Tutorial on Deep Learning

Mengmi Zhang

Postdoc in Kreiman Lab
Boston Children’s Hospital, Harvard Medical School

Feb, 25, 2020
Topics to cover

• How to use google colab (folder navigation; execute shell commands; mount google drive)
• How to train a classifier for image recognition
• How to write a custom dataset loader
• How to fine-tune a pre-trained model on a small dataset (transfer learning)
• How to visualize model, learnt weights and learnt representations
Object classification/Image recognition

Image

What is the predicted image label?

10 object classes:
- airplane
- automobile
- bird
- cat
- deer
- dog
- frog
- horse
- ship
- truck

1x10 “confidence” vector

Input:

Output:
Class label with the highest confidence
Which deep learning platforms to pick

- Pytorch
- Tensorflo
- Caffe2
- Keras

- Preferred programming languages (C++, python, java)
- Industrial standards (scalability)
- Popularity among researchers (more open source codes)
- Hardware supports (multiple GPUs and TPUs)

Platform comparisons: https://www.netguru.com/blog/deep-learning-frameworks-comparison
Tensor

Tensor - A multi-dimensional array which can hold gradients with respect to the tensor

Image credit: http://corochann.com/understanding-convolutional-layer-1227.html
Feature extraction - 2D Convolution

Image (3 input channels x 5x5)

Convolution kernel (3 input channels x 3x3)

Image credit: http://deeplearning.net/software/theano/tutorial/conv_arithmetic.html
Feature extraction - Rectifier Linear Unit (ReLU)

>>> torch.nn.ReLU()
Feature extraction - 2D Max pooling

Input feature maps
(#input channels x 4x4)

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>3</td>
<td>5</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>1</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>9</td>
<td>3</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>8</td>
<td>4</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

Output feature maps
(#input channels x 2x2)

maxpool

Stride = 2
Kernel size = 2

>>> torch.nn.MaxPool2d(kernel_size=2, stride=2)

Image credit: https://computersciencewiki.org/index.php/Max-pooling_/_Pooling
Classification - Flattern (vectorize the tensor)

Feature Extraction
Conv + relu + pooling
(spatial information is preserved)

Image (3x224x224)

Output feature maps
(10 x 2 x 2)

#outputChannels = 10

Flattern

Output feature vector
(1x 40)

>>> torch.nn.View(output_dim = 40)
Classification - Fully connected layer

\[ W_{10 \times 40} \]

Matrix multiplication

Output feature vector (40 x 1)

Logit score vector (10 x 1)

```python
>>> torch.nn.Linear(inputDimension = 40, outputDimension = 10)
```
Classification – Softmax layer

Logit score vector 
(1x10)

\[ f(z_j) = \frac{e^{z_j}}{\sum_k e^{z_k}} \]

10 object classes:
airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck

Example:
\[
\begin{align*}
z &= \{8, 4, 2\} \\
\sum_k e^{z_k} &= 2981 + 54.6 + 7.4 = 3043 \\
f(8) &= 2981 / 3043 = 0.98 \\
f(4) &= 54.6 / 3043 = 0.018 \\
f(2) &= 7.4 / 3043 = 0.002
\end{align*}
\]

1x10 “confidence” vector

\[ \text{Example:} \]

Sum adds to 1

\>>> torch.nn.functional.softmax(input = logit score vector)
Network (stacks of layers)

```
import torch.nn as nn
import torch.nn.functional as F

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 18)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x

model = Net()
```

Import pytorch libraries

Layer definition

- **x** is input
- (batch_size, channel_size, width, height)

Define how inputs pass through each layer

Output logit score vector

(batch_size x #classes)

Construct a model

Save checkpoint:

```python
torch.save(
    {'epoch': epoch,
    'model_state_dict': model.state_dict(),
    'optimizer_state_dict': optimizer.state_dict(),
    'loss': loss,
    ...}, PATH)
```

Load checkpoint to resume training:

```python
model = Net()
optimizer = TheOptimizerClass(*args, **kwargs)
checkpoint = torch.load(PATH)
model.load_state_dict(checkpoint['model_state_dict'])
optimizer.load_state_dict(checkpoint['optimizer_state_dict'])
epoch = checkpoint['epoch']
loss = checkpoint['loss']
```
Loss function, optimizer, and epoch

```python
import torch.optim as optim

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)

for epoch in range(2):
    # loop over the dataset multiple times
    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        # get the inputs; data is now of shape: [N, C, H, W]
        inputs, labels = data

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()
        if i % 2000 == 1999:  # print every 2000 mini-batches
            print('[%d, %5d] loss: %.3f' %
                  (epoch + 1, i + 1, running_loss / 2000))
        running_loss = 0.0

print('Finished Training')
```

**Import pytorch libraries**

**Define classification loss**

**Define optimizer; learning rate (typical range: 0.01 – 0.0001), momentum (typical range: 0.5-2)**

**Epoch:** how many times do you want to loop your entire dataset

**Iteration:** we normally call “an iteration” as “processing of one image batch”

**Input images and corresponding class labels (a scalar from 0 – 9; e.g. cat = 3)**

**Clear ∆W**

**Given inputs, forward network to get predicted outputs**

**Compare predicted outputs and labels, and compute loss**

**Compute ∆W for each layer in the network**

**Update Wnew = ∆W + Wold for each layer in the network**

**It is a good practice to save checkpoint in the end of each epoch**
Other considerate – data batch

At each iteration, the network updates its weights based on multiple images (image batch) instead of one image.

Informal and intuitive understanding:
Update rule: $W_{\text{new}} = lr \times \Delta W + W_{\text{old}}$

E.g. Batch size = 3

Average of $(\Delta W(\text{img1}), \Delta W(\text{img2}), \Delta W(\text{img3}))$

Update rule: $W_{\text{new}} = lr \times \text{averaged } \Delta W + W_{\text{old}}$ $\quad$ More stable training, less variance

Input image batch: batch_size x 3 x 224 x 224

Batch feature maps: batch_size x feature map size

Batch logit score vectors: batch_size x 10

Batch output vectors: batch_size x 10

>>> torch.utils.data.DataLoader (dataset, batch_size = 3)
Other considerate – data pre-processing

Normalize the pixel value from \([0,1]\) to \([-1,1]\)

\[
input = \frac{input - \text{mean}}{\text{standard deviation}}
\]

Image augmentation: random crop, random flip

Image augmentation: but not random in validation

Original
Random Rotation
Horizontal Flip
Vertical Flip
Random Crop around Center

mean  std

\([0.485, 0.456, 0.406], [0.229, 0.224, 0.225]\)

# Data augmentation and normalization for training
# Just normalization for validation

data_transforms = {
    'train': transforms.Compose([
        transforms.RandomResizedCrop(224),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]),
    'val': transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]),
}
Other considerate – data splits & cross validation

Dataset (100 images per class, 10 classes)

20 images per class, 10 classes  |  20 images per class, 10 classes  |  20 images per class, 10 classes  |  20 images per class, 10 classes  |  20 images per class, 10 classes

Splits to 5 groups

5-fold Cross-validation

cross validation – test model generalization
validation set – prevent overfitting
What to do when dataset size is small

• Self-collection
• Merge with existing open-source datasets
• Crowd sourcing platforms (mturk, or just ask)
• Use 3D modeling softwares (e.g. Blender and Unity3D) to construct realistic 3D scenes
• Use generative models (train another model to generate fake data)
• Data augmentation (random scaling, crops, rotations, etc)
• Use pre-trained models
Dummy framework

For each fold cross-validation
  Construct dataloader
  Construct model
  Define loss and optimizer
  for each epoch
    for each iteration (image batch) in dataloader
      optimizer.zero_grad()
      output = model.forward(input)
      loss = loss(outputs, ground truth)
      loss.backward()
      optimizer.step()
    end each iteration
  test on validation set
  save model at this epoch
  end each epoch
test on test set
Google Co-lab hands-on

***Download All colab source codes and datasets from here: https://drive.google.com/open?id=17cYN2Je1jsxLsaZ83SL7UMBecABftVAY

• How to use google colab (folder navigation; execute shell commands; mount google drive)
  https://colab.research.google.com/drive/1Yse4xR0RECZBfNQFeef3z5iH0AiB37ZT

• How to train a classifier for image recognition
  https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html

• How to write a custom dataset loader
  https://pytorch.org/tutorials/beginner/data_loading_tutorial.html

• How to fine-tune a pre-trained model on a small dataset (transfer learning)
  https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html

• How to visualize learnt weights and learnt activation maps