CS4501: Introduction to Computer Vision
Neural Networks + Backpropagation
Last Class

• Softmax Classifier
• Generalization / Overfitting
• Pytorch
Today’s Class

• Global Features
• The perceptron model
• Neural Networks – multilayer perceptron model (MLP)
• Backpropagation
Supervised Machine Learning Steps

Training

- Training Images
- Image Features
- Training
- Training Labels
- Learned model

Testing

- Test Image
- Image Features
- Prediction
Supervised Learning – Softmax Classifier

\( x_i = [x_{i1} \ x_{i2} \ x_{i3} \ x_{i4}] \)

Extract features

Run features through classifier

\[
\begin{align*}
g_c &= w_{c1}x_{i1} + w_{c2}x_{i2} + w_{c3}x_{i3} + w_{c4}x_{i4} + b_c \\
g_d &= w_{d1}x_{i1} + w_{d2}x_{i2} + w_{d3}x_{i3} + w_{d4}x_{i4} + b_d \\
g_b &= w_{b1}x_{i1} + w_{b2}x_{i2} + w_{b3}x_{i3} + w_{b4}x_{i4} + b_b
\end{align*}
\]

Get predictions

\[
\hat{y}_i = [f_c \ f_d \ f_b]
\]

\[
\begin{align*}
f_c &= e^{g_c}/(e^{g_c} + e^{g_d} + e^{g_b}) \\
f_d &= e^{g_d}/(e^{g_c} + e^{g_d} + e^{g_b}) \\
f_b &= e^{g_b}/(e^{g_c} + e^{g_d} + e^{g_b})
\end{align*}
\]
Last Class:
(mini-batch) Stochastic Gradient Descent (SGD)

\[ \lambda = 0.01 \]

Initialize \( w \) and \( b \) randomly

\[ l(w, b) = \sum_{i \in B} -\log f_i, label(w, b) \]

\( B \) is a small set of training examples.

\( \text{for } e = 0, \text{num\_epochs} \text{ do} \)
\( \text{for } b = 0, \text{num\_batches} \text{ do} \)
  \( \text{Compute: } \frac{dl(w, b)}{dw} \text{ and } \frac{dl(w, b)}{db} \)
  \( \text{Update } w: \quad w = w - \lambda \frac{dl(w, b)}{dw} \)
  \( \text{Update } b: \quad b = b - \lambda \frac{dl(w, b)}{db} \)
  \( \text{Print: } l(w, b) \quad // \text{Useful to see if this is becoming smaller or not.} \)
\( \text{end} \)
\( \text{end} \)
Generalization

- Generalization refers to the ability to correctly classify never before seen examples
- Can be controlled by turning “knobs” that affect the complexity of the model

Training set (labels known)  Test set (labels unknown)
Overfitting

\( f \) is linear

\( f \) is cubic

\( f \) is a polynomial of degree 9

\( \text{Loss}(w) \) is high

\( \text{Loss}(w) \) is low

\( \text{Loss}(w) \) is zero!

Underfitting

Overfitting

High Bias

High Variance
Pytorch: Project Assignment 4

- http://vicenteordonez.com/vision/
Image Features

- In your Project 4: Nearest Neighbors + Softmax Classifier features are:
  
  Image: 3x32x32

  Feature: 3072-dim vector
Image Features: Color

Photo by: marielito

slide by Tamara L. Berg
80 million tiny images: a large dataset for non-parametric object and scene recognition

Antonio Torralba, Rob Fergus and William T. Freeman
However, these are all images of people but the colors in each image are very different.

Scikit-image implementation
Image Features: HoG

+ Block Normalization

Figure from Zhuolin Jiang, Zhe Lin, Larry S. Davis, ICCV 2009 for human action recognition.
Image Features: GIST

The “gist” of a scene: Oliva & Torralba, 2001
Image Features: GIST

Oriented edge response at multiple scales (5 spatial scales, 6 edge orientations)

Hays and Efros, SIGGRAPH 2007
Image Features: GIST

Aggregated edge responses over 4x4 windows

Hays and Efros, SIGGRAPH 2007
Image Features: Bag of (Visual) Words Representation

Object → Bag of ‘words’
Summary: Image Features

- Largely replaced by Neural networks
- Still useful to study for inspiration in designing neural networks that compute features.

- Many other features proposed
  - LBP: Local Binary Patterns: Useful for recognizing faces.
  - Dense SIFT: SIFT features computed on a grid similar to the HOG features.
  - etc.
Perceptron Model

Frank Rosenblatt (1957) - Cornell University

\[ f(x) = \begin{cases} 
1, & \text{if } \sum_{i=0}^{n} w_i x_i + b > 0 \\
0, & \text{otherwise}
\end{cases} \]

More: https://en.wikipedia.org/wiki/Perceptron
Perceptron Model

Frank Rosenblatt (1957) - Cornell University

\[ f(x) = \begin{cases} 
1, & \text{if } \sum_{i=0}^{n} w_i x_i + b > 0 \\
0, & \text{otherwise}
\end{cases} \]

More: https://en.wikipedia.org/wiki/Perceptron
Perceptron Model

Frank Rosenblatt (1957) - Cornell University

\[ f(x) = \begin{cases} 
1, & \text{if } \sum_{i=0}^{n} w_i x_i + b > 0 \\
0, & \text{otherwise} 
\end{cases} \]

More: https://en.wikipedia.org/wiki/Perceptron
Activation Functions

ReLU ($x$) = max(0, $x$)

Tanh ($x$)

Sigmoid ($x$)

Step ($x$)

ReLU ($x$) = max(0, $x$)
import torch
import torch.nn as nn
import torch.autograd

network = nn.Sequential(
    nn.Linear(4, 1),
    nn.Sigmoid()
)

batch_size = 16
input_vector = torch.autograd.Variable(torch.Tensor(batch_size, 4))
predictions = network(input_vector)
print(predictions.size())

torch.Size([16, 1])
Two-layer Multi-layer Perceptron (MLP)

\[ \sum \rightarrow \text{Activation} \rightarrow a_1 \rightarrow \sum \rightarrow a_2 \rightarrow \sum \rightarrow a_3 \rightarrow \sum \rightarrow a_4 \rightarrow \sum \rightarrow \hat{y}_1 \rightarrow \text{Loss / Criterion} \]

- \( x_1, x_2, x_3, x_4 \) are inputs.
- \( a_1, a_2, a_3, a_4 \) are activations in the "hidden" layer.
- \( \hat{y}_1 \) is the predicted output.
- \( y_1 \) is the actual output.

\( \sum \) denotes summation.
Forward pass

\[ z_i = \sum_{i=0}^{n} w_{1ij}x_i + b_1 \]

\[ a_i = \text{Sigmoid}(z_i) \]

\[ p_1 = \sum_{i=0}^{n} w_{2i}a_i + b_2 \]

\[ y_1 = \text{Sigmoid}(p_i) \]

\[ \text{Loss} = L(y_1, \hat{y}_1) \]
Backward pass

\[
\frac{\partial L}{\partial x_k} = \left( \frac{\partial}{\partial x_k} \sum_{i=0}^{n} w_{1ij} x_i + b_1 \right) \frac{\partial L}{\partial z_i}
\]

\[
\frac{\partial L}{\partial z_i} = \frac{\partial}{\partial z_i} \text{Sigmoid}(z_i) \frac{\partial L}{\partial a_k}
\]

\[
\frac{\partial L}{\partial a_k} = \left( \frac{\partial}{\partial a_k} \sum_{i=0}^{n} w_{2i} a_i + b_2 \right) \frac{\partial L}{\partial p_1}
\]

\[
\frac{\partial L}{\partial p_1} = \frac{\partial}{\partial p_1} \text{Sigmoid}(p_i) \frac{\partial L}{\partial \hat{y}_1}
\]

\[
\frac{\partial L}{\partial \hat{y}_1} = \frac{\partial}{\partial \hat{y}_1} L(y_1, \hat{y}_1)
\]

\[
\frac{\partial L}{\partial w_{1ij}} = \frac{\partial x_k}{\partial w_{1ij}} \frac{\partial L}{\partial x_k}
\]

\[
\frac{\partial L}{\partial w_{2i}} = \frac{\partial a_k}{\partial w_{2i}} \frac{\partial L}{\partial a_k}
\]
Pytorch – Two-layer MLP + Regression

```python
In [9]:
import torch
import torch.nn as nn
import torch.autograd

network = nn.Sequential(
    nn.Linear(4, 4),
    nn.Sigmoid(),
    nn.Linear(4, 1),
    nn.Sigmoid()
)

batch_size = 16
input_vector = torch.autograd.Variable(torch.Tensor(batch_size, 4))
predictions = network(input_vector)
predictions.size()
print(predictions.size())
torch.Size([16, 1])

In [10]:
criterion = nn.MSELoss()
loss = criterion(predictions, labels)
```
Pytorch – Two-layer MLP + LogSoftmax

```
In [16]:
import torch
import torch.nn as nn
import torch.autograd

network = nn.Sequential(
    nn.Linear(3072, 512),
    nn.Sigmoid(),
    nn.Linear(512, 10),
    nn.LogSoftmax()
)

batch_size = 16
input_vector = torch.autograd.Variable(torch.Tensor(batch_size, 3072))
predictions = network(input_vector)
print(predictions.size())

torch.Size([16, 10])

In [13]:
criterion = nn.NLLLoss()
loss = criterion(predictions, labels)
```
Pytorch – Two-layer MLP + LogSoftmax

In [17]:
```python
import torch
import torch.nn as nn
import torch.autograd

network = nn.Sequential(
    nn.Linear(3072, 512),
    nn.Sigmoid(),
    nn.Linear(512, 10),
)

batch_size = 16
input_vector = torch.autograd.Variable(torch.Tensor(batch_size, 3072))
predictions = network(input_vector)
print(predictions.size())
```

LogSoftmax + Negative Likelihood Loss

In [13]:
```python
criterion = nn.CrossEntropyLoss()
loss = criterion(predictions, labels)
```
PyTorch documentation

PyTorch is an optimized tensor library for deep learning using GPUs and CPUs.

Notes

- Autograd mechanics
- Broadcasting semantics
- CUDA semantics
- Extending PyTorch
- Multiprocessing best practices
- Serialization semantics

Package Reference

- torch
- torch.Tensor
- torch.sparse
- torch.Storage
- torch.nn
- torch.nn.functional
Questions?