Convolutionary Neural Networks

Lingzhou Xue

Penn State Department of Statistics
lingzhou@psu.edu
http://sites.psu.edu/sldm/deeplearning/
Materials

• C. Lee Giles (Penn State)
  • http://clgiles.ist.psu.edu/IST597/index.html

• Andrew Ng (Coursera, Stanford U., Google Brain, Biadu)
  • https://www.deeplearning.ai/

• Fei-Fei Li (Stanford U., Google Cloud)
  • http://vision.stanford.edu/teaching.html

• Aaron Courville (U Montreal), Ian Goodfellow (Google Brain), and Yoshua Bengio (U Montreal)
  • http://www.deeplearningbook.org/
Outline

1. Convolutional Neural Networks
   • Development
   • Application
   • Architecture
   • Optimization

2. Deep Learning Software

3. TensorFlow Demos
1. Convolutional Neural Networks
Convolutional Neural Networks (CNNs/ConvNets)

- Convolution: a specialized type of linear operation
- CNNs: a specialized type of neural network using the convolution (instead of general matrix multiplication) in at least one of its layers
- CNNs are biologically-inspired to emulate the animal visual cortex
The dendrites in biological neurons perform complex nonlinear computations. The synapses are not just a single weight, they’re a complex non-linear system. The activation function takes the decision of whether or not to pass the signal.
Activation Functions: Sigmoid/Logistic and Tanh

• (-) Gradients at tails are almost zero (vanishing gradients)
• (-) Outputs (y-axis) are not zero-centered for Sigmoid/Logistic
• (-) Computationally expensive
Rectified Linear Unit (ReLU)

• (+) Accelerate the convergence of SGD compared to sigmoid/tanh  
• (+) Easy to implement by simply thresholding at zero  
• First introduced by Hahnloser et al. (Nature, 2000) with strong biological motivations and mathematical justifications  
• First demonstrated in 2011 to enable better training of deeper neural networks  
• The most popular activation as of 2018.
**ILSVRC Milestones**

- **LeNet-5 (1998):** 7 layers, 60k parameters
  (4 main operations: convolution, nonlinearity, pooling, FC Layer)

- **AlexNet (2012):** 16.4% top-5 error rate, 8 layers, 60m parameters

- **VGGNet (2014):** 7.3% top-5 error rate, 16 layers, 138m parameters

- **GoogLeNet (2014):** 6.7% top-5 error rate, 22 layers, 5m parameters

- **ResNet (2015):** 3.57% top-5 error rate, 152 layers
Applications

Classification

Retrieval

Applications (Cont’d)

Detection

Segmentation

Figures copyright Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun. 2015. Reproduced with permission.

[Faster R-CNN: Ren, He, Girshick, Sun 2015]


[Farabet et al., 2012]
Applications (Cont’d)

self-driving cars

NVIDIA Tesla line
(these are the GPUs on rye01.stanford.edu)

Note that for embedded systems a typical setup would involve NVIDIA Tegras, with integrated GPU and ARM-based CPU cores.
Applications (Cont’d)

No errors

Minor errors

Somewhat related

Image Captioning

[Vinyals et al., 2015]
[Karpathy and Fei-Fei, 2015]

A white teddy bear sitting in the grass

A man in a baseball uniform throwing a ball

A woman is holding a cat in her hand

A man riding a wave on top of a surfboard

A cat sitting on a suitcase on the floor

A woman standing on a beach holding a surfboard

All images are CC0 Public domain:
https://dls-Odhp.com/cryp3/3pgd1-anique-car-13431958
https://dls-Odhp.com/crypt/4g3p1-aas2-3e3e2-l1-b3b3-1343148
https://dls-Odhp.com/crypt/3y4n-summer-apoth-1698218
https://dls-Odhp.com/crypt/wm-fealn-model-port-fall-1343687
https://dls-Odhp.com/crypt-knopu-letter-1343600
https://dls-Odhp.com/crypt-knopu-letter-1343600
Captions generalized by Justin Johnson using Numps182
Applications (Cont’d)
Application: Predicting Poverty

From left to right: urban areas, nonurban areas, water, and roads

Jean et al. (Science, 2016)
Application: Predicting Poverty (Cont’d)

Predicting Multidimensional Poverty Index (MPI) for Senegal

Pokhriyal and Jacques (PNAS, 2017)
Application: Galaxy Morphology Prediction

Kaggle Galaxy Challenge
Training: 61578 galaxies
Testing: 79975 galaxies

Dieleman, Willett & Dambre (2015)
Application: Galaxy Morphology Prediction (Cont’d)

Dieleman, Willett & Dambre (2015)
Architecture

- CNNs are usually built by stacking the following types of layers:
  - Convolutional Layer (CONV)
  - Pooling Layer (POOL)
  - RELU Layer will apply an elementwise activation function such as ReLU
  - Fully-Connected Layer (FC)
  - Normalization Layer
- CONV/FC have parameters
- CONV/FC/POOL have hyperparameters
LeNet-5 (LeCun et al., 1998)
AlexNet (Krizhevsky et al., 2012)
Convolutional Layer

• The core building block (it does most of the computational heavy lifting)
• Require 4 hyperparameters (K filters, F width, S stride, P zero padding)
• Accept a volume of size W x H x D
• Produce a volume of size \((W-F+2P)/S+1 \times (H-F+2P)/S+1 \times K\)
• Include a total of \(FxFxDxK\) weights and \(K\) biases for \(K\) filters
• Next, we will focus on two important ideas in CONV Layer
  • Sparse Connectivity
  • Shared Parameters
The Convolution Operation

- Convolution is a new function \( s \), the weighted average of \( x \)
  \[
  s(t) = \int x(a)w(t-a)\,da
  \]
- This operation is typically denoted with an asterisk
  \[
  s(t) = (x \ast w)(t)
  \]
- \( w \) needs to be a valid pdf, or the output is not a weighted average.
- \( w \) needs to be 0 for negative arguments, or we will look into the future.
- In convolution network terminology the first function \( x \) is referred to as the *input*, the second function \( w \) is referred to as the *kernel*.
- The output \( s \) is referred to as the *feature map*. 
The Convolution Operation (Cont’d)

- If we use a 2D image $I$ as input and use a 2D kernel $K$ we have

$$S(i,j) = (I * K)(i,j) = \sum_{m} \sum_{n} I(m,n)K(i-m,j-n)$$
Traditional vs Convolutional Neural Networks

- Traditional neural network layers use matrix multiplication by a matrix of parameters with a separate parameter describing the interaction between each input unit and each output unit:
  \[ s = g(W^T x) \]

- With \( m \) inputs and \( n \) outputs, matrix multiplication requires \( m \times n \) parameters and \( O(m \times n) \) runtime per example.

- This means every output unit interacts with every input unit.

- Convolutional network layers have sparse interactions.
Sparse Connectivity

• CNNs exploit the **spatially-local correlation** by enforcing a local connectivity pattern between neurons of adjacent layers.

• The learnt “filters” produce the strongest response to a spatially local input pattern. However, stacking many such layers leads to (non-linear) “filters” that is increasingly “global” (to a larger width).
Sparse Connectivity (Cont’d)

- Highlight one input $x_3$ and output units $s$ affected by it
- Top: when $s$ is formed by convolution with a kernel of width 3, only three outputs are affected by $x_3$
- Bottom: when $s$ is formed by matrix multiplication connectivity is no longer sparse
  - So all outputs are affected by $x_3$
Sparse Connectivity (Cont’d)

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

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

- Parameter sharing scheme is used in Convolutional Layers to control the number of parameters.

- Each filter is replicated to share the same parameterization (weight vector and bias) and form a feature map.

- It only introduces a total of $F \times F \times D$ weights and 1 bias per filter.
Shared Parameters (Cont’d)

Share the same parameters across different locations (assuming input is stationary):

Convolutions with learned kernels
Efficiency of Convolution for Edge Detection

- Image on right formed by taking each pixel of input image and subtracting the value of its neighboring pixel on the left.
  - This is a measure of all the vertically oriented edges in input image which is useful for object detection.

Both images are 280 pixels tall
Input image is 320 pixels wide
Output image is 319 pixels wide

- Transformation can be described by a convolution kernel containing two elements and requires $319 \times 320 \times 3 = 267,960$ flops (2 mpy's, one add).
- Same transformation would require $320 \times 280 \times 319 \times 280$, i.e., 8 billion entries in the matrix.
- Convolution is 4 billion times more efficient.
Pooling Layer

• The function of POOL is to progressively reduce the spatial size of the representation, which helps reduce the amount of parameters and computation in CNNs, and hence to also control overfitting.

• Examples of pooling functions
  • MaxPooling
  • AveragePooling
  • $L_2$-NormPooling

• MaxPooling partitions the input into non-overlapping rectangles and, for each such sub-region, outputs the maximum value only.
MaxPooling

- It is another important concept of CNNs.
- It is a form of non-linear down-sampling.
- By eliminating non-maximal values, it reduces computation for upper layers.
- Also, it provides a form of translation invariance. The translation invariance is important when we care about whether a feature is present rather than exactly where it is.
  - For example, we just need to know that an eye is present in a region (not its exact location) when detecting a face.
MaxPooling Introduces Invariance to Translation

- View of middle of output of a convolutional layer

- Same network after the input has been shifted by one pixel

- Every input value has changed, but only half the values of output have changed because maxpooling units are only sensitive to maximum value in neighborhood not exact value
Effect of Pooling after ReLU
Now: non-linearity, dimension reduction, translation invariance

The 2nd CONV layer performs convolution on the output of the 1st POOL Layer using six filters to produce a total of six feature maps. ReLU is then applied on all of these six feature maps. We then perform Max Pooling operation separately on each of the six rectified feature maps.
A Deep Classification Network

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

• “Fully Connected” implies that every neuron in the previous layer is connected to every neuron on the next layer, as in regular NNs.

• CONV layers provide a meaningful, low-dimensional, and invariant feature space. FC layer learns a (non-linear) function in the space
Training CNNs

\[ E_{\text{total}} = \frac{1}{2} \sum (\text{target} - \text{output})^2 \]
Optimization: Stochastic Gradient Descent (SGD)

• Once the analytic gradient is computed with backpropagation, the gradients are used to perform a parameter update.
• SGD: follow the negative gradient of the objective using a few data points.
• Motivation:
  • often in practice computing the cost and gradient for the entire training set can be very slow and sometimes intractable.
  • give an easy way to incorporate new data in an ‘online’ setting.
• SGD overcomes the high cost of running back propagation over the full training set, and it still leads to fast convergence for CNNs.
**SGD + Momentum**

- Momentum is one method for pushing the objective more quickly along the shallow ravine. When the gradient keeps changing its direction, the momentum will smooth out the variations.

- *With Momentum update, the parameter vector will build up velocity in any direction that has consistent gradient.*
SGD + Momentum (Cont’d)

SGD

$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

while True:
    dx = compute_gradient(x)
    x += learning_rate * dx

SGD+Momentum

$$v_{t+1} = \rho v_t + \nabla f(x_t)$$

$$x_{t+1} = x_t - \alpha v_{t+1}$$

vx = 0
while True:
    dx = compute_gradient(x)
    vx = rho * vx + dx
    x += learning_rate * vx

- Build up “velocity” as a running mean of gradients
- Rho gives “friction”; typically rho=0.9 or 0.99
2. Deep Learning Software
Neural Network Frameworks

• Low level frameworks/libraries
  • TensorFlow (Python, wrappers): https://www.tensorflow.org/
  • Theano (Python): http://www.deeplearning.net/software/theano/
  • pyTorch (Python, based on Torch): https://pytorch.org/
  • Torch (Lua): http://torch.ch/
  • Caffe (C++): http://caffe.berkeleyvision.org/
  • DeepLearning4J (Java): https://deeplearning4j.org/

• High level frameworks/APIS
  • Keras: https://keras.io/
  • Chainer: https://chainer.org/
  • Lasagne: https://lasagne.readthedocs.io/en/latest/
Neural Network Frameworks (Cont’d)

- Caffe (UC Berkeley) → Caffe2 (Facebook)
- Torch (NYU/Facebook) → PyTorch (Facebook)
- Theano (U Montreal) → TensorFlow (Google)
- Paddle (Baidu)
- CNTK (Microsoft)
- MXNet (Amazon, main framework of choice at AWS)
TensorFlow/Theano vs. Scikit-learn

  - Machine Learning in Python
  - Models already built, “off-the-shelf”
  - Paradigm: fit/ predict style
- TensorFlow/Theano
  - Build model from ground up
  - Paradigm: Describe model in form of a “data-graph”
  - Operators/functions (like add, max, etc.)
  - APIs like Keras play nicely with these frameworks (abstraction)
- Advantages
  - Cross platform: Android, Linux, etc.
  - Quick turn around to production
  - Efficient computation utilizing CPUs & GPUs
Theano: Overview

- Theano was the priestess of Athena in Troy [source: Wikipedia].
- It is also a Python package for symbolic differentiation.\(^a\)
- Open source project primarily developed at the University of Montreal.
- Symbolic equations compiled to run efficiently on CPU and GPU.
- Computations are expressed using a NumPy-like syntax:
  - `numpy.exp()` – `theano.tensor.exp()`
  - `numpy.sum()` – `theano.tensor.sum()`

\(^a\)TensorFlow (Google’s Theano alternative) is similar.
How does it work?

Internally, Theano builds a graph structure composed of:

- interconnected variable nodes (red),
- operator (op) nodes (green),
- and “apply” nodes (blue, representing the application of an op to some variables)

```python
1 import theano.tensor as T
2 x = T.dmatrix('x')
3 y = T.dmatrix('y')
4 z = x + y
```
Theano: Pros and Cons

- (+) Python + numpy
- (+) Computational graph is nice abstraction
- (+) RNNs fit nicely in computational graph
- (-) Raw Theano is somewhat low-level
- (+) High level wrappers (Keras, Lasagne) ease the pain
- (-) Error messages can be unhelpful
- (-) Large models can have long compile times
- (-) Much “fatter” than Torch; more magic
- (-) Patchy support for pretrained models
TensorFlow: Overview

• Open source software library from Google for machine learning

• Especially good for training and implementing deep neural networks

• Example applications include image recognition, automated translation. Think Google Photos, Translate

• Used in production at Uber, SnapChat, Google, & many others
How does TensorFlow work?

• Uses data flow graphs to represent a learning model
  
  • Comprise of nodes and edges
  
  • Nodes represent mathematical operations
  
  • Edges represent multi-dimensional data arrays (tensors)
  
  • “TensorFlow”

• C based with Python and C++ APIs
Computation as a Dataflow Graph
Dataflow Graph Paradigm

- Computations are done in 2 steps
  - First: Build the graph
  - Second: Execute the graph

- Both steps can be done in many languages (Python, C++, R, Scala?)
- Best supported so far is Python
# Numpy vs. TensorFlow

<table>
<thead>
<tr>
<th>Numpy</th>
<th>TensorFlow</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>a = np.zeros((2,2)); b = np.ones((2,2))</code></td>
<td><code>a = tf.zeros((2,2)), b = tf.ones((2,2))</code></td>
</tr>
<tr>
<td><code>np.sum(b, axis=1)</code></td>
<td><code>tf.reduce_sum(a, reduction_indices=[1])</code></td>
</tr>
<tr>
<td><code>a.shape</code></td>
<td><code>a.get_shape()</code></td>
</tr>
<tr>
<td><code>np.reshape(a, (1,4))</code></td>
<td><code>tf.reshape(a, (1,4))</code></td>
</tr>
<tr>
<td><code>b * 5 + 1</code></td>
<td><code>b * 5 + 1</code></td>
</tr>
<tr>
<td><code>np.dot(a,b)</code></td>
<td><code>tf.matmul(a, b)</code></td>
</tr>
<tr>
<td><code>a[0,0], a[:,0], a[0,:]</code></td>
<td><code>a[0,0], a[:,0], a[0,:]</code></td>
</tr>
</tbody>
</table>
TensorFlow: Pros and Cons

- (+) Python + numpy
- (+) Computational graph abstraction, like Theano; great for RNNs
- (+) Much faster compile times than Theano
- (+) Slightly more convenient than raw Theano?
- (+) TensorBoard for visualization
- (+) Data AND model parallelism; best of all frameworks
- (+/−) Distributed models, but not open-source yet
- (-) Slower than other frameworks right now
- (-) Much “fatter” than Torch; more magic
- (-) Not many pretrained models
Keras: Overview

What is Keras?
- Neural Network library written in Python
- Designed to be minimalistic & straightforward yet extensive
- Built on top of either Theano as newly TensorFlow

Why use Keras?
- Simple to get started, simple to keep going
- Written in python and highly modular; easy to expand
- Deep enough to build serious models
Keras: Pros and Cons

Pros:
- Easy to implement
- Lots of choice
- Extendible and customizable
- GPU
- High level
- Active community
- keras.io

Cons:
- Lack of generative models
- High level
- Theano overhead

Great to use, especially if you are just starting out and trying to learn!
References

• **Tutorials/Installation**
  • TensorFlow
    • https://www.tensorflow.org/
  • Theano
    • http://www.iro.umontreal.ca/~pift6266/H10/notes/deepintro.html
  • Keras
    • https://keras.io/getting-started/sequential-model-guide/
    • https://elitedatascience.com/keras-tutorial-deep-learning-in-python

• **Setup (Keras/Theano/TensorFlow)**
  • https://medium.com/learning-machine-learning/getting-tensorflow-theano-and-keras-on-windows-70c18f2c533b
3. TensorFlow Demos

Amal Agarwal

Penn State Department of Statistics
https://www.amalag.com
TensorFlow Demos

• Installing TensorFlow
  • https://www.tensorflow.org/install/

• Neural Network Playground
  • https://playground.tensorflow.org/

• Demo 1: Linear Regression

• Demo 2: Two-Layer Neural Network

• Demo 3: MNIST database using One-Layer Neural Network

• Demo 4: MINIST database using Convolutional Neural Network