Introduction of Deep Learning

Xinxiang Zhang

2/16/2017
Abstract

• Deep learning becomes increasingly important
  • Automatic Machine Translation
  • Object Classification in Photographs
  • Image Caption Generation
  • Automatic Game Playing (AlphaGO)
  • ...
Outline

• Introduction of Neural Network
• Introduction of popular Deep Learning Libraries
• Introduction of Deep Neural Network
  • Convolutional Neural Network
  • Auto-encoder
• Implementation of several Deep models
  • Convolutional Neural Network via Tensorflow
  • Auto-encoder via Matlab
• Applications of Deep models in ImageNet (AlexNet)
Introduction of Neural Network

• Basic Architecture
• Linear Classifier
• Transfer Function
• Gradient Descent
Popular Deep Learning Libraries

• Theano
• DeepLearnToolbox
• MatConvNet
• Caffe
• Tensorflow
• Keras
Theano

• What is Theano?
  • Symbolic computation library
  • CPU and GPU infrastructure
  • Optimized compiler

• Theano introduction, installation guides, tutorials, and documents
  • [http://deeplearning.net/software/theano/index.html](http://deeplearning.net/software/theano/index.html)

• GitHub Page
  • [https://github.com/Theano/Theano](https://github.com/Theano/Theano)
DeepLearnToolbox

• DeepLearnToolbox
  • A open-source Matlab toolbox for Deep Learning
  • Download in: https://github.com/rasmusbergpalm/DeepLearnToolbox

• Advantage
  • Matlab, easy to use
  • Open-source

• Disadvantage
  • Only CPU version, slow
MatConvNet

• MatConvNet
  • A open-source Matlab toolbox for Convolution Network
  • Download in: https://github.com/vlfeat/matconvnet

• Advantage
  • Matlab, easy to use
  • Pretrained models(VGG, AlexNet)
  • Support GPU

• Disadvantage
  • Complicated than DeepLearnToolbox
  • Support only Convolution Network
Caffe

• What is Caffe?
  • Open source deep learning framework maintained by Berkeley Vision and Learning Center (BVLC)
  • Mainly written in C++ and CUDA C with Python and Matlab interfaces

• Why Using Caffe?
  • Open source
  • Reliability, especially for large scale problem
  • Speed
  • Popularity
Caffe

• Official website (http://caffe.berkeleyvision.org)
• Download from the GitHub page (https://github.com/BVLC/caffe)
• Try the tutorials and reference models (http://caffe.berkeleyvision.org/tutorial/)
• Look through the detailed API documentations (http://caffe.berkeleyvision.org/doxygen/annotated.html)
Tensorflow

• What is Tensorflow?
  • Open source software library for numerical computation using data flow graphs.
  • Mainly written in C++, and defined handy new compositions of operators as writing a Python function.

• Why using Tensorflow?
  • Flexible architecture allows computation to one or more CPUs or GPUs in a desktop, server, or mobile device with a single API.
Tensorflow

- Official website: https://www.tensorflow.org/
- Tutorials: https://www.tensorflow.org/tutorials/
- GitHub page: https://github.com/tensorflow/tensorflow

- Recommended installation and Python coding IDE:
  - Anaconda: https://anaconda.org/
  - Jupyter Notebook (IDE): http://jupyter.org/
Keras

• What is Keras?
  • Keras is a high-level neural networks library, written in Python and capable of running on top of either Tensorflow and Theano

• Why using Keras?
  • Allows for easy and fast prototyping.
  • Supports both CNN and RNN, as well as combinations of the two.
  • Supports arbitrary connectivity schemes.
  • Runs seamlessly on CPU and GPU.
Keras

• Official website: https://keras.io/
• GitHub page: https://github.com/fchollet/keras
Introduction of Deep Neural Network

- Convolutional Neural Network (CNN)
- Auto-encoder
Convolutional Neural Network (CNN)

- Problem of fully connected NN:
  - The number of weights grows largely with the size of the input image
  - Pixels in distance are less correlated
Convolutional Neural Network (CNN)

• Locally connected NN:
  • Sparse connectivity: a hidden unit is only connected to a local patch (weights connected to the patch are called filter or kernel)
  • The learned filter is a spatially local pattern
Convolutional Neural Network (CNN)

• Shared weights:
  • Hidden nodes at different locations share the same weights.
  • It greatly reduces the number of parameters to learn.
Convolutional Neural Network (CNN)

• Convolution:
  • Computing the responses at hidden nodes is equivalent to convoluting the input image $x$ with a learned filter $w$
  • After convolution, a filter map $net$ is generated at the hidden layer:

$$net[i, j] = (X \ast W)[i, j] = \sum_{m} \sum_{n} X[m, n] W[i - m, j - n]$$
Convolutional Neural Network (CNN)
Convolutional Neural Network (CNN)

• Zero-padding (optional):
  • The valid feature map is smaller than the input after convolution
  • Implementation of neural networks needs to zero-pad the input $x$ to make it wider
Convolutional Neural Network (CNN)

• Downsampling convolutional layer (optional):
  • To reduce computational cost, we may want to skip some positions of the filter and sample only every \( s \) pixels in each direction.
  • A downsampling convolution function is defined as:
    \[
    net[i,j] = (X * W)[i \times s, j \times s]
    \]
    Where \( s \) is referred as the stride of this downsampling convolution.
Convolutional Neural Network (CNN)

- Multiple filters:
  - Multiple filters generate multiple feature maps
  - Detect the spatial distributions of multiple visual patterns
Convolutional Neural Network (CNN)

• Multiple filters: \( net = \sum_{k=1}^{K} X^k \ast W^k \)
Convolutional Neural Network (CNN)
Convolutional Neural Network (CNN)

• Local contrast normalization
  • Normalization can be done within a neighborhood along both spatial and feature dimensions:

\[
\begin{align*}
  h_{i+1,x,y,k} &= \frac{h_{i,x,y,k} - m_{i,N(x,y,k)}}{\sigma_{i,N(x,y,k)}} \\
\end{align*}
\]
Convolutional Neural Network (CNN)

- **Pooling**
  - Max-pooling outputs the maximum value for each sub-region
  - The number of output maps is the same as input, but the resolution is reduced
  - Reduce the computational complexity for upper layers
  - Average pooling can also be applied
Convolutional Neural Network (CNN)

• Typical architecture of CNN
  • Convolutional layer increases the number of feature maps
  • Pooling layer decreases spatial resolution
  • LCN and pooling are optional at each stage

After a few stages, residual spatial resolution is very small. We have learned a descriptor for the whole image. Ranzato CVPR’13
Convolutional Neural Network (CNN)

• Backpropagation on Convolution Neural Network
  • Calculate sensitivity (back propagate errors) \( \delta = - \frac{\partial J}{\partial \text{net}} \) and update weights in the convolutional layer and pooling layer
  • Calculating sensitivity in the convolutional layer is the same as multilayer neural network
Convolutional Neural Network (CNN)

- Calculate sensitivities in the pooling layer
  - The input of a pooling layer $l$ is the output feature map $y^l$ of the previous convolutional layer. The output $x^{l+1}$ of the pooling layer is the input of the next convolutional layer $l + 1$
  - For max pooling, the sensitivity is propagated according to the corresponding indices built during max operation
  - If pooling regions are overlapped and one node in the input layer corresponds to multiple nodes in the output layer, the sensitivities are added
  - Average pooling
CNN Implementation via Tensorflow

- Model Architecture
CNN Implementation via Tensorflow

- Conv-Pooling-LRN structure implementation

```python
# conv1
with tf.variable_scope('conv1') as scope:
    kernel = _variable_with_weight_decay('weights',
        shape=[5, 5, 3, 64],
        stddev=5e-2,
        wd=0.0)
    conv = tf.nn.conv2d(images, kernel, [1, 1, 1, 1], padding='SAME')
    biases = _variable_on_cpu('biases', [64], tf.constant_initializer(0.0))
    pre_activation = tf.nn.bias_add(conv, biases)
    conv1 = tf.nn.relu(pre_activation, name=scope.name)
    _activation_summary(conv1)

# conv2
with tf.variable_scope('conv2') as scope:
    kernel = _variable_with_weight_decay('weights',
        shape=[5, 5, 64, 64],
        stddev=5e-2,
        wd=0.0)
    conv = tf.nn.conv2d(norm1, kernel, [1, 1, 1, 1], padding='SAME')
    biases = _variable_on_cpu('biases', [64], tf.constant_initializer(0.1))
    pre_activation = tf.nn.bias_add(conv, biases)
    conv2 = tf.nn.relu(pre_activation, name=scope.name)
    _activation_summary(conv2)

# pool1
pool1 = tf.nn.max_pool(conv1, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1],
    padding='SAME', name='pool1')

# norm1
norm1 = tf.nn.lrn(pool1, 4, bias=1.0, alpha=0.001 / 9.0, beta=0.75,
    name='norm1')

# pool2
pool2 = tf.nn.max_pool(norm1, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME', name='pool2')
```
CNN Implementation via Tensorflow

- Fully-connected layer with rectified linear activation
- Linear transformation to produce logits

```python
# local3
with tf.variable_scope('local3') as scope:
  # Move everything into depth so we perform a single matrix multiply.
  reshape = tf.reshape(pool2, [FLAGS.batch_size, -1])
  dim = reshape.get_shape()[1].value
  weights = _variable_with_weight_decay('weights', [dim, 384],
                                        stddev=0.04, wd=0.004)
  biases = _variable_on_cpu('biases', [384], tf.constant_initializer(0.1))
  local3 = tf.nn.relu(tf.matmul(reshape, weights) + biases, name=scope.name)
  _activation_summary(local3)

# local4
with tf.variable_scope('local4') as scope:
  weights = _variable_with_weight_decay('weights', [384, 192],
                                        stddev=0.04, wd=0.004)
  biases = _variable_on_cpu('biases', [192], tf.constant_initializer(0.1))
  local4 = tf.nn.relu(tf.matmul(local3, weights) + biases, name=scope.name)
  _activation_summary(local4)

return softmax_linear
```

# tf.nn.sparse_softmax_cross_entropy_with_logits accepts the unscaled logits
# and performs the softmax internally for efficiency.
with tf.variable_scope('softmax_linear') as scope:
  weights = _variable_with_weight_decay('weights', [192, NUM_CLASSES],
                                        stddev=1/192.0, wd=0.0)
  biases = _variable_on_cpu('biases', [NUM_CLASSES],
                            tf.constant_initializer(0.0))
  softmax_linear = tf.add(tf.matmul(local4, weights), biases, name=scope.name)
  _activation_summary(softmax_linear)

return softmax_linear
CNN Implementation via Tensorflow

• Objective function:
  • cross entropy loss
  • all weight decay terms
CNN Implementation via Tensorflow

- Optimization of trainable variables:

```python
# Variables that affect learning rate.
num_batches_per_epoch = NUM_EXAMPLES_PER_EPOCH_FOR_TRAIN / FLAGS.batch_size
decay_steps = int(num_batches_per_epoch * NUM_EPOCHS_PER_DECAY)

# Decay the learning rate exponentially based on the number of steps.
lr = tf.train.exponential_decay(INITIAL_LEARNING_RATE,
global_step,
decay_steps,
LEARNING_RATE_DECAY_FACTOR,
staircase=True)

tf.summary.scalar('learning_rate', lr)

# Generate moving averages of all losses and associated summaries.
loss_averages_op = _add_loss_summaries(total_loss)

# Compute gradients.
with tf.control_dependencies([loss_averages_op]):
opt = tf.train.GradientDescentOptimizer(lr)
grads = opt.compute_gradients(total_loss)

# Apply gradients.
apply_gradient_op = opt.apply_gradients(grads, global_step=global_step)

# Add histograms for trainable variables.
for var in tf.trainable_variables():
tf.summary.histogram(var.op.name, var)

# Add histograms for gradients.
for grad, var in grads:
    if grad is not None:
        tf.summary.histogram(var.op.name + '/gradients', grad)

# Track the moving averages of all trainable variables.
variable_averages = tf.train.ExponentialMovingAverage(MOVING_AVERAGE_DECAY, global_step)
variables_averages_op = variable_averages.apply(tf.trainable_variables())

with tf.control_dependencies([apply_gradient_op, variables_averages_op]):
    train_op = tf.no_op(name='train')

return train_op
```
CNN Implementation via Tensorflow

- Train the deep model via CPU implementation
- Code GitHub resource: https://github.com/tensorflow/models/tree/master/tutorials/image/cifar10

```python
def train():
    """Train CIFAR-10 for a number of steps."""
    with tf.Graph().as_default():
        global_step = tf.contrib.framework.get_or_create_global_step()
    images, labels = cifar10.distorted_inputs()
    # Build a Graph that computes the logits predictions from the
    # inference model.
    logits = cifar10.inference(images)
    # Calculate loss.
    loss = cifar10.loss(logits, labels)
    # Build a Graph that trains the model with one batch of examples and
    # updates the model parameters.
    train_op = cifar10.train(loss, global_step)
```
Auto-encoder

- So far, we have described the application of neural networks to supervised learning, in which we have labeled training examples.
- Now suppose we have only a set of unlabeled training examples.

- An autoencoder neural network is an unsupervised learning algorithm that applies backpropagation, setting the target values to be equal to the inputs:

\[ y^{(i)} = x^{(i)}. \]
Auto-encoder Implementation via Matlab

- Classify MNIST Dataset
  - 9 digits (0~9)
  - Input size: $28 \times 28 = 784$
  - Encoder size: 100
  - Decoder size: 784
  - Output size: 784
Auto-encoder Implementation via Matlab

• You can see that the features learned by the autoencoder represent curls and stroke patterns from the digit images.

• These features are, not surprisingly, useful for such tasks as object recognition and other vision tasks.
Image Classification Application

• Applications of Deep models in ImageNet Challenge
  • Introduction of ImageNet
  • Introduction of AlexNet model (Krizhevsky 2012)
  • Introduction of other different CNN structures (optional)
Image Classification Application

• What is ImageNet?
  • **ImageNet** is an image database organized according to the WordNet hierarchy (currently only the nouns), in which each node of the hierarchy is depicted by hundreds and thousands of images
Image Classification Application

- CNN for object recognition on ImageNet challenge
  - Krizhevsky, Sutskever, and Hinton, NIPS 2012
  - Trained on ImageNet with two GPU. 2GB RAM on each GPU. 5GB of system memory
  - The first time deep model is shown to be effective on large scale computer vision task.
  - Training lasts for one week
Image Classification Application

- Model architecture - AlexNet Krizhevsky 2012
Image Classification Application

- Model architecture - AlexNet Krizhevsky 2012
  - 5 convolutional layers and 2 fully connected layers for learning features.
  - Max-pooling layers follow first, second, and fifth convolutional layers.
  - The number of neurons in each layer is given by 253440, 186624, 64896, 64896, 43264, 4096, 4096, 1000.
  - 650000 neurons, 60000000 parameters, and 630000000 connections.
Image Classification Application

• Choice of activation function

![Activation Functions](image-url)
Image Classification Application

• Reducing Overfitting
  • What is overfitting?

• Useful Methods
  • Data augmentation
  • Dropout
Image Classification Application

• Data augmentation
  • The neural net has 60M real-valued parameters and 650,000 neurons
  • It overfits a lot. 224 × 224 image regions are randomly extracted from 256 images, and also their horizontal reflections
Image Classification Application

• Dropout
  • Independently set each hidden unit activity to zero with 0.5 probability
  • Do this in the two globally-connected hidden layers
Image Classification Application

• Stochastic Gradient Descent Learning
  • Momentum Update

\[ v_{i+1} = 0.9v_i - 0.0005\epsilon w_i - \epsilon \left( \frac{\partial L}{\partial w} \right|_{w_i} \right)_{D_i} \]

\[ w_{i+1} = w_i + v_{i+1} \]

Where 0.9 is momentum (damping parameter), 0.0005\epsilon w_i is weight decay, \( \epsilon \) is learning rate (initialized with 0.01), and \( \epsilon \left( \frac{\partial L}{\partial w} \right|_{w_i} \right)_{D_i} \) is gradient of loss w.r.t weight averaged over batches (batch size: 128)
Image Classification Application

• Results: ILSVRC-2010
  • Achieves top-1 and top-5 test set error rates of 37.5% and 17.0%
  • The best performance achieved during the ILSVRC-2010 competition was 47.1% and 28.2%
  • Shows the outperformance of deep learning to traditional methods

<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>

Table 1: Comparison of results on ILSVRC-2010 test set. In italics are best results achieved by others.
Image Classification Application

• 96 learned low-level filters
Image Classification Application

• Classification result
  • The correct label is written under each image, and the probability assigned to the correct label is also shown with a red bar
Image Classification Application

- Top hidden layer can be used as feature for retrieval
Image Classification Application

• Other different CNN structures for image classification
  • Clarifai
  • Overfeat
  • VGG
  • DeepImage of Baidu
  • Network-in-network
  • GoogLeNet
  • ...
References


Thank you!