DL: Assignment 1

Jonathan Mitnik
MSc Artificial Intelligence
University of Amsterdam
jonathan@student.uva.nl

Abstract

In this report, the main focus is to go through the assignment, starting with a rough Numpy implementation, and then using State-of-the-art PyTorch.

1 Derivatives

In this section, we will note down the basic derivatives that form the basis of our neural network.

1.1 Linear Module

In this part, we will derive the formula's of three derivatives:

1. $\frac{\delta L}{\delta \mathbf{W}}$
2. $\frac{\delta L}{\delta \mathbf{b}}$
3. $\frac{\delta L}{\delta \mathbf{X}}$

To do so, we start with $\frac{\delta L}{\delta \mathbf{W}}$.

$\frac{\delta L}{\delta \mathbf{W}}$:

$$
\begin{aligned}
\frac{\delta L}{\delta \mathbf{W}} &=> \frac{\delta L}{\delta W}_{ij} \\
&= \sum_{mn} \frac{\delta L}{\delta Y_{mn}} \frac{\delta Y_{mn}}{\delta W_{ij}} \\
&= \sum_{mn} \frac{\delta L}{\delta Y_{mn}} \frac{\delta}{\delta W_{ij}} \sum_{o} X_{mo} * W_{on} \\
&= \sum_{mn} \frac{\delta L}{\delta Y_{mn}} \sum_{o} X_{mo} * \frac{\delta W_{on}}{\delta W_{ij}} \\
&= \sum_{mn} \frac{\delta L}{\delta Y_{mn}} \sum_{o} X_{mo} * \delta_{oi} \delta_{nj} \\
&= \sum_{mn} \frac{\delta L}{\delta Y_{mn}} X_{mi} \delta_{nj} \\
&= \sum_{m} \frac{\delta L}{\delta Y_{mi}} X_{mi} \\
&=> \frac{\delta L}{\delta Y}^{T} * X
\end{aligned}
$$

Next, we calculate for beta:

$$\frac{\delta L}{\delta \beta} : \boxed{\begin{aligned} \frac{\delta L}{\delta \beta} &= \sum_{pq} \frac{\delta L}{\delta Y_{pq}} \frac{Y_{pq}}{\delta b_j} \\ &= \sum_{pq} \frac{\delta L}{\delta Y_{pq}} \frac{\delta}{\delta b_j} B_{pq} \\ &= \sum_{pq} \frac{\delta L}{\delta Y_{pq}} \frac{\delta}{\delta b_j} b_q \\ &= \sum_{pq} \frac{\delta L}{\delta Y_{pq}} \delta_{qj} \\ &= \sum_{p} \frac{\delta L}{\delta Y_{pj}} * 1 \\ &=> [\frac{\delta L}{\delta Y} * 1] \end{aligned}}$$

where 1 is a (S x 1) vector.

Finally, we calculate X.

$$\frac{\delta L}{\delta \mathbf{X}} : \boxed{\begin{aligned} \frac{\delta L}{\delta \mathbf{X}} &=> \frac{\delta L}{\delta X}_{ij} \\ &= \sum_{pq} \frac{\delta L}{\delta Y_{pq}} \frac{\delta Y_{pq}}{\delta X_{ij}} \\ &= \sum_{pq} \frac{\delta L}{\delta Y_{pq}} \frac{\delta}{\delta X_{ij}} X W^T_{pq} + B_{pq} \\ &= \sum_{pq} \frac{\delta L}{\delta Y_{pq}} \frac{\delta}{\delta X_{ij}} \sum_r X_{pr} W^T_{rq} \\ &= \sum_{pq} \frac{\delta L}{\delta Y_{pq}} \sum_r \frac{\delta X_{pr}}{\delta X_{ij}} W^T_{rq} \\ &= \sum_{pq} \frac{\delta L}{\delta Y_{pq}} \sum_r \delta_{pi} \delta_{rj} W^T_{rq} \\ &= \sum_{q} \frac{\delta L}{\delta Y_{iq}} W^T_{jq} \\ &= \sum_{q} \frac{\delta L}{\delta Y_{iq}} W_{qj} \\ &=> \frac{\delta L}{\delta Y} W \end{aligned}}$$

1.2 Activation Module

We start by defining our chain again. We will a generic letter, H, to define the generic element-wise product.

$$\frac{\delta L}{\delta X} = \frac{\delta L}{\delta X_i j} = \sum_{pq} \frac{\delta L}{\delta Y_{pq}} * \frac{\delta Y_{pq}}{\delta X_i j} \tag{1}$$

$$= \sum_{pq} \frac{\delta L}{\delta Y_{pq}} * \frac{\delta}{\delta X_i j} H_{pq} \cdot X_{pq} \tag{2}$$

$$= \sum_{pq} \frac{\delta L}{\delta Y_{pq}} * H_{pq} \cdot \delta X_{pq} \delta X_{ij} \tag{3}$$

$$= \sum_{pq} \frac{\delta L}{\delta Y_{pq}} * H_{pq} * \delta_{pi} \delta_{qj} \tag{4}$$

$$= \frac{\delta L}{\delta Y_{ij}} * H_{ij} \tag{5}$$

$$= [\frac{\delta L}{\delta Y} \cdot H]_{ij} \tag{6}$$

$$=> \frac{\delta L}{\delta Y} \cdot H \tag{7}$$

$$\tag{8}$$

1.3 Softmax and Loss

For the softmax, we start by using the quotient rule to define the entries. We leave this in index notation.

$$\frac{\delta y_{pq}}{\delta X_{ij}} = \frac{\frac{\delta e^{X_{pq}}}{\delta X_{ij}} * \sum_k e^{X_{pk}} - e^{X_{pq}} * \frac{\delta \sum_k e^X_{pk}}{\delta X_{ij}}}{(\sum_k e^{X_p k})^2} \tag{9}$$

$$= \frac{\delta_{pi}\delta_{qj} e^{X_{pq}} * \sum_k e^{X_{pk}} - e^{X_{pq}} * e^{X_{pk}} \delta_{pi}\delta_{kj}}{(\sum_k e^{X_p k})^2} \tag{10}$$

$$= e^{X_{pq}} * \frac{\delta_{pi}\delta_{qj} * \sum_k e^{X_{pk}}}{-} e^{X_{pk}} \delta_{pi}\delta_{kj} (\sum_k e^{X_p k})^2 \tag{11}$$

$$= \frac{e^{X_{pq}}}{\sum_k e^{X_p k}} * \frac{\delta_{pi}\delta_{qj} * \sum_k e^{X_{pk}}}{\sum_k e^{X_p k}} - \frac{e^{X_{pk}} \delta_{pi}\delta_{kj}}{\sum_k e^{X_p k}} \tag{12}$$

$$= \sigma(X_{pq}) * (\delta_{pi}\delta_{qj} - \sigma(X_{pk} * \delta_{pi}\delta_{kj})) \tag{13}$$

$$= \sigma(X_{iq}) * (\delta_{qj} - \sigma(X_{ij})) \tag{14}$$

For the loss function:

$$\frac{\delta L}{\delta X_i j} = \frac{\delta}{X_{ij}} \frac{1}{S} \sum_{k,s} T_{sk} log(X_{sk}) \tag{15}$$

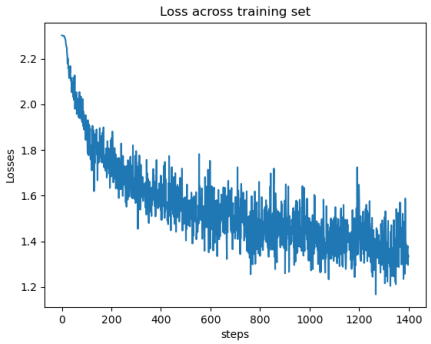
$$= \frac{1}{S} \sum_{k,s} T_{sk} \frac{log(X_{sk})}{\delta X_{ij}} \tag{16}$$

$$= \frac{1}{S} \sum_{k,s} T_{sk} \frac{\delta_{si}\delta_{kj}}{X_{sk}} \tag{17}$$

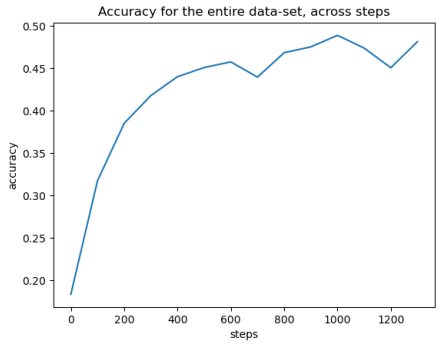
$$= \frac{1}{S} \frac{T_{ij}}{X_{ij}} \tag{18}$$

$$=> \frac{1}{S} T \cdot 1/X \tag{19}$$

where, all elements are inverted (not the matrix itself).



(a) Numpy: Training loss for best settings



(b) Numpy: Testing accuracy

Figure 1: Numpy: Performances for training and testing

1.4 Numpy Implementation

In these figures, we see the loss decreasing with the steps and the accuracy incerasing. The score reaches the 0.48, and dances around that value consequently. The loss is slowing down already as well, indicating that the model is unlikely to learn a lot more. Similarily goes for testing: it reaches its plateau, unable to further improve much more given the current parameters.

2 PyTorch Implementation

In this part of the report, results of the PyTorch experimentation are described, as well as a quick hint to the differnece between ELU and TanH.

2.1 Experimental design: Getting the MLP to 0.52 and beyond

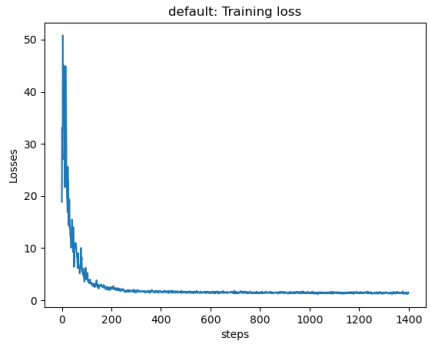
In the experiment, the goal was to get the MLP to 0.52 accuracy. To do this, a number of test-cases were considered. To prevent an explosion of the parameter space (and learn to understand simple changes better for the scope of this project), changes were alternated most of the time, rather than combining them together. In the following table, the results of various experiments. To ensure an easy overview, all the cells that contain a "-" use the default parameters as provided in the original code.

Furthermore, for batchnorm, an 'X' indicates that it has been used. The batchnorm was utilized only on hidden layers, and not on the output layer.

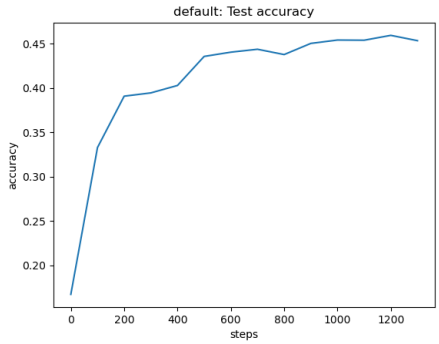
The best performing is the bottom row in the following table:

optimizer	hidden_layers	batchnorm	batch-size	act_fn	max_steps	results
-	-	-	-	-	-	0.451
adam	-	-	-	-	-	0.461
-	-	-	64	-	-	0.41
-	-	-	256	-	-	0.456
-	-	X	-	-	-	0.412
-	-	-	-	-	10000	0.480
-	-	-	-	tanh	-	0.353
-	64,128	-	-	-	-	0.426
adam	64,128	-	-	-	-	0.421
adam	64,128	-	-	-	10000	0.462
adam	64, 128	X	-	-	10000	0.524
adam	64,128,256	X	-	-	10000	**0.531**

Table 1: Runs with various permutations

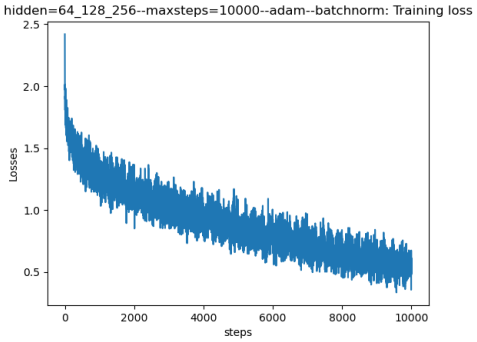


(a) Training loss for default settings

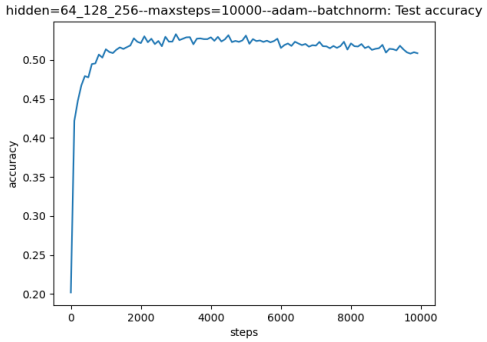


(b) Test accuracy for default settings

Figure 2: Default performance, test and train



(a) Training loss for best settings



(b) Test accuracy for best settings

Figure 3: Best performance, test and train

In the end, the best performance turned out to be the inclusion of batch-norm with more steps and more layers. Important to note, was that especially when adding batch-norm, the ability to generalize with more layers started improving tremendously.

2.2 ELU vs TanH

Some of the main benefits of using TanH, is that it provides a very clear bound for the activation values. ELU has no upper boundary, meaning that values could theroretically become huge, and rely on regularization techniques.

Drawbacks of Tanh relate to both slightly more expensive computation of the gradient of Tanh (more complex than the very simple form of the ELU), and the dense layer weights it allows. With ELU, many insignificant weights go to practically 0, whereas with Tanh, the weights are scaled somewhere between -1 and 1. Furthermore, networks with Tanh activation functions have converged a lot slower in this example for instance.

Tanh could be used for outputting your values to a particular characteristic (e.g [-1, 1]). If the output of your model requires this, Tanh can be used. ELU should never be used as output activation for your network.

3 Layer normalization

3.2a: Manual implementation of backward pass

I will start by writing down, in essence, all known shapes.

1. $\boldsymbol{X} \in \mathbb{R}^{S*M}$
2. $\boldsymbol{Y} \in \mathbb{R}^{S*M}$ (same shape as X)
3. $L \in \mathbb{R}^1$
4. $\gamma \in \mathbb{R}^M$
5. $\beta \in \mathbb{R}^M$
6. $\frac{\delta L}{\delta \gamma} \in \mathbb{R}^M$
7. $\frac{\delta L}{\delta \beta} \in \mathbb{R}^M$
8. $\frac{\delta L}{\delta \boldsymbol{Y}} \in \mathbb{R}^{S*M}$
9. $\frac{\delta L}{\delta \boldsymbol{X}} \in \mathbb{R}^{S*M}$

To start, the first derivative we calculate is that of $\frac{\delta L}{\delta \gamma}$.

$\frac{\delta L}{\delta \gamma}$:

$$
\begin{aligned}
\frac{\delta L}{\delta \gamma} &=> [\frac{\delta L}{\delta \gamma}]_i = \frac{\delta L}{\delta \gamma}_i = \sum_{sj} \frac{\delta L}{\delta Y_{sj}} * \frac{\delta Y_{sj}}{\delta \gamma_i} \\
&= \sum_{sj} \frac{\delta L}{\delta Y_{sj}} * \frac{\delta \gamma_j \hat{X}_{sj}}{\delta \gamma_i} \\
&= \sum_{sj} \frac{\delta L}{\delta Y_{sj}} * \delta_{ji} \hat{X}_{sj} \\
&= \sum_{s} \frac{\delta L}{\delta Y_{si}} * \hat{X}_{si} \\
&= \sum_{s} \mathbf{1}_s * \frac{\delta L}{\delta Y_{si}} * \hat{X}_{si} \\
&=> \mathbf{1}^T * [\frac{\delta L}{\delta Y} \circ \hat{X}] \qquad \text{where } \mathbf{1} \text{ is a 1xS vector}
\end{aligned}
$$

In a similar way we can calculate the derivative with respect to β.

$\frac{\delta L}{\delta \beta}$:

$$
\begin{aligned}
\frac{\delta L}{\delta \beta} &=> [\frac{\delta L}{\delta \beta}]_i = \frac{\delta L}{\delta \beta}_i = \sum_{sj} \frac{\delta L}{\delta Y_{sj}} * \frac{\delta Y_{sj}}{\delta \beta_i} \\
&= \sum_{sj} \frac{\delta L}{\delta Y_{sj}} * \frac{\delta \gamma_j \hat{X}_{sj} + \beta_J}{\delta \beta_i} \\
&= \sum_{sj} \frac{\delta L}{\delta Y_{sj}} * \delta_{ji} \\
&= \sum_{s} \frac{\delta L}{\delta Y_{si}} \\
&= \mathbf{1}^T * \frac{\delta L}{\delta Y} \qquad \text{where } \mathbf{1} \text{ is a 1xS vector}
\end{aligned}
$$

Now on to calculating the derivative with respect to X. It would be good to first define the starting chain:

$$\frac{\delta L}{\delta \boldsymbol{X}} => \frac{\delta L}{\delta X_{ri}} = \sum_{sj} \frac{\delta L}{\delta Y_{sj}} * \frac{\delta Y_{sj}}{\delta X_{ri}} \tag{20}$$

When focusing on $\frac{\delta Y_{sj}}{\delta X_{ri}}$, there are a number of aspects that come to play. To start, we can break it down into a chain, where we explicitly focus on the "contribution" that μ and σ have on X.

$$\frac{\delta Y_{sj}}{\delta X_{ri}} = \frac{\delta Y_{sj}}{\delta \hat{X}_{sj}} * \frac{\delta \hat{X}_{sj}}{\delta X_{ri}} + \frac{\delta Y_{sj}}{\delta \mu_s} * \frac{\delta \mu_s}{\delta X_{ri}} + \frac{\delta Y_{sj}}{\delta \sigma_s^2} * \frac{\delta \sigma_s^2}{\delta X_{ri}} \tag{21}$$

This contains quite a number of steps. It would be good to start from the first term we encounter.

$\frac{\delta Y_{sj}}{\delta \hat{X}_{sj}}$:

$$\boxed{\frac{\delta Y_{sj}}{\delta \hat{X}_{sj}} = \frac{\delta \gamma_j * \hat{X}_{sj}}{\delta \hat{X}_{sj}} \quad = \gamma_j}$$

$\frac{\delta \hat{X}_{sj}}{\delta X_{ri}}$:

$$\boxed{\begin{aligned} \frac{\delta \hat{X}_{sj}}{\delta X_{ri}} &= \frac{\delta}{\delta X_{ri}} * ((X_{sj} - \mu_s) * (\sigma_s^2 + \epsilon)^{-0.5}) \\ &= (\delta_{sr}\delta ji * (\sigma_s^2 + \epsilon)^{-0.5}) \\ &= (\frac{\delta_{sr}\delta_{ji}}{(\sigma_s^2 + \epsilon)^{0.5}}) \end{aligned}}$$

These were the first two terms, and relatively simple. However, from the third term onwards, we deal with a μ which occurs in σ as well. As such, to account for this, we need to calculate the derivative based on the amount of contribution it has in sigma as well. Essentially, we calculate the following:

$$\frac{\delta Y_{sj}}{\delta \mu_s} = \frac{\delta Y_{sj}}{\delta \hat{X}_{sj}} * \frac{\delta \hat{X}_{sj}}{\delta \mu_s} + \frac{\delta Y_{sj}}{\delta \sigma_s^2} * \frac{\delta \sigma_s^2}{\delta \mu_s} \tag{22}$$

$\frac{\delta \hat{X}_{sj}}{\delta \mu_s}$:

$$\boxed{\frac{\delta \hat{X}_{sj}}{\delta \mu_s} = -\frac{1}{(\sigma_s^2 + \epsilon)^{0.5}}}$$

$\frac{\delta Y_{sj}}{\delta \sigma_s^2}$:

$$\boxed{\begin{aligned} \frac{\delta Y_{sj}}{\delta \sigma_s^2} = \frac{\delta Y_{sj}}{\delta \hat{X}_{sj}} * \frac{\delta \hat{X}_{sj}}{\delta \sigma_s^2} &= \frac{\delta Y_{sj}}{\delta \hat{X}_{sj}} * \frac{\delta}{\delta \sigma_s^2}(X_{sj} - \mu_s) * (\sigma_s^2 + \epsilon)^{-0.5} \\ = \frac{\delta Y_{sj}}{\delta \hat{X}_{sj}} * -\frac{1}{2}(X_{sj} - \mu_s) * (\sigma_j^2 + \epsilon)^{-1.5} \end{aligned}}$$

$$\frac{\delta\sigma_s^2}{\delta\mu_s}: \boxed{\begin{aligned}\frac{\delta\sigma_s^2}{\delta\mu_s} &= \frac{\delta}{\delta\mu_s}\frac{1}{M}\sum_{i=1}^{M}(X_{si}-\mu_s)^2 \\ &= 2*-1*\frac{1}{M}\sum_{i=1}^{M}(X_{si}-\mu_s) \\ &= \frac{-2}{M}\sum_{i=1}^{M}(X_{si}-\mu_s)\end{aligned}}$$

So when combined, we get the following:

$$\frac{\delta Y_{sj}}{\delta\mu_j}: \boxed{\begin{aligned}\frac{\delta Y_{sj}}{\delta\mu_j} &= (\gamma_j * -\frac{1}{(\sigma_s^2+\epsilon)^{0.5}}) + (\gamma_j * -\frac{1}{2}(X_{sj}-\mu_s)*(\sigma_s^2+\epsilon)^{-1.5} * \frac{-2}{M}\sum_{i=1}^{M}(X_{si}-\mu_s)) \\ &= (\gamma_j * -\frac{1}{(\sigma_s^2+\epsilon)^{0.5}}) + (\gamma_j * (X_{sj}-\mu_s)*(\sigma_s^2+\epsilon)^{-1.5} * (\frac{1}{M}\sum_{i=1}^{M}(X_{si}) - \frac{1}{M}\sum_{i=1}^{M}\mu_s) \\ &= (\gamma_j * -\frac{1}{(\sigma_s^2+\epsilon)^{0.5}}) + (\gamma_j * (X_{sj}-\mu_s)*(\sigma_s^2+\epsilon)^{-1.5} * \mu_s - \frac{M}{M}\mu_s) \\ &= (\gamma_j * -\frac{1}{(\sigma_s^2+\epsilon)^{0.5}}) + (\gamma_j * (X_{sj}-\mu_s)*(\sigma_s^2+\epsilon)^{-1.5} * 0) \\ &= (\gamma_j * -\frac{1}{(\sigma_s^2+\epsilon)^{0.5}})\end{aligned}}$$

We are almost there, we only need two more terms before we can finish this up.

$$\frac{\delta\mu_s}{\delta X_{ri}}: \boxed{\begin{aligned}\frac{\delta\mu_s}{\delta X_{ri}} &= \frac{\delta}{\delta X_{ri}}\frac{1}{M}\sum_{k=1}^{M}X_{sk} \\ &= \frac{1}{M}\sum_{k=1}^{M}\frac{\delta X_{sk}}{\delta X_{ri}} \\ &= \frac{1}{M}\sum_{k=1}^{M}\delta_{sr}\delta_{ki} \\ &= \frac{1}{M}\delta_{sr} \\ &= \frac{\delta_{sr}}{M}\end{aligned}}$$

$$\frac{\delta\sigma_s^2}{\delta X_{ri}}: \boxed{\begin{aligned}\frac{\delta\sigma_s^2}{\delta X_{ri}} &= \frac{\delta}{\delta X_{ri}}\frac{1}{M}\sum_{k=1}^{M}(X_{ki}-\mu_s)^2 \\ &= \frac{1}{M}\sum_{k=1}^{M}2*(X_{sk}-\mu_s)*\delta_{sr}\delta_{ki} \\ &= \frac{2\delta_{sr}}{M}\sum_{k=1}^{M}(X_{sk}-\mu_s)\delta_{ki} \\ &= \frac{2\delta_{sr}}{M}X_{si}-\mu_s\end{aligned}}$$

To give a main overview, these are the main terms that will play a role:

1. $\frac{\delta Y_{sj}}{\delta \hat{X}_{sj}} = \gamma_j$

2. $\frac{\delta \hat{X}_{sj}}{\delta X_{ri}} = \frac{\delta_{sr}\delta_{ji}}{(\sigma_s^2+\epsilon)^{0.5}}$

3. $\frac{\delta Y_{sj}}{\delta \sigma_s^2} = \frac{\delta Y_{sj}}{\delta \hat{X}_{sj}} * -\frac{1}{2}(X_{sj}-\mu_s)*(\sigma_s^2+\epsilon)^{-1.5}$

4. $\frac{\delta Y_{sj}}{\delta \mu_s} = \gamma_j * -\frac{1}{(\sigma_s^2+\epsilon)^{0.5}}$

5. $\frac{\delta \mu_s}{\delta X_{ri}} = \frac{\delta_{sr}}{M}$

6. $\frac{\delta \sigma_s^2}{\delta X_{ri}} = \frac{2\delta_{sr}}{M}X_{si}-\mu_s$

To reiterate, the index notation for this formula goes as follows:

$$
\begin{aligned}
\frac{\delta L}{\delta \boldsymbol{X}} &=> \frac{\delta L}{\delta X_{ri}} = \sum_{sj} \frac{\delta L}{\delta Y_{sj}} \frac{\delta Y_{sj}}{\delta X_{ri}} \\
&= \sum_{sj} \left[\frac{\delta L}{\delta Y_{sj}} \frac{\delta Y_{sj}}{\delta \hat{X}_{sj}} \frac{\delta \hat{X}_{sj}}{\delta X_{ri}} \right] + \left[\frac{\delta L}{\delta Y_{sj}} \frac{\delta Y_{sj}}{\delta \mu_s} \frac{\delta \mu_s}{\delta X_{ri}} \right] + \left[\frac{\delta L}{\delta Y_{sj}} \frac{\delta Y_{sj}}{\delta \sigma_s^2} \frac{\delta \sigma_s^2}{\delta X_{ri}} \right] \\
&= \sum_{sj} \left[\frac{\delta L}{\delta Y_{sj}} \gamma_j \frac{\delta_{sr}\delta_{ji}}{(\sigma_s^2 + \epsilon)^{0.5}} \right] \\
&+ \left[\frac{\delta L}{\delta Y_{sj}} \left(\gamma_j * - \frac{1}{(\sigma_s^2 + \epsilon)^{0.5}} \right) \frac{\delta_{sr}}{M} \right] \\
&+ \left[\frac{\delta L}{\delta Y_{sj}} \left(\gamma_j * - \frac{1}{2} (X_{sj} - \mu_s) * (\sigma_s^2 + \epsilon)^{-1.5} \right) \frac{2\delta_{sr}}{M} (X_{si} - \mu_s) \right] \\
&= \sum_{sj} \left[\frac{\delta L}{\delta Y_{sj}} \gamma_j \frac{\delta_{sr}\delta_{ji}}{(\sigma_s^2 + \epsilon)^{0.5}} \right] \\
&- \left[\frac{\delta L}{\delta Y_{sj}} \left(\gamma_j * \frac{1}{(\sigma_s^2 + \epsilon)^{0.5}} \right) \frac{\delta_{sr}}{M} \right] \\
&- \left[\frac{\delta L}{\delta Y_{sj}} \left(\gamma_j * \frac{1}{2} (X_{sj} - \mu_s) * (\sigma_s^2 + \epsilon)^{-1.5} \right) \frac{2\delta_{sr}}{M} (X_{si} - \mu_s) \right] \\
&= \sum_{sj} \frac{\delta L}{\delta Y_{sj}} \gamma_j \frac{1}{(\sigma_s^2 + \epsilon)^{0.5}} \delta_{sr} \\
&* \left(\left[\delta_{ji} \right] - \left[\frac{1}{M} \right] - \left[\frac{1}{2} (X_{sj} - \mu_s) * (\sigma_s^2 + \epsilon)^{-1} * \frac{2}{M} (X_{si} - \mu_s) \right] \right) \\
&= \sum_{j} \frac{\delta L}{\delta Y_{rj}} \gamma_j \frac{1}{(\sigma_r^2 + \epsilon)^{0.5}} \\
&* \left(\left[\delta_{ji} \right] - \left[\frac{1}{M} \right] - \left[\frac{1}{2} (X_{rj} - \mu_r) * (\sigma_r^2 + \epsilon)^{-1} * \frac{2}{M} (X_{ri} - \mu_r) \right] \right) \\
&= \sum_{j} \frac{\delta L}{\delta Y_{rj}} \gamma_j \frac{1}{(\sigma_r^2 + \epsilon)^{0.5}} \\
&* \left(\left[\delta_{ji} \right] - \left[\frac{1}{M} \right] - \left[(X_{rj} - \mu_r) * (\sigma_r^2 + \epsilon)^{-1} * \frac{1}{M} (X_{ri} - \mu_r) \right] \right) \\
&= \frac{1}{(\sigma_r^2 + \epsilon)^{0.5}} \\
&* \left(\left[\sum_{j} \frac{\delta L}{\delta Y_{rj}} \gamma_j * \delta_{ji} \right] - \left[\sum_{j} \frac{\delta L}{\delta Y_{rj}} \gamma_j \frac{1}{M} \right] \right. \\
&\left. - \left[\sum_{j} \frac{\delta L}{\delta Y_{rj}} \gamma_j * (X_{rj} - \mu_r) * (\sigma_r^2 + \epsilon)^{-0.5} * (\sigma_r^2 + \epsilon)^{-0.5} * \frac{1}{M} (X_{ri} - \mu_r) \right] \right) \\
&= \frac{1}{(\sigma_r^2 + \epsilon)^{0.5}} \\
&* \left(\left[\sum_{j} \frac{\delta L}{\delta Y_{rj}} \gamma_j * \delta_{ji} \right] - \left[\sum_{j} \frac{\delta L}{\delta Y_{rj}} \gamma_j \frac{1}{M} \right] \right. \\
&\left. - \left[\sum_{j} \frac{\delta L}{\delta Y_{rj}} \gamma_j * (X_{rj} - \mu_r) * (\sigma_r^2 + \epsilon)^{-0.5} * (\sigma_r^2 + \epsilon)^{-0.5} * \frac{1}{M} (X_{ri} - \mu_r) \right] \right)
\end{aligned}
$$

$$
\begin{aligned}
&= \frac{1}{(\sigma_r^2 + \epsilon)^{0.5}} \\
&* \left(\left[\sum_j \frac{\delta L}{\delta Y_{rj}} \gamma_j * \delta_{ji} \right] - \left[\sum_j \frac{\delta L}{\delta Y_{rj}} \gamma_j \frac{1}{M} \right] \right. \\
&\left. - \left[\sum_j \frac{\delta L}{\delta Y_{rj}} \gamma_j * \hat{X}_{rj} * (\sigma_r^2 + \epsilon)^{-0.5} * \frac{1}{M} (X_{ri} - \mu_r) \right] \right)
\end{aligned}
$$

$$
\begin{aligned}
&= \frac{1}{(\sigma_r^2 + \epsilon)^{0.5}} \\
&* \left(\left[\sum_j \frac{\delta L}{\delta Y_{rj}} \gamma_j * \delta_{ji} \right] - \left[\sum_j \frac{\delta L}{\delta Y_{rj}} \gamma_j \frac{1}{M} \right] \right. \\
&\left. - \left[\sum_j \frac{\delta L}{\delta Y_{rj}} \gamma_j * \hat{X}_{rj} * \frac{1}{M} \hat{X_{ri}} \right] \right)
\end{aligned}
$$

$$
\begin{aligned}
&= \frac{1}{M} * \frac{1}{(\sigma_r^2 + \epsilon)^{0.5}} \\
&* \left(\left[M * \sum_j \frac{\delta L}{\delta Y_{rj}} \gamma_j * \delta_{ji} \right] - \left[\sum_j \frac{\delta L}{\delta Y_{rj}} \gamma_j \right] \right. \\
&\left. - \left[\sum_j \frac{\delta L}{\delta Y_{rj}} \gamma_j * \hat{X}_{rj} \hat{X_{ri}} \right] \right)
\end{aligned}
$$

3.1 Batch norm vs Layer norm

Batch normalization ensures that our features are scaled to have 0 mean and standard deviation of 1, which creates an implicit boundary for the loss of these features, and essentially smoothes out the surface.

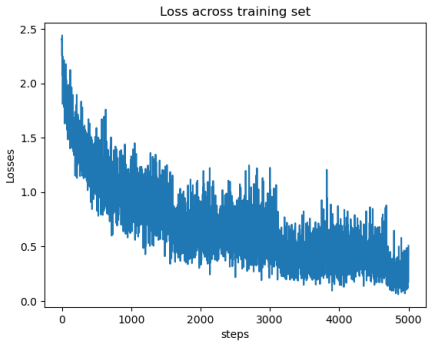
One of the problems of batch-norm, is how the mean and sigma are not invariant on the batch size itself. This means that the definition of your batch-size can influence the perforamnce of batch-normalization immensely. We are required to enforce a higher batch-size to prevent noise from coming in. And also, we need to keep into account for each iteration of an RNN the differnet means and standard deviations, which can become storage expensive.

Layer norm normalizes based on features rather than batch size. That essentially that each example is normalized different from other examples (no shared mean and standard deviation). However, this assumes a major flaw: not all neurons are always equally important, which is something the layer normalization implicitly uses as assumption. This can work to its detriment in for instance ConvNets, even though RNNs have better performance

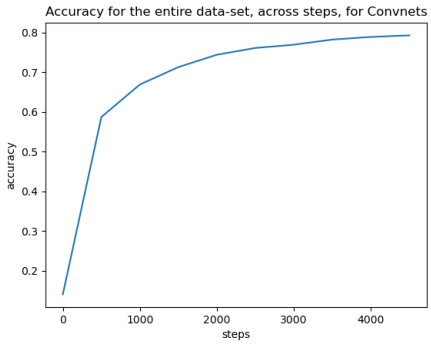
4 Conv nets

In this section, a simple and light-weight version of the Resnet version has been implemented, trained and evaluated.

By the end of the default number of steps, the model was not necessarily showing any sign of "stopping". What is interesting to note, is that the model seems to reach a point where the accuracy's linear increase (around 800 steps as seen in testing scores) slopes down a bit. While it does not reach



(a) Training loss for Convnet



(b) Test accuracy for Convnet

Figure 4: Default performance, test and train, for Convnet

convergence yet, it seems that from this point the model slows down its learning considerably; this is likely a decrease in learning rate as set by Adam's optimizer.

It is surprising nonetheless how fast this model learns on "limited" data. The gradients are not vanished, due to the resnet blocks, which means that even the deeper layers get enough of a signal to keep training.

The loss curve here (admittedly, I should have used averaged scores over steps, but time constraints and LISA server errors proved an extra challenge to run on time again) indicates "hops": around step 700, 1500, 3000, and 4700, the loss make significant "average" drops.