Homework 4

4190.408 001, Artificial Intelligence

November 19, 2024

1 Submission

In this homework we will learn and practice the basics of deep learning by building a deep learning model and training the model by ourselves. You may search the internet, but you should not copy and paste it as a whole.

- Assignment due: Dec 3, 11:59 pm
- Individual Assignment
- Submission through ETL. Please read the instructions in the PDF, and **fill in the TODOs in the ipynb file and submit the ipynb file including your outputs**. The file name should be in the format of "{student_ID}_{first name}_{last name}.ipynb", e.g. "2024-11111_Seungho_Lee.ipynb". Please write your name in English.
- We recommend to use **Google colab** for running the code.
- Collaborations on solving the homework is allowed. Discussions are encouraged but you should think about the problems on your own.
- If you do collaborate with someone or use a book or website, you are expected to write up your solution independently. That is, close the book and all of your notes before starting to write up your solution.
- Make sure you cite your work/collaborators at the end of the homework.
- **Honor Code**: This document is exclusively for Fall 2024, 4190.408 students with Professor Hanbyul Joo at Seoul National University. Any student who references this document outside of that course during that semester (including any student who retakes the course in a different semester), or who shares this document with another student who is not in that course during that semester, or who in any way makes copies of this document (digital or physical) without consent of Professor Hanbyul Joo is guilty of cheating, and therefore subject to penalty according to the Seoul National University Honor Code.

2 Implementing Models

In this homework, we will implement and train a simple image classifier based on PyTorch. We first have to implement a model, which receives image tensors as input and computes classified labels as output. There are many possible choices, but in this homework we will implement ResNet-50. ResNet is a widely used model, known to solve the vanishing gradient problem which appeared in models with deep network structures, by exploiting residual learning. For detailed explanation, please look up the paper here.

2.1 Bottleneck

ResNet-50 is made up of building blocks called "bottlenecks". The structure of a building block is as follows:

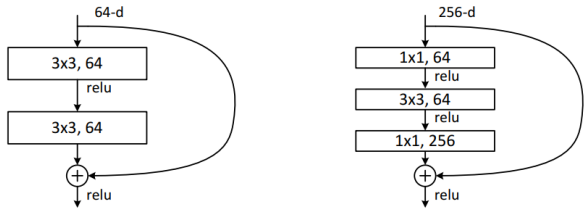

Figure 5. A deeper residual function $\mathcal{F}$ for ImageNet. Left: a building block (on 56×56 feature maps) as in Fig. 3 for ResNet-34. Right: a "bottleneck" building block for ResNet-50/101/152.

First implement a building block by filling in the following skeleton code.

- Each weight layer consists of `conv2d` - `batchnorm2d` - `ReLU` activation
- To match the dimension for residual learning, implement and use the `self.shortcut` in the skeleton code.
- The number 1 and 3 in the figure indicates kernel sizes.
- For first and last `conv2d` layer, fix the stride to 1. For the second `conv2d` layer, stride is determined by input.

```
class Bottleneck(nn.Module):
 def __init__(self, in_channels, channels, stride=1):
   ...
 def forward(self, x):
   ...
```

Input: number of input channels and channels of the convolution layer.
Output: Bottleneck module
Description: You will implement the building block for ResNet-50 model.

2.2 ResNet-50

With the bottleneck block, we will implement ResNet-50 with the blocks. The structure of ResNet-50 is in the following table (3rd column):

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Implementation details:

- First layer (the `conv1` layer) is made up of `conv2d` layer with kernel size 7, channel 64, and stride 2, (and set padding to 3) followed by a `batchnorm2d` layer.
- `conv2_x`, `conv3_x`, `conv4_x`, `conv5_x` layer consists of the bottleneck blocks, with different number of channels.
- There should be a `MaxPool2d` layer before `conv2_x` with kernel size 3 and stride 2, padding 1.
- For `conv2_x`, set stride to 1. For `conv3_x`, `conv4_x`, `conv5_x` set stride to 2.
- The output after `conv5_x` should pass average pooling, and a fully connected layer.
  Use `torch.nn.AdaptiveAvgPool2d` for average pooling and `nn.Linear` for fully connected layer.
- You do not have to implement softmax. (In training, we will use cross entropy loss)

```
class ResNet50(nn.Module):
 def __init__(self, num_blocks=[3,4,6,3], num_classes=10):
   ...
 def forward(self,x):
   ...
```

Input: Number of blocks for each layer and number of labels to classify. Both are given in this case.
Output: ResNet-50 model
Description: You will implement ResNet-50 model with the bottleneck blocks.

3 Training

As we finished building the model, we will implement optimizer and criterion for training. The details are as follows:

- SGD optimizer (you can change and test with different parameters if you want)
  Use `torch.optim.SGD` for implementation.
- Use cross entropy loss (as this is a classification task)
  Use `torch.nn.CrossEntropyLoss` for implementation.

We will move on to training our model with the data loaded above. For each training epoch, the following has to be done:

- Get data (batches), load images and labels from the data. Use the `trainloader` in the skeleton code.
- Set parameter gradients to zero. Use `optimizer.zero_grad()`.
- Feedforward data to get the model output.
- Compute losses, and do backpropagation for optimization. Use `loss.backward()` and `optimizer.step()`.

You may add additional functions to print or plot the losses.

4 Testing & Result Report

Evaluate the trained model with testing dataset. The overall process is similar with training the model, except that (1) the model is in evaluation mode, (2) the `autograd` is turned off.

Once you finish the implementation, write the results in the ipynb file. The results should include:

- results of the training loss per epoch (doesn't have to print/plot the results for every epoch. Printing/plotting within a certain frequency is fine)
- results (losses) for evaluation.
- accuracy of classification in percentage for evaluation. Note: The accuracy percentage would not affect your grade. That is, regardless of the accuracy percentage value, you will get your grade if you measure and write it on your report.

You may do additional experiments (e.g., changing parameters, etc) but this will not affect your grade, which means there will be no points on additional experiments.