Deep Learning

Joseph Margaryan

January 25, 2024

1 Neural Network from Scratch

1.1 Neural Network Architecture

We start off with our features as arrays

$$x_1, x_2 \cdots x_k$$

```
# Extract features
    feature1 = [50,25,75]
    feature2 = [0.2,0.01,1.2]
    feature3 = [756 938 683]

# Reshape to 2-dimenions (k, 1) where k is datapoints and 1 is batch dimension
    featrue1 = np.array(featrue1, ndmin=2).T
    featrue2 = np.array(featrue2, ndmin=2).T
    featrue3 = np.array(featrue3, ndmin=2).T

# Concatenate features along the second axis
    concatenated_features = np.concatenate((feature1, feature2, feature3), axis=1)

# Convert to PyTorch tensor for compatibility with the neural network
    features_tensor = torch.tensor(concatenated_features, dtype=torch.float32)
```

Target value

$$y$$

```
    target = [0, 1, 2]
    target. np.array(target, dmin=2).T
    target = torch.tensor(target)
```

Our input layer consists of the count of features. We can then use a Linear function to the features

$$y = wx + b$$

y is the output
x is the input features
w is a matrix of weights
b is the bias term

$$x_1 = \begin{bmatrix} x_{11} \\ x_{12} \\ \vdots \\ x_{1k} \end{bmatrix}, x_2 = \begin{bmatrix} x_{21} \\ x_{22} \\ \vdots \\ x_{2k} \end{bmatrix}, x_3 = \begin{bmatrix} x_{31} \\ x_{32} \\ \vdots \\ x_{3k} \end{bmatrix}, w = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ \vdots & \vdots & \vdots \\ w_{1k} & w_{2k} & w_{3k} \end{bmatrix}, b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_k \end{bmatrix}$$

in our case the linear function would take in the input features and an output feature

```
linear = nn.Linear(3, 3)
linear(features_tensor)
```

The operation that would take place here would look like this

$$\hat{y} = \begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{bmatrix} \cdot \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

We would then wrap an activation function around this whole operation. Different activation functions include:
ReLU (Rectified Linear Unit)
Tanh (Hyperbolic tangent)

$$ReLU_{x_i} = max(0, x)$$

$$ReLU_{x_i} = \begin{cases} x, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases}$$

$$Tanh = \frac{e^{2x} - 1}{e^{2x} + 1}$$

The activation function is denoted as sigma σ
so our expression will look like this:

$$y = RelU(Linear(x, \textbf{output features})$$

$$y = \text{ReLU}(Wx + b)$$

$$y_i = \text{ReLU}\left(\sum_{j=1}^{n} W_{ij}x_j + b_i\right)$$

Adding a hidden layer to our network would add complexity to our model and make our model recognize more complex patterns in our data since we

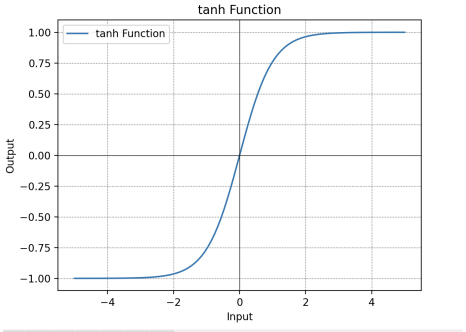

Figure 1: Tanh

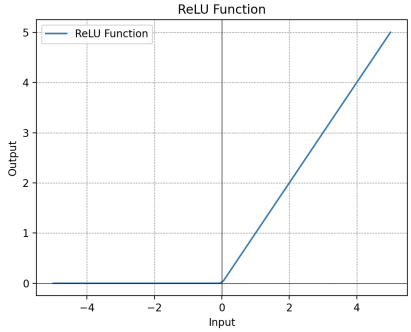

Figure 2: ReLU

would be adding another layer of weight. Each weight is a parameter to our model that aims to capture patterns and trends in our data
If we add a hidden layer to our network with another Linear function and another ReLU activation function, our new formula could be expressed as follows:

$$y = \text{ReLU}(W_2 \cdot \text{ReLU}(W_1 \cdot x + b_1) + b_2)$$

The underlying operation would look like this:

$$\begin{aligned}
\text{Linear Transformation (Hidden Layer 2):} \quad & Z_2 = \text{Linear}(A_1, W_2, b_2) \\
& Z_2 = A_1 \cdot W_2 + b_2 \\
& Z_2 = \text{ReLU}(\text{Linear}(\text{ReLU}(\text{Linear}(X, W_1, b_1)), W_2, b_2))
\end{aligned}$$

$$\begin{aligned}
\text{ReLU Activation (Hidden Layer 2):} \quad & A_2 = \text{ReLU}(Z_2) \\
& A_2 = \begin{cases} Z_2, & \text{if } Z_2 > 0 \\ 0, & \text{otherwise} \end{cases}
\end{aligned}$$

So we start off by having our feature matrix:

$$\begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{21} & x_{22} & x_{23} \end{bmatrix}$$

First hidden layer:

$$Z_1 = X * W_1 + B_1 = \begin{bmatrix} x_{11} \cdot w_{11} + x_{12} \cdot w_{12} + x_{13} \cdot w_{13} + b_1 \\ x_{21} \cdot w_{21} + x_{22} \cdot w_{22} + x_{23} \cdot w_{23} + b_1 \\ x_{31} \cdot w_{31} + x_{32} \cdot w_{32} + x_{33} \cdot w_{32} + b_1 \end{bmatrix}$$

$$A_1 = ReLU(Z_1) = \begin{bmatrix} ReLU(x_{11} \cdot w_{11} + x_{12} \cdot w_{12} + x_{13} \cdot w_{13} + b_1) \\ ReLU(x_{21} \cdot w_{21} + x_{22} \cdot w_{22} + x_{23} \cdot w_{23} + b_1) \\ ReLU(x_{31} \cdot w_{31} + x_{32} \cdot w_{32} + x_{33} \cdot w_{32} + b_1) \end{bmatrix}$$

Second hidden layer Here we use the weights as connections between the layers. These weights corresponds to the strength of connection between the neurons

$$Z_2 = A_1 * W_2 + b_2 = \begin{bmatrix} A_1 \cdot w_{11} + A_2 \cdot w_{21} + A_3 \cdot w_{31} + b_{1,2} \\ A_1 \cdot w_{12} + A_2 \cdot w_{22} + A_3 \cdot w_{32} + b_{2,2} \\ A_1 \cdot w_{13} + A_2 \cdot w_{23} + A_3 \cdot w_{33} + b_{3,2} \end{bmatrix}$$

$$A_2 = ReLU(Z_2 = \begin{bmatrix} ReLU(A_1 \cdot w_{11} + A_2 \cdot w_{21} + A_3 \cdot w_{31} + b_{1,2}) \\ ReLU(A_1 \cdot w_{12} + A_2 \cdot w_{22} + A_3 \cdot w_{32} + b_{2,2}) \\ ReLU(A_1 \cdot w_{13} + A_2 \cdot w_{23} + A_3 \cdot w_{33} + b_{3,2}) \end{bmatrix}$$

based on our problem, wee need to choose the appropriate activation function

for our output layer. If we are dealing with a binary classification problem, we may use a **sigmoid function**, if were dealing with a multiclass classification we may use a **softmax function**, if we are dealing with a regression problem, we may use a **Linear function**

- **Sigmoid** for binary classification
- **Softmax** for multiclass classification
- **Linear** for regression
- **Sigmoid function (σ)** for binary classification:

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

- **Softmax function** for multiclass classification:

$$\text{Softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}} \quad \text{where } K \text{ is the number of classes}$$

- **Linear function** for regression:

$$f(x) = wx + b$$

The number of neurons in our output layer needs to correspond to the number of target classes we are dealing with
In our example, the neural network architecture would look like this:

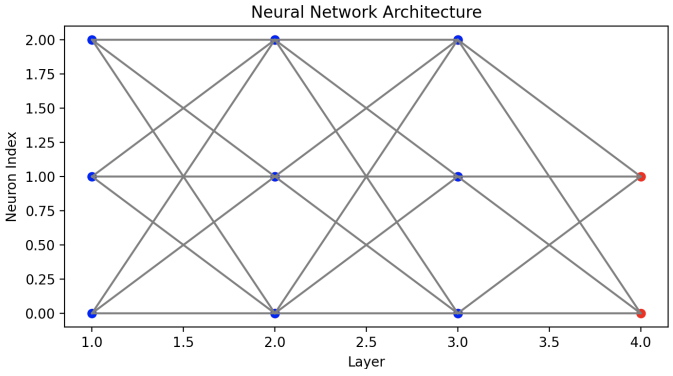

Figure 3: Neural Network Architecture

2 Training and Testing

2.1 Cost Functions in Neural Networks

1. **Mean Squared Error (MSE) / L2 Loss:**

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_{\text{true}}^{(i)} - y_{\text{pred}}^{(i)})^2$$

2. **Mean Absolute Error (MAE) / L1 Loss:**

$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}|y_{\text{true}}^{(i)} - y_{\text{pred}}^{(i)}|$$

3. **Binary Cross Entropy Loss:**

$$\text{BCE} = -\frac{1}{N}\sum_{i=1}^{N}\left[y_{\text{true}}^{(i)} \cdot \log(y_{\text{pred}}^{(i)}) + (1 - y_{\text{true}}^{(i)}) \cdot \log(1 - y_{\text{pred}}^{(i)})\right]$$

4. **Categorical Cross Entropy Loss:**

$$\text{CE} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C} y_{\text{true},c}^{(i)} \cdot \log(y_{\text{pred},c}^{(i)})$$

5. **Hinge Loss:**

$$\text{Hinge} = \frac{1}{N}\sum_{i=1}^{N}\max(0, 1 - y_{\text{true}}^{(i)} \cdot y_{\text{pred}}^{(i)})$$

6. **Huber Loss:**

$$\text{Huber} = \frac{1}{N}\sum_{i=1}^{N}\begin{cases}\frac{1}{2}(y_{\text{true}}^{(i)} - y_{\text{pred}}^{(i)})^2, & \text{if } |y_{\text{true}}^{(i)} - y_{\text{pred}}^{(i)}| \leq \delta \\ \delta \cdot |y_{\text{true}}^{(i)} - y_{\text{pred}}^{(i)}| - \frac{1}{2}\delta^2, & \text{otherwise}\end{cases}$$

2.2 Backpropagation

Our first goal is to understand how sensitive our Cost function is to small changes in our weights.
Mathematically, we can rephrase it as: What is the partial derivative of C with respect to W

$$\frac{\partial C_0}{\partial w^L} = \frac{\partial z^L}{\partial w^L}\frac{\partial a^L}{\partial z^L}\frac{\partial C_0}{\partial a_L}$$

where L denotes the L'th layer in the network
If we denote the cost function to be:

$$C_0 = (a^L - y)^2$$

then

$$\frac{\partial C_0}{\partial a^L} = 2(a^L - y)$$

if

$$a^L = \sigma(z^L)$$

then the derivative is just the derivative of the activation function, sigmoid, softmax etc

$$\frac{\partial a^L}{\partial z^L} = \sigma'(z^L)$$

if

$$z^L = w^L a^{(L-1)} + b^L$$

then

$$\frac{\partial z^L}{\partial w^L} = a^{L-1}$$

This is an example of a specific single training example.
The full cost function involves averages together all those costs across many different training examples:

$$\frac{\partial C}{\partial w^L} = \sum_{k=0}^{n-1} \frac{\partial C_k}{\partial w^L}$$

That is just one component of the gradient vector

$$\nabla C = \begin{bmatrix} \frac{\partial C}{\partial w^1} \\ \\ \frac{\partial C}{\partial b^1} \\ \\ \vdots \\ \frac{\partial C}{\partial w^L} \\ \\ \frac{\partial C}{\partial b^L} \end{bmatrix}$$