ANLY-590 Assignment 2

September 2020

1 Feedforward: Building a ReLU neural network

Consider the rectified linear activation function : $h_j = \max(0, a_j)$.

1. Draw a network with:
   - 2 inputs
   - 1 hidden layers with 4 hidden units and a
   - 1-class output (for binary classification)
2. Write out the mathematical equation for the output of this network (feel free to break the input-output relationship into multiple equations).
3. Write out the forward-pass function in python, call it `ff_nn__ReLu(...)`
4. Suppose you have the following set of weight matrices:

$$W^{(1)} = \begin{bmatrix} 1 & -1 & 0 & 1 \\ 0 & 0 & .5 & 1 \end{bmatrix} \qquad b^{(1)} = [0, 0, 1, 0]^T \tag{1}$$

$$V = \begin{bmatrix} 1 \\ 0 \\ -1 \\ 1 \end{bmatrix} \qquad c = [1] \tag{2}$$

$$\tag{3}$$

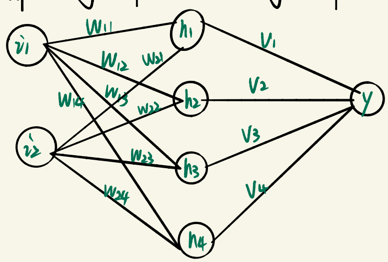
   and a few inputs:

$$X = \begin{bmatrix} 1 & -1 \\ 0 & -1 \\ 0 & 1 \end{bmatrix}$$

   what are the class probabilities associated with the forward pass of each sample?

1. Feedforward : Building a RELU neural network

1. input layer | hidden layer | output layer |



2. mathematical equation for input-output relationship :

The activation function is $h_j = \max(0, a_j)$ in hidden layer, and sigmoid function in output layer

$$y = \sigma(V_1 h_1 + V_2 h_2 + V_3 h_3 + V_4 h_4 + c)$$

$$\text{where} \begin{cases} h_1 = h_j(W_{11} i_1 + W_{21} i_2 + b_1) \\ h_2 = h_j(W_{12} i_1 + W_{22} i_2 + b_2) \\ h_3 = h_j(W_{13} i_1 + W_{23} i_2 + b_3) \\ h_4 = h_j(W_{14} i_1 + W_{24} i_2 + b_4) \end{cases}$$

In matrix form :

$$\vec{Y} = \sigma\left(\sum_{i=1}^{4} V_i h_i + C\right) = \sigma\left(\underbrace{\vec{V}^T_{1\times4}\, \vec{h}_{4\times1}}_{\text{vectors}} + C_{1\times1}\right)$$

$$\text{and} \quad \vec{h} = h_j\left(\sum_{i=1}^{4}(W_{1i} i_1 + W_{2i} i_2 + b_i)\right) = h_j(\vec{W}_{4\times2}\, \vec{I}_{2\times1} + b_{4\times1})$$

3. Write ff_nn_ReLu(...) in python :

Have written in .ipynb and test it.

4. What's the output given a set of weight matrices :

Have computed in .ipynb and make comments.

2 Gradient Descent

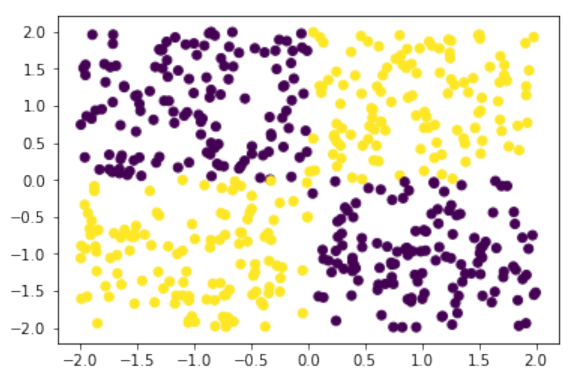
Consider a simple non-convex function of two variables:

$$f(x, y) = (3 - x^3) + 50 * (2y^2 - x)^2$$

1. What are the partial derivatives of f with respect to x and to y?
2. Create a visualization of the contours of this function.
3. Write a Gradient Descent algorithm for finding the minimum of the function. Visualize your results with a few different learning rates.
4. Write a Gradient Descent With Momentum algorithm for finding the minimum. Visualize your results with a few different settings of the algorithm's hyperparameters.

3 Backprop

1. For the same network as in Question 1, derive expressions of the gradient of the Loss function with respect to each of the model parameters.
2. Write a function `grad_f(...)` that takes in a weights vector and returns the gradient of the Loss at that location.
3. Generate a synthetic dataset like the XOR pattern (see below).
4. Fit your network using Gradient Descent. Keep track of the total Loss at each iteration and plot the result.
5. Repeat the exercise above using Momentum. Comment on whether your algorithm seems to converge more efficiently.
6. Plot a visualization of the final decision boundary that your model has learned. Overlay the datapoints in this plot.



2. Gradient Descent

$$f(x,y) = (3-x^3) + 50(2y^2-x)^2$$

1. What are the partial derivatives of f with respect to x and y?

$$\frac{\partial f}{\partial x} = -3x^2 + 50 \cdot 2 \cdot (2y^2-x) \cdot (-1)$$
$$= -3x^2 - 200y^2 + 100x$$

$$\frac{\partial f}{\partial y} = 50 \cdot 2 \cdot (2y^2-x) \cdot (4y)$$
$$= 400y(2y^2-x)$$

2. Create a visualization of f:

   See .ipynb for contours of f

3. Write Gradient Descent for finding minimum:

   see .ipynb

4. Write Gradient Descent + Momentum:

   see .ipynb

3. Backprop

1. gradient of loss function with respect to each parameter

$$\frac{\partial L}{\partial v_i} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial v_i} = -\left(\frac{y_i}{\hat{y}_i} - \frac{1-y_i}{1-\hat{y}_i}\right) \cdot out_i$$

$$= -\left(\frac{y_i}{\hat{y}_i} - \frac{1-y_i}{1-\hat{y}_i}\right) \cdot h_j(w_{1i} i_1 + w_{2i} i_2 + b_i)$$

$$\frac{\partial L}{\partial w_{ij}} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial out_j} \cdot \frac{\partial out_j}{\partial in_j} \cdot \frac{\partial in_j}{\partial w_{ij}}$$

$$= \begin{cases} -\left(\frac{y_i}{\hat{y}_i} - \frac{1-y_i}{1-\hat{y}_i}\right) \cdot v_j \cdot 0 \cdot x_i = 0 \text{ , if } out_j \leq 0 \\ -\left(\frac{y_i}{\hat{y}_i} - \frac{1-y_i}{1-\hat{y}_i}\right) \cdot v_j \cdot x_i \text{ , if } out_j > 0 \end{cases}$$

2. Write a function that returns gradient of the Loss
3. Generate XOR pattern:
4. Track Loss with Gradient Descent:
5. Repeat with Momentum:
6. Viz final decision boundary:

See ipynb