MLP and CNN

Yiangos Georgiou
11807288
University of Amsterdam
ygeorgiou12@gmail.com

1 MLP backprop and NumPy implementation

1.1 Analytical derivation of gradients

1.1.1 Question 1.1.a

1) Cross-Entropy Loss

We can remove the sum because our result is one-hot vector.

$$\begin{aligned}\frac{\partial L}{\partial x^{(N)}} &\Rightarrow \frac{\partial(-\log x^{(N)}_{argmax(t)})}{\partial x^{(N)}}\\ &\Rightarrow \frac{-1}{x_{argmax(t)}}\\ &\delta_{out} dimensions = [1, N]\end{aligned}$$

2) Soft-max Function

Since soft-max is a $^N \rightarrow ^N$ function, the most general derivative we compute for it is the Jacobian matrix.

$$\begin{aligned}\frac{\partial x_i^{(N)}}{\partial \hat{x}_j^{(N)}} &\Rightarrow \frac{\partial(\frac{\exp(\hat{x}^{(N)})}{\sum_{k=1}^{dN}\exp(\hat{x}^{(N)})})}{\partial \hat{x}^{(N)}}\\ &\Rightarrow \frac{(\exp(\hat{x}_i^{(N)}))\prime * \sum_{k=1}^{dN}\exp(\hat{x}^{(N)}) - \exp(\hat{x}_i^{(N)}) * (\sum_{k=1}^{dN}\exp(\hat{x}^{(N)}))\prime}{\sum_{k=1}^{dN}\exp(\hat{x}^{(N)})^2}\\ &\Rightarrow \frac{(\exp(\hat{x_i}^{(N)}))\prime * \sum \hat{x}^{(N)} - \exp(x_i) * \exp(x_j)}{\sum_{k=1}^{dN}\exp(\hat{x}^{(N)})^2}\\ \text{if i==j} &\Rightarrow \frac{\exp(\hat{x}_i^{(N)}}{\sum_{k=1}^{dN}\exp(\hat{x}^{(N)})}\frac{\sum_{k=1}^{dN}\exp(\hat{x}^{(N)}) - \exp\hat{x}_j}{\sum_{k=1}^{dN}\exp(\hat{x}^{(N)})}\\ &\Rightarrow Softmax(\hat{x}_i)(1 - Softmax(\hat{x}_j))\end{aligned}$$

$$\begin{aligned}\text{if i!=j} &\Rightarrow \frac{(\exp(\hat{x_i}^{(N)}))\prime * \sum \hat{x}^{(N)} - \exp(x_i) * \exp(x_j)}{\sum_{k=1}^{dN}\exp(\hat{x}^{(N)})^2}\\ &\Rightarrow \frac{0 * \exp(x_j)}{\sum_{k=1}^{dN}\exp(\hat{x}^{(N)})^2}\\ &\Rightarrow -Softmax(\hat{x}_i^{(N)}) * Softmax(\hat{x}_j^{(N)})\end{aligned}$$

The jacobian matrix's dimensions are [N x N] where N is the number of outputs.

3) Relu Function

For the Relu function we do not need to create a jacobian matrix due to the fact that the output depends only in a single input.

$$\frac{\partial x_i^{(l<N)}}{\partial x_j^{(l<N)}} \Rightarrow \frac{\partial(max(0,x_i))}{\partial x_j^{(l<N)}}$$
$$\text{if i==j } \Rightarrow x\prime \Rightarrow 1$$
$$\text{if i!=j } \Rightarrow 0\prime \Rightarrow 0$$

4) Linear Function w.r.t. input

$$\frac{\partial \hat{x}_i^{(l)}}{\partial x_j^{(l-1)}} \Rightarrow \frac{\partial(W^{(l)} * x^{(l-1)} + b^{(l)})}{\partial x_j^{(l-1)}}$$
$$\Rightarrow W_j^{(l)}$$

5) Linear Function w.r.t. weights

$$\frac{\partial \hat{x}_i^{(l)}}{\partial W_{jk}^{(l)}} \Rightarrow \frac{\partial(W^{(l)} * x^{(l-1)} + b^{(l)})}{\partial W_{jk}^{(l)}}$$
$$\Rightarrow x_k^{(l-1)}$$

6) Linear Function w.r.t. biases

$$\frac{\partial \hat{x}_i^{(l)}}{\partial b_j^{(l)}} \Rightarrow \frac{\partial(W^{(l)} * x^{(l-1)} + b^{(l)})}{\partial b_j^{(l)}}$$
$$\text{if i==j } \Rightarrow b_{ij}\prime \Rightarrow 1$$
$$\text{if i!=j } \Rightarrow 0$$

1.1.2 Question 1.1.b

1)

$$\frac{\partial L}{\partial \hat{x}^{(N)}} = \frac{\partial L}{\partial x^{(N)}}\frac{\partial x^{(N)}}{\partial \hat{x}^{(N)}} \Rightarrow \underset{1\times N}{\delta_{out}} \cdot \underset{N\times N}{Jacobian\_matrix}$$
$$\Rightarrow \underset{N\times 1}{\delta^{(N)}}$$

$$Jacobian\_matrix = \begin{bmatrix} S_1(\delta_{11}-S_1) & S_1(\delta_{12}-S_2) & S_1(\delta_{13}-S_3) & \dots & S_1(\delta_{1N}-S_N) \\ \dots & \dots & \dots & \dots & \dots \\ S_N(\delta_{N1}-S_1) & S_N(\delta_{N2}-S_2) & S_N(\delta_{N3}-S_3) & \dots & S_N(\delta_{NN}-S_N) \end{bmatrix}$$
$$\text{if i==j } \delta_{ij} = 1$$
$$\text{else } \delta_{ij} = 0$$

2)

$$\frac{\partial L}{\partial \hat{x}^{l<N}} = \frac{\partial L}{\partial x^{(l)}}\frac{\partial x^{(l)}}{\partial \hat{x}^{(l)}} \Rightarrow \underset{N\times 1}{\delta_{out}} * \underset{N\times 1}{relu\prime(x)}$$
$$\underset{N\times 1}{\frac{\partial L}{\partial \hat{x}^{l}}} \Rightarrow \begin{bmatrix} \delta_1 * relu\prime(x_1) \\ \delta_2 * relu\prime(x_2) \\ \dots \\ \delta_{dim_l} * relu\prime(x_l) \end{bmatrix}$$

3) Linear backward w.r.t inputs

$$\frac{\partial L}{\partial x^{(l+1)}} = \frac{\partial L}{\partial \hat{x}^{(l+1)}} \frac{\partial \hat{x}^{(l+1)}}{\partial x^{(l)}} \Rightarrow \underset{dim_{l+1}\times 1}{\delta^{(l+1)}} \cdot \frac{\partial (W^{(l+1)} x^{(l)} + b^{l+1})}{\partial x^{(l)}}$$
$$\Rightarrow \underset{d_{l+1}\times d_l}{W^T} \underset{d_{l+1}\times 1}{\delta^{(l+1)}} \Rightarrow \underset{d_l \times 1}{\delta^l}$$

4) Linear backward w.r.t. weights

$$\frac{\partial L}{W^l} = \frac{\partial L}{\partial \hat{x}^{(l)}} \frac{\hat{x}^{(l)}}{\partial W^{(l)}} \Rightarrow \delta^{(l+1)} \cdot \frac{\partial (W^{(l)} x^{(l-1)} + b^{(l)})}{\partial W^{(l)}}$$
$$\Rightarrow \underset{d_l\times 1}{\delta^l} \cdot \underset{d_l\times 1}{x^l} \Rightarrow \underset{d_l\times 1}{x^l} \cdot \underset{d_l\times 1}{(\delta^l)^T} \Rightarrow \underset{d_l\times d_{l+1}}{\frac{\partial L}{\partial W}}$$

5) Linear backward w.r.t. biases

$$\frac{\partial L}{\partial b^{(l)}} = \frac{\partial L}{\partial \hat{x}^{(l)}} \frac{\partial \hat{x}^{(l)}}{\partial b^{(l)}} \Rightarrow \delta^{(l)} \cdot \frac{\partial (W^{(l)} x^{(l-1)} + b^l)}{\partial b^{(l)}}$$
$$\Rightarrow \underset{d_l\times 1}{\delta^{(l)}} \cdot 1 \Rightarrow \underset{d_l\times 1}{\frac{\partial L}{\partial b^{(l)}}}$$

1.1.3 Question 1.1.c

The difference is that the loss derivatives between the layers are matrices if B>1, so some equations have to be changed and summed up to fit on the back-propagation or on the updates of the weighs. For instance, on the update of the biases you have to sum the batch to get a vector, or in cases such as soft-max where if your batch size is more than 1 you have to multiply with a 3d-array. However, batch size it doesn't make a lot of different in learning or testing procedures, except in the implementation.

1.2 NumPy implementation

NumPy implementation based on the equations derived from the sections 1.1.a and 1.1.b. Test accuracy reached 46.5% with default parameter setup. Test and train accuracies can be seen in Figure 1.

2 PyTorch MLP

Experiments were focused on two optimizers, ADAM and SGD. illustrated on Figure 2 are the best models for these optimization algorithms. As it can be seen the ADAM optimizer outperform SGD. However, different parameters and architectures boost the performance of each one.

The architecture of our best SGD model contained 3 hidden layers with 300 neuron each, the learning rate was initialized to 0.05 and was reducing to half every 2000 iterations.

Regarding ADAM best model, its architecture contained 2 hidden layers with 300 and 200 neurons respectively. The learning rate was initialized to 0.0006 and was reducing to half every 2000 iterations. Batch normalization seem benefit this model.

Batch normalization : Batch normalization works well only for ADAM, and boost its accuracy to 51.5%. Although, that does not apply to SGD where the accuracy after the batch norm is getting worst.

Batch size: Although we kept the original batch size, we experimented with higher and lower values for this parameter. We observed that high batch size cause a drop of accuracy while it makes the model faster, and vice versa.

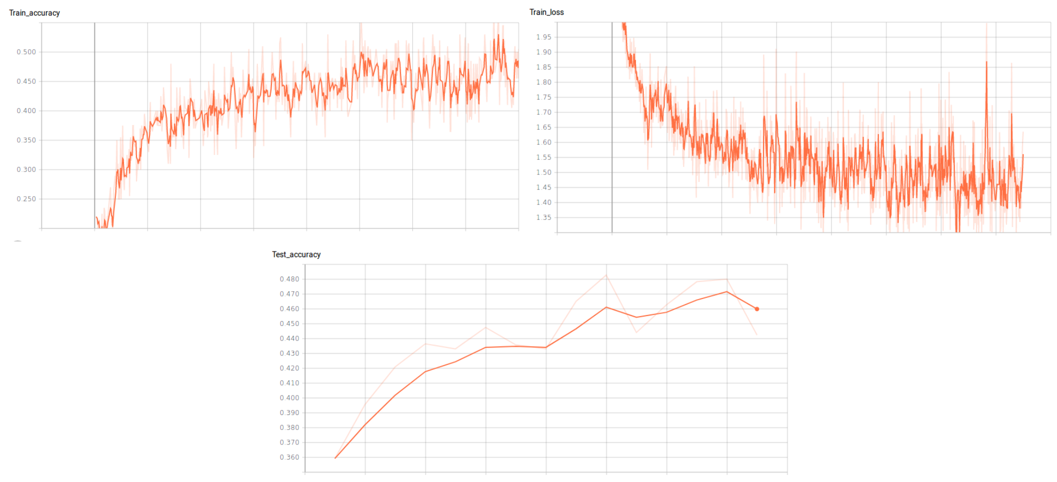

Figure 1: MLP loss and accuracy measurements (train accuracy (up,left), train loss (up,right), test accuracy(down,middle))

Learning rate: The current configuration is not the ideal for our model, so we tried to find the best learning rate for each model. The learning rate is affected by the size of the network and the optimization algorithm. We noticed that, if the network is too big is preferable to have a small learning rate. Moreover, higher accuracy was achieved when the learning rate was reducing over the iterations.

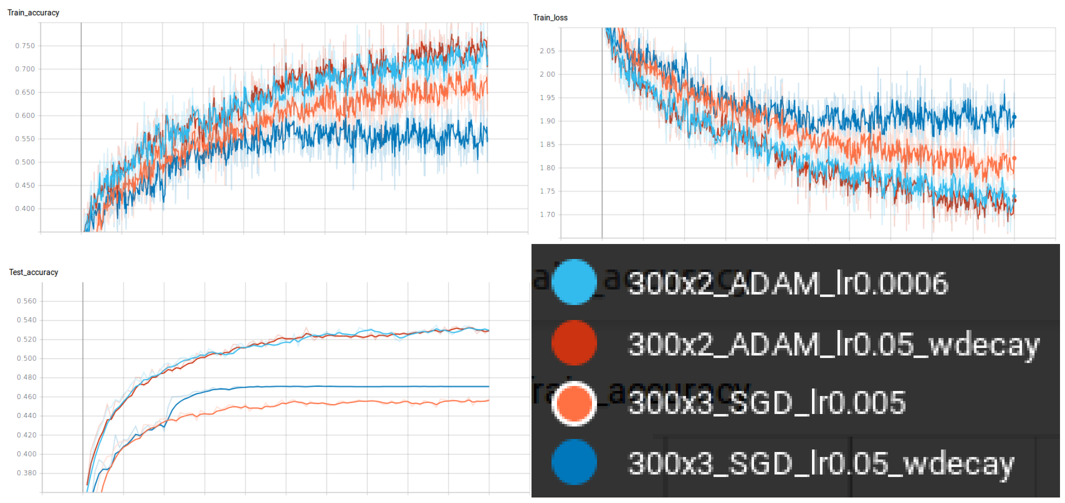

Figure 2: MLP pytorch loss and accuracy measurements (train accuracy (up,left), train loss (up,right), test accuracy(down,middle))

3 Custom Module: Batch Normalization

3.1 Automatic differentiation

3.2 Manual implementation of backward pass

1) gamma derivative

$$\frac{\partial L}{\partial \gamma} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial \gamma} \Rightarrow \delta_{out} * \frac{\partial(\gamma * \hat{x} + \beta)}{\partial \gamma}$$
$$\Rightarrow \underset{N\times D}{\delta_{out}} * \underset{N\times D}{\hat{x}} \Rightarrow \sum_{n=1}^{N} \delta_{outn} * \hat{x}_n \Rightarrow \underset{D\times 1}{\frac{\partial L}{\partial \gamma}}$$

2) beta derivative

$$\frac{\partial L}{\partial \beta} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial \beta} \Rightarrow \delta_{out} * \frac{\partial(\gamma * \hat{x} + \beta)}{\partial \beta} \Rightarrow \sum_{n=1}^{N} \delta_{outn} * 1$$

3) x derivative

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial x} \Rightarrow \delta_{out} * \frac{\partial(\gamma * \hat{x} + \beta)}{\partial x} \Rightarrow \delta_{out} * \frac{\partial(\gamma * \hat{x} + \beta)}{\partial \hat{x}}\frac{\partial \hat{x}}{x} \Rightarrow \delta_{out} * \frac{\partial(\gamma * \hat{x} + \beta)}{\partial \hat{x}}\frac{\partial(x - mean) * \frac{1}{\sqrt{\sigma^2+\epsilon}}}{\partial(x - mean)}$$
$$\Rightarrow dxm_1 = \delta_{out} * \gamma \frac{1}{\sqrt{\sigma^2 + \epsilon}}$$
$$\Rightarrow \delta_{out} * \gamma \frac{\partial(x - mean) * \frac{1}{\sqrt{\sigma^2+\epsilon}}}{(\frac{1}{\sqrt{\sigma^2+\epsilon}})} \Rightarrow \delta_{out} * \gamma * (x - mean) * \frac{\partial(\frac{1}{\sqrt{\sigma^2+\epsilon}})}{\partial\sqrt{\sigma^2 + \epsilon}}$$
$$\Rightarrow \delta_{out} * \gamma * (x - mean) * \frac{1}{\sqrt{\sigma^2 + \epsilon}^2} * \frac{\partial\sqrt{\sigma^2 + \epsilon}}{\partial \sigma^2}$$
$$\Rightarrow \delta_{out} * \gamma * (x - mean) * \frac{1}{\sqrt{\sigma^2 + \epsilon}^2} * 0.5 * \frac{1}{\sqrt{\sigma^2 + \epsilon}} * \frac{\partial \frac{1}{B}\sum_{s=1}^{B}(x - mean)^2}{\partial(x - mean)}$$
$$\Rightarrow dxm_2 = \delta_{out} * \gamma * (x - mean) * \frac{1}{\sqrt{\sigma^2 + \epsilon}^2} * 0.5 * \frac{1}{\sqrt{\sigma^2 + \epsilon}} * \frac{2}{B} * (x - mean) * \frac{\partial(x - mean)}{\partial x}$$
$$dx = dxm_1 + dxm_2$$

4 PyTorch CNN

Convolutional networks proved to be undoubtedly more suitable for this task by observing the derived results in Figure 3. The thing is that convolution networks have the ability to exploit spatially local correlation present in natural images. We can observe that convolution network outperforms MLP in both accuracy and loss.

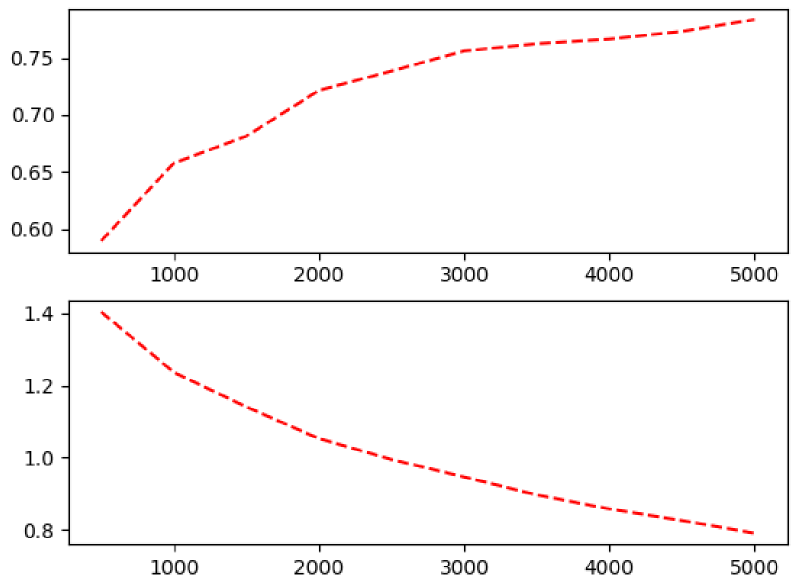

Figure 3: Convolution network loss and accuracy measurements (accuracy (up), loss (down)