Programming with TensorFlow

Autoencoders and CNNs

Soham Pal

Department of Computer Science and Automation
Indian Institute of Science

E0 302: ML for SE

Outline

Outline

- Recall that the (training) Dataset is denoted by:

$$\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{n}$$

where $x_i \in \mathbb{R}^{d_1}$ and $y_i \in \mathbb{R}^{d_2}$.

- Look for $W_1 \in \mathbb{R}^{d_1 \times m}$, $W_2 \in \mathbb{R}^{m \times d_2}$, $b_1 \in \mathbb{R}^m$, $b_2 \in \mathbb{R}^{d_2}$ such that:

$$h_i = \sigma(W_1^T x_i + b_1)$$

$$\hat{y}_i = \sigma(W_2^T h_i + b_2)$$

where $\forall i, \hat{y}_i \approx y_i$ and generalizes well to unseen x.

Training

- Cross-entropy loss:

$$\mathcal{L}(\hat{y}_i, y_i) = -\sum_i y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)$$

  where $\hat{y}_i$ is the predicted label and y_i is the expected label.
- Optimization routines
  - `GradientDescentOptimizer`
  - `RMSPropOptimizer`
  - `MomentumOptimizer`
  - `AdamOptimizer`
    Default parameters "good enough", do not need hyperparameter optimization.

Implementation

Inputs

```
input = tf.placeholder(tf.float32, [None, in_size])
label = tf.placeholder(tf.float32, [None, out_size])
```

Weights

```
w = tf.Variable(tf.random_normal(shape, stddev=0.01))
b = tf.Variable(tf.zeros(shape))
```

Putting it together

```
hidden = tf.sigmoid(tf.matmul(input, W1) + b1)
output = tf.sigmoid(tf.matmul(hidden, W2) + b2)
```

Outline

The curse of dimensionality

- Consider handwritten digits of size 28×28 pixels.
  Dimension of vector space $\dim(\mathcal{X}) = 784$!
- Not all information is useful: most of it is just a black background.
- Automatically learn what is important – unsupervised learning.

Autoencoders

- Dataset:

$$\mathcal{D} = \{(x_i, x_i)\}_{i=1}^{n}$$

  where $x_i \in \mathbb{R}^d$.

- Look for $W_1 \in \mathbb{R}^{d \times m}, W_2 \in \mathbb{R}^{m \times d}, b_1 \in \mathbb{R}^m, b_2 \in \mathbb{R}^d$ such that:

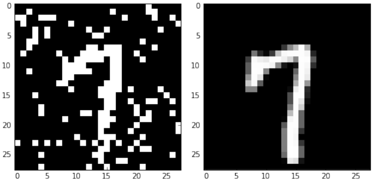
$$h_i = \sigma(W_1^T x_i + b_1)$$

$$\hat{x}_i = \sigma(W_2^T h_i + b_2)$$

  where $\forall i, x_i \approx \hat{x}_i$ and generalizes well to unseen x.

Autoencoders

- What's the catch?
  The hidden layer is a vector in $\mathbb{R}^m$, where $m \neq_n d$.
- Can "compress" information by enforcing $m \ll d$.
- Can impose sparsity constraints on the hidden units h_i for further compression.



- Train pairs of noisy and original samples.
  Input $(z_i) = \text{AddNoise}(x_i; \mu, \sigma)$
  Output $= \hat{x}_i$
- Calculate loss with respect to the original input:

$$\mathcal{L}(\hat{x}_i, x_i) = -\sum_i x_i \log(\hat{x}_i) + (1 - x_i) \log(1 - \hat{x}_i)$$

and **not** the input to the autoencoder $\mathcal{L}(\hat{x}_i, z_i)$.

Xavier Initialization

$$W_{ij} \sim U\left[-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}\right]$$

where:

- $U[-a, a]$ is the uniform distribution in the interval $(-a, a)$.
- n is the size of the previous layer (the number of columns in W)

Tied Autoencoder

- Set $W_2 = W_1^T$
- Look for $W_1 \in \mathbb{R}^{d \times m}, b_1 \in \mathbb{R}^m, b_2 \in \mathbb{R}^d$ such that:

$$h_i = \sigma(W_1^T x_i + b_1)$$

$$\hat{x}_i = \sigma((W_1^T)^T h_i + b_2)$$

where $\forall i, x_i \approx \hat{x}_i$ and generalizes well to unseen x.

Tied Autoencoder

- Look for $W \in \mathbb{R}^{d \times m}, b_1 \in \mathbb{R}^m, b_2 \in \mathbb{R}^d$ such that:

$$h_i = \sigma(W^T x_i + b_1)$$

$$\hat{x}_i = \sigma(W h_i + b_2)$$

where $\forall i, x_i \approx \hat{x}_i$ and generalizes well to unseen x.

Outline

Discriminative vs Generative Models

- **Discriminative:** Models the conditional probability distribution

$$P(Y|X)$$

  Example: label a handwritten digit.

- **Generative:** Models the joint probability distribution

$$P(X, Y)$$

  Allows us to sample from the learned distribution.

Generative Models

Let $\mathcal{X} = \{x_i\}_{i=1}^{n}$ be the MNIST dataset of 28×28 pixel images.

A generative model would give us $P(X)$, i.e. the probability distribution over the space of 28×28 pixel images from which the training examples were drawn, this is called the "underlying distribution".

Autoencoders as a Generative Model

Let $\tilde{X}$ denote a corrupted image and $\hat{X}$ denote a reconstruction.

- $R(\hat{X}|\tilde{X})$ is a reconstructing distribution.
- $Q(\tilde{X}|X)$ is a corrupting distribution.

Corruption-Reconstruction Markov Chain

1. Start with a draw from the distribution.
2. Corrupt it using Q.
3. Reconstruct it using R.

Theorem (Y. Bengio et al.)

The stationary distribution of this Markov Chain converges to P(X).

Let $\tilde{X}$ denote a corrupted image and $\hat{X}$ denote a reconstruction.

- $R(\hat{X}|\tilde{X})$ is a denoising autoencoder trained on $\mathcal{X}$ (i.e. reconstructing distribution).
- $Q(\tilde{X}|X)$ is a corrupting distribution.

Algorithm 1 sample from $P(X)$

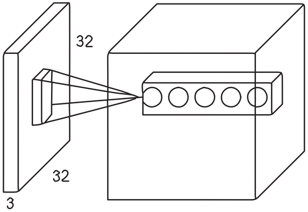
1: *previous* $\leftarrow x_i \sim \mathcal{X}$
2: **while** The Markov Chain has not Converged **do**
3:     *corrupted* $\sim Q(\tilde{X}|\hat{X} = \text{previous})$
4:     *previous* $\sim R(\hat{X}|\tilde{X} = \text{corrupted})$
5: **end while**
6: **return** *previous*

Outline

Collection of Homogenous Filters

- A *collection of filters* is a weight matrix of dimensions $h \times v \times n_{in} \times n_{out}$ where $h \times v$ is the size of the locally receptive field, $n_i n$ is the number of "channels" in the input and n_{out} is the number of such filters.
- From now on, we shall call this a *filter*.

Such a filter transforms an input volume into an output volume, for example:



Outline

2D Convolution in TensorFlow

`tf.nn.conv2d(input, filter, strides, padding)`

- `input` is the input image tensor of dimensions
  `batch_size` × `height` × `width` × `channels`
- `filter` is a filter collection of dimensions
  `filter_height` × `filter_width` × `channels` × `# filters`
  It is initialized randomly with small weights $W \sim \mathcal{N}(0, 0.01)$
- `strides` specifies the stride in each dimension.
  Typically want this to be $[1, \texttt{v\_stride}, \texttt{h\_stride}, 1]$
- `padding` is one of:
  - "SAME": pad with 0 to ensure all input pixels covered by convolution.
  - "VALID": no padding, some input pixels may not be covered.

  Typically, "SAME" is used.

Max Pooling in TensorFlow

`tf.nn.max_pool(value, ksize, strides, padding)`

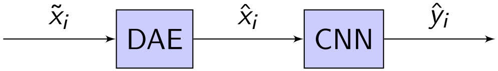
- `value` is the output volume of the filter.
- `filter` specifies the sliding window size in each dimension.
  Typically want this to be $[1, \texttt{v\_size}, \texttt{h\_size}, 1]$
- `strides` specifies the stride in each dimension.
  Typically want this to be equal to `filter`.
- padding is one of:
  - "SAME": pad with $-\infty$ to ensure all input pixels covered by pooling.
  - "VALID": no padding, some input pixels may not be covered.

  Typically, "SAME" is used.

Outline

Classifying Noisy Handwritten Digits

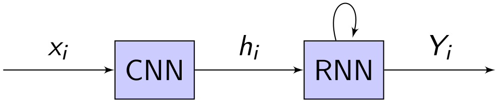
Front end: Denoising Autoencoder (DAE)
Back end: Convolutional Neural Network (CNN)



- The DAE is pretrained on the MNIST dataset to reconstruct noisy samples.
- The CNN then uses the trained DAE to reconstruct the input before convolution.

Image Captioning

Front end: Convolutional Neural Network (CNN)
Back end: Recurrent Neural Network (RNN)



- RNNs provide variable length output – the result is thus not a vector but a matrix, Y_i.
- The network is trained end-to-end on image-word pairs.

Outline

Input size: $128 \times 244 \times 244 \times 3$

Table: Benchmarks of public domain implementations [1]

Library	Time (ms)	Forward (ms)	Back (ms)
Nervana-neon-fp16	230	72	157
Nervana-neon-fp32	270	84	186
TensorFlow	445	135	310
CuDNN[R4]-fp16 (Torch)	462	112	349
CuDNN[R4]-fp32 (Torch)	470	130	340
Caffe	1935	786	1148

[1] `https://github.com/soumith/convnet-benchmarks`

Input size: $128 \times 244 \times 244 \times 3$

Table: Benchmarks of public domain implementations [2]

Library	Time (ms)	Forward (ms)	Back (ms)
CuDNN[R4]-fp16 (Torch)	71	25	46
Nervana-neon-fp16	78	25	52
Nervana-neon-fp32	81	27	53
TensorFlow	81	26	55
CuDNN[R4]-fp32 (Torch)	81	27	53
Caffe	324	121	203

[2] https://github.com/soumith/convnet-benchmarks

Outline

Overfitting: How to detect it

- Always validate.
- Do **not** provide the correct labels to the network for validation even if you think the graph doesn't use them. This prevents accidentally training the network on the validation set.
  Better be safe than sorry – use `np.zeros_like(valid_y)` instead.

Outline

Assignments to tensors

Code Sample

```
x = tf.placeholder(...)
z = tf.zeros(...)
w = tf.add(x, z)
print sess.run(w, feed_dict={x:[1], z:[1]})
```

What will this code fragment print?

1. 1
2. 2
3. Garbage
4. Assertion Failure in framework

Be careful with assignments

Code Sample

```
x = tf.placeholder(...)
z = tf.zeros(...)
w = tf.add(x, z)
print sess.run(w, feed_dict={x:[1], z:[1]})
```

- The value of *w* printed is 2!
- Every Tensor is mutable in Tensorflow – **there are no constants**!

Summary

- Extensive community support – is currently the most used framework on Github. Caffe is a close second.
- Major weakness: static computational graph, that boils down to a series of matrix operations – makes beam search, etc. difficult to implement.
  - Symbolic conditionals
  - Symbolic loops

For Further Reading I

Google
TensorFlow API Documentation.
https://www.tensorflow.org/api_docs

Soham Pal
Tensorflow Tutorial (source code for the examples).
https://github.com/peteykun/TensorflowTutorial

Soumith Chintala
convnet-benchmarks
https://github.com/soumith/convnet-benchmarks

For Further Reading II

Yoshua Bengio et al.
Generalized denoising autoencoders as generative models.
NIPS, 2013.

Yoshua Bengio et al.
Deep Generative Stochastic Networks Trainable by Backprop.
ICML, 2014.