Deep Learning

CMSC 422
MARINE CARPUAT
marine@cs.umd.edu

Based on slides by Vlad Morariu
Standard Application of Machine Learning to Computer Vision

- Features: e.g., Scale Invariant Feature Transform (SIFT)
- Classifiers: SVM, Random Forests, KNN, ...
- Features are hand-crafted, not trained
  - eventually limited by feature quality

Cat image credit: https://raw.githubusercontent.com/BVLC/caffe/master/examples/images/cat.jpg
• Deep learning
  – multiple layer neural networks
  – learn features and classifiers directly ("end-to-end" training)
  – breakthrough in Computer Vision, now in other AI areas

Speech Recognition

According to Microsoft’s speech group:

Using DL

Word error rate on Switchboard

1990 2000 2010

Slide credit: Bohyung Han
Image Classification Performance

Today’s lecture: key concepts

- Convolutional Neural Networks
- Revisiting Backpropagation and Gradient Descent for Deep Networks
Multi-Layer Perceptron (MLP)

Neural Networks Applied to Vision


- USPS digit recognition, later check reading
- Convolution, pooling (“weight sharing”), fully connected layers

Architecture overview

Components:

– Convolution layers
– Pooling/Subsampling layers
– Fully connected layers

Convolutional Layer

32x32x3 image

32 height

32 width

3 depth

Slide credit: Fei-Fei Li, Andrej Karpathy, and Justin Johnson
Convolutional Layer

32x32x3 image

5x5x3 filter

Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”
Convolutional Layer

- **32x32x3 image**
- **5x5x3 filter**

**Convolve** the filter with the image, i.e., "slide over the image spatially, computing dot products"

Filters always extend the full depth of the input volume

Slide credit: Fei-Fei Li, Andrej Karpathy, and Justin Johnson
Convolutional Layer

32x32x3 image
5x5x3 filter \( w \)

1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image (i.e. \( 5 \times 5 \times 3 = 75 \)-dimensional dot product + bias)

\[ w^T x + b \]
Convolutional Layer

32x32x3 image
5x5x3 filter

convolve (slide) over all spatial locations

activation map

Slide credit: Fei-Fei Li, Andrej Karpathy, and Justin Johnson
Convolutional Layer

32x32x3 image
5x5x3 filter

convolve (slide) over all spatial locations

consider a second, green filter

activation maps

Slide credit: Fei-Fei Li, Andrej Karpathy, and Justin Johnson
For example, if we had 6 5x5 filters, we’ll get 6 separate activation maps:

We stack these up to get a “new image” of size 28x28x6!
ConvNet is a sequence of Convolutional Layers, interspersed with activation functions.

Slide credit: Fei-Fei Li, Andrej Karpathy, and Justin Johnson
ConvNet is a sequence of Convolutional Layers, interspersed with activation functions.

- **3x3 CONV, ReLU** e.g. 6 filters
- **5x5 CONV, ReLU** e.g. 10 filters
- **3x3 CONV, ReLU** e.g. 6 filters

Slide credit: Fei-Fei Li, Andrej Karpathy, and Justin Johnson
Rectified Linear Units (ReLU)

- Use rectified linear function instead of sigmoid
  \[ \text{ReLU}(x) = \max(0, x) \]

- Advantages
  - Fast
  - No vanishing gradients
- makes the representations smaller and more manageable
- operates over each activation map independently

Slide credit: Fei-Fei Li, Andrej Karpathy, and Justin Johnson
Pooling Layer

MAX POOLING

Single depth slice

max pool with 2x2 filters and stride 2

Slide credit: Fei-Fei Li, Andrej Karpathy, and Justin Johnson
Convolutional filter visualization

Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Slide credit: Fei-Fei Li, Andrej Karpathy, and Justin Johnson
We call the layer convolutional because it is related to convolution of two signals:

\[ f[x,y] \ast g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1,n_2] \cdot g[x-n_1,y-n_2] \]

elementwise multiplication and sum of a filter and the signal (image)

example 5x5 filters (32 total)

Slide credit: Fei-Fei Li, Andrej Karpathy, and Justin Johnson
Today’s lecture: key concepts

• Convolutional Neural Networks

• Revisiting Backpropagation and Gradient Descent for Deep Networks
Multi-Layer Perceptron (MLP)

Single neuron gradient

\[ z = b + \sum_i w_i x_i \]
\[ \hat{y} = \frac{1}{1 + e^{-z}} \]
\[ L = \frac{1}{2} \sum_n (y^n - \hat{y}^n)^2 \]

\[ \frac{\partial z}{\partial w_i} = x_i \]
\[ \frac{d\hat{y}}{dz} = \hat{y}(1 - \hat{y}) \]
\[ \frac{\partial L}{\partial \hat{y}^n} = -(y^n - \hat{y}^n) \]

Chain rule: If \( y = f(x), z = g(y) \), then \( \frac{dz}{dx} = \frac{dy}{dx} \frac{dz}{dy} \)

\[ \frac{\partial L}{\partial w_i} = \sum_i \frac{\partial \hat{y}^n}{\partial w_i} \frac{\partial L}{\partial \hat{y}^n} = \sum_i \frac{\partial z^n}{\partial w_i} \frac{d\hat{y}^n}{dz^n} \frac{\partial L}{\partial \hat{y}^n} = -\sum_i x_i^n \hat{y}^n(1 - \hat{y}^n)(y^n - \hat{y}^n) \]
Single neuron training

for $t = 1, \ldots, T$

\[
\hat{y}^n = f(x^n, w_t) \quad (n = 1, \ldots, N)
\]

\[
\frac{\partial L}{\partial w_i} = - \sum_n x_i^n \hat{y}^n (1 - \hat{y}^n)(y^n - \hat{y}^n) \quad (i = 1, \ldots, d)
\]

\[
w_{t+1} = w_t + \Delta w
\]

endfor

an epoch

Slide credit: Adapted from Bohyung Han
Multi-Layer: Backpropagation

\[
\frac{\partial L}{\partial z_j} = \frac{d\hat{y}_j}{dz_j} \frac{\partial L}{\partial \hat{y}_j}
\]

\[
\frac{\partial L}{\partial \hat{y}_i} = \sum_j \frac{d\hat{y}_i}{d\hat{y}_j} \frac{\partial L}{\partial z_j} = \sum_j w_{ij} \frac{\partial L}{\partial z_j} = \sum_j w_{ij} \frac{d\hat{y}_j}{dz_j} \frac{\partial L}{\partial \hat{y}_j}
\]

\[
\frac{\partial L}{\partial w_{ki}} = \sum_n \frac{\partial z_i^n}{\partial w_{ki}} \frac{d\hat{y}_i^n}{dz_i^n} \frac{\partial L}{\partial \hat{y}_i^n} = \sum_n \frac{\partial z_i^n}{\partial w_{ki}} \frac{d\hat{y}_i^n}{dz_i^n} \sum_j w_{ij} \frac{d\hat{y}_j^n}{dz_j^n} \frac{\partial L}{\partial \hat{y}_j^n}
\]

Slide credit: Bohyung Han
Backpropagation in practice

Two passes per iteration:

- **Forward pass:** compute value of loss function (and intermediate neurons) given inputs

- **Backward pass:** propagate gradient of loss (error) backwards through the network using the chain rule
Stochastic Gradient Descent (SGD)

- Update weights for each sample

\[ E = \frac{1}{2} (y^n - \hat{y}^n)^2 \quad w_i(t + 1) = w_i(t) - \epsilon \frac{\partial E^n}{\partial w_i} \]

+ Fast, online
- Sensitive to noise

- Minibatch SGD: Update weights for a small set of samples

\[ E = \frac{1}{2} \sum_{n \in B} (y^n - \hat{y}^n)^2 \quad w_i(t + 1) = w_i(t) - \epsilon \frac{\partial E^B}{\partial w_i} \]

+ Fast, online
+ Robust to noise
SGD improvements: Momentum

• Remember the previous direction

\[ v_i(t) = \alpha v_i(t - 1) - \epsilon \frac{\partial E}{\partial w_i}(t) \]

\[ w(t + 1) = w(t) + v(t) \]

+ Converge faster
+ Avoid oscillation

Slide credit: Bohyung Han
SGD improvements: Weight Decay

• Penalize the size of the weights

\[ C = E + \frac{1}{2} \sum_i w_i^2 \]

\[ w_i(t + 1) = w_i(t) - \epsilon \frac{\partial C}{\partial w_i} = w_i(t) - \epsilon \frac{\partial E}{\partial w_i} - \lambda w_i \]

+ Improve generalization a lot!

Slide credit: Bohyung Han
Key concepts

- Convolutional Neural Networks
- Revisiting Backpropagation and Gradient Descent for Deep Networks
History: NN Revival in the 1980’s

Backpropagation discovered in 1970’s but popularized in 1986


MLP is a universal approximator

- Can approximate any non-linear function in theory, given enough neurons, data

Generated lots of excitement and applications

Neural Networks Applied to Vision

LeNet – vision application

- USPS digit recognition, later check reading
- Convolution, pooling (“weight sharing”), fully connected layers

Issues in Deep Neural Networks

• Prohibitive training time
  – Especially with lots of training data
  – Many epochs typically required for optimization
  – Expensive gradient computations

• Overfitting
  – Learned function fits training data well, but performs poorly on new data (high capacity model, not enough training data)

Slide credit: adapted from Bohyung Han
Issues in Deep Neural Networks

Vanishing gradient problem

\[
\frac{\partial E}{\partial w_{ki}} = \sum_n \frac{\partial z_i^n}{\partial w_{ki}} \frac{d\hat{y}_i^n}{dz_i^n} \frac{\partial E}{d\hat{y}_i^n} = \sum_n \frac{\partial z_i^n}{\partial w_{ki}} \frac{d\hat{y}_i^n}{dz_i^n} \sum_j w_{ij} \frac{d\hat{y}_j^n}{dz_j^n} \frac{\partial E}{d\hat{y}_j^n}
\]

– Gradients in the lower layers are typically extremely small
– Optimizing multi-layer neural networks takes huge amount of time

Slide credit: adapted from Bohyung Han
New “winter” and revival in early 2000’s

New “winter” in the early 2000’s due to

• problems with training NNs
• Support Vector Machines (SVMs), Random Forests (RF) – easy to train, nice theory

Revival again by 2011-2012

• Name change (“neural networks” -> “deep learning”)
• + Algorithmic developments
  – unsupervised layer-wise pre-training
  – ReLU, dropout, layer normalization
• + Big data + GPU computing =
• Large outperformance on many datasets (Vision: ILSVRC’12)

Big Data

- **ImageNet Large Scale Visual Recognition Challenge**
  - 1000 categories w/ 1000 images per category
  - 1.2 million training images, 50,000 validation, 150,000 testing

AlexNet Architecture

60 million parameters!

Various tricks

- ReLU nonlinearity
- Overlapping pooling
- Local response normalization
- Dropout – set hidden neuron output to 0 with probability .5
- Data augmentation
- Training on GPUs

GPU Computing

• Big data and big models require lots of computational power

• GPUs
  – thousands of cores for parallel operations
  – multiple GPUs
  – still took about 5-6 days to train AlexNet on two NVIDIA GTX 580 3GB GPUs (much faster today)
Recurrent Neural Networks

**Networks with loops**
- The output of a layer is used as input for the same (or lower) layer
- Can model dynamics (e.g. in space or time)

Image credit: Christopher Olah’s blog [http://colah.github.io/posts/2015-08-Understanding-LSTMs/](http://colah.github.io/posts/2015-08-Understanding-LSTMs/)
Recurrent Neural Networks

Let’s unroll the loops

- Now a standard feed-forward network with many layers
- Suffers from vanishing gradient problem
- In theory, can learn long term memory, in practice not (Bengio et al, 1994)

Image credit: Christopher Olah’s blog http://colah.github.io/posts/2015-08-Understanding-LSTMs/
Y. Bengio, P. Simard, P. Frasconi. Learning Long-Term Dependencies with Gradient Descent is Difficult. In TNN
1994.
Long Short Term Memory (LSTM)

- A type of RNN explicitly designed not to have the vanishing or exploding gradient problem
- Models long-term dependencies
- Memory is propagated and accessed by gates
- Used for speech recognition, language modeling ...

Autoencoders

- Encode then decode the same input
- No supervision needed