Deep Learning Final Project Report

Alexander B. Powers

December 18th 2019

Contents

1 Introduction

This project explores different ways of leveraging multiple labels to improve the performance of deep neural networks. Multi-Task Learning (MTL) is the idea of leveraging commonalities between problems or tasks to improve model performance. To the best of my knowledge, Multi-Task Learning dates back to 1993[1].

1.1 Hard Parameter Sharing

Hard-parameter sharing is the sharing of hidden layer weights in a neural network across multiple tasks. Hard parameter sharing has the benefit of reducing the chance of over fitting, as the shared layer weights must be representative of multiple tasks[2]. Below is a diagram for standard hard parameter sharing.

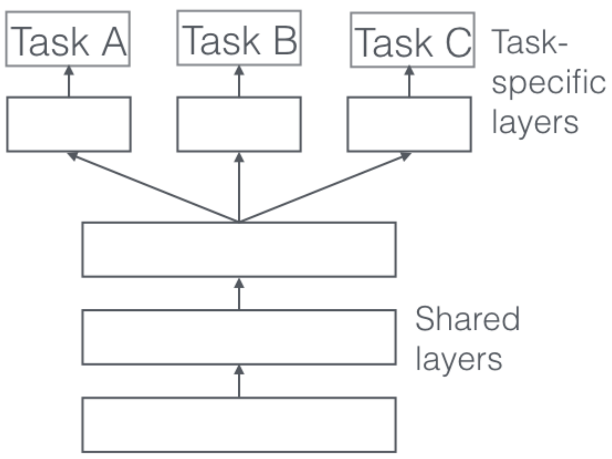


Figure 1: Hard Parameter Sharing Diagram [2]

1.2 Benefits of Multi-Task Learning [1][2]

- **Eavesdropping:** Eavesdropping is the idea of one task learning more quickly due to another tasks effects on the shared weight values. Assume TaskA is easier to learn than TaskB, and that they share common representations. As TaskA is learned, TaskB indirectly benefits as the shared representations are extracted, which allows TaskB to be learned more easily than in TaskA's absence.
- **Representation Bias:** Representation bias is the idea that training on multiple tasks biases the share weights to extract representation that benefit other tasks as well. This intuition is that MTL generates more generalizable weights, and would hypothetically lead to better transfer learning performance.
- **Regularization:** Regularization occurs naturally in MTL systems because tasks have different output distributions given the same input. Because of this the gradient of the shared layer weights with respect to the loss functions does not generate an optimal step for either task, but instead a step that benefits both suboptimally.

1.3 Dangers of Multi-Task Learning [1][2]

- **Task Interference:** The gradient of the shared layers with respect to each of the task losses may conflict. This interference can result in slower network convergence, and actually hurt performance.
- **Different Task Convergence:** Tasks can converge to their optimal performance at different rates. This component of MTL networks can result in a network that never optimally performs across all tasks.

2 The Experiment

The goal of these experiments is to develop an intuition for how different weight sharing techniques impact model performance on different tasks. The goal of all of these networks is to accurately classify the coarse and fine labels given and input image.

2.1 The CIFAR100 Dataset

The CIFAR100 dataset consists of 100 classes of images, which are grouped into 20 super classes. Throughout this report these will be referred to as 'coarse' and 'fine' labels. The data consists of 500 training images, and 100 test images for each fine class. Each image has exactly one fine label and one coarse label. The images in the CIFAR100 dataset are 32x32 RGB (three channel) images, making the network input shape: (batch_size, 32, 32, 3).

Coarse Label	**Fine Label**
aquatic mammals	beaver, dolphin, otter, seal, whale
fish	aquarium fish, flatfish, ray, shark, trout
flowers	orchids, poppies, roses, sunflowers, tulips
food containers	bottles, bowls, cans, cups, plates
fruit and vegetables	apples, mushrooms, oranges, pears, sweet peppers
household electrical devices	clock, computer keyboard, lamp, telephone, television
household furniture	bed, chair, couch, table, wardrobe
insects	bee, beetle, butterfly, caterpillar, cockroach
large carnivores	bear, leopard, lion, tiger, wolf
large man-made outdoor things	bridge, castle, house, road, skyscraper
large natural outdoor scenes	cloud, forest, mountain, plain, sea
large omnivores and herbivores	camel, cattle, chimpanzee, elephant, kangaroo
medium-sized mammals	fox, porcupine, possum, raccoon, skunk
non-insect invertebrates	crab, lobster, snail, spider, worm
people	baby, boy, girl, man, woman
reptiles	crocodile, dinosaur, lizard, snake, turtle
small mammals	hamster, mouse, rabbit, shrew, squirrel
trees	maple, oak, palm, pine, willow
vehicles 1	bicycle, bus, motorcycle, pickup truck, train
vehicles 2	lawn-mower, rocket, streetcar, tank, tractor

Table 1: CIFAR100 Structure

2.2 Network Architectures

The following network architectures, to varying degrees, leverage the fact that there are multiple labels for a single image. All of the following networks use the same number of parameters, except for the network outputs that are reused to improve the performance of the other tasks.

2.2.1 Independent networks

This is the control architecture for determining the effect of the following hard-parameter sharing techniques. The two tasks are learned independently, and share no representations.

```
input_image --> conv_layers --> fc_layers --> fine_label
input_image --> conv_layers --> fc_layers --> coarse_label
```

Figure 2: Independent Network Architecture

2.2.2 Hard parameter sharing in convolutional layers

The model most closely matches Figure 1. The weights of the convolutional layers will be shared across both tasks and the error propagation in these shared layers will be a function of both the coarse and fine label losses.

```
                           /--> fc_layers --> fine_label
input_image --> conv_layers
                           \--> fc_layers --> coarse_label
```

Figure 3: Shared Convoluion Architecture

2.2.3 Using coarse label output as weights

This network concatenates the coarse labels onto one of the fully connected layers of the fine label classifier. This allows the coarse label classification to inform the fine label classification. It is important to note that knowing the coarse label does not necessarily tell us the fine label, as each coarse class is composed of 5 fine classes.

```
input_image   ---->   conv_layers   ---->   fc_layers_1
                                                      \
                                                    concat -> fc_layers_2  -> fine_label
                                                      /
input_image -> conv_layers -> fc_layers -> coarse_label
```

Figure 4: Reuse Coarse Labels Architecture

2.2.4 Using fine label output as weights

This network concatenates the fine labels onto one of the fully connected layers of the coarse label classifier. As above in Figure 4 the final classification leverages the earlier classification. We should note that if we achieved 100% accuracy in fine_label classification, the coarse_label classification would become trivial.

```
input_image   ---->   conv_layers   ---->   fc_layers_1
                                                      \
                                                    concat -> fc_layers_2  -> coarse_label
                                                      /
input_image -> conv_layers -> fc_layers -> fine_label
```

Figure 5: Reuse Fine Labels Architecture

2.2.5 Combination of 2.2.2 & 2.2.3

This network is a combination of 2.2.2 and 2.2.3, and should theoretically carry the benefits of both networks.

```
                         / -------------> fc_layers_1
                        /                           \
input_image -> conv_layers                        concat -> fc_layers_2 -> fine_label
                        \                           /
                         \-> fc_layers -> coarse_label
```

Figure 6: Shared Convoluions & Reuse Coarse Labels Architecture

2.2.6 Combination of 2.2.2 & 2.2.4

This network is a combination of 2.2.2 and 2.2.4, and should theoretically carry the benefits of both networks.

```
                         / -------------> fc_layers_1
                        /                            \
input_image -> conv_layers                        concat -> fc_layers_2 -> coarse_label
                        \                            /
                         \-> fc_layers -> fine_label
```

Figure 7: Shared Convoluions & Reuse Fine Labels Architecture

2.3 Training Notes

For consistency, all of these architectures were trained for the same number of epochs, with the same batch size, and with the same learning rate. The error in fine_label classification was weighted twice as hevaily as the error in coarse_label classification, because accurate fine label classification makes coarse label classification trivial. The values noted above were as follows:

- Learning rate: 5e-4
- Epochs: 1e+2
- Batch Size: 1e+2

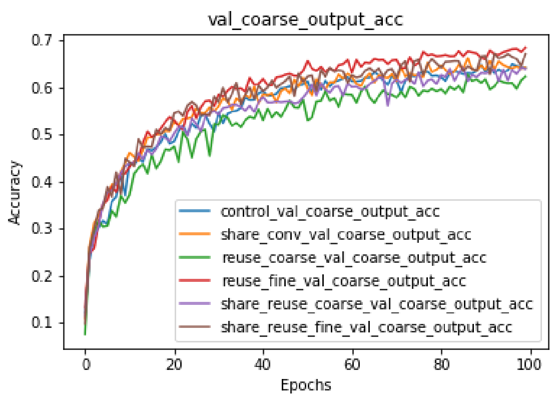
3 Results

Table 2 shows the final loss and accuracy of each model after 100 epochs of training.

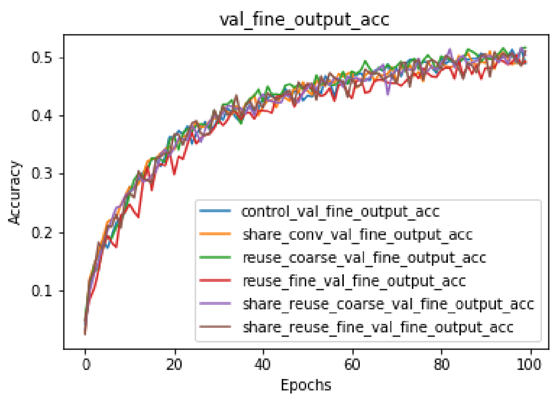
Model	**Fine Loss**	**Coarse Loss**	**Fine Accuracy**	**Coarse Accuracy**
Independent	1.96	1.15	48.86%	64.16%
Shared Convolutions	1.94	1.16	49.09%	64.20%
Reuse Coarse Labels	1.83	1.21	51.50%	62.29%
Reuse Fine Labels	1.95	1.07	49.14%	68.38%
Shared & Reuse Coarse	1.86	1.16	50.34%	63.91%
Shared & Reuse Fine	1.84	1.06	50.91%	67.11%

Table 2: Results on Validation Data

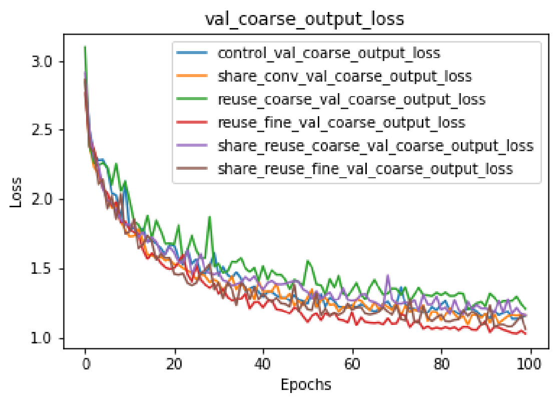
Below we can see the plot of network losses and accuracies across all models and coarse/fine labels. The networks all have fairly similar convergence, but there are some outlying networks like the Architecture 2.2.3 and Architecture 2.2.4, which generally mark the upper and lower bound of performance across each of the metrics.



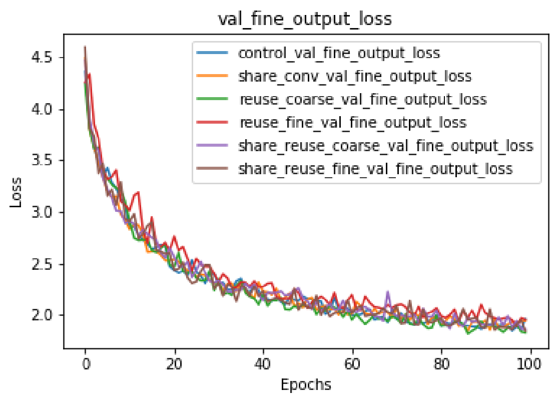
(a) Coarse Accuracy



(b) Fine Accuracy



(c) Coarse Loss



(d) Fine Loss

Figure 8: Loss and Accuracy Plots of All Models

4 Discussion & Conclusion

We can observe that the reuse of coarse/fine labels significantly benefits the performance of the network on the other task that is not being reused. We should additionally note that the architectures that reuse the coarse labels (2.2.3) and (2.2.5) perform worse on coarse label prediction than the independent network (2.2.1). I hypothesize that this is because the error *w.r.t.* the fine labels is propagated through the layers that predict the coarse label, and that these gradients, while improving fine label classification, slow down and even hurt the coarse label classification. We can also observe that in models that share convolutional layers and reuse outputs (2.2.6 and 2.2.3) the impacts of reusing outputs is minimized. I would attribute this to the regularizing behavior of hard parameter sharing.

We can see some very clear benefits to multi-task learning and it's ability to improve network performance. These benefits, however, come with pitfalls that we must be conscious of and account for when designing networks.

References

[1] Richard Caruana. Multitask learning: A knowledge-based source of inductive bias. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 41–48. Morgan Kaufmann, 1993.

[2] Sebastian Ruder. An overview of multi-task learning in deep neural networks, 2017.