APS360 Fundamentals of AI

Lisa Zhang

Lecture 8; May 30, 2019
Agenda

Last Class:

- Convolutional Neural Networks

Today:

- CNN Architectures
- Fully Convolutional Networks
- Neural Network Debugging
- Train/Test Split
CNN Architectures
Named Architectures

- LeNext
- AlexNet
- VGG
- ResNet

You should know:

- How do we interpret CNN figures?
- How were these architectures different from the previous?
- What new idea was introduced?
import torchvision.models
alexNet = torchvision.models.alexnet(pretrained=False)
Q: What is new in AlexNet (compared to LeNet)?
There are many VGG versions

vgg16 = torchvision.models.vgg.vgg16(pretrained=False)
vgg19 = torchvision.models.vgg.vgg19(pretrained=False)
Q: What is new in VGG (compared to AlexNet)?
GoogLeNet (Inception)

torchvision.models.inception.inception_v3(pretrained=False)

Q: What is new in GoogLeNet that we haven’t seen yet?
Inception Module
torchvision.models.resnet.resnet18(pretrained=False)
torchvision.models.resnet.resnet152(pretrained=False)

Q: What is new in ResNet that we haven’t seen yet?
Q: What is new in ResNet that we haven’t seen yet?
Skip connections to make very deep neural networks

torchvision.models.resnet.resnet18(pretrained=False)
torchvision.models.resnet.resnet152(pretrained=False)
# normal layer application:
next_activation = layer(activation)

# residual layer application
next_activation = activation + layer(activation)
Skip Connections

- Made it easier to train deeper neural networks
- Information about weight updates are passed **backwards** from the output towards the input
- Difficult for information to propagate to the earlier layers

Note: You don’t need to know the math behind why skip connections are better
Q: How is this network different from what we have seen so far?
Why avoid fully connected layers?

▶ So that the neural network can (theoretically) take arbitrary dimension images as input
Instead of fully connected layers:

- Use a convolution layer with the same kernel size as hidden unit size and no padding
- Use global average-pooling
Neural Network Debugging
Most bugs are invisible and manifest only in poor performance.

How do you know whether poor performance is due to:
- a bug
- poor architecture/hyperparameter choice
- data quantity/quality
- something else?

Please make sure you flip through the reading:


Slides 1-34, 46-47, 52-75 (for now, there are more useful information here later)
Steps to building a neural network

1. Start with a simple model
2. Get your training code to run without syntax and runtime errors
3. Get your code to overfit on a small subset of the training set (single batch)
4. Actual training
1. Simple Model

- Start with something like LargeNet modified to fit the new problem
- Some number of convolutions, then 1 or 2 fully-connected layer(s)
2. Common Runtime Errors

- Labels out of order
- Incorrect shapes for tensors
- Incompatible types of tensors (float32 vs float64 vs long)
- Incorrect pre-processing of images (not scaling the pixels to the range \([0, 1]\), or normalize to mean 0, std 1)
- Incorrect input to the loss function (pre-softmax vs post-softmax)
- Forgetting `optimizer.zero_grad()` cleanup step
- **Learning rate too high**

Recommended solutions for some of these in:

Q: What does overfitting on a small data set achieve?
Q: What does overfitting on a small data set achieve?

▶ “Quickly” = maybe ~100 iterations
▶ Check that your learning rate isn't too low or too high
▶ You can use the Adam optimizer:
  ```python
  optim.Adam(model.parameters(), lr=learning_rate)
  ```
  ▶ Adam generally trains faster than SGD
  ▶ Usually the go-to optimizer for modern practitioners
Questions?
Train/Test Split Strategies for Lab 3
Proposed Strategy #1

Strategy:

▶ Each student has three sets of gesture images submitted
▶ Place two of those sets in the training/validation set
▶ Place one of those sets in the test set

Q: What do you think about this strategy?
Proposed Strategy #2

Strategy:
  ▶ Randomly split the images into training, validation and test

Q: What do you think about this strategy?
Proposed Strategy #3

Strategy:

- Split **students** into training/validation and test
  - If a student is in the test set, then all images generated by that student is in the test set.
- Hand pick which students are in which set

Q: What do you think about this strategy?
Data splitting is hard
You will need to make some trade-offs, especially with limited data
Be honest when reporting what you did, and explain your choices
Other thoughts: Recommend use ImageFolder

Sample code:

```python
from google.colab import drive
drive.mount('/content/gdrive')
# Upload data
!unzip '/content/gdrive/My Drive/train_data.zip'
images = datasets.ImageFolder(root='train_data/', transform=transforms.ToTensor())
images = list(images)
```
Other thoughts: Saving the AlexNet output

I don’t think anyone is there yet, but when you get there...

- Don’t compute AlexNet features every time during training!
- Save the features for each input image
- When training your model, start with the saved features (rather than the image pixels)
Lab Today

- I will be there too
- Walk-through of lab 2 code
- Office Hour Monday 4pm-5pm