + \begin{pmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \end{pmatrix} = \begin{pmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \end{pmatrix} \]

\[ \hat{y} = a^{[2]} = \begin{pmatrix} a_1^{[2]} \\ a_2^{[2]} \end{pmatrix} = \begin{pmatrix} \hat{y}_1 \end{pmatrix} \]

The Weight Matrix for the entire layer 1:

\[ W^{[1]} = \begin{pmatrix} w_1^1 & w_2^1 & w_3^1 & w_4^1 & w_5^1 \\ w_1^2 & w_2^2 & w_3^2 & w_4^2 & w_5^2 \\ w_1^3 & w_2^3 & w_3^3 & w_4^3 & w_5^3 \end{pmatrix} \]

\[ a^{[1]} = \begin{pmatrix} a_1^{[1]} \\ a_2^{[1]} \end{pmatrix} = \begin{pmatrix} \sigma(z_1^{[1]}) \\ \sigma(z_2^{[1]}) \end{pmatrix} = \sigma(z^{[1]}) \]
A 2-layer FC-NN Example:

\[ \mathbf{a}^{[1]} = \sigma(\mathbf{z}^{[1]}) \]

\[ \mathbf{z}^{[1]} = \mathbf{W}^{[1]} \mathbf{x} + \mathbf{b}^{[1]} \]

Steps to compute the output for logistic regression for one input sample:

\[ \hat{y} = \mathbf{a}^{[2]} = \left[ \begin{array}{c} a_1^{[2]} \\ a_2^{[2]} \\ a_3^{[2]} \end{array} \right] = \left[ \begin{array}{c} \sigma(z_1^{[1]}) \\ \sigma(z_2^{[1]}) \\ \sigma(z_3^{[1]}) \end{array} \right] \]
Computation with less for-loop

\[x_1 \quad x_2 \quad x_3 \quad \hat{y}\]

\[X = \begin{bmatrix} x^{(1)} & x^{(2)} & \cdots & x^{(m)} \end{bmatrix}\]

\[A^{[1]} = \begin{bmatrix} a^{[1](1)} & a^{[1](2)} & \cdots & a^{[1](m)} \end{bmatrix}\]

\((Z^{[1]} \text{ has the same shape as } A^{[1]}))\]

Algorithm 1:

\[
\begin{align*}
\text{for } i = 1 \text{ to } m \\
z^{[1](i)} &= W^{[1]}x^{(i)} + b^{[1]} \\
a^{[1](i)} &= \sigma(z^{[1](i)}) \\
z^{[2](i)} &= W^{[2]}a^{[1](i)} + b^{[2]} \\
a^{[2](i)} &= \sigma(z^{[2](i)})
\end{align*}
\]

Algorithm 2:

\[
\begin{align*}
Z^{[1]} &= W^{[1]}X + b^{[1]} \\
A^{[1]} &= \sigma(Z^{[1]}) \\
Z^{[2]} &= W^{[2]}A^{[1]} + b^{[2]} \\
A^{[2]} &= \sigma(Z^{[2]})
\end{align*}
\]

\(m = \text{number of training samples}\)
Another 2 layer FC NN Example

Input Layer
With 6 features

Hidden Layer
with 4 units

Output Layer
with 2 units

A 2 layer NN with 6 inputs and 2 outputs

\[ \mathbf{x} = \mathbf{a}^{[0]} \]

\[ \mathbf{a}^{[1]} \]

\[ \mathbf{a}^{[2]} \]

\[ \hat{\mathbf{y}} = \mathbf{a}^{[2]} \]

\[ \hat{\mathbf{y}} = \mathbf{a}^{[2]} \]

\[ \mathbf{a}^{[1]} = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \\ a_4^{[1]} \end{bmatrix}_{4 \times 1} \]

\[ \hat{\mathbf{y}} = \begin{bmatrix} a_1^{[2]} \\ a_2^{[2]} \end{bmatrix}_{2 \times 1} = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \end{bmatrix}_{2 \times 1} \]

QUESTION: What are the dims of \( W^{[1]} \) and \( W^{[2]} \)?

Dims = (a x b); a=? b=?

a=4 and b=6 for \( W^{[1]} \)
Common Activation Functions

Sigmoid: \( a(z) = \frac{1}{1 + e^{-z}} \)

ReLU: \( a(z) = \max(0, z) \)

tanh: \( a(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \)

Leaky ReLU: \( a(z) = \max(0.01z, z) \)
Activation Function as: $g(z)$

Algorithm 2:

\[
Z^{[1]} = W^{[1]}X + b^{[1]}
\]

\[
A^{[1]} = \sigma(Z^{[1]})
\]

\[
Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}
\]

\[
A^{[2]} = \sigma(Z^{[2]})
\]
With or Without the Activation Function

Lets have a look at the case where we do not use any activation function. (That is also equivalent to setting $g(z^{[1]}) = z^{[1]}$)

\[ z^{[1]} = W^{[1]} x + b^{[1]} \]
\[ a^{[1]} = g(z^{[1]}) \]
\[ z^{[2]} = W^{[2]} a^{[1]} + b^{[2]} \]
\[ a^{[2]} = g(z^{[2]}) \]
\[ \hat{y} = a^{[2]} = \begin{bmatrix} a_1^{[2]} \\ \hat{y}_1 \end{bmatrix} = \begin{bmatrix} \hat{y}_1 \end{bmatrix} \]

The output is always a linear function of the input!

\[ \hat{y} = z^{[2]} = W^{[2]} z^{[1]} + b^{[2]} \]
\[ = W^{[2]} [W^{[1]} x + b^{[1]}] + b^{[2]} \]
\[ = W^{[2]} W^{[1]} x + W^{[2]} b^{[1]} + b^{[2]} \]
\[ = W x + b \]

\[ z^{[1]} = W^{[1]} x + b^{[1]} \]
\[ a^{[1]} = z^{[1]} \]
\[ z^{[2]} = W^{[2]} z^{[1]} + b^{[2]} \]
\[ a^{[2]} = z^{[2]} = W^{[2]} z^{[1]} + b^{[2]} \]
\[ = W^{[2]} [W^{[1]} x + b^{[1]}] + b^{[2]} \]
\[ = W^{[2]} W^{[1]} x + W^{[2]} b^{[1]} + b^{[2]} \]
\[ = W x + b \]
• Remember that the updating process of the parameters depends on the derivatives!

• That also depends on the derivative of the chosen activation function!
  • (we used sigmoid function previously in our logistic regression implementation).
Derivatives for the Activation Functions

ReLU: $a = g(z) = \max(0,z)$

- $g'(z) = \frac{dg(z)}{dz} = \begin{cases} 
0, & \text{if } z < 0 \\
1, & \text{if } z > 0 \\
\text{undefined, if } z = 0
\end{cases}$

Sigmoid: $a = g(z) = \frac{1}{1 + e^{-z}}$

- $g'(z) = \frac{dg(z)}{dz} = g(z)(1 - g(z)) = a(1 - a)$

Tanh: $a = g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$

- $g'(z) = \frac{dg(z)}{dz} = (1 - (\tanh(z))^2) = (1 - a^2)$

Leaky ReLU: $a = g(z) = \max(0.01z, z)$

- $g'(z) = \frac{dg(z)}{dz} = \begin{cases} 
0.01, & \text{if } z < 0 \\
1, & \text{if } z \geq 0
\end{cases}$
\[ db[2] = \frac{1}{m} \text{np. sum}(dZ[2], axis = 1, keepdims = True) \]
\[ dW[1] = \frac{1}{m} dZ[1] x^T \]
\[ db[1] = \frac{1}{m} \text{np. sum}(dZ[1], axis = 1, keepdims = True) \]