Advances in Convolutional Neural Networks Programming Assignment

20233611 Hojun Lee

November 29, 2023

1 Knowledge Distillation

In general, larger and deeper models in classification tasks are expected to have better performance. However, there are some situations in which memory usage is restricted or requires immediate predictions. It would be admirable if a smaller and simpler model achieves a similar performance to a bigger and deeper model. If a pre-trained larger and deeper model (teacher model) can transfer knowledge to a smaller and simpler model (student model), it would be more helpful than the student model training alone.

Then, how can we define knowledge? When we use a neural network in a classification task, we often add a softmax layer at the end of the layer to let the logit z_i mean the probability of the input belonging in the ith class.

$$p_i = \frac{exp(z_i)}{\sum_j exp(z_j)} \tag{1}$$

And then, the model predicts the class that has the greatest probability as the answer. If a class has a slightly big probability (e.g. $p = 0.1$) even if it is not an accurate class, we can expect that there are some similar properties between the class and the accurate class. Otherwise, it means that the class has completely different properties to the accurate class. Hinton considered the probabilities of other classes as knowledge. [1] Since the probabilities are too small to use for training, he softened the probability as the following.

$$q_i = \frac{exp(z_i/T)}{\sum_j exp(z_j/T)} \tag{2}$$

Here, T is called a temperature.

We use a weighted sum of two losses, the original loss, which is a cross-entropy loss between student output and the label, and a Kullback–Leibler divergence between soft targets of the student and teacher model. Since the magnitude of a gradient of the Kullback-Leibler divergence of softened targets scales $1/T^2$, let's multiply T^2 for this loss. The object function is

$$\mathcal{L} = \mathcal{L}_{CE}(p_s, y) + \lambda T^2 \mathcal{L}_{KL}(q_s, q_t), \tag{3}$$

where p_s is the output probability of the student model, and q_s, q_t are softened output probability of the student model and the teacher model, respectively.

2 Training Process

2.1 Training algorithm implementation

I divided the training data into `batch_size=64` data and optimized the cost function for each minibatch. Figure 1 is the code that implements a training process for a single minibatch.

```
self.optimizer.zero_grad()
soft_teacher_output = self.teacher(train_data)/self.temp
student_output = self.student(train_data)
soft_student_output = student_output/self.temp

pred = torch.max(student_output, dim=1)

loss = self.lamb * self.kl(F.log_softmax(soft_student_output,dim=1), F.softmax(soft_teacher_output,dim=1)) * (self.temp**2)
loss += self.ce(student_output, train_label)
epoch_mean_loss.append(loss.item())
loss.backward()
self.optimizer.step()
```

Figure 1: Training process for a single minibatch. Since `KLDivLoss` method in PyTorch is defined as `loss_pointwise = target * (target.log() - input)`, we have to apply log function to the input.

2.2 Hyperparameter search

For hyper-parameter search, I used Adam optimizer with `lr=1e-3, betas=(0.9,0.999), weight_decay = 1e-4`. I used a random search with a log-uniform distribution (See Figure 2 for details) rather than using a grid search. [2] I compared the test accuracy for each hyper-parameter combination after 30 epochs.

```
def sample_hyp_params():
  temp = 10**random.uniform(np.log10(20),np.log10(80))
  lamb = (10**random.uniform(np.log10(10),np.log10(80)))
  temp = round(temp,2)
  lamb = round(lamb,2)
  # temp = 43.25
  # lamb = 38.21
  print("Temp: ",temp, " Lambda: ",lamb)
  return temp, lamb
```

Figure 2: A code for sampling hyper-parameters. Initially, `temp` and `lamb` are sampled log-uniformly between $10^0 \sim 10^2$ and $10^{-2} \sim 10^2$, respectively, and then refine the scope to the region where the test accuracy is high.

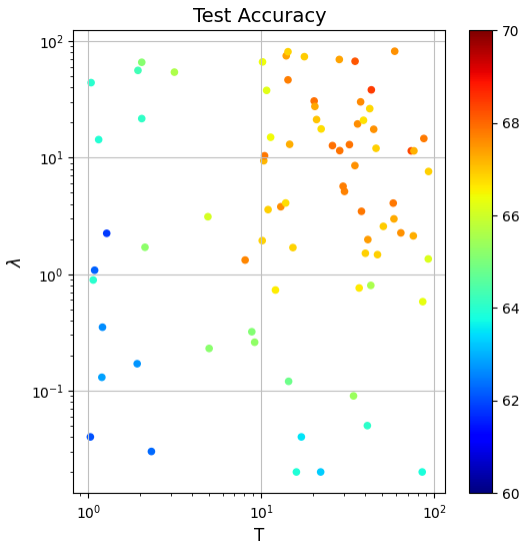


Figure 3: Test accuracy for various temperature T and λ in percent.

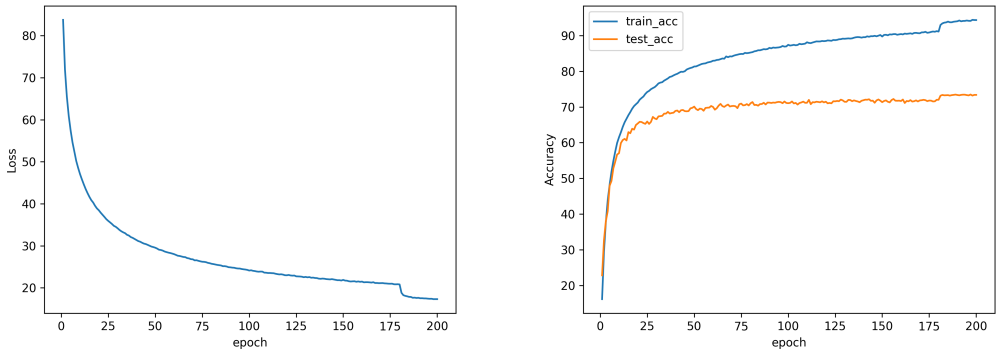
The hyper-parameter combination that achieves the highest test accuracy was $T = 43.25$, $\lambda = 38.21$, and it achieved a test accuracy of 68.44%.

2.3 Train

For actual training, I used hyper-parameters $T = 43.25$, $\lambda = 38.21$ that I found, and Adam optimizer with `lr=1e-3, betas=(0.9,0.999), weight_decay = 1e-4` for 180 epochs. And then, I reduced the learning rate to `1e-4` and trained 20 additional epochs.

3 Result

Figure 4 shows the loss, training accuracy, and test accuracy for the training process.



(a) Loss plot for 200 epochs

(b) Training and test accuracy for 200 epochs

Figure 4: Plot of loss, training accuracy, and test accuracy.

The final test accuracy was 73.380%.

```
Student model test accuracy: 73.380 %
Is above threshold performance? True
```

References

[1] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. "Distilling the knowledge in a neural network". In: *arXiv preprint arXiv:1503.02531* (2015).

[2] James Bergstra and Yoshua Bengio. "Random search for hyper-parameter optimization." In: *Journal of machine learning research* 13.2 (2012).