Deep Learning - Assignment 1

Mathias Parisot
12993700
parisot.mathias.31@gmail.com

1 MLP backprop and NumPy Implementation

1.1 Evaluating the Gradients

1.1.1 Linear Module

Let $\mathbf{X} \in \mathcal{R}^{S\times M}$, $\mathbf{W}^T \in \mathcal{R}^{N\times M}$, and $\mathbf{B} \in \mathcal{R}^{S\times N}$. We define a linear module as $\mathbf{Y} = \mathbf{X}\mathbf{W}^T + \mathbf{B}$ and express the gradients of the loss with respect to the weights, bias, and input in terms of the gradients of the loss with respect to the output features $\frac{\partial L}{\partial \mathbf{Y}}$:

$$
\begin{aligned}
\left[\frac{\partial L}{\partial \mathbf{W}}\right]_{ij} &= \frac{\partial L}{\partial W_{ij}} \\
&= \sum_{m,n} \frac{\partial L}{\partial Y_{mn}} \frac{\partial Y_{mn}}{\partial W_{ij}} \\
&= \sum_{m,n} \frac{\partial L}{\partial Y_{mn}} \frac{\partial [\mathbf{X}\mathbf{W}^T + \mathbf{B}]_{mn}}{\partial W_{ij}} \\
&= \sum_{m,n} \frac{\partial L}{\partial Y_{mn}} \frac{\partial (\sum_p X_{mp} W^T_{pn} + B_{mn})}{\partial W_{ij}} \\
&= \sum_{m,n} \frac{\partial L}{\partial Y_{mn}} \sum_p X_{mp} \frac{\partial W_{np}}{\partial W_{ij}} \\
&= \sum_{m,n} \frac{\partial L}{\partial Y_{mn}} \sum_p X_{mp} \delta_{ni} \delta_{pj} \\
&= \sum_m \frac{\partial L}{\partial Y_{mi}} X_{mj} \\
&= \sum_m X^T_{jm} \frac{\partial L}{\partial Y_{mi}} \\
&= \left[\mathbf{X}^T \frac{\partial L}{\partial \mathbf{Y}}\right]_{ji} \\
&= \left[\left(\frac{\partial L}{\partial \mathbf{Y}}\right)^T \mathbf{X}\right]_{ij} \\
\frac{\partial L}{\partial \mathbf{W}} &= \left(\frac{\partial L}{\partial \mathbf{Y}}\right)^T \mathbf{X}
\end{aligned}
$$

32nd Conference on Neural Information Processing Systems (NeurIPS 2018), Montréal, Canada.

$$
\begin{aligned}
\left[\frac{\partial L}{\partial \mathbf{b}}\right]_i &= \frac{\partial L}{\partial b_i} \\
&= \sum_{m,n} \frac{\partial L}{\partial Y_{mn}} \frac{\partial Y_{mn}}{\partial b_i} \\
&= \sum_{m,n} \frac{\partial L}{\partial Y_{mn}} \frac{\partial [\mathbf{X}\mathbf{W}^T + \mathbf{B}]_{mn}}{\partial b_i} \\
&= \sum_{m,n} \frac{\partial L}{\partial Y_{mn}} \frac{\partial (\sum_p X_{mp} W^T_{pn} + B_{mn})}{\partial b_i} \\
&= \sum_{m,n} \frac{\partial L}{\partial Y_{mn}} \frac{\partial B_{mn}}{\partial b_i} \\
&= \sum_{m,n} \frac{\partial L}{\partial Y_{mn}} \delta_{ni} \\
&= \sum_{m} \frac{\partial L}{\partial Y_{mi}} \\
&= \left[\mathbf{1}^T \frac{\partial L}{\partial \mathbf{Y}}\right]_i \\
\frac{\partial L}{\partial \mathbf{b}} &= \mathbf{1}^T \frac{\partial L}{\partial \mathbf{Y}}
\end{aligned}
$$

$$
\begin{aligned}
\left[\frac{\partial L}{\partial \mathbf{X}}\right]_{ij} &= \frac{\partial L}{\partial X_{ij}} \\
&= \sum_{m,n} \frac{\partial L}{\partial Y_{mn}} \frac{\partial Y_{mn}}{\partial X_{ij}} \\
&= \sum_{m,n} \frac{\partial L}{\partial Y_{mn}} \frac{\partial [\mathbf{X}\mathbf{W}^T + \mathbf{B}]_{mn}}{\partial X_{ij}} \\
&= \sum_{m,n} \frac{\partial L}{\partial Y_{mn}} \frac{\partial (\sum_p X_{mp} W^T_{pn} + B_{mn})}{\partial X_{ij}} \\
&= \sum_{m,n} \frac{\partial L}{\partial Y_{mn}} \sum_p \frac{\partial X_{mp}}{\partial X_{ij}} W^T_{pn} \\
&= \sum_{m,n} \frac{\partial L}{\partial Y_{mn}} \sum_p \delta_{mi}\delta_{pj} W^T_{pn} \\
&= \sum_{n} \frac{\partial L}{\partial Y_{in}} W^T_{jn} \\
&= \sum_{n} \frac{\partial L}{\partial Y_{in}} W_{nj} \\
&= \left[\frac{\partial L}{\partial \mathbf{Y}} \mathbf{W}\right]_{ij} \\
\frac{\partial L}{\partial \mathbf{X}} &= \frac{\partial L}{\partial \mathbf{Y}} \mathbf{W}
\end{aligned}
$$

1.1.2 Activation Module

We define an element-wise activation function h and express the gradients of the loss with respect to the input in terms of the gradients of the loss with respect to the output features $\frac{\partial L}{\partial \mathbf{Y}}$:

$$
\begin{aligned}
\left[\frac{\partial L}{\partial \mathbf{X}}\right]_{ij} &= \frac{\partial L}{\partial X_{ij}} \\
&= \sum_{m,n} \frac{\partial L}{\partial Y_{mn}} \frac{\partial Y_{mn}}{\partial X_{ij}} \\
&= \sum_{m,n} \frac{\partial L}{\partial Y_{mn}} \frac{\partial h(X_{mn})}{\partial X_{ij}} \\
&= \frac{\partial L}{\partial Y_{ij}} \frac{\partial h(X_{ij})}{\partial X_{ij}} \\
&= \left[\frac{\partial L}{\partial \mathbf{Y}} \odot h'(\mathbf{X})\right]_{ij} \\
\frac{\partial L}{\partial \mathbf{X}} &= \frac{\partial L}{\partial \mathbf{Y}} \odot h'(\mathbf{X})
\end{aligned}
$$

1.1.3 Softmax and Loss Modules

We now consider the softmax and categorical cross entropy modules and express the gradients of the loss with respect to the input in terms of the gradients of the loss with respect to the output features $\frac{\partial L}{\partial \mathbf{Y}}$:

$$
\begin{aligned}
\left[\frac{\partial L}{\partial \mathbf{X}}\right]_{ij} &= \frac{\partial L}{\partial X_{ij}} \\
&= \sum_{m,n} \frac{\partial L}{\partial Y_{mn}} \frac{\partial Y_{mn}}{\partial X_{ij}} \\
&= \sum_{m,n} \frac{\partial L}{\partial Y_{mn}} \frac{\partial \left(\frac{e^{X_{mn}}}{\sum_k e^{X_{mk}}}\right)}{\partial X_{ij}} \\
&= \sum_{m,n} \frac{\partial L}{\partial Y_{mn}} \frac{\frac{\partial e^{X_{mn}}}{\partial X_{ij}} \sum_k e^{X_{mk}} - e^{X_{mn}} \frac{\partial \sum_k e^{X_{mk}}}{\partial X_{ij}}}{(\sum_k e^{X_{mk}})^2} \\
&= \sum_{m,n} \frac{\partial L}{\partial Y_{mn}} \frac{\delta_{mi}\delta_{nj} e^{X_{mn}} \sum_k e^{X_{mk}} - e^{X_{mn}} \sum_k \delta_{mi}\delta_{kj} e^{X_{mk}}}{(\sum_k e^{X_{mk}})^2} \\
&= \sum_{m,n} \frac{\partial L}{\partial Y_{mn}} \frac{\delta_{mi}\delta_{nj} e^{X_{mn}} \sum_k e^{X_{mk}} - \delta_{mi} e^{X_{mn}} e^{X_{mj}}}{(\sum_k e^{X_{mk}})^2} \\
&= \sum_{m,n} \frac{\partial L}{\partial Y_{mn}} \delta_{mi} \frac{e^{X_{mn}}}{\sum_k e^{X_{mk}}} (\delta_{nj} - \frac{e^{X_{mj}}}{\sum_k e^{X_{mk}}}) \\
&= \sum_{n} \frac{\partial L}{\partial Y_{in}} \frac{e^{X_{in}}}{\sum_k e^{X_{ik}}} (\delta_{nj} - \frac{e^{X_{ij}}}{\sum_k e^{X_{ik}}})
\end{aligned}
$$

$$
\begin{aligned}
\left[\frac{\partial L}{\partial \mathbf{X}}\right]_{ij} &= \frac{\partial L}{\partial X_{ij}} \\
&= -\frac{1}{S}\sum_{m,k} T_{mk}\frac{\partial \log X_{mk}}{\partial X_{ij}} \\
&= -\frac{1}{S}\sum_{m,k} T_{mk}\frac{\delta_{mi}\delta_{kj}}{X_{ij}} \\
&= -\frac{1}{S}\frac{T_{ij}}{X_{ij}} \\
&= \left[-\frac{1}{S}\mathbf{T}\odot f(\mathbf{X})\right]_{ij} \text{ with } f \text{ the element-wise inverse function} \\
\frac{\partial L}{\partial \mathbf{X}} &= -\frac{1}{S}\mathbf{T}\odot f(\mathbf{X})
\end{aligned}
$$

1.1.4 Bonus

TODO

1.2 Numpy Implementation

We now create our own Multilayer Perceptron (MLP) by implementing the modules discussed in the previous section using NumPy. We present the training loss and test accuracy on CIFAR10 for a MLP with a single hidden layer of 100 units followed by ELU activation in Figure 1.

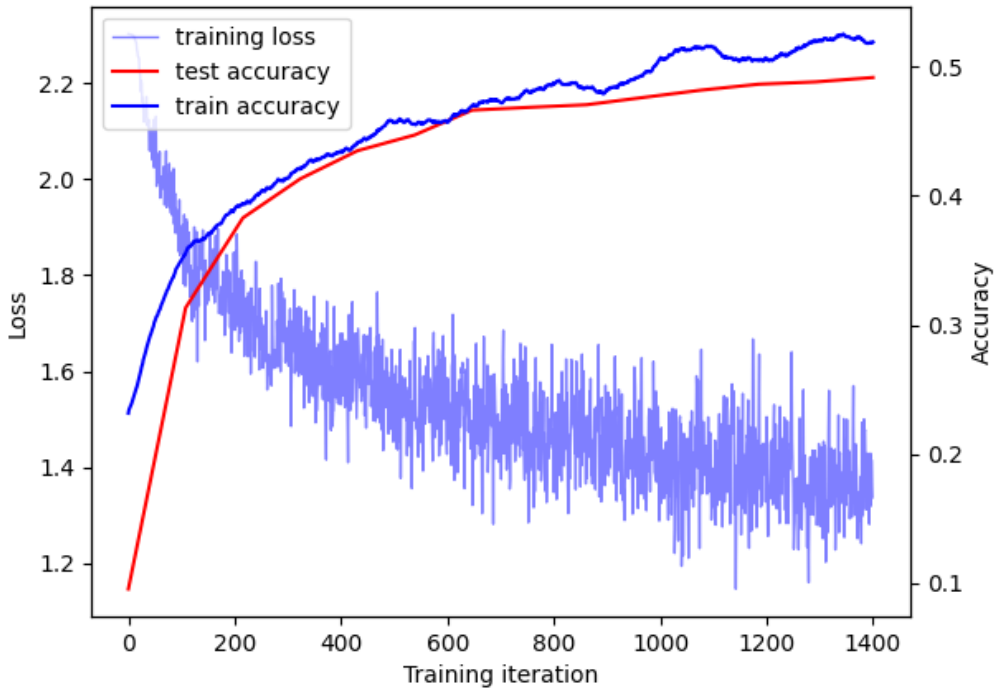

Figure 1: Training loss and test accuracy for a Numpy implementation of a MLP. note: train accuracy is a rolling average over 100 steps, test accuracy is evaluated every 500 steps.

2 Pytorch Implementation

We started the experimentation with 1400 training steps, a network with a single hidden units of 100 neurons, trained using Stochastic Gradient Descent (SGD) with a learning rate lr=0.001, a batch-size=200.

- First we increased the number of training steps to 10000 to observe the behavior of the test and train accuracies and the training loss. The loss kept decreasing (slower and slower but still decreasing) and the training accuracy kept increasing (up to 68%) while the test accuracy oscillated between 49 and 51%. The model was over-fitting to the training data.
- Because of the oscillations, we thought that the learning rate was to large and the lower areas of the loss surface were "jumped over". So we reduced the learning rate to lr=0.0005. The training accuracy still reached 68% while the test accuracy kept oscillating between 50-51%. We therefore, reduced the learning rate again to lr=0.0001 and the training accuracy reached 54% and the test accuracy 50.3% without oscilations, the over-fitting was mitigated but still present. Lower learning rates were not as successful.
- Keeping the same architecture, we tested other optimizers: SGD with momentum (0.9) and Adam, but none helped to increase the accuracy or reduce the over-fitting.
- Changing the activation function did not help neither, $\tanh$ reached 39% train and 36% test accuracy. ReLU performed similarly as ELU.
- The model seemed to have reached its maximal learning capacity. So we decided to increase its complexity by adding another hidden layer of 100 neurons. With a learning rate of 0.0001 the model did not learn and both the train and test accuracies oscillated between 10 and 12%. Using SGD with momentum, the model started to learn after 5000 training steps and reached 38% train and test acccuracies. Since the accuracies did not seem to stagnate yet, we increased the number of training steps to 20000 and reached 50% accuracy but the model was still over-fitting with 70% training accuracy. Using the Adam optimizer, it seems that the model learns faster (test accuracy stagnated after about 4000 steps),the test accuracy oscillated between 51 and 53%.
- We kept increasing the depth of the network up to 4 hidden layers but deeper networks seem to have trouble learning with SGD and SGD with momentum. Adam seems to be learning relatively faster compared to them.
- We had the idea that the number of parameters of the model could be the reason of the slow convergence. However, reducing the number of units in each hidden layers to 50 did not show to increase the convergence speed of the network significantly.
- Because the train and test accuracies were still oscilliating, we decided to reduce the learning rate after a predefined number of steps. We used a learning rate scheduler starting at a given value and dividing the learning rate by 10 after several predefined number of iterations.
- The best test accuracy was reached using 2 hidden layers of 100 units with ELU activation, SGD without momentum, and a step schedule learning rate starting at lr=0.01 and dividing by 10 after 3000 steps and 5000 steps. A test accuracy of 55% was reached after 6000 training steps. We experimented with the previous configurations and a learning rate scheduler and overall, using one was a simple way to add a between 1 and 3 percentage points after convergence of the test accuracy.

We present the training loss and test accuracy on CIFAR10 for the best found configuration in Figure 2.

2.1 Tanh vs. ELU

The *Exponential Linear Unit (ELU)* activation keeps one of the advantages of the ReLU activation: it has strong gradients for positive inputs. However, it tackles one of the drawback of ReLU, the 0 gradients for negative inputs which can create dead neurons, by adding an exponential term inducing negative decreasing gradients when the input become more negative. ELU still has some of the inconvenients of ReLU: it is not differentiable at 0, and it has strong gradients for very large positive inputs which can results in exploding gradients since any positive input is passed to the next module

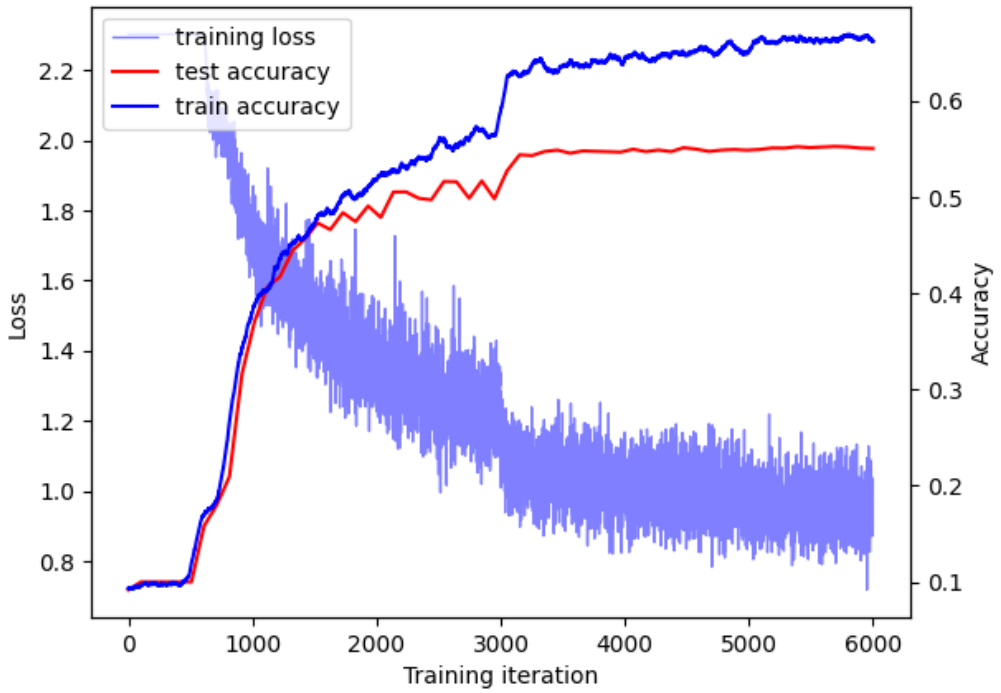

Figure 2: Training loss, train and test accuracy for a Pytorch implementation of a MLP. note: train accuracy is a rolling average over 100 steps, test accuracy is evaluated every 500 steps.

unmodified.
The *Hyperbolic tangent (tanh)* activation has the advantage of keeping its output in the $[-1, 1]$ range. It is also differentiable everywhere. However, for large negative and positive inputs, the gradients tends towards 0. Because of that, $\tanh$ generates smaller gradients when the signals are strong which causes the weights to only be slightly updated (even when the error is large), the model becomes "overconfident". Moreover, the gradients generated by $\tanh$ are strictly smaller than 0, which means that there is a problem of vanishing gradients for deep networks.

3 Custom Module: Layer Normalization

3.1 Automatic differentiation

See code implementation.

3.2 Manual implementation of backward pass

We define the following entities:

$$
\begin{aligned}
\mu_s &= \frac{1}{M}\sum_{i=1}^{M} X_{si} \\
\sigma_s^2 &= \frac{1}{M}\sum_{i=1}^{M} (X_{si} - \mu_s)^2 \\
\hat{X}_{si} &= \frac{(X_{si} - \mu_s)}{\sqrt{\sigma_s^2 + \varepsilon}} = \left(\sigma_s^2 + \varepsilon\right)^{-0.5} (X_{si} - \mu_s) \\
Y_{si} &= \gamma_i \hat{X}_{si} + \beta_i
\end{aligned}
$$

We want to express $\frac{\partial L}{\partial \boldsymbol{\gamma}}$:

$$
\begin{aligned}
\left[\frac{\partial L}{\partial \boldsymbol{\gamma}}\right]_i &= \frac{\partial L}{\partial \gamma_i} \\
&= \sum_{s,j} \frac{\partial L}{\partial Y_{sj}} \frac{\partial Y_{sj}}{\partial \gamma_i} \\
&= \sum_{s,j} \frac{\partial L}{\partial Y_{sj}} \frac{\partial \left(\gamma_j \hat{X}_{sj} + \beta_j\right)}{\partial \gamma_i} \\
&= \sum_{s,j} \frac{\partial L}{\partial Y_{sj}} \delta_{ji} \hat{X}_{sj} \\
&= \sum_{s} \frac{\partial L}{\partial Y_{si}} \hat{X}_{si} \\
\frac{\partial L}{\partial \boldsymbol{\gamma}} &= \left(\frac{\partial L}{\partial \mathbf{Y}} \odot \hat{\mathbf{X}}\right)^T \mathbf{1}
\end{aligned}
$$

We now want to express $\frac{\partial L}{\partial \boldsymbol{\beta}}$:

$$
\begin{aligned}
\left[\frac{\partial L}{\partial \boldsymbol{\beta}}\right]_i &= \frac{\partial L}{\partial \beta_i} \\
&= \sum_{s,j} \frac{\partial L}{\partial Y_{sj}} \frac{\partial Y_{sj}}{\partial \beta_i} \\
&= \sum_{s,j} \frac{\partial L}{\partial Y_{sj}} \frac{\partial \left(\gamma_j \hat{X}_{sj} + \beta_j\right)}{\partial \beta_i} \\
&= \sum_{s,j} \frac{\partial L}{\partial Y_{sj}} \delta_{ji} \\
&= \sum_{s} \frac{\partial L}{\partial Y_{si}} \\
\frac{\partial L}{\partial \boldsymbol{\beta}} &= \left(\frac{\partial L}{\partial \mathbf{Y}}\right)^T \mathbf{1}
\end{aligned}
$$

We want to express $\frac{\partial L}{\partial \mathbf{X}}$:

$$\begin{aligned}\left[\frac{\partial L}{\partial \mathbf{X}}\right]_{ri} &= \frac{\partial L}{\partial X_{ri}} \\ &= \sum_{s,j} \frac{\partial L}{\partial Y_{sj}} \frac{\partial Y_{sj}}{\partial X_{ri}} \\ &= \sum_{s,j} \frac{\partial L}{\partial Y_{sj}} \frac{\partial Y_{sj}}{\partial \hat{X}_{sj}} \frac{\partial \hat{X}_{sj}}{\partial X_{ri}} \\ &= \sum_{s,j} \frac{\partial L}{\partial Y_{sj}} \gamma_j \frac{\partial \hat{X}_{sj}}{\partial X_{ri}}\end{aligned}$$

We further extend $\frac{\partial \hat{X}_{sj}}{\partial X_{ri}}$ knowing that $\hat{X}_{sj}$ is a function of the 3 variables X_{ri}, μ_s, and σ_s^2 :

$$\frac{\partial \hat{X}_{sj}}{\partial X_{ri}} = \frac{\partial \hat{X}_{sj}}{\partial X_{ri}} + \frac{\partial \hat{X}_{sj}}{\partial \mu_s} \frac{\partial \mu_s}{\partial X_{ri}} + \frac{\partial \hat{X}_{sj}}{\partial \sigma_s^2} \frac{\partial \sigma_s^2}{\partial X_{ri}}$$

First we compute $\frac{\partial \hat{X}_{sj}}{\partial X_{ri}}$:

$$\begin{aligned}\frac{\partial \hat{X}_{sj}}{\partial X_{ri}} &= \frac{\partial \left((\sigma_s^2 + \varepsilon)^{-0.5} X_{sj}\right)}{\partial X_{ri}} \\ &= \left(\sigma_s^2 + \varepsilon\right)^{-0.5} \delta_{sr} \delta_{ji}\end{aligned}$$

Second, we compute $\frac{\partial \hat{X}_{sj}}{\partial \mu_s}$:

$$\begin{aligned}\frac{\partial \hat{X}_{sj}}{\partial \mu_s} &= \frac{\partial \left(\sigma_s^2 + \varepsilon\right)^{-0.5} (X_{si} - \mu_s)}{\partial \mu_s} \\ &= -\left(\sigma_s^2 + \varepsilon\right)^{-0.5}\end{aligned}$$

Third, we compute $\frac{\partial \mu_s}{\partial X_{ri}}$:

$$\begin{aligned}\frac{\partial \mu_s}{\partial X_{ri}} &= \frac{\partial \left(\frac{1}{M} \sum_{i=1}^{M} X_{si}\right)}{\partial X_{ri}} \\ &= \frac{1}{M} \sum_{i=1}^{M} \frac{\partial X_{si}}{\partial X_{ri}} \\ &= \frac{\delta_{sr}}{M}\end{aligned}$$

Fourth, we compute $\frac{\partial \hat{X}_{sj}}{\partial \sigma_s^2}$:

$$\begin{aligned}\frac{\partial \hat{X}_{sj}}{\partial \sigma_s^2} &= \frac{\partial \left((\sigma_s^2 + \epsilon)^{-0.5} (X_{sj} - \mu_s)\right)}{\partial \sigma_s^2} \\ &= (X_{sj} - \mu_s) \frac{\partial (\sigma_s^2 + \epsilon)^{-0.5}}{\partial \sigma_s^2} \\ &= -0.5 (X_{sj} - \mu_s) (\sigma_s^2 + \epsilon)^{-1.5}\end{aligned}$$

Fifth, we compute $\frac{\partial \sigma_s^2}{\partial X_{ri}}$:

$$
\begin{aligned}
\frac{\partial \sigma_s^2}{\partial X_{ri}} &= \frac{\partial \left(\frac{1}{M} \sum_{k=1}^{M} \left(X_{sk} - \mu_s \right)^2 \right)}{\partial X_{ri}} \\
&= \frac{1}{M} \sum_{k=1}^{M} \frac{\partial (X_{sk} - \mu_s)^2}{\partial X_{ri}} \\
&= \frac{1}{M} \sum_{k=1}^{M} 2(X_{sk} - \mu_s) \left(\frac{\partial X_{sk}}{\partial X_{ri}} - \frac{\partial \mu_s}{\partial X_{ri}} \right) \\
&= \frac{1}{M} \sum_{k=1}^{M} 2(X_{sk} - \mu_s) \left(\delta_{sr}\delta_{ki} - \frac{\delta_{sr}}{M} \right) \\
&= \frac{1}{M} \sum_{k=1}^{M} 2\delta_{sr}(X_{sk} - \mu_s) \left(\delta_{ki} - \frac{1}{M} \right) \\
&= \frac{2\delta_{sr}}{M} \sum_{k=1}^{M} \left((X_{sk} - \mu_s)\delta_{ki} - \frac{(X_{sk} - \mu_s)}{M} \right) \\
&= \frac{2\delta_{sr}}{M} \sum_{k=1}^{M} (X_{sk} - \mu_s)\delta_{ki} - \frac{2\delta_{sr}}{M} \sum_{k=1}^{M} \frac{(X_{sk} - \mu_s)}{M} \\
&= \frac{2\delta_{sr}}{M} (X_{si} - \mu_s) - \frac{2\delta_{sr}}{M} \sum_{k=1}^{M} \frac{(X_{sk} - \mu_s)}{M} \\
&= \frac{2\delta_{sr}}{M} (X_{si} - \mu_s) - \frac{2\delta_{sr}}{M^2} \sum_{k=1}^{M} X_{sk} + \frac{2\delta_{sr}}{M^2} \sum_{k=1}^{M} \mu_s \\
&= \frac{2\delta_{sr}}{M} (X_{si} - \mu_s) - \frac{2\delta_{sr}}{M} \mu_s + \frac{2\delta_{sr}}{M^2} M\mu_s \\
&= \frac{2\delta_{sr}}{M} (X_{si} - \mu_s)
\end{aligned}
$$

We can then express $\frac{\partial \hat{X}_{sj}}{\partial X_{ri}}$ by grouping the previous derivatives:

$$
\begin{aligned}
\frac{\partial \hat{X}_{sj}}{\partial X_{ri}} &= \frac{\partial \hat{X}_{sj}}{\partial X_{ri}} + \frac{\partial \hat{X}_{sj}}{\partial \mu_s} \frac{\partial \mu_s}{\partial X_{ri}} + \frac{\partial \hat{X}_{sj}}{\partial \sigma_s^2} \frac{\partial \sigma_s^2}{\partial X_{ri}} \\
&= (\sigma_s^2 + \varepsilon)^{-0.5} \delta_{sr}\delta_{ji} - (\sigma_s^2 + \varepsilon)^{-0.5} \frac{\delta_{sr}}{M} - 0.5(X_{sj} - \mu_s)(\sigma_s^2 + \epsilon)^{-1.5} \frac{2\delta_{sr}}{M} (X_{si} - \mu_s) \\
&= \delta_{sr}(\sigma_s^2 + \varepsilon)^{-0.5} \left(\delta_{ji} - \frac{1}{M} - (X_{sj} - \mu_s)(\sigma_s^2 + \epsilon)^{-1} \frac{1}{M} (X_{si} - \mu_s) \right) \\
&= \delta_{sr}(\sigma_s^2 + \varepsilon)^{-0.5} \left(\delta_{ji} - \frac{1}{M} - (X_{sj} - \mu_s)(\sigma_s^2 + \epsilon)^{-0.5} (\sigma_s^2 + \epsilon)^{-0.5} \frac{1}{M} (X_{si} - \mu_s) \right) \\
&= \delta_{sr}(\sigma_s^2 + \varepsilon)^{-0.5} \left(\delta_{ji} - \frac{1}{M} - \frac{1}{M} \hat{X}_{sj} \hat{X}_{si} \right)
\end{aligned}
$$

Finally, we can express the derivative of the loss:

$$\begin{aligned}
\left[\frac{\partial L}{\partial \mathbf{X}}\right]_{ri} &= \sum_{s,j} \frac{\partial L}{\partial Y_{sj}} \gamma_j \frac{\partial \hat{X}_{sj}}{\partial X_{ri}} \\
&= \sum_{s,j} \frac{\partial L}{\partial Y_{sj}} \gamma_j \delta_{sr} (\sigma_s^2 + \varepsilon)^{-0.5} \left(\delta_{ji} - \frac{1}{M} - \frac{1}{M} \hat{X}_{sj} \hat{X}_{si}\right) \\
&= \sum_{j} \frac{\partial L}{\partial Y_{rj}} \gamma_j (\sigma_s^2 + \varepsilon)^{-0.5} \left(\delta_{ji} - \frac{1}{M} - \frac{1}{M} \hat{X}_{rj} \hat{X}_{ri}\right) \\
&= (\sigma_s^2 + \varepsilon)^{-0.5} \left(\sum_j \frac{\partial L}{\partial Y_{rj}} \gamma_j \delta_{ji} - \sum_j \frac{\partial L}{\partial Y_{rj}} \gamma_j \frac{1}{M} - \sum_j \frac{\partial L}{\partial Y_{rj}} \gamma_j \frac{1}{M} \hat{X}_{rj} \hat{X}_{ri}\right) \\
&= (\sigma_s^2 + \varepsilon)^{-0.5} \left(\frac{\partial L}{\partial Y_{ri}} \gamma_i - \frac{1}{M} \sum_j \frac{\partial L}{\partial Y_{rj}} \gamma_j - \frac{\hat{X}_{ri}}{M} \sum_j \frac{\partial L}{\partial Y_{rj}} \gamma_j \hat{X}_{rj}\right)
\end{aligned}$$

3.2.1 Layer Normalization vs. Batch Normalization

Batch Normalization computes the mean and variance of each feature across the batch, then normalizes the batch using those statistics and uses two learnable parameters to scale and shift the normalized batch. The idea is to tackle the covariate shift happening during training: the distribution of the input data of a given module may change as the training progress. This means that a module needs to adapt to the new distribution at each training step which can slow the learning. The goal of batch normalization is then to ensure that the distribution of the input of a given module is constant as the training progress. One of the issue of batch normalization is that it is not practical for Recurrent Neural Networks (RNN) because each time-step has a different mean and variance, which would require its own batch normalization module.
Layer Normalization is an alternative to batch normalization which computes the mean and variance of the input across the features. The statistics are computed for each of the instances in the batch.
With batch normalization, the larger the batch size, the better the estimates of the statistics are. It is also not possible to use batch normalization if the batch size is 1 since the variance would be 0. On the other hand, layer normalization is not impacted by the size of the batch.

4 Pytorch CNN

4.1 VGG13 with pre-activation skip connections

We now implement a modified version of the VGG13 architecture with added pre-activation skip connections. We present the training loss and test accuracy on CIFAR10 in Figure 3.

4.2 Transfer learning

We now study transfer learning techniques by using a Resnet18 architecture pre-trained on Imagenet. We use the weights of the pre-trained model as initialization for another model that we train on the CIFAR10 dataset. We replace the last fully-connected layer and train the model for 5000 steps using 32 images per batch, the Adam optimizer with a learning rate of $1e^{-4}$. In Figure 4, we present a comparison of two models: Resnet18 pre-trained on Imagenet and fine-tuned on CIFAR10 against Resnet18 only trained on CIFAR10. As we can observe, the pre-trained model reaches a higher test accuracy than the non-pre-trained one: it takes the pre-trained model only 500 steps to reach a larger accuracy than the non-pre-trained one after 5000 steps. Transfer Learning, because it make use of already well-performing models on similar tasks, is therefore a way to reduce the training time and therefore the computational resources needed to reach a target accuracy.

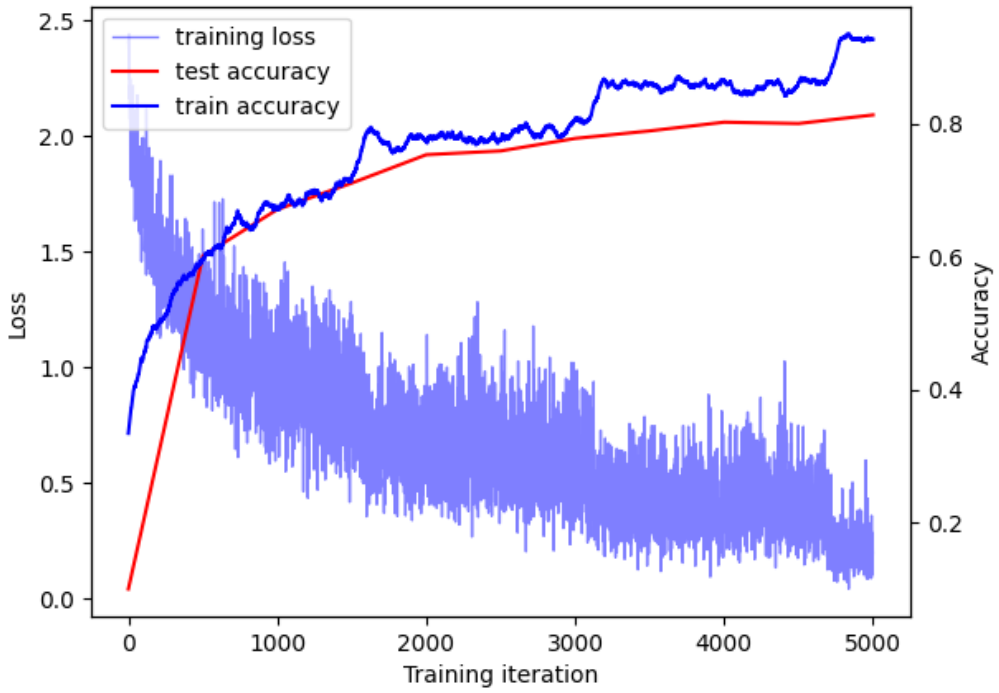

Figure 3: Training loss, train and test accuracy of the modified VGG13. note: train accuracy is a rolling average over 100 steps, test accuracy is evaluated every 500 steps.

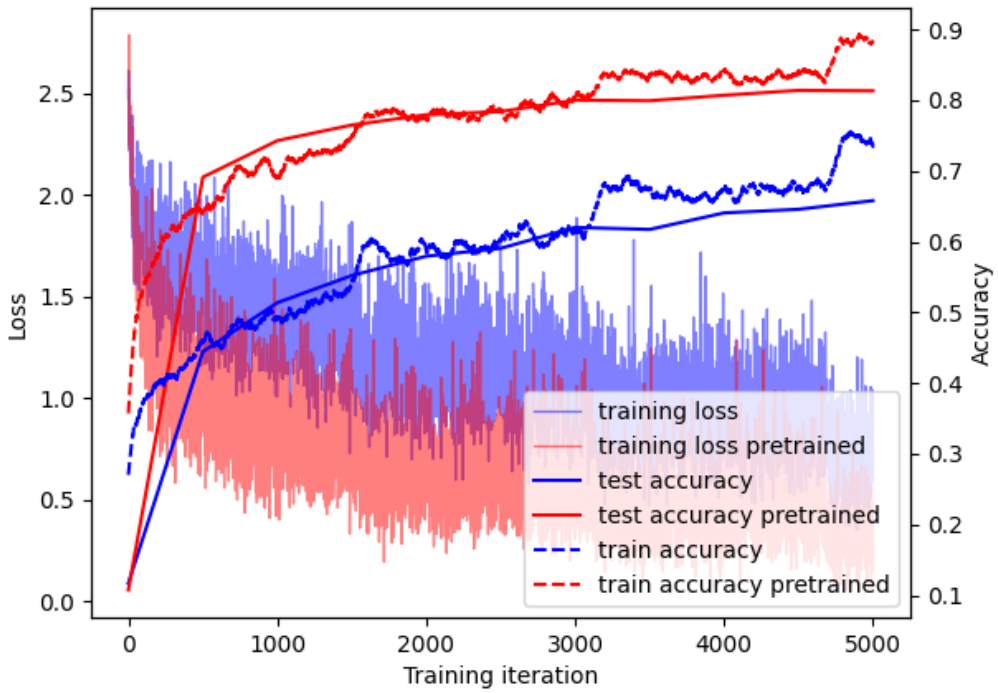

Figure 4: Comparison of the training loss and test accuracy of Resnet18 trained on CIFAR10 (blue) vs. Resnet18 trained on Imagenet and fine-tuned on CIFAR10 (red). note: train accuracy is a rolling average over 100 steps, test accuracy is evaluated every 500 steps.