CNNs From Scratch

Advay Singh

CONTENTS

0.1. Introduction

0.2. Image Processing

0.2.1. Convolution.

0.2.2. Pooling.

0.3. Fully-Connected Multi Layer Perception

Now, with our inputs in place, will begin discussion regarding the a Fully-Connected Multi Layer Perception.

$$x_i = ith \text{ input} \tag{1}$$
$$w_{ij}^k = \text{weight from } ith \text{ input to } jth \text{ output in the } kth \text{ layer} \tag{2}$$
$$b_{ji}^k = \text{bias from } ith \text{ input to } jth \text{ output in the } kth \text{ layer} \tag{3}$$
$$z_j^k = jth \text{ input for activation at layer } k \tag{4}$$
$$\alpha_j^k = jth \text{ activated val at layer } k \tag{5}$$
$$y_i = ith \text{ true label} \tag{6}$$
$$l = \text{number of hidden layers} \tag{7}$$
$$n^k = \text{number of activations for layer } k \tag{8}$$
$$Y_i = ith \text{ predicted output} \tag{9}$$
$$E_i = ith \text{ error} \tag{10}$$
$$\Delta w = \text{change to weights} \tag{11}$$
$$\Delta b = \text{change to biases} \tag{12}$$
$$a = \text{learning rate.} \tag{13}$$

0.3.1. Forward Propagation. The x and y are both lists of inputs and labels respectively. Other terms, require derivation.

0.3.1.1. *Weights and Biases.* Weights and Biases are both 2-dimensional arrays. Note that there are multiple of these, intact, $l+1$ of each in a Fully-Connected Neural Network.
Initially, these arrays are filled with random values which are eventually redfined through gradient-decent. The length of these arrays can be determined through matrix multiplication. The matrix w^k's aXb where $a = n^{k-1}$ and $b = n^k$. For this, the initial layer $k-1$ is reshaped to $n^{k-1} \cdot 1$ and k is reshaped $1 \cdot n^k$. It's a similar process for the biases.

0.3.1.2. *Zs (Activation Inputs).* With the weights biases, and inputs in-place, we may derive our zs.

$$z_j^k = \sum_i x_i^{k-1} w_{ij}^k. \tag{14}$$

Note we need not derive zs for $k \geq l$.

0.3.1.3. *Activation Function.* There are many activation functions. ReLU, Sigmoid, and TanH to name a few. The following defines the **Sigmoid Activation Function**, $\sigma(z) : \mathbb{R} \to (0,1)$. The reason for these function is to **introduce non-linearity** in the dataset. The more hidden layers there are, the more complex problems the CNN is capable of solving. The cost? Increased computation time.

$$\alpha_j^k = \sigma(z_j^k) = \frac{1}{1 + e^{-(z_j^k)}}. \tag{15}$$

Note that $\forall \alpha,\ \alpha \in (0,1)$. Such normalization is often seen in activation functions. However, the purpose is the non-linear nature of the sigmoid graph.
We also have the **ReLU** activation function: $R'(x) =$

$$\begin{cases} 0 & x < 0 \\ x & x \geq 0. \end{cases}$$

More on this as we compute their derivatives.

0.3.1.4. *Softmax Activation Function.* Because $\forall \alpha,\ \alpha \in (0,1)$ it's impossible to use them to predict values outside that range or non-numerical objects. In the latter case, we apply **Softmax** to the z^l list while a simple regression activation function often suffices for numerical classifications.

$$Y_i = S(x_i) = \frac{e^{x_i}}{\sum_j^{nj} e^{x_j}}. \tag{16}$$

Due to the Softmax function, $\forall Y_i, Y_i \in (0,1)$ meaning that $S(x) : \mathbb{R} \to [0,1]$. To compare this to y_i, we convert $\forall y_i \in \{0,1\}$. Where only the true classification $y_{true} = 1$ while every other $y_i = 0$.

With this, we have the forward propagation function can be generated. The inputs for the function are x, the inputs and true labels. The outputs: Y an array of the predicted outputs. Additionally, other functors and parameters can be used, including which activation function, etc.

0.3.2. Back Propagation & Gradient Decent. Firstly, we must convert the labels y_i to values $\in \{0,1\}$

0.3.2.1. *Error Function.* The error function compares $Y_i \wedge y_i$ and is responsible for generating $\Delta w \wedge \Delta b$ for gradient decent. For this, we define a cost function.

$$E_i = C(i) = \frac{(Y_i - y_i)^2}{2} \tag{17}$$

In our computations, we use

$$E_{tot} = C(n^l) = \sum_{i}^{n^l} \frac{(Y_i - y_i)^2}{2} \tag{18}$$

Therefore, through each iteration, we derive $\Delta w_{ji}^l \wedge \Delta b_{ji}^l$ based on E_i. And update the w and b through back propagation. Iterating through the training set is considered one **epoch**. Notably, every epoch increases accuracy for regular gradient decent, however, **Stochastic Gradient Decent**'s accuracy plateaus over time. More on the later.

0.3.2.2. *Calculating Δw And Δb.* Firstly, we begin with our final layer. $\frac{\delta E_i}{\delta w_i j}$ and $\frac{\delta E_i}{\delta b_i j}$.
Breaking down the function:

$$\frac{\delta E_i}{\delta wij} = \frac{\delta E_i}{\delta S(z_i)} \cdot \frac{\delta S(z_i)}{\delta z_i^l} \cdot \frac{\delta z_i^l}{\delta w_{ij}}. \tag{19}$$

Hence we derive all the parts.

$$\frac{\delta E_i}{\delta S(x_i)} = \frac{\delta}{\delta S(x_i)} \frac{(Y_i - y_i)^2}{2} \tag{20}$$

$$= \frac{\delta}{\delta S(x_i)} \frac{(S(x_i) - y_i)^2}{2} \tag{21}$$

$$= (S(x_i) - y_i) \tag{22}$$

$$= (Y_i - y_i). \tag{23}$$

Note that for our purposes, x_i refers to z_i^l. For the following calculation, l is implicit for z_i because we are deriving for the final layer.

$$\frac{S(z_i)}{\delta z_i} = S(z_i)(1 - S(Z_i)). \tag{24}$$

$J_{softmax}$ is known as the **Jacobian Matrix** and as modeled like such

$$\begin{bmatrix} \frac{\delta S(1)}{z_1} & \frac{\delta S(1)}{\delta z_2} & \cdots & \frac{\delta S(1)}{\delta z_i} \\ \frac{\delta S(2)}{\delta z_1} & \frac{\delta S(2)}{\delta z_2} & \cdots & \frac{\delta S(2)}{\delta z_i} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\delta S(i)}{\delta z_1} & \frac{\delta S(i)}{\delta z_2} & \cdots & \frac{\delta S(i)}{\delta z_i} \end{bmatrix}$$

where $i = n^l$. We will derive the $S(x_i)$ function later.

Now we derive Δw for the final layer (layer going from jth input to ith output.

$$\frac{\delta z_i}{\delta w_{ij}} = \frac{\delta}{\delta w_{ij}} \sum_{p}^{n^{i-1}} x_p w_i p + b_i p \tag{25}$$

Note that for $f(w) = \dfrac{\delta}{\delta w_{ij}} \displaystyle\sum_{p}^{n^{i-1}} x_p w_i p + b_i p$, $(p = j) \iff f(z) = x_j$ and $(j \neq i \iff f(z) = 0)$. Therefore,

$$\frac{\delta}{\delta w_{ij}} \sum_{p}^{n^{i-1}} x_p w_i p + b_i p = x_j \tag{26}$$

Note that the $\dfrac{\delta E_i}{\delta S(x_i)} \cdot \dfrac{\delta S(x_i)}{\delta z_i^l}$ remain the same for Δb, however, $\dfrac{\delta z_i}{\delta b_{ij}} = 1$. So we have

$$\Delta w_{ij} = \frac{\delta E_i}{\delta w_{ij}} = (Y_i - y_i) \cdot S(z_i)(1 - S(z_i)) \cdot x_j \tag{27}$$

$$\Delta b_{ij} = \frac{\delta E_i}{\delta b_{ij}} = (Y_i - y_i) \cdot S(z_i)(1 - S(z_i)). \tag{28}$$

0.3.2.3. *Softmax Derivative and Jacobin Matrix.* Our goal in this segment is to define $\dfrac{dS(z_i)}{dz_i}$. Firstly,

$$\frac{\delta \ln(S(z_i))}{\delta z_i} = \frac{1}{S(z_i)} \frac{\delta S(z_i)}{\delta z_i}. \tag{29}$$

Hence, we initially compute $\frac{\delta \ln(S(z_i))}{\delta z_i}$.

$$\frac{\delta \ln(S(z_i))}{\delta z_i} = \frac{\delta}{\delta z_i} \ln(\frac{e^{z_i}}{\sum_j^{n^j} e^{z_j}}) \tag{30}$$

$$= \frac{\delta}{\delta z_i} \ln(e^{z_i}) - \frac{\delta}{\delta z_i} \ln(\sum_j^{n^j} e^{z_j}) \tag{31}$$

$$= 1 - \frac{1}{\sum_j^{n^j} e^{z_j}} \frac{\delta}{\delta z_i}(\sum_j^{n^j} e^{z_j}). \tag{32}$$

Note that for $g(z) = \frac{\delta}{\delta z_i}(\sum_j^{n^j} e^{z_j})$, $(j = i) \iff g(z) = 1$ and $(j \neq i \iff g(z) = 0)$. Therefore,

$$\frac{\delta \ln(S(z_i))}{\delta z_i} = 1 - S(z_i). \tag{33}$$

And so we have that

$$\frac{\delta S(z_i)}{\delta z_i} = S(z_i)\frac{\delta \ln(S(z_i))}{\delta z_i} \tag{34}$$

$$= S(z_i)(1 - S(z_i)). \tag{35}$$

There is also an idea of **Temperature** T which adds another layer of non-linearity to S. A smaller temperature favors greater z_i and vice-versa.

$$Y_i = S(x_i) = \frac{e^{\frac{x_i}{t}}}{\sum_j^{n^j} e^{\frac{x_j}{t}}} \tag{36}$$

0.3.2.4. *Activation Function Derivative.* The derivative of the activation function plays a similar role as the Softmax function derivative, just for earlier, hidden layers. For now, we will be focusing on the Sigmoid function.

$$\frac{\delta}{\delta z_i}\alpha_i = \frac{\delta}{\delta z_i}\sigma(z_i) \tag{37}$$

$$= \frac{\delta}{\delta z_i}\frac{1}{1+e^{-(z_i)}} \tag{38}$$

$$= \frac{\delta}{\delta z_i}(1+e^{-(z_i)})^{-1} \tag{39}$$

$$= \frac{-1}{(1+e^{-(z_i)})^2}\frac{\delta}{\delta z_i}e^{-(z_i)} \tag{40}$$

$$= \frac{-1}{(1+e^{-(z_i)})^2} - e^{-(z_i)} \tag{41}$$

$$= \frac{e^{-(z_i)}}{(1+e^{-(z_i)})^2} \tag{42}$$

$$= \frac{1}{1+e^{-(z_i)}} \cdot \frac{e^{-(z_i)}}{1+e^{-(z_i)}} \tag{43}$$

$$= \frac{1}{1+e^{-(z_i)}} \cdot (1 - \frac{1}{1+e^{-(z_i)}}) \tag{44}$$

$$= \sigma(z_i)(1-\sigma(z_i)). \tag{45}$$

Look familiar? It's the same as S.
Notably, calculating the derivative of the ReLU function is much simpler. $R'(x) =$

$$\begin{cases} 0 & x < 0 \\ 1 & x \geq 0. \end{cases}$$

Computationally, this often yields a much greater $\Delta w \wedge \Delta b$ allowing for a much faster back propagation process. Hence, we can now proceed to

calculate $\Delta w \wedge \Delta b$ for the hidden layers (layers $< l$). The following computes δw^1_{ji} in a Neural Network with two layers ($l = 2$) with p inputs.

$$\frac{\delta E_i}{\delta w^1_{jp}} = \frac{\delta E_i}{\delta S(z^l_i)} \cdot \frac{\delta S(z^l_i)}{\delta z^l_i} \cdot \frac{\delta z^l_i}{\delta w^1_{jp}} \tag{46}$$

$$= \frac{\delta E_i}{\delta S(z^l_i)} \cdot \frac{\delta S(z^l_i)}{\delta z^l_i} \cdot w^l_{ij} \cdot \frac{\delta a^1_j}{\delta w^1_{jp}} \tag{47}$$

$$= \frac{\delta E_i}{\delta S(z^l_i)} \cdot \frac{\delta S(z^l_i)}{\delta z^l_i} \cdot w^l_{ij} \cdot \frac{\delta \sigma(z^1_j)}{\delta z^1_j} \cdot \frac{\delta z^1_j}{\delta w^1_{jp}} \tag{48}$$

$$= (Y_i - y_i) \cdot S(z^2_i)(1 - S(z^2_i)) \cdot w^2_{ij} \cdot \sigma(z^1_j)(1-\sigma(z^1_j)) \cdot x_p. \tag{49}$$

This way, we can also obtain Δb.

$$\frac{\delta E_i}{\delta b^1_{jp}} = \frac{\delta E_i}{\delta S(z^l_i)} \cdot \frac{\delta S(z^l_i)}{\delta z^l_i} \cdot \frac{\delta z^l_i}{\delta b^1_{jp}} \tag{50}$$

$$= \frac{\delta E_i}{\delta S(z^l_i)} \cdot \frac{\delta S(z^l_i)}{\delta z^l_i} \cdot w^l_{ij} \cdot \frac{\delta a^1_j}{\delta b^1_{jp}} \tag{51}$$

$$= \frac{\delta E_i}{\delta S(z^l_i)} \cdot \frac{\delta S(z^l_i)}{\delta z^l_i} \cdot w^l_{ij} \cdot \frac{\delta \sigma(z^1_j)}{\delta z^1_j} \cdot \frac{\delta z^1_j}{\delta b^1_{jp}} \tag{52}$$

$$= (Y_i - y_i) \cdot S(z^2_i)(1 - S(z^2_i)) \cdot w^2_{ij} \cdot \sigma(z^1_j)(1 - \sigma(z^1_j)). \tag{53}$$

Now that we know how to compute $\Delta w \wedge \Delta b$ for all matrices, it's essential to understand how to automate the process for any computational application.

0.3.2.5. *Stochastic Gradient Decent.* The learning rate a has an impact but can cause for less overall accuracy.

0.4. Efficiency Testing

0.5. Application in Full-Stack Web Application