Learning Neural Networks in TensorFlow

Dan Salo

Duke University

January 11th, 2017

Overview

- **Learning Neural Networks:** Some basics
- **TensorFlow:** Learning made easy!
- **Autoencoders:** Leveraging unlabeled data
- **Code Demo:** Wind it up, let it run.

Multi-Layer Perceptron

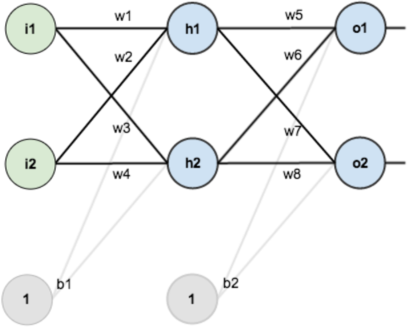
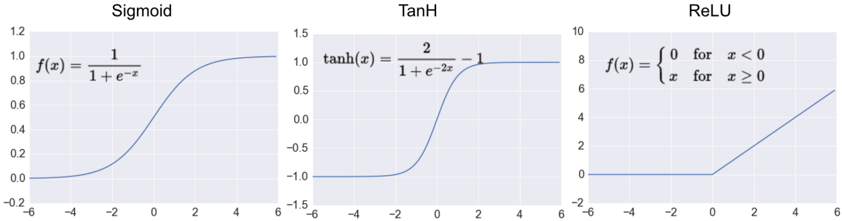


Figure: 2-Layer MLP

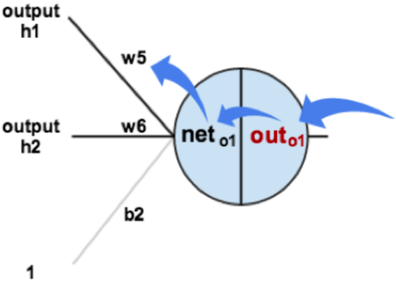
$$
\begin{aligned}
h_1 &= w_1 * i_1 + w_2 * i_2 + b_1 \\
&= \sum_{j=1}^{2} w_j * i_j + b_1 \\
o_1 &= w_6 * h_2 + w_5 * h_1 + b_2 \\
&= \sum_{k=5}^{6} w_k * h_{k-4} + b_2 \\
&= \sum_{k=5}^{6} f(g(i_{k-4}))
\end{aligned}
$$

"Classical" Activation Functions



- ReLU can lead to sparser networks
- Initialization of weights is an active research area

Learning Gradients



$$out = \sigma(net)$$

$$net = w * x + b$$

Chain Rule

$$f \circ g = f(g(x))$$

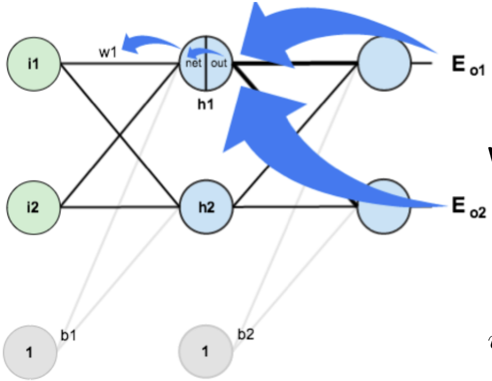
$$\frac{\delta}{\delta x}[f \circ g] = g'(x)f(x)$$

Back Propagation

$$L_{o_1} = ||out_{o_1} - label_{o_1}||^2$$

$$\frac{\delta L_{o_1}}{\delta w_5} = \frac{\delta L_{o_1}}{\delta out_{o_1}} \frac{\delta out_{o_1}}{\delta net_{o_1}} \frac{\delta net_{o_1}}{\delta w_5}$$

Updating Weights



Weight Update

$$w_5^{(i+1)} = w_5^{(i)} - \eta * \left[\frac{\delta L_{o_1}}{\delta w_5}\right]$$

$$w_1^{(i+1)} = w_1^{(i)} - \eta * \left[\frac{\delta L_{o_1}}{\delta w_1} + \frac{\delta L_{o_2}}{\delta w_1}\right]$$

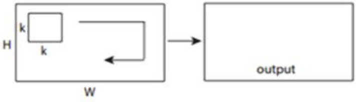
$$out = \sigma(net)$$

$$net = w * x + b$$

Extension to Convolutional Layer

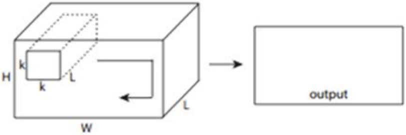
Input: Single 2D Feature Map

$$x_{ij} = \sum_{a=0}^{k-1} \sum_{b=0}^{k-1} w_{ab} * y_{(i+a)(j+b)}$$



Input: Stack of 2D Feature Maps

$$x_{ij} = \sum_{l=0}^{L} \sum_{a=0}^{k-1} \sum_{b=0}^{k-1} w_{abl} * y_{(i+a)(j+b)(l)}$$



Deep Learning Frameworks and Packages

Frameworks

MatConvNet
TensorFlow™
Caffe
torch
theano

Packages

Chainer
K
Lasagne

TensorFlow

TensorFlow™

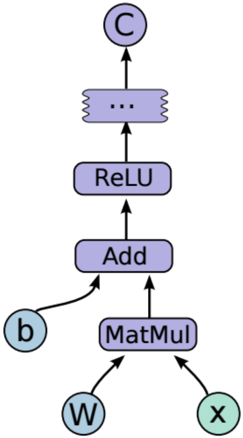
- Graphs, Sessions, Nodes and Ops, Tensors, Variables
- Computational Graph for Symbolic Differentation
- Distributed Learning
- Queueing and Threading
- C++ and Python API

Basic TensorFlow Network

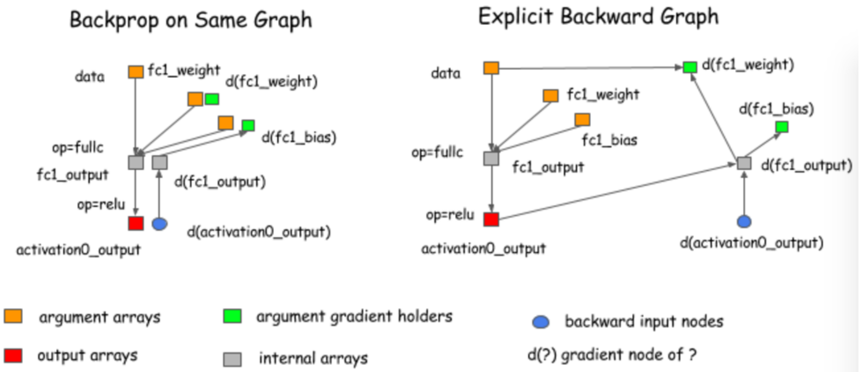
```
import tensorflow as tf

b = tf.Variable(tf.zeros([100]))
W = tf.Variable(tf.random_uniform([784,100],-1,1))
x = tf.placeholder(name="x")
relu = tf.nn.relu(tf.matmul(W, x) + b)
C = [...]


s = tf.Session()
for step in xrange(0, 10):
  input = ...construct 100-D input array ...
  result = s.run(C, feed_dict={x: input})
  print step, result
```



Computational Graphs



Distributed Learning

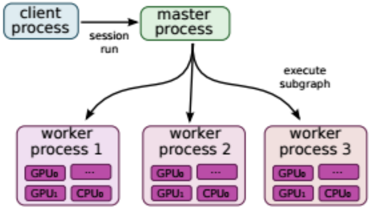


Figure: Multiple Device Learning

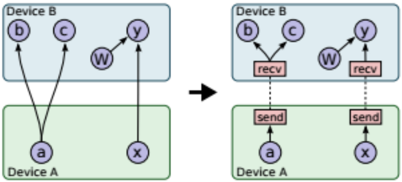


Figure: Message Passing

Comparison to other Frameworks

TF	Theano	Caffe	Torch
Google	U. of Montreal	U.C. Berkeley	Facebook
Python, C++	Python	Python, C++	Lua
Symbolic	Symbolic	Non-Symbolic	Non-Symbolic
Apache 2.0	BSD	BSD	BSD

- Variance in Module Creation, Model Selection
- Each has it's own start-up cost, community

Autoencoder

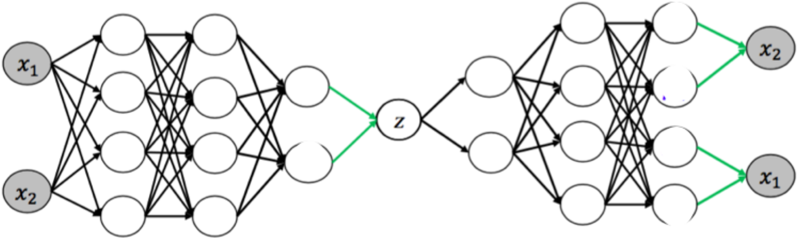


Figure: Inputs, Latent, Reconstruction

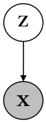
- Encoder and Decoder
- Dimensionality Reduction

Probabilistic Generative Models

Assume: Image data, $\mathbf{x}$, is described by underlying hidden variables, $\mathbf{z}$.

$$\mathbf{z} \sim p(\mathbf{z}; \phi)$$

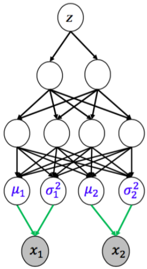
$$\mathbf{x} \sim p(\mathbf{x}|\mathbf{z}; \theta)$$



Consider: $p(\mathbf{x}|\mathbf{z})$ is Normal and described by a neural network with parameters θ.

$$\mathbf{z} \sim \mathcal{N}(0,1), \quad \phi \leftarrow 0,1$$

$$x_i \sim \mathcal{N}(\mu_i, \sigma_i^2), \quad \theta \leftarrow w_j$$

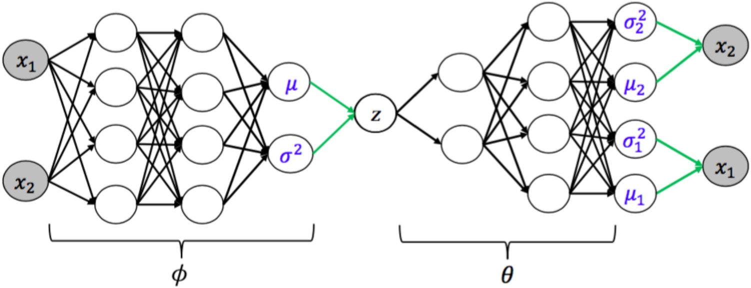


Semi-supervision with Autoencoders

Assume: Hidden variables, **z**, are related to data, **x**. Employ Bayes Rule:

$$p(\mathbf{z}|\mathbf{x}) \propto p(\mathbf{z})p(\mathbf{x}|\mathbf{z})$$

Consider: $p(\mathbf{z}|\mathbf{x};\phi)$ is Normal and described by a neural network with parameters ϕ.



Infer: Label from **z**.

Variational Autoencoder Loss

Unlabeled Data:

$$\mathcal{U}_{\theta,\phi}(\mathbf{x}) = -D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z})) + \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})]$$
$$= \text{KL Loss (Regularizer)} + \text{Recon Loss (L2 Loss)}$$

Labeled Data:

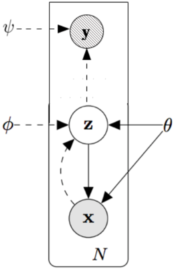
$$\mathcal{L}_{\theta,\phi,\psi}(\mathbf{x},\mathbf{y}) = \mathcal{U}_{\theta,\phi}(\mathbf{x}) + \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log s_\psi(\mathbf{y}|\mathbf{z},\mathbf{x})]$$
$$= \text{KL Loss} + \text{Recon Loss} + \text{Label Loss (Cross-Entropy)}$$

Total Loss:

$$\mathcal{J} = \sum_{\mathbf{x},\mathbf{y}\in\mathcal{D}_L} \mathcal{L}_{\theta,\phi,\psi}(\mathbf{x}) + \sum_{\mathbf{x}\in\mathcal{D}_U} \mathcal{U}_{\theta,\phi}(\mathbf{x})$$

Conv-VAE Models

Network	MNIST	BAGS
$q_\phi(\mathbf{z}\vert\mathbf{x})$	4-Layer CNN	12-Layer CNN
$p_\theta(\mathbf{x}\vert\mathbf{z})$	4-Layer DNN	12-Layer DNN
$s_\psi(\mathbf{y}\vert\mathbf{x},\mathbf{z})$	1-Layer MLP	2-Layer MLP



$$\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x}))$$

$$s_\psi(\mathbf{y}|\mathbf{x},\mathbf{z}) = \text{Softmax}\left(f\left(\mathbf{z}\right)\right)$$

MNIST Classification Results

# of Labels	CNN	Conv-VAE
100	7.32%	6.87%
300	3.64%	3.41%
500	2.49%	2.06%
1000	1.54%	1.50%
3000	1.16%	0.87%

Figure: Error on Test Set

- Conv-VAE trained with balanced minibatches
- CNN uses same network as Conv-VAE

Code

MNIST Digit Reconstruction