COMPUTER VISION: DEEP LEARNING LAB COURSE
DAY 3 – CONVOLUTIONAL NEURAL NETWORKS

SEBASTIAN HOUBEN
Schedule

Today

- Neural Nets
- Training of Neural Nets
- Gradient Computation
- Deep Neural Nets
- Bare Necessities for Training Deep Neural Nets
- Tensorflow
Neural Net – Multilayer Perceptron

Feature Extraction

[\begin{pmatrix} 2 \\ 5 \\ 1 \\ 8 \end{pmatrix}]

Classifier

\{\text{cat, dog}\}
Neural Net – Multilayer Perceptron
Neural Net – Multilayer Perceptron

\[
\begin{align*}
\sigma(z) &= \frac{1}{1 + e^{-z}} \\
W &= \begin{pmatrix}
w_{11} & w_{12} & w_{13} \\
w_{21} & w_{22} & w_{23} \\
w_{31} & w_{32} & w_{33} \\
w_{41} & w_{42} & w_{43} \\
w_{51} & w_{52} & w_{53}
\end{pmatrix} \\
U &= \begin{pmatrix}
u_{11} & u_{12} & u_{13} & u_{14} & u_{15} \\
u_{21} & u_{22} & u_{23} & u_{24} & u_{25}
\end{pmatrix} \\
x &= \begin{pmatrix}
x_1 \\
x_2 \\
x_3
\end{pmatrix} \\
\tilde{y} &= \begin{pmatrix}
\tilde{y}_1 \\
\tilde{y}_2
\end{pmatrix} \\
\tilde{y} &= U \sigma (W x)
\end{align*}
\]
Neural Net – Multilayer Perceptron

\[ \sigma(z) = \frac{1}{1 + e^{-z}} \]

\[
W := \begin{pmatrix}
w_{11} & w_{12} & w_{13} \\
w_{21} & w_{22} & w_{23} \\
w_{31} & w_{32} & w_{33} \\
w_{41} & w_{42} & w_{43} \\
w_{51} & w_{52} & w_{53}
\end{pmatrix}
\]

\[
U := \begin{pmatrix}
u_{11} & u_{12} & u_{13} & u_{14} & u_{15} \\
u_{21} & u_{22} & u_{23} & u_{24} & u_{25}
\end{pmatrix}
\]

\[
x := \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}, \quad \tilde{y} := \begin{pmatrix} \tilde{y}_1 \\ \tilde{y}_2 \end{pmatrix}, \quad y := \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}
\]

\[
\tilde{y} = U \sigma \left( W x \right)
\]
Neural Net – Non-Linearities

\[ \sigma(z) = \frac{1}{1 + e^{-z}} = \frac{1}{2} \left( 1 + \tanh \frac{z}{2} \right) \]

\[ \sigma'(z) = \sigma(z)(1 - \sigma(z)) \]

\[ \tanh'(z) = 1 - \tanh^2(z) \]
Neural Net – Interpretation

- Input norm should be limited
- Nothing should fire for zero input
- Shift by mean and normalize by standard deviation (over training set)
  \[ x := \frac{\hat{x} - \text{mean}}{\text{std}} \]
- Hidden neuron reacts if input is similar to weight vector
- Hidden neurons code regions of feature space
- More hidden neurons can divide the feature space in more regions
Neural Net – Interpretation

- Second layer weights control output for each region
- Net can approximate each continuous function
- Polynomials can
- Sine functions can (Fourier series)
Neural Net – Training

- Training data: \((x^{(i)}, y^{(i)}); i = 1, \ldots, n; y^{(i)} \in \{0; 1\}^m; x^{(i)} \in \mathbb{R}^d\)
- Predictions (!): \(\tilde{y}^{(i)} = \sigma \left( U \sigma \left( W x^{(i)} \right) \right)\)
- Accuracy: \(\frac{1}{n} \sum_{i=0}^{n} \min_j \left[ \arg\max_j \tilde{y}_j^{(i)} = \arg\max_j y_j^{(i)} \right]\)
- Loss: \(L(x, y, W, U) = \sum_{i=1}^{n} \sum_{j=1}^{m} \left( \tilde{y}_j^{(i)} - y_j \right)^2\)
  \[= \sum_{i=1}^{n} \sum_{j=1}^{m} \left( \sigma \left( U \sigma \left( W x^{(i)} \right) \right)_j - y_j \right)^2\]
- Training:
  \[W^{(k+1)} = W^{(k)} - \eta \frac{\partial}{\partial W} L(x, y, W^{(k)}, U^{(k)})\]
  \[U^{(k+1)} = U^{(k)} - \eta \frac{\partial}{\partial U} L(x, y, W^{(k)}, U^{(k)})\]

\[y^{(i)} = \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix}, \quad \tilde{y}^{(i)} = \begin{pmatrix} 0.1 \\ \vdots \\ 0.44 \end{pmatrix}\]

- \(n\) samples
- \(m\) classes
- \(d\) input size
Neural Net – Training

- Simple case: \( d = 1, m = 1 \)

\[
L(x, y, w, u) = \sum_{i=1}^{n} \left( \sigma \left( u \sigma \left( wx^{(i)} \right) \right) - y \right)^2
\]

\[
\frac{\partial}{\partial w} L(x, y, w, u) = \sum_{i=1}^{n} 2 \left( \sigma \left( u \sigma \left( wx^{(i)} \right) \right) - y \right) \cdot \sigma' \left( u \sigma \left( wx^{(i)} \right) \right) \cdot u \sigma' \left( wx^{(i)} \right) \cdot x^{(i)}
\]

\[
\frac{\partial}{\partial u} L(x, y, w, u) = \sum_{i=1}^{n} 2 \left( \sigma \left( u \sigma \left( wx^{(i)} \right) \right) - y \right) \cdot \sigma' \left( u \sigma \left( wx^{(i)} \right) \right) \cdot \sigma \left( wx^{(i)} \right)
\]

- Vanishing Gradient

- \( n \) samples
- \( m \) classes
- \( d \) input size
Neural Net – Training

\[ W^{(k+1)} = W^{(k)} - \eta \frac{\partial}{\partial W} L(x, y, W^{(k)}, U^{(k)}) \]
\[ U^{(k+1)} = U^{(k)} - \eta \frac{\partial}{\partial U} L(x, y, W^{(k)}, U^{(k)}) \]

- Basic form of gradient: \( \sum_{i=1}^{n} d(x^{(i)}, y^{(i)}, W, U) \)
- Gradient descent: \( W^{(k+1)} = W^{(k)} - \eta \sum_{i=1}^{n} d(x^{(i)}, y^{(i)}, W^{(k)}, U^{(k)}) \)
- Stochastic gradient descent: \( W^{(k+1)} = W^{(k)} - \eta d(x^{(i)}, y^{(i)}, W^{(k)}, U^{(k)}) \)
- Batch gradient descent \( W^{(k+1)} = W^{(k)} - \eta \sum_{i \in \text{BATCH}} d(x^{(i)}, y^{(i)}, W^{(k)}, U^{(k)}) \)
Deep Neural Nets

- 3-Layer network can approximate any continuous function
- More layers tend to work better
  - Not quite clear why
  - Handwavy: Natural phenomenons are hierarchically structured
  - Hopefully layers will adapt to those different phenomenons
- Vanishing Gradient Problem
- Many, many parameters
Deep Neural Nets - Adaptations

- **Non-Linearity:** $\sigma(z) \rightarrow \text{ReLU}(z) = \max\{0, z\}$

Output:

$$\text{softmax}(z) = \frac{\exp(z_i)}{\sum_k \exp(z_k)}$$

- **Loss function:** $(f(x) - y)^2 \rightarrow -\sum_i y_i \log \text{softmax}(z)_i$

Cross-entropy

- **Weight initialization:** $W \sim \mathcal{N}(0, 0.1)$

- **Data preparation:** $x^{(k)} := \frac{x^{(k)} - \text{mean}(x^{(1)}, ..., x^{(n)})}{\text{std}(x^{(1)}, ..., x^{(n)})}$

$$\begin{pmatrix} 21 \\ 34 \\ \vdots \\ 10,000 \\ 680 \end{pmatrix} \rightarrow \begin{pmatrix} 0.002 \\ 0.004 \\ \vdots \\ 0.927 \\ 0.03 \end{pmatrix}$$
Convolutional Neural Nets

- Neural net with parameter reuse
- Each layer gets an image with $c$ channels as input
- This is convoluted with $d$ filters of size $n \times n \times c$
- resulting in an image with $d$ channels
- Idea: Find certain local image patches / patterns

Alex Krizhevsky et al.
Convolutional Neural Nets

- Idea: Exact location of image patch is not so important
- Compress information
- Maxpool-Layers: Take small window (e.g., 2x2) and only propagate maximum value to next layer

Alex Krizhevsky et al.
Convolutional Neural Nets

- Idea: at the end only relevant information is propagated
- Use classical neural net (fully-connected) to classify results

Alex Krizhevsky et al.
Tensorflow

- Python library for Deep Learning
  - Gradient computation
  - Backpropagation
  - 2000+ operations (e.g., convolution, maxpooling)
- Symbolic computation
  - Write a program that writes (and executes) a program
  - Similar to Numpy
Tensorflow

- GPU paralellization (via CUDA kernels)
- Caveats:
  - Slightly hard to learn
  - Hard to debug

- Alternatives:
  - PyTorch (Torch)
  - Theano (basically the same)
  - Caffe (C++)
  - Keras (Simplification of Theano / Tensorflow)
import tensorflow as tf
import numpy as np

tf.reset_default_graph()  # tensorflow internal reset

x = tf.Variable(np.array([2, 1]), dtype=tf.float32, name="x")  # a variable in the program our program writes
y = tf.constant(np.array([3, 5]), dtype=tf.float32, name="y")  # a constant in the program our program writes

z = tf.placeholder(shape=[None, 2], dtype=tf.float32, name="z")  # an input in the program our program writes

loss = tf.reduce_sum((x - y + z)**2)  # many other numpy operations are implemented

train_step = tf.train.GradientDescentOptimizer(0.1).minimize(loss)  # a subroutine that takes one gradient descent step on loss

z_ = np.array([[2, 0]])

with tf.Session() as sess:
sess.run(tf.global_variables_initializer())  # initialize variables in program

print(loss.eval(feed_dict={z:z_}), x.eval())  # 17.0, [2. 1.]

for k in range(100):
    train_step.run(feed_dict={z:z_})  # compute loss, compute backpass (derivative), one step downwards

print(loss.eval(feed_dict={z:z_}), x.eval())  # 9.66338e-13, [1.00000024 4.9999905]
import tensorflow as tf
import numpy as np

tf.reset_default_graph()  # tensorflow internal reset

# fancy model implemented here ...

train_step = tf.train.GradientDescentOptimizer(0.1).minimize(loss)
saver = tf.train.Saver()
epoch_cnt = 0

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())  # initialize variables in program
    saver.restore(sess, tf.train.latest_checkpoint(os.path.dirname(os.path.realpath(__file__))))  # restore last checkpoint
    for ...
        epoch_cnt += 1
        # optimization going on here ...
        saver.save(sess, os.path.dirname(os.path.realpath(__file__)) + '/tsd_model', global_step=epoch_cnt, write_meta_graph=False)  # save current state of variables (but not the model)
Tensorflow

- NWHC order
  - stacking of images
  - number, width, height, channel
  - try to adapt to this order
QUESTIONS?
EXERCISES.