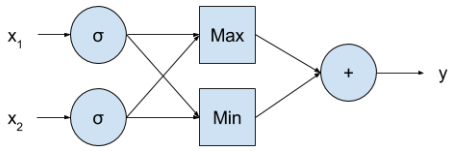
Deep Learning Teaching Kit

Lab 2, Sample Solution 1

1 More Backpropagation

1.1 Backpropagation through a DAG of modules



This module can be written as below.

$$\begin{aligned} y &= \max(s_1, s_2) + \min(s_1, s_2) \\ &= s_1 + s_2, \end{aligned}$$

where $s_1 = \sigma(x_1)$ and $s_2 = \sigma(x_2)$. Also, the backward pass can be written as below.

$$\begin{aligned} \frac{\partial E}{\partial x_1} &= \frac{\partial E}{\partial y}\frac{\partial y}{\partial s_1}\frac{\partial s_1}{\partial x_1} \\ &= \frac{\partial E}{\partial y} \cdot 1 \cdot s_1(1 - s_1) \\ &= \frac{\partial E}{\partial y}\sigma(x_1)(1 - \sigma(x_1)) \end{aligned}$$

$$\begin{aligned} \frac{\partial E}{\partial x_2} &= \frac{\partial E}{\partial y}\frac{\partial y}{\partial s_2}\frac{\partial s_2}{\partial x_2} \\ &= \frac{\partial E}{\partial y} \cdot 1 \cdot s_2(1 - s_2) \\ &= \frac{\partial E}{\partial y}\sigma(x_2)(1 - \sigma(x_2)), \end{aligned}$$

where $s_i(1- \ s_i)$ is the derivative of the sigmoid function derived in the Lab 1 question.

1.2 Batch Normalization

Let values of x over a mini-batch be $\mathcal{B} = \{x_{1\cdots m}\}$.

Batch Normalization:

$$y_i = \frac{x_i - \mu_\mathcal{B}}{\sqrt{\sigma_\mathcal{B}^2}},$$

where i is an index of samples within a mini-batch, $\mu_\mathcal{B} = \frac{1}{m}\sum_{i=1}^{m} x_i$, and $\sigma_\mathcal{B}^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_\mathcal{B})^2$.

(i)

The backward pass can be written as below.

$$\frac{\partial E}{x_i} = \frac{\partial E}{\partial y_i}\frac{\partial y_i}{\partial x_i} + \frac{\partial E}{\partial \mu_\mathcal{B}}\frac{\partial \mu_\mathcal{B}}{\partial x_i} + \frac{\partial E}{\partial \sigma_\mathcal{B}^2}\frac{\partial \sigma_\mathcal{B}^2}{\partial x_i}$$

Now, we compute each partial derivative.

$$\begin{aligned}
\frac{\partial y_i}{\partial x_i} &= \frac{1}{\sqrt{\sigma_\mathcal{B}^2}} \\
\frac{\partial \mu_\mathcal{B}}{\partial x_i} &= \frac{1}{m} \\
\frac{\partial \sigma_\mathcal{B}^2}{\partial x_i} &= \frac{2}{m}(x_i - \mu_\mathcal{B}) \\
\frac{\partial E}{\partial \sigma_\mathcal{B}^2} &= \sum_{i=1}^{m}\frac{\partial E}{\partial y_i}\frac{\partial y_i}{\partial \sigma_\mathcal{B}^2} \\
&= \sum_{i=1}^{m}\frac{\partial E}{\partial y_i}\left(-\frac{1}{2}\right)(x_i - \mu_\mathcal{B})\left(\sigma_\mathcal{B}^2\right)^{-\frac{3}{2}} \\
\frac{\partial E}{\partial \mu_\mathcal{B}} &= \sum_{i=1}^{m}\frac{\partial E}{\partial y_i}\frac{\partial y_i}{\partial \mu_\mathcal{B}} + \frac{\partial E}{\partial \sigma_\mathcal{B}^2}\frac{\partial \sigma_\mathcal{B}^2}{\mu_\mathcal{B}} \\
&= \sum_{i=1}^{m}\frac{\partial E}{\partial y_i}\left(-\frac{1}{\sqrt{\sigma_\mathcal{B}^2}}\right) + \frac{\partial E}{\partial \sigma_\mathcal{B}^2}\left(-\frac{2}{m}\right)\sum_{i=1}^{m}(x_i - \mu_\mathcal{B}) \\
&= \sum_{i=1}^{m}\frac{\partial E}{\partial y_i}\quad -\frac{1}{\sqrt{\sigma_\mathcal{B}^2}}\right) + \frac{1}{m}\sum_{i=1}^{m}\frac{\partial E}{\partial y_i}(x_i - \mu_\mathcal{B})^2\left(\sigma_\mathcal{B}^2\right)^{-\frac{3}{2}}
\end{aligned}$$

Therefore,

$$\begin{aligned}
\frac{\partial E}{x_i} = \frac{\partial E}{\partial y_i}\frac{1}{\sqrt{\sigma_\mathcal{B}^2}} + \left[\sum_{i=1}^{m}\frac{\partial E}{\partial y_i}\quad -\frac{1}{\sqrt{\sigma_\mathcal{B}^2}}\right) + \frac{1}{m}\sum_{i=1}^{m}\frac{\partial E}{\partial y_i}(x_i - \mu_\mathcal{B})^2\left(\sigma_\mathcal{B}^2\right)^{-\frac{3}{2}}\right]\frac{1}{m} \\
+ \sum_{j=1}^{m}\frac{\partial E}{\partial y_j}\left(-\frac{1}{2}\right)(x_j - \mu_\mathcal{B})\left(\sigma_\mathcal{B}^2\right)^{-\frac{3}{2}}\left[\frac{2}{m}(x_i - \mu_\mathcal{B})\right]
\end{aligned}$$

(ii)

Forward:

$$y_i = \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon_2}} + \epsilon_1$$

Backward:

$$\frac{\partial E}{x_i} = \frac{\partial E}{\partial y_i}\frac{1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon_2}} + \left[\sum_{i=1}^{m}\frac{\partial E}{\partial y_i} \quad -\frac{1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon_2}}\right) + \frac{1}{m}\sum_{i=1}^{m}\frac{\partial E}{\partial y_i}(x_i - \mu_{\mathcal{B}})^2\left(\sigma_{\mathcal{B}}^2 + \epsilon_2\right)^{-\frac{3}{2}}\Bigg]\frac{1}{m}$$
$$+\sum_{j=1}^{m}\frac{\partial E}{\partial y_j}\left(-\frac{1}{2}\right)(x_j - \mu_{\mathcal{B}})\left(\sigma_{\mathcal{B}}^2 + \epsilon_2\right)^{-\frac{3}{2}}\left[\frac{2}{m}(x_i - \mu_{\mathcal{B}})\right]$$

Backprop to parameters:

$$\begin{aligned}\frac{\partial E}{\partial \epsilon_1} &= \sum_{i=1}^{m}\frac{\partial E}{\partial y_i}\frac{\partial y_i}{\partial \epsilon_1} \\ &= \sum_{i=1}^{m}\frac{\partial E}{\partial y_i}\cdot 1\end{aligned}$$

$$\begin{aligned}\frac{\partial E}{\partial \epsilon_2} &= \sum_{i=1}^{m}\frac{\partial E}{\partial y_i}\frac{\partial y_i}{\partial \epsilon_2} \\ &= \sum_{i=1}^{m}\frac{\partial E}{\partial y_i}\left(-\frac{1}{2}\right)(x_i - \mu_{\mathcal{B}})\left(\sigma_{\mathcal{B}}^2 + \epsilon_2\right)^{-\frac{3}{2}}\end{aligned}$$

2 STL-10: Semi-supervised Image Recognition

Approach

Our general approach was:

- Start with the baseline code (**accuracy 72%**)
- Experimented with variants of the original architecture (**accuracy 77%**)
- Expanded the data set via various data augmentations (**accuracy 77%**)
- Attempted to obtain a better initialization of the filters in the first convolutional layer,by utilizing the approach of [1]. (**accuracy 86%**)
- After fine-tuning the network with the steps above, we then tried several psuedo-labeling procedures, inspired by [2]. (**accuracy 87%**)

All of the models were trained on 4000 train images with the remaining 1000 images used for validation. We ran each of the experiments for 130 epochs, and once we identified the top performing model, we retrained on the entire data set before making prediction on the test data set.

Architecture

We experimented with many variations on the base-architecture provided. We quickly noticed that the base model overfit the training data. We attempted to combat this with dropout in the convolutional layers, but that did not work well. We opted for a smaller/less complex model and this was successful in reducing overfitting.
After many iterations, the model architecture we settled on was shallower than the baseline model, with 9 layers instead of 11. The other significant difference was the kernel size of the three max-pooling layers. The base model had kernels of dimensions (4,3,2,2) respectively, and our model had max-pooling kernels of sizes (3,4,4). Please see appendix for more details.

Data Augmentation

Our data augmentation involved five different sub-procedures.

- translation of $+/-10\%$, chosen at random
- rotation of $+/-20$ degrees, chosen at random
- scaling 1: crop to size 87x87, and scale back to 96x96
- scaling 2: scale from 96x96 to 86x86, and add zero-padding along border of image
- horizontal flip 50% of the time, randomly

K-means

We started off with a small number of base-images and only a few patches generated per image. We then iteratively scaled up both the number of base-image and patches generated per image. Additionally, we tested out pre-processing the patches with whitening. Our final model was initialized with centroids obtained from running k-mean procedure on 10000 base-images with 5 whitened patches generated based on each image.

Psuedo-labeling

After training our modified architecture with the first convolutional layer initialized using k-means centorids, we then attempted psuedo-labeling the 100000 unlabeled images. We labeled the data following two sets of rules:

- Label only the top K most confident images in each class
- Label the top K most confident images in the entire dataset

Where confidence is measured by the maximum softmax output.
The first approach did not provide good results, especially because the model performed particularly poorly for some classes, yet still displayed "high-confidence". The model was very confident in predicting class 1, therefore, the second approach basically consisted of adding 1000 "class 1" images to the training dataset.

Results and Analysis

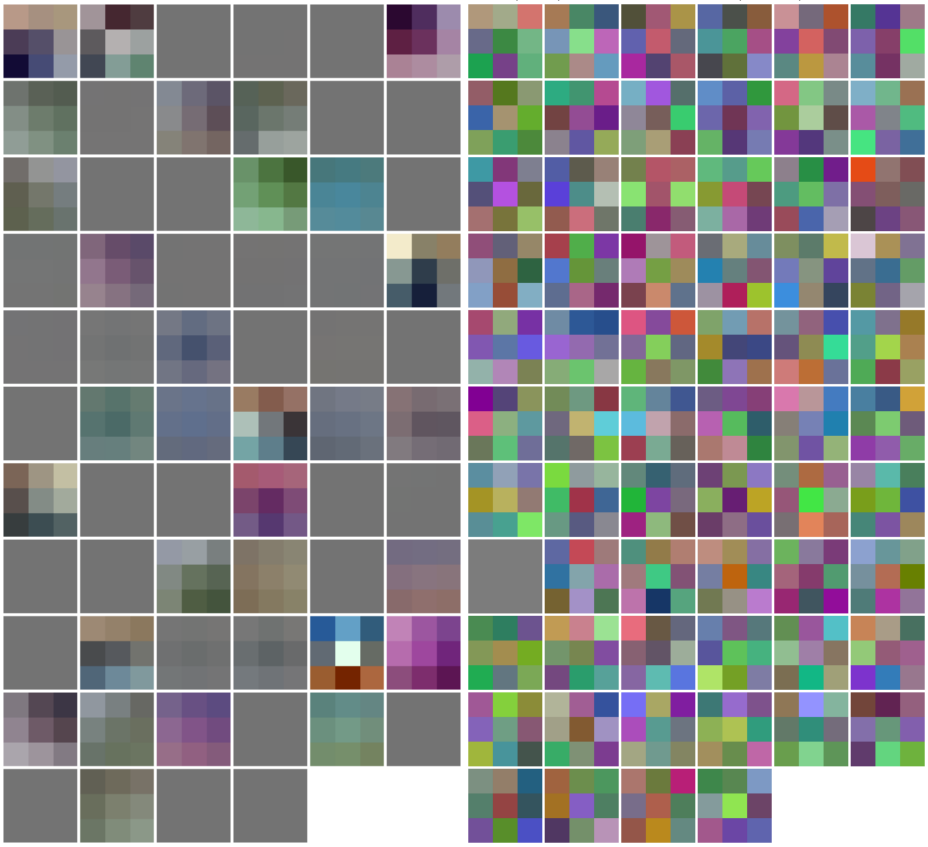
During training, we noticed a large discontinuous jump in accuracy around the 50th and 80th epochs. In the 20 epochs before each of the breakpoint, the training accuracy would remain flat and then make big sudden improvement. This observation was consistent regardless of the model which was being run, which leads us to believe that it is a property of the data rather than the model or choice of architecture.

We made an error in the code, and forgot to normalize the data, neither globally or locally. We noticed that this actually improved our performance, so we didn't fix this "bug". We hypothesize that batch-normalization probably enabled us to not do any other normalization.

2.1 Visualization

2.1.1 Filters in the First Layer

The filters from the first layer. Best model (left) and k-means (right).

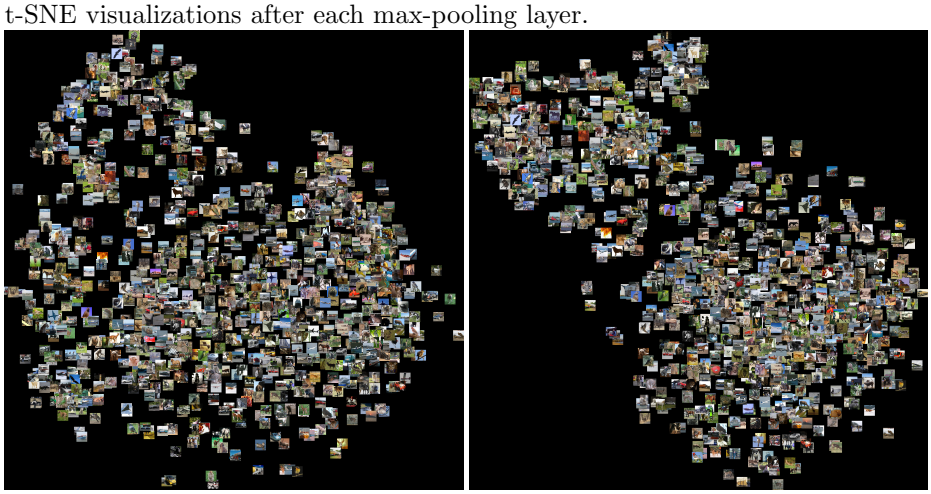


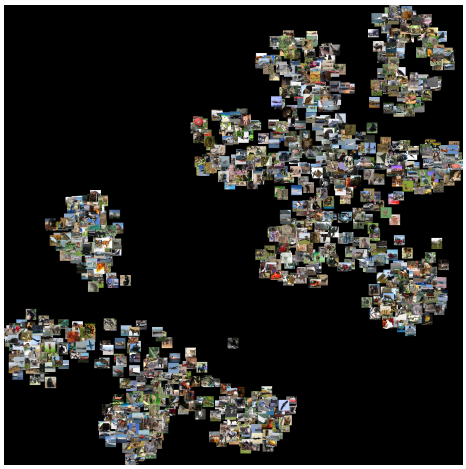
2.1.2 Data Augmentation

Each of our data-augmentation methods presented below.

2.2 t-SNE

t-SNE visualizations after each max-pooling layer.





References

[1] Adam Coates, Andrew Y Ng, and Honglak Lee. An analysis of single-layer networks in unsupervised feature learning. In *International conference on artificial intelligence and statistics*, pages 215–223, 2011.

[2] Dong-Hyun Lee. Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks. *Workshop on Challenges in Representation Learning, ICML*, 3, 2013.

Appendix

Model Architecture

(1): nn.SpatialConvolutionMM(3 -> 64, 3x3, 1,1, 1,1)
(2): nn.SpatialBatchNormalization
(3): nn.ReLU
(4): nn.SpatialConvolutionMM(64 -> 64, 3x3, 1,1, 1,1)
(5): nn.SpatialBatchNormalization
(6): nn.ReLU
(7): nn.SpatialMaxPooling(3x3, 3,3)
(8): nn.SpatialConvolutionMM(64 -> 128, 3x3, 1,1, 1,1)
(9): nn.SpatialBatchNormalization
(10): nn.ReLU
(11): nn.SpatialConvolutionMM(128 -> 128, 3x3, 1,1, 1,1)
(12): nn.SpatialBatchNormalization
(13): nn.ReLU
(14): nn.SpatialMaxPooling(4x4, 4,4)
(15): nn.SpatialConvolutionMM(128 -> 256, 3x3, 1,1, 1,1)
(16): nn.SpatialBatchNormalization
(17): nn.ReLU
(18): nn.SpatialConvolutionMM(256 -> 256, 3x3, 1,1, 1,1)
(19): nn.SpatialBatchNormalization
(20): nn.ReLU
(21): nn.SpatialMaxPooling(4x4, 4,4)
(22): nn.View(1024)
(23): nn.Sequential {
(1): nn.Dropout(0.500000)
(2): nn.Linear(1024 -> 256)
(3): nn.BatchNormalization
(4): nn.ReLU
(5): nn.Dropout(0.500000)
(6): nn.Linear(256 -> 10)
}