Fall 2024 CS7643 Deep Learning Assignment 2

Jacob Blevins

September 30, 2024

1 Theory

1.1 Problem 1

Working with a CNN with stride-4 and zero padding of size 2, on an input of 3x3 and kernel of 3x3, one can visualize that the sliding of the kernel over the padded input will result in

$$y = x_{(0,0)}w_{(2,2)} + x_{(0,2)}w_{(2,0)} + x_{(2,0)}w_{(0,2)} + x_{(2,2)}w_{(0,0)} \tag{1}$$

Which, put in linear form $y = Ax$, is below where A matrix has dimensions 4×9,

$$y = \begin{bmatrix} w_{(2,2)} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & w_{(2,0)} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & w_{(0,2)} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & w_{(0,0)} \end{bmatrix} \begin{bmatrix} x_{(0,0)} \\ x_{(0,1)} \\ x_{(0,2)} \\ x_{(1,0)} \\ x_{(1,1)} \\ x_{(1,2)} \\ x_{(2,0)} \\ x_{(2,1)} \\ x_{(2,2)} \end{bmatrix}. \tag{2}$$

1.2 Problem 2

A 2 hidden layer ReLU network can be represented by an overall weight W and bias b to map an input to the output. The network is modeled as

$$h(x) = W^{(3)} \max\left\{0, W^{(2)} \max\left\{0, W^{(1)}x + b^{(1)}\right\} + b^{(2)}\right\} + b^{(3)} \tag{3}$$

Using given weights and biases,

a) $x_0 = 2$, no ReLU layers zero their inputs as shown, along with the result:

$$max\{0, 2\begin{bmatrix}1.5\\0.5\end{bmatrix} + \begin{bmatrix}0\\1\end{bmatrix}\} \rightarrow \begin{bmatrix}3\\2\end{bmatrix}, \tag{4}$$

$$max\{0, \begin{bmatrix}1 & 2\\2 & 1\end{bmatrix}\begin{bmatrix}3\\2\end{bmatrix} + \begin{bmatrix}0\\1\end{bmatrix}\} \rightarrow \begin{bmatrix}7\\9\end{bmatrix}, \tag{5}$$

$$\begin{bmatrix}1 & 1\end{bmatrix}\begin{bmatrix}7\\9\end{bmatrix} - 1 \rightarrow h(x) = 15, \tag{6}$$

Since all ReLU layers take their positive values, the final network can be represented as

$$h(x) = W^{(3)}W^{(2)}W^{(1)}x + W^{(3)}W^{(2)}b^{(1)} + W^{(3)}b^{(2)} + b^{(3)} \tag{7}$$

Where the overall W and b are

$$W = W^{(3)}W^{(2)}W^{(1)} = 6 \tag{8}$$

$$b = W^{(3)}W^{(2)}b^{(1)} + W^{(3)}b^{(2)} + b^{(3)} = 3 \tag{9}$$

b) $x_0 = -1$ results in a final network with some max functions taking the 0 part, as shown below:

$$max\{0, -1 \begin{bmatrix}1.5\\0.5\end{bmatrix} + \begin{bmatrix}0\\1\end{bmatrix}\} \rightarrow \begin{bmatrix}0\\0.5\end{bmatrix}, \tag{10}$$

$$max\{0, \begin{bmatrix}1 & 2\\2 & 1\end{bmatrix} \begin{bmatrix}0\\0.5\end{bmatrix} + \begin{bmatrix}0\\1\end{bmatrix}\} \rightarrow \begin{bmatrix}1\\1.5\end{bmatrix}, \tag{11}$$

$$\begin{bmatrix}1 & 1\end{bmatrix} \begin{bmatrix}1\\1.5\end{bmatrix} - 1 \rightarrow h(x) = 1.5, \tag{12}$$

Where the overall W and b are

$$W = W^{(3)}W^{(2)} \begin{bmatrix}0 & 0\\0 & 1\end{bmatrix} W^{(1)} = 1.5 \tag{13}$$

$$b = W^{(3)}W^{(2)}b^{(1)} + W^{(3)}b^{(2)} + b^{(3)} = 3 \tag{14}$$

c) $x_0 = 1$ results in the same as in part *a)* where no ReLUs zero out,

$$max\{0, 1 \begin{bmatrix}1.5\\0.5\end{bmatrix} + \begin{bmatrix}0\\1\end{bmatrix}\} \rightarrow \begin{bmatrix}1.5\\1.5\end{bmatrix}, \tag{15}$$

$$max\{0, \begin{bmatrix}1 & 2\\2 & 1\end{bmatrix} \begin{bmatrix}1.5\\1.5\end{bmatrix} + \begin{bmatrix}0\\1\end{bmatrix}\} \rightarrow \begin{bmatrix}4.5\\5.5\end{bmatrix}, \tag{16}$$

$$\begin{bmatrix}1 & 1\end{bmatrix} \begin{bmatrix}4.5\\5.5\end{bmatrix} - 1 \rightarrow h(x) = 9, \tag{17}$$

With

$$W = W^{(3)}W^{(2)}W^{(1)} = 6 \tag{18}$$

$$b = W^{(3)}W^{(2)}b^{(1)} + W^{(3)}b^{(2)} + b^{(3)} = 3 \tag{19}$$

1.3 Problem 3

ReLU layers often produce dead gradients when the output is zero'd, but in the case of problem 2 above, it does not affect the final result. This phenomenon is because the ReLU is piecewise linear so the positive side of the input to the ReLU layer still learns even if some negative part dies.

2 Paper Review - Taskonomy: Disentangling Task Transfer Learning

Paper Summary *Taskonomy: Disentangling Task Transfer Learning* proposes that visual tasks have commonality that can be discovered in order to reduce complexity in visual neural network tasks. The authors provide a computational approach that utilizes dependencies in transfer learning between various tasks. As a result, the required amount of labeled data is significantly reduced. All-in-all, the authors are increasing efficiency in supervised training. Put in simpler terms, learned features of various visual tasks can be utilized to efficiently train other tasks on less data. Training a model from scratch is time consuming and data-expensive. Diving deeper on the actual approach, the authors utilize an affinity matrix to read representations trained for tasks to apply to new tasks. The method is tested on common datasets such as "ImageNet" and "Places". Since the method is computational, there are no imposed prior assumptions on the task space that are typically not found in fact, but rather assumption. The methodology is split into four steps: Step 1 is training the source tasks S from scratch, step 2 is training feasible transfers (or relationships) between sources and targets, step 3 normalization occurs, and in step 4, a hypergraph is synthesized which predicts the performance of any transfer policy. The tasks are trained on various models such as ResNet-50 and others. As a result, the authors find that the new method allows on average a higher "win rate" (or efficiency) over models trained from scratch!

My Takeaway This paper is eye opening for me in that neural networks are shown to be utilized not only for training models on data for specific tasks, but can be stepped forward to study relationships between already trained models. This fact opens a whole new world of deep learning possibility. I hope to apply this knowledge to my research wherein I may train a model to estimate a theoretically proven near-impossible network estimator (neural network vs neural network)... my research is in cyber secure robotics. More specifically for this paper, the evidence of the importance of features specifically shown in correlation to one another between tasks is outstanding and demonstrates the power of deep learning.

To answer this paper's first question *Do the task pairs with stronger arrows (better transfer) make sense in terms of why they would transfer better? Pick one positive pair (with 4 good transfer) and one negative pair (with bad transfer) and conjecture why it might be the case. Note that there are several types of features in deep learning, including low-level (e.g. edges), mid-level (components), and high-level (abstract concepts and classification layer) that you might reason about.*
An edge between a group of sourse tasks and a target task as represented by an arrow is a feasible transfer case and the arrow weight is the performance prediction between the two nodes. A greater weight correlates to a stronger correlation between the two models and thus an increase in efficiency of the targets' training. A pair with good transfer is 3D keypoints which connects to occlusion edges, reshading, depth, etc. This transfer is perhaps strong because these features are highly correlated from both a mathematical standpoint and by logical reasoning, and from low to high level. A bad transfer is between reshading and distance since these features are typically less correlated.

To answer this paper's second question *What does this say in terms of practical usage of deep learning across tasks? How might we use this information to guess where to transfer from if we have a new target task?* In utilizing these techniques, it is beneficial to mentally process where transfer might be useful before implementation. Practical usage would be found in transfer between tasks with highly correlated features, as mentioned in the previous comment. These highly correlated tasks may find great benefit with the techniques in the paper in increasing efficiency of training models, thus requiring less data and computational effort for model training. Amazing!

3 Programming

3.1 Implementing CNN from Scratch

The result of the CNN training is shown in Figure 1.

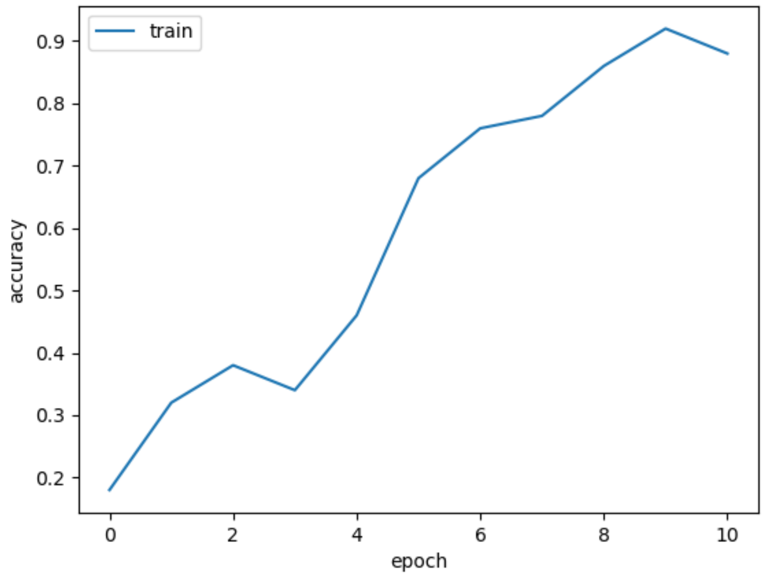

Figure 1: Training Accuracy for CNN from Scratch

3.2 Implementing CNN with PyTorch

Describe and justify your model design in plain text here:
My CNN model is loosely inspired by ResNet, with extensions such that I can compare them to the classic ResNet performance and understand how these mainstream architecures compare to similar counterparts. My model is structured such that like ResNet, it utilizes a series of convolutional layers with increasing numbers of filters/out channels, however, I take it one step further to a bigger model, ending in 1024 filters, and with leaky ReLU layers between the models instead of only ReLU layers. All-in-all, the model did not seem to perform better than the classic ResNet, but that could be due to hyperparamter tuning. The described model is as follows:

1. Convolution: in channels 3, out channels 32, kernel size 7, stride 1, padding 3
2. LeakyReLU
3. MaxPool: kernel size 3, stride 2
4. Convolution: in channels 64, out channels 64, kernel size 3, stride 1, padding 1
5. BatchNorm
6. LeakyReLU
7. Convolution: in channels 64, out channels 64, kernel size 3, stride 1, padding 1
8. BatchNorm
9. Convolution: in channels 64, out channels 128, kernel size 3, stride 2, padding 1
10. BatchNorm
11. LeakyReLU
12. Convolution: in channels 128, out channels 128, kernel size 3, stride 1, padding 1
13. BatchNorm
14. Convolution: in channels 128, out channels 256, kernel size 3, stride 2, padding 1
15. BatchNorm
16. LeakyReLU
17. Convolution: in channels 256, out channels 256, kernel size 3, stride 1, padding 1
18. BatchNorm
19. Convolution: in channels 256, out channels 512, kernel size 3, stride 2, padding 1
20. BatchNorm
21. LeakyReLU
22. Convolution: in channels 512, out channels 512, kernel size 3, stride 1, padding 1
23. BatchNorm
24. Convolution: in channels 512, out channels 1024, kernel size 3, stride 2, padding 1
25. BatchNorm
26. LeakyReLU
27. Convolution: in channels 1024, out channels 1024, kernel size 3, stride 1, padding 1
28. BatchNorm

29. Average Pool

30. Linear: in channels 1024, out features 10

Describe and justify your choice of hyper-parameters:
The hyperparameters are set as $v = 0.9(\text{momentum}), epochs = 15, reg = 0.0005, lr = 0.005, batchsize = 64$. Realistically the epochs are chosen due to computation time constraints. The momentum is chosen as 0.9 since this value seemed to perform well in formerly utilized models. The learning rate was found to perform better when increased, but there is fear that making the learning rate too large would cause oscillations around optimal points. A happy medium learning rate was finally chosen at 0.005. Batch size did not show much training accuracy improvement for both smaller and larger batches.

What's your final accuracy on validation set?
My final accuracy on the validation set is 0.8005.

3.3 Data Wrangling

What's your result of training with regular CE loss on imbalanced CIFAR-10? Tune appropriate parameters and fill in your best per-class accuracy in the table

	Class 0	**Class 1**	**Class 2**	**Class 3**	**Class 4**	**Class 5**	**Class 6**	**Class 7**	**Class 8**	**Class 9**
CE Loss	0.8790	0.7300	0.5390	0.4000	0.0660	0.001	0.001	0.0000	0.0000	0.0000

What's your result of training with CB-Focal loss on imbalanced CIFAR-10? Additionally tune the hyper-parameter beta and fill in your per-class accuracy in the table; add more rows as needed.

	Class 0	**Class 1**	**Class 2**	**Class 3**	**Class 4**	**Class 5**	**Class 6**	**Class 7**	**Class 8**	**Class 9**
β=0.9999	0.4790	0.2980	0.1380	0.1240	0.5700	0.0970	0.1960	0.1150	0.3340	0.1580
β=0.5000	0.8439	0.7490	0.6050	0.3500	0.0630	0.0000	0.0000	0.0000	0.0000	0.0000
β=0.8000	0.8800	0.6860	0.5670	0.3970	0.007	0.0630	0.0000	0.0000	0.0000	0.0000

Put your results of CE loss and CB-Focal Loss(best) together:

	Class 0	**Class 1**	**Class 2**	**Class 3**	**Class 4**	**Class 5**	**Class 6**	**Class 7**	**Class 8**	**Class 9**
CE Loss	0.8790	0.7300	0.5390	0.4000	0.0660	0.001	0.001	0.0000	0.0000	0.0000
CB Focal	0.4790	0.2980	0.1380	0.1240	0.5700	0.0970	0.1960	0.1150	0.3340	0.1580

Describe and explain your observation on the result:

The imbalanced data shows expected results with the typical cross entropy loss as shown in the first table. Since the samples are skewed, the lower numbered classes are trained much stronger with greater resulting weights. This phenomenon makes logical sense, because the less represented data has little opportunity to train, and connects logically to many real world scenarios related to representation. With the focal loss function balancing the weighting, given the number of samples for each class, the classes show a much greater balance in training for the $\beta = 0.9999$ case. The other cases with $\beta = 0.8$ and 0.5 show poor performance because the weighting factor β is having little effect on the unbalanced data. The resulting comparison shows the loss comparison between the methods and demonstrates the importance of data wrangling/data engineering in the training process.