CS6501: Deep Learning for Visual Recognition

CNN Architectures
Today’s Class

Recap
• The Convolutional Layer
• Spatial Pooling Operations

CNN Architectures
• LeNet (LeCun et al 1998)
• AlexNet (Krizhevsky et al 2012)
• VGG-Net (Simonyan and Zisserman, 2014)
Automatic Differentiation

You only need to write code for the forward pass, backward pass is computed automatically.

Pytorch (Facebook -- mostly): https://pytorch.org/

Tensorflow (Google -- mostly): https://www.tensorflow.org/

DyNet (team includes UVA Prof. Yangfeng Ji): http://dynet.io/
import torch.nn as nn
import torch.nn.functional as F

class TwoLayerNN(nn.Module):
    def __init__(self):
        super(TwoLayerNN, self).__init__()

        self.linear1 = nn.Linear(1 * 28 * 28, 512)
        self.linear2 = nn.Linear(512, 10)

    def forward(self, x):
        x = x.view(batchSize, 1 * 28 * 28)
        z = F.relu(self.linear1(x))
        return self.linear2(z)
1. Creating Model, Loss, Optimizer

```python
# Create the model.
model = TwoLayerNN()
loss_fn = nn.CrossEntropyLoss()

# Define a learning rate.
learningRate = 5e-2

# Optimizer.
optimizer = optim.SGD(model.parameters(), lr=learningRate,
momentum = 0.9, weight_decay = 1e-4)
```
2. Running forward and backward on a batch

```python
# Forward pass. (Prediction stage)
scores = model(inputs)
loss = loss_fn(scores, labels)
```

```python
# Zero the gradients in the network.
optimizer.zero_grad()

# Backward pass. (Gradient computation stage)
loss.backward()

# Parameter updates (SGD step) -- if done with torch.optim!
optimizer.step()
```

Compare this to what we had to do for toynn
Convolutional Layer

Weights
Convolutional Layer

Weights
Convolutional Layer
Convolutional Layer (with 4 filters)

Input: 1x224x224

weights: 4x1x9x9

Output: 4x224x224

if zero padding, and stride = 1
Convolutional Layer (with 4 filters)

Input: 1x224x224

weights: 4x1x9x9

Output: 4x112x112

if zero padding, but stride = 2
**Convolutional Layer in pytorch**

```
class torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True)  [source]
```

**Input**
- `in_channels` (e.g. 3 for RGB inputs)

**Output**
- `out_channels` (equals the number of convolutional filters for this layer)
Convolutional Network: LeNet

Yann LeCun
LeNet in Pytorch

```python
# LeNet is French for The Network, and is taken from Yann Lecun's 98 paper
# on digit classification http://yann.lecun.com/exdb/lenet/
# This was also a network with just two convolutional layers.

class LeNet(nn.Module):
    def __init__(self):
        super(LeNet, self).__init__()
        # Convolutional layers.
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.conv2 = nn.Conv2d(6, 16, 5)

        # Linear layers.
        self.fc1 = nn.Linear(16*5*5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        out = F.relu(self.conv1(x))
        out = F.max_pool2d(out, 2)
        out = F.relu(self.conv2(out))
        out = F.max_pool2d(out, 2)

        # This flattens the output of the previous layer into a vector.
        out = out.view(out.size(0), -1)
        out = F.relu(self.fc1(out))
        out = F.relu(self.fc2(out))
        out = self.fc3(out)
        return out
```
SpatialMaxPooling Layer

take the max in this neighborhood
LeNet Summary

• 2 Convolutional Layers + 3 Linear Layers

• + Non-linear functions: ReLUs or Sigmoids
  + Max-pooling operations
New Architectures Proposed

- Alexnet (Krizhevsky et al NIPS 2012) [Required Reading]
- VGG (Simonyan and Zisserman 2014)
- GoogLeNet (Szegedy et al CVPR 2015)
- ResNet (He et al CVPR 2016)
- DenseNet (Huang et al CVPR 2017)
Convolutional Layers as Matrix Multiplication

https://petewarden.com/2015/04/20/why-gemm-is-at-the-heart-of-deep-learning/
Convolutional Layers as Matrix Multiplication

Pros?
Cons?

https://petewarden.com/2015/04/20/why-gemm-is-at-the-heart-of-deep-learning/
CNN Computations are Computationally Expensive

- However highly parallelizable
- GPU Computing is used in practice (Why is GPU computing Good?)
- CPU Computing in fact is prohibitive for training these models
ILSVRC:
Imagenet Large Scale Visual Recognition Challenge
[Russakovsky et al 2014]
The Problem: Classification

Classify an image into 1000 possible classes:
e.g. Abyssinian cat, Bulldog, French Terrier, Cormorant, Chickadee, red fox, banjo, barbell, hourglass, knot, maze, viaduct, etc.

cat, tabby cat (0.71)
Egyptian cat (0.22)
red fox (0.11)
.....
The Data: ILSVRC

Imagenet Large Scale Visual Recognition Challenge (ILSVRC): Annual Competition

1000 Categories

~1000 training images per Category

~1 million images in total for training

~50k images for validation

Only images released for the test set but no annotations, evaluation is performed centrally by the organizers (max 2 per week)
The Evaluation Metric: Top K-error

True label: Abyssinian cat

cat, tabby cat (0.61)
Egyptian cat (0.22)
red fox (0.11)
Abyssinian cat (0.10)
French terrier (0.03)
.....

Top-1 error: 1.0   Top-1 accuracy: 0.0
Top-2 error: 1.0   Top-2 accuracy: 0.0
Top-3 error: 1.0   Top-3 accuracy: 0.0
Top-4 error: 0.0   Top-4 accuracy: 1.0
Top-5 error: 0.0   Top-5 accuracy: 1.0
Top-5 error on this competition (2012)
Alexnet (Krizhevsky et al NIPS 2012)
Alexnet

https://www.saagie.com/fr/blog/object-detection-part1
Pytorch Code for Alexnet

• In-class analysis

https://github.com/pytorch/vision/blob/master/torchvision/models/alexnet.py
Dropout Layer

Srivastava et al 2014
What is happening?

https://www.saagie.com/fr/blog/object-detection-part1
Feature extraction (SIFT) → Feature encoding (Fisher vectors) → Classification (SVM or softmax)

Deep Learning

Convolutional Network (includes both feature extraction and classifier)
Preprocessing and Data Augmentation
Preprocessing and Data Augmentation
Preprocessing and Data Augmentation

224x224
Preprocessing and Data Augmentation

224x224
True label: Abyssinian cat
Other Important Aspects

- Using ReLUs instead of Sigmoid or Tanh
- Momentum + Weight Decay
- Dropout (Randomly sets Unit outputs to zero during training)
- GPU Computation!

<table>
<thead>
<tr>
<th>Model</th>
<th>Top-1</th>
<th>Top-5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sparse coding [2]</td>
<td>47.1%</td>
<td>28.2%</td>
</tr>
<tr>
<td>SIFT + FVs [24]</td>
<td>45.7%</td>
<td>25.7%</td>
</tr>
<tr>
<td>CNN</td>
<td>37.5%</td>
<td>17.0%</td>
</tr>
</tbody>
</table>
VGG Network

Simonyan and Zisserman, 2014.

https://github.com/pytorch/vision/blob/master/torchvision/models/vgg.py

BatchNormalization Layer

**Input:** Values of $x$ over a mini-batch: $\mathcal{B} = \{x_1...m\}$; Parameters to be learned: $\gamma, \beta$

**Output:** $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$

\[
\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^{m} x_i \quad \text{ // mini-batch mean}
\]

\[
\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \quad \text{ // mini-batch variance}
\]

\[
\tilde{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad \text{ // normalize}
\]

\[
y_i \leftarrow \gamma \tilde{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i) \quad \text{ // scale and shift}
\]
Questions?