Deep Learning: Image Recognition

Using Python, Tensorflow, and Tensorboard

Anthony Hershberger

February 22, 2018

Outline

ImageNet Large Scale Visual Recognition Challenge 2018

- Started in 2010, ImageNet is a competition where research groups, commercial enterprises, and individuals compete on several image challenges.
- This is the first year that the competition will be hosted by Kaggle.
- The winner of the object localization challenge will be the team which achieves the minimum error across all test images.
- The winner of the object detection challenge will be the team which achieves the highest accuracy on the most amount of object categories

Tasks of Image Recognition

- When you read papers, there are many references to object localization, object detection, image segmentation.
- **Object localization** is the task of detecting a *single instance* of an object and localizing it with a bounding box.
- **Object detection** is the task of detecting *all instances* given a known class label such as cars, boats, or people.
- **Image Segmentation** is a process of assigning a class label to every pixel such that every pixel is within the boundaries of the object.

ImageNet Dataset

- ImageNet is an image database of over 1.2 million images that are human-annotated with synsets, or synonym sets, that classify the images within a word hierarchy.
- The training data for the object localization challenge is a subset of 150,000 images of ImageNet.
- The training set for the object detection challenge is a subset of 450k images and 475k objects for classification. The test set includes 40k images.
- ImageNet Website

Object Localization

- For each photo input, the algorithm will output 5 class labels denoted

$$c_i, i = 1, 2, ...5$$

in decreasing order of confidence.

- The algorithm will also output 5 bounding boxes denoted

$$b_i, i = 1, 2, ..., 5$$

for each class label.

- The ground truth labels are for the image

$$C_k, k = 1, 2, ..., n$$

class labels.

Object Localization

- For each ground truth class label C_k, the ground truth bounding boxes are

$$B_{km}, m = 1, 2, ..., M_k$$

where M_k is the number of instances of the k^{th} object in the current image.

- Let $d(c_i, C_k) = 0$ if $c_i = C_k$ and 1 otherwise.

Object Localization

- The function that we are trying to minimize is

$$e = \frac{1}{n} \sum (min_i min_m max(d(c_i, C_k), f(b_i, B_{km})$$

where d is 0 if the your algorithm predicts the same class as the bounded class label and $f(b_i, B_{km}) = 0$ if your algorithm's bounding box overlaps the ground truth bounding box by more than 50

Object Detection Challenge

- For each input image, your algorithm must produce a set of annotations

$$(c_i, s_i, b_i)$$

  where c_i are the class labels, s_i are the confidence scores, and b_i are the bounding boxes.
- There are 200 categories labeled for the test set that are fully annotated.
- Any objects that are not annotated or create duplicate bounding boxes will be penalized.

Current Research in Image Recognition

1. ImageNet Classification with Deep Convolutional Neural Networks* (2012)
2. ZFNet*(2013)
3. Going Deeper with Convolutions*(2014)
4. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*(2015)
5. Squeeze and Excitation Networks* (2017)
6. Mask R-CNN (2018)

*ImageNet winner

Current Research in Image Recognition

- In 2012, AlexNet significantly outperformed all submissions with a 16 percent error by introducing a deep CNN that featured **stacked convolutional layers**.
- In 2013, ZFNet improved on AlexNet's work by adjusting the stride and filter sizes to assist in the tuning of hyperparameters.
- In 2014, GoogLeNet developed **inception modules** and implemented the use of **average pooling layers** to reduce parameter sizes while reducing error.
- In 2015, Resnet introduced the concept of **skip connections** and **batch normalization**

Convolutional Neural Networks

- **Convolutional Neural Networks** are a subset of traditional feed-forward neural networks and have seen much success in the area of image recognition for their ability to build deep accurate networks, reduce parameters for efficient training, and quick recall for deployment.
- A basic CNN architecture involves one or more convolutional layers attached by non-linear activation functions to pooling layers and output through a fully connected layers to obtain the class scores and make predictions.

Convolutional Neural Networks

- In image recognition, a three-layer CNN will learn the edges from raw pixels on the first ConvLayer, learn simple shapes on the second ConvLayer, and learn higher level features on the 3rd layer. The 3rd layer will then be connected to the class probabilities through a fully connected layer and
- The sliding window approach of CNN's leads to **location invariance** which means the network is not affected by scaling, rotation and translation.

Layers of Convolutional Neural Networks

- **Convolutional Layers** are the fundamental layers of CNN's and involve any layer where convolutional operations occur. They are the most computational expensive layers but where most information is learned in the model. These layers are connected through other layers by non-linear activation functions.
- **Pooling Layers** are typically applied after convolution layers to reduce output dimensionality while minimizing the information loss of the ConvLayer. *Max Pooling* using a 2x2 filter and stride of 2 has been a popular operation in recent research.
- **Fully connected Layers** attach all neurons to all the activations in the previous layer

Spatial Arrangements

- Spatial arrangement deals with neuron arrangement and the volume of the output on the next layer.
- The output volume on the layer following the convolutional layer is determined by three hyper parameters: **depth**,**stride** and **zero padding**.

Depth, Stride, and Zero Padding

- **Depth** describes the output volume and directly corresponds to the number of filters. The filters *convolve* around the input by shifting a fixed number of times. The fixed unit that the filter convolves is called the stride.
- **Stride** describes the unit of time that we shift or slide the filter when convolving an input layer.
- **Zero padding** is used to pad an input with zeros. Under some circumstances, zero padding is used for convenience. The most important feature of zero padding is to preserve the spatial sizes of outputs.

Activation Functions

- In a CNN, the activation function of a node decides whether that neuron should fire or activate. Mathematically speaking, *activation functions* look at the coefficients of the convolution matrix, impose a sigmoid function, and attempt to map those values to 0 (don't fire) or 1 (fire).
- Activation functions impose non-linearity on the model. Without an activation function the weight and bias terms would transform linearly.
- CNN's are tasked with complex feature extraction that would perform poorly with a linear activation function

Backpropogation

- First we will look at some standard notation regarding the concept of backpropogation.

X: Tensor pairs of inputs of x_i and class labels y_i
x_i^ℓ: Input x of node i at layer ℓ
y_i^ℓ: Output y of node i at layer ℓ
w_{ij}^k: The weight of node j in layer ℓ_k coming from node i
b_i^k: The bias of node i in layer k
θ: the grouping of parameters, w and b
$y_i^{'}$: the actual predicted class label
α: the learning rate of our update function

Backpropogation

- When training a convolutional neural network using the gradient descent algorithm, the network requires gradient computations of the chosen error function $w.r.t$ the weights w_{ij}^k and biases b_i^k commonly grouped together as θ.
- Gradient descent will iteratively update the weight and bias at each node and its speed is set by the hyperparamter α, which is the learning rate. The function to iteratively update the weights and bias is:

$$\theta^{t+1} = theta^t - \alpha \frac{dE(X, \theta^t}{d\theta}$$

backpropogation

- Commonly used error functions used in CNN's are the mean squared error defined as

$$E(X, \theta) = \frac{1}{2N} \sum_{i=1}^{N} (y_i^{'} - y_i)^2$$

as well as the cross entropy loss

$$-\sum y_i^{'} \cdot \log(y_i)$$

- The objective of backpropogation is to minimize the the error function using an optimization method such as gradient descent.

Computational Complexity of CNNs

- CNN's are computationally expensive due to the large *WxHxD* tensor volumes.
- On a given layer, a $28x28x1$ grayscale image would have 784 weights and a $28x28x3$ RGB image would have 2352 weights.
- For most computer systems, the largest bottleneck for a CNN is the GPU memory. (at most 24 gb)
- The usage of memory depends on the number of activations of intermediate tensors(backpropogation), the number of total network parameters, and overhead.

Tools for Neural Networks

TensorFlow

- Tensorflow is compiled on the use of static computational graphs
- Encapsulation: Define the network architecture within the TF session and then execute.
- TF uses static computational graphs because under the hood they can pre-compile functions, pre-allocate buffers in memory allowing for faster optimization.

PyTorch

- PyTorch implements dynamic computational graphs.
- This allows for dynamic data structures including mixed lists, stacks, you name it.
- Since the neural networks are modular and each part of the model is considered a separate piece, you can be very creative when building your network.

Tools for Neural Networks:TensorFlow

- TensorFlow was developed internally by the Google Brain Team to replace a distributed system called DistBelief.
- TF allows the user to schedule tasks between CPU and GPUs, or between Multiple GPUs.
- TF provides computationally efficient algorithms for optimization, node operations, and tools for debugging.
- TF uses static computational graphs to structure the numerical computations.
- Installing TensorFlow on Windows

TensorFlow installed for CUDA

- TensorFlow programs typically run significantly faster on a GPU than on a CPU.
- An Nvidia 1080 has 2560 CUDA cores, 8GB GDDRX memory, and 256 bit memory interface(320GB/s)

CNN Code Explanation

1. Import Tensorflow into your environment

```
import tensorflow as tf
```

2. Access the MNIST data through the TF module

```
from tensorflow.examples.tutorials.mnist import input_data
MNIST = input_data.read_data_sets("/tmp/data/", one_hot=True)
```

CNN Code Explanation

3. Define your placeholders. Placeholders are nodes that will be fed values or labels at the time of execution.

```
x = tf.placeholder(tf.float32, [None, 784], name='X')
y = tf.placeholder(tf.float32, [None, 10], name='Labels')
```

4. Define your variables. Variables are stateful nodes that will output their current value.

```
W = tf.Variable(tf.zeros([784, 10]), name='Weights')
b = tf.Variable(tf.zeros([10]), name='Bias')
```

5. Initialize your variables.

```
init = tf.global_variables_initializer()
```

CNN Code Explanation

6. Scope your operations

```
with tf.name_scope('Model'):
    # Model
    pred = tf.nn.softmax(tf.matmul(x, W) + b) # Softmax
with tf.name_scope('Loss'):
    cross_entropy = tf.reduce_mean(-tf.reduce_sum(y*tf.log(pred), r
with tf.name_scope('SGD'):
    optimizer = tf.train.GradientDescentOptimizer(learning_rate).mi
with tf.name_scope('Accuracy'):
    acc = tf.equal(tf.argmax(pred, 1), tf.argmax(y, 1))
    acc = tf.reduce_mean(tf.cast(acc, tf.float32))
```

Note: By scoping your operations, TensorFlow can define the structure of your computational graph for data visualization. Scoping also assists in scheduling tasks to different components.

CNN Code Explanation

7. Define the metrics that you want to output to monitor the perfomance of your network

```
tf.summary.scalar("loss", cross_entropy)
tf.summary.scalar("accuracy", acc)
merged_summary_op = tf.summary.merge_all()
```

Note: TensorFlow requires that you place summaries into the log files in order to be read by Tensorboard.

CNN Code Explanation

7. tf.Session() executes the model.

```
with tf.Session() as sess:
    sess.run(init)
    ...
for i in range(combined_batch):
  batch_xs, batch_ys =   MNIST.train.next_batch(batch_size)
  _, c, summary = sess.run([optimizer, cross_entropy, merged_summar
  feed_dict={x: batch_xs, y: batch_ys})
```

Note: tf.Session() deploys the defined computational graph onto the device maps. We must call Sess.run(fetches, feeds) to execute the nodes in our graph. **Fetches** are lists of graph nodes that return outputs. **Feeds** represent mappings of graph nodes to concrete values. *Keys* are graph nodes. The *items* are numpy data.

Backpropogation

- We calculate our gradients by creating an optimizer object.
- In this model we have chosen to use gradient descent optimizer and create a node operation that minimizes the function using cross entropy loss.
- TensorFlow has gradient operations attached to every node. When a node is called and executed, TF calculates the gradients with respect to the loss parameters using backpropagation.
- TF calculates gradients with respect to the variables which is why it is important to define variables and placeholders separately.

TensorBoard

- TensorBoard is a suite of visualization tools to help decompose, optimize and debug even the most complex neural networks.
- TensorBoard operates by reading TensorFlow event files.

How do you set up TensorBoard?

1. Create a TensorFlow graph that you would like to collect summary data from and decide which nodes you would like to add to your list of summary operations.
2. Store the summary data into a temporary log file within the Tensorboard environment.
3. Open up your computer's web browser and enter in your log address that is attached to the TB port.

MNIST Data Demo

- Let's use TensorFlow and Python to train a Convolution Neural Network and output the results to TensorBoard.
- Special Thanks to Martin Gorner's video on YouTube for TensorFlow Tutorial.
- Tensorflow and deep learning - without a PhD by Martin Gorner