Deep Neural Networks for Scalable Prediction

Lynd Bacon
Loma Buena Associates
Notre Dame University
Northwestern University
lbacon@LBA.com

Conference on Statistical Practice
Portland OR

February 2018

www.LBA.com/library

Agenda

- Neural Networks (NN's) basics and inspirations
- Problem domains
- Architectures: various and popular
- Training and testing basics
- A simple application example:
  - Marketing campaign response application
- As time allows:
  - “Try it at Home” using a downloadable Jupyter Notebook

www.LBA.com/library

NN's, ML, and Prediction

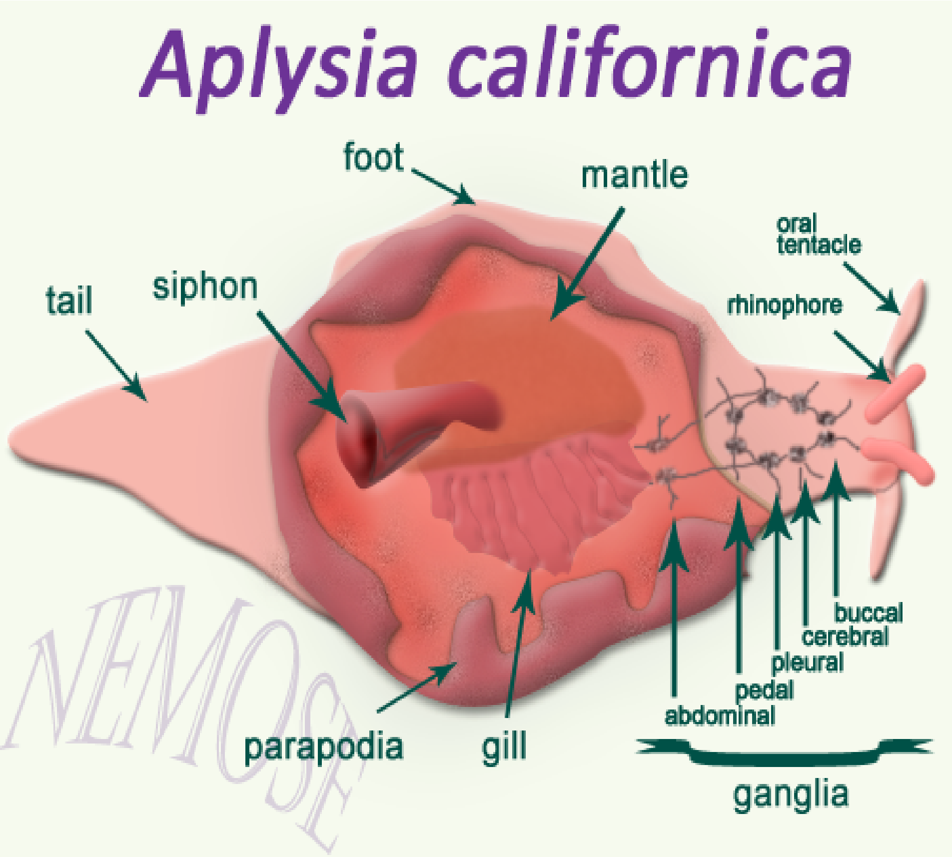
- NN's are a class of machine learning (ML) methods
- Like many other ML methods, a common objective is maximizing predictive accuracy
- ML methods are in general not for theory or hypothesis testing (Not yet, at least)

www.LBA.com/library

NN Progress in Problem Domains

- NLP
- Image recognition
- Autonomous vehicles
- Drug repurposing and discovery
- Security

Alypsia

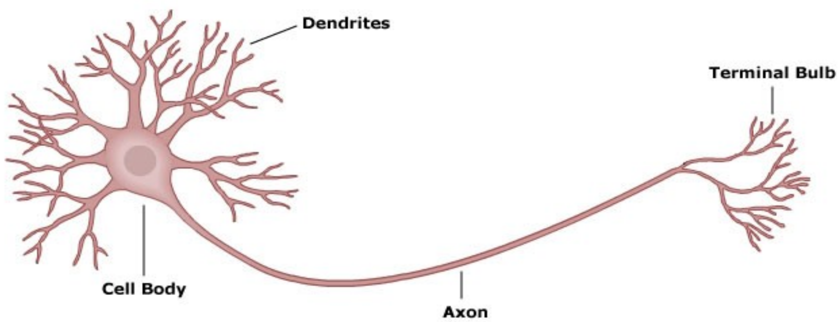


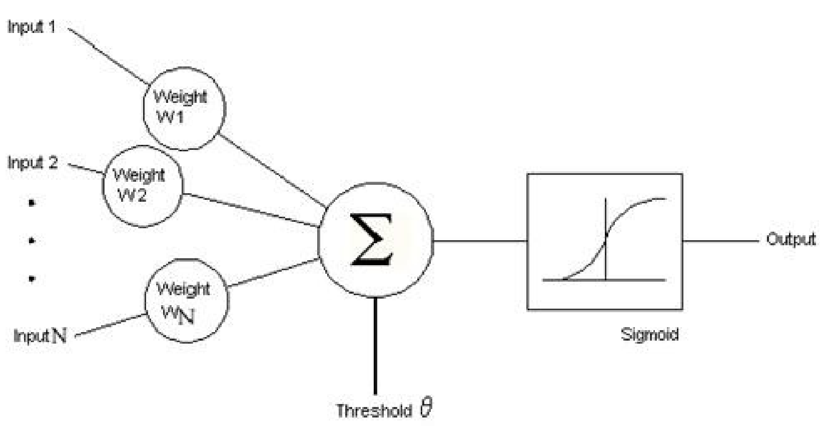
www.LBA.com/library

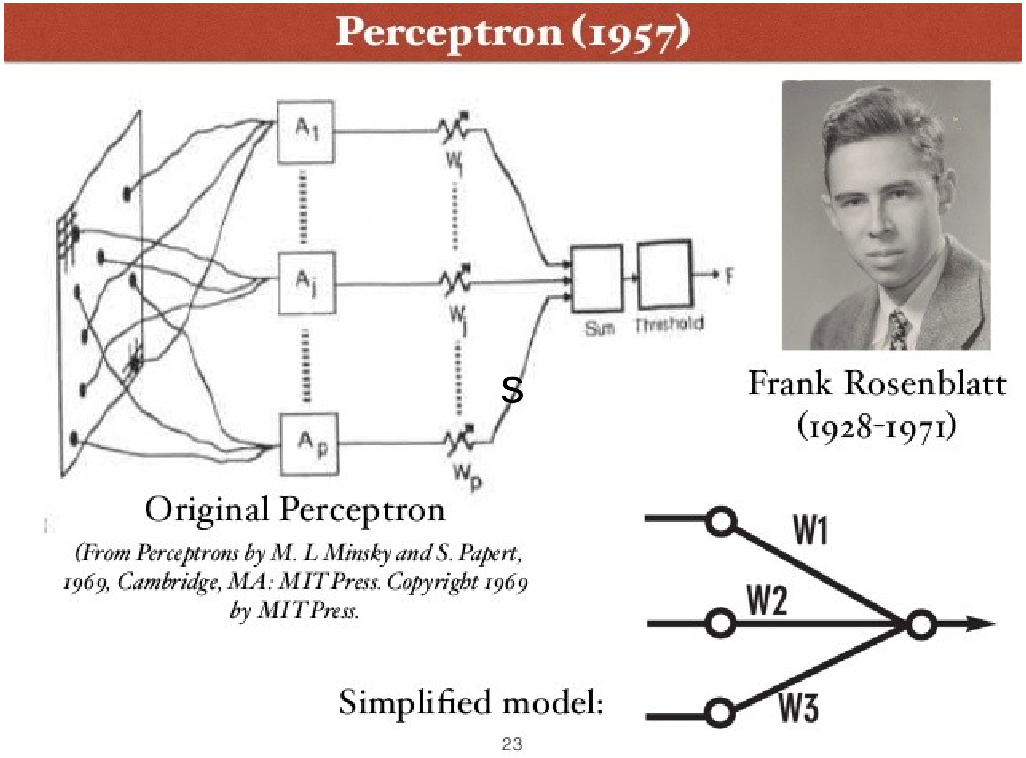
Getting from There to Here: Some Key Developments

- Carew, Castellucci & Kandel (1971): learning at the level of the single neuron
- McCulloch & Pitts (1943): artificial neuron
- Hubel & Wiesel (1959): organization of central nervous system in cats, primates
- Rosenblatt (1957), Minsky & Papert (1969, 1987): “perceptron”
- Rumelhart & McClelland (1986, 1987): parallel distributed processing

Perceptron

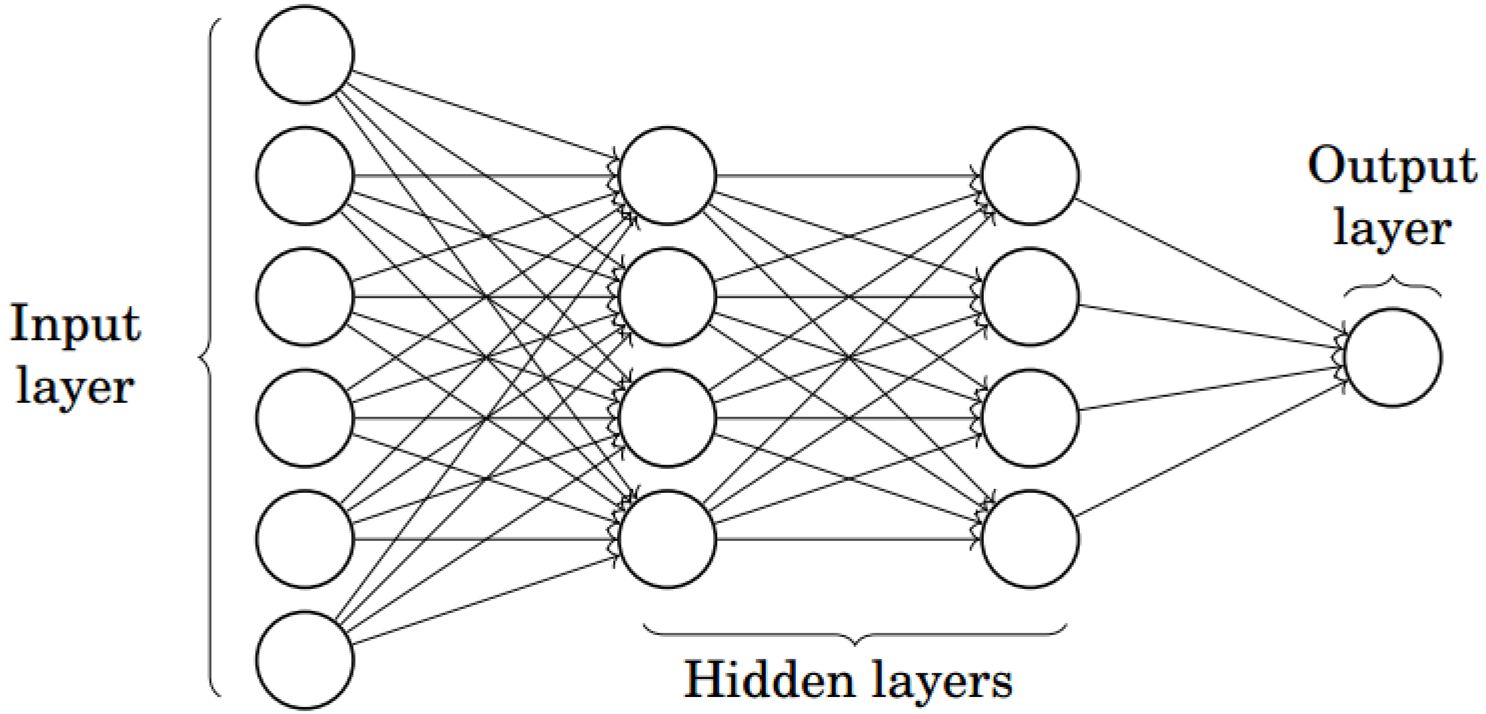






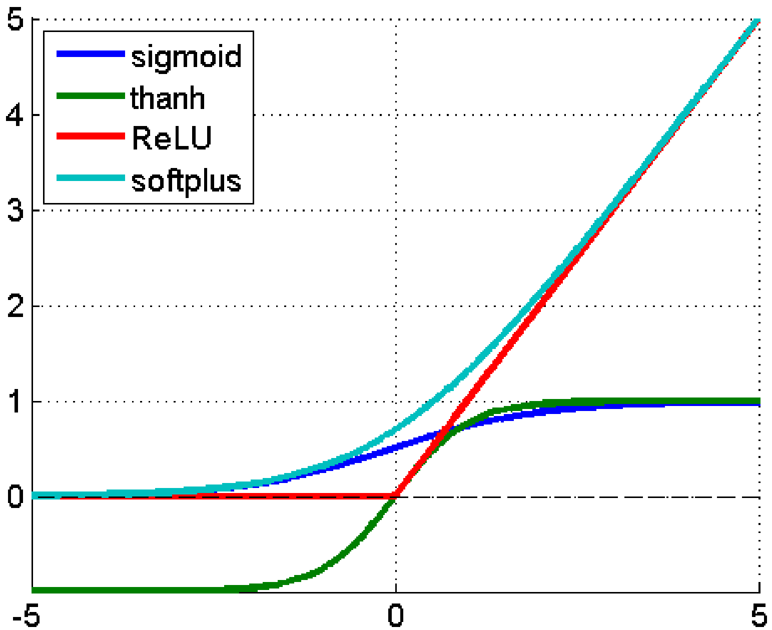
www.LBA.com/library

Multilayer Perceptron

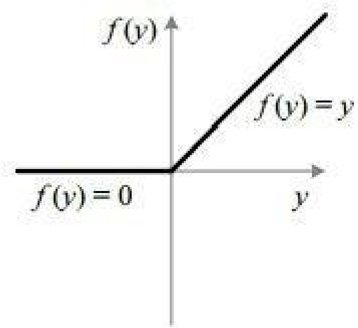


www.LBA.com/library

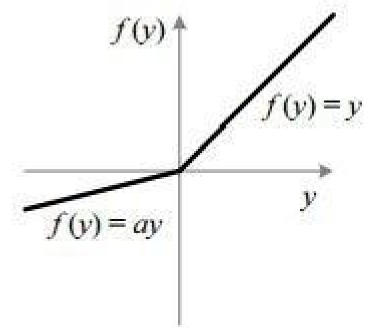
Common Activation Functions

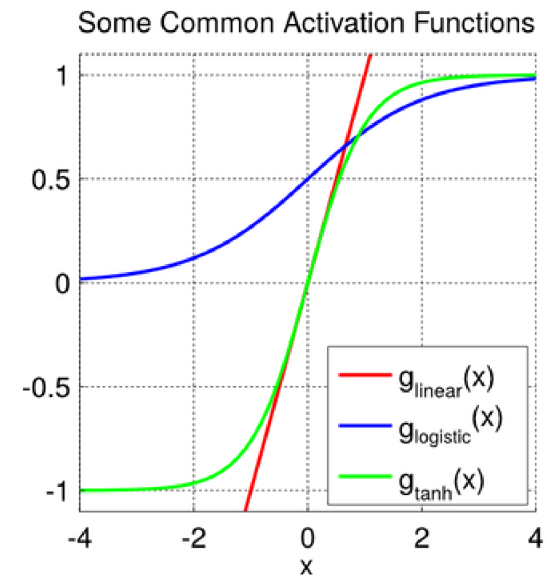


ReLU



Leaky ReLU

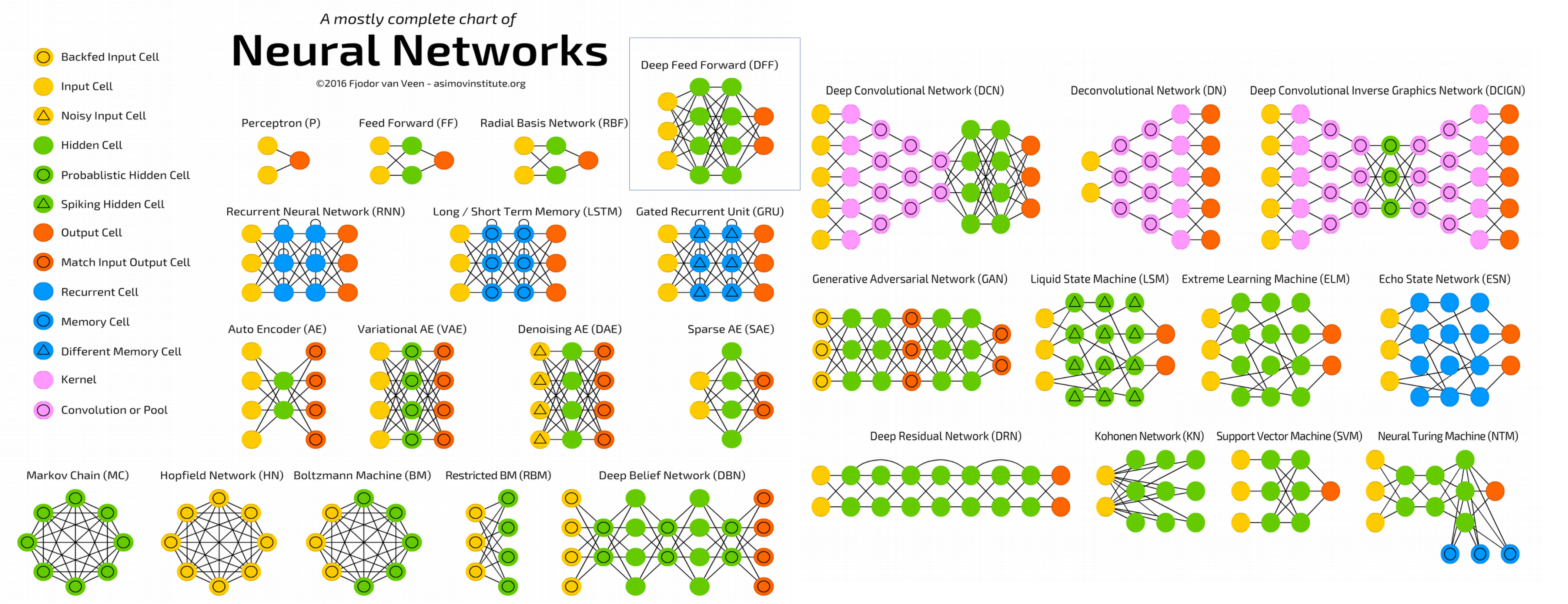




www.LBA.com/library

A Diversity of Network Architectures

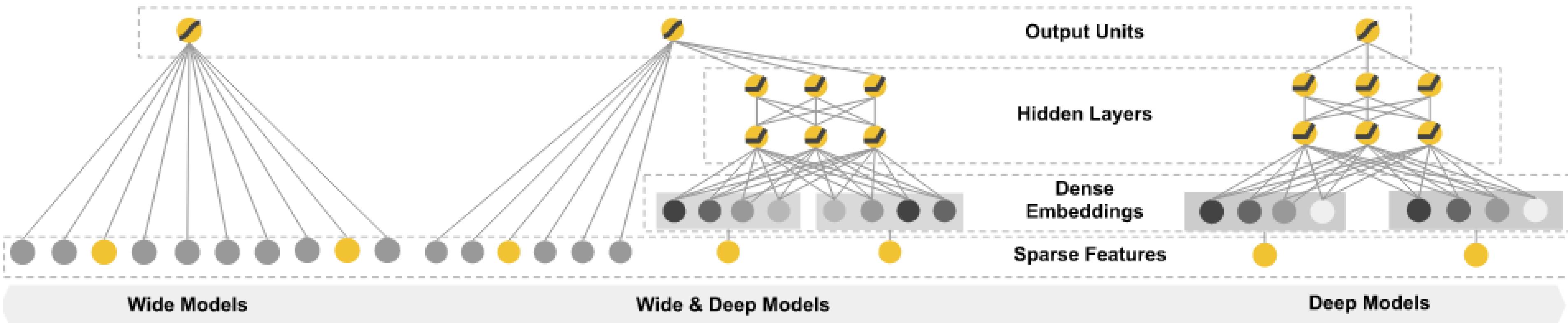
van Veen, 2016



http://www.asimovinstitute.org/neural-network-zoo/

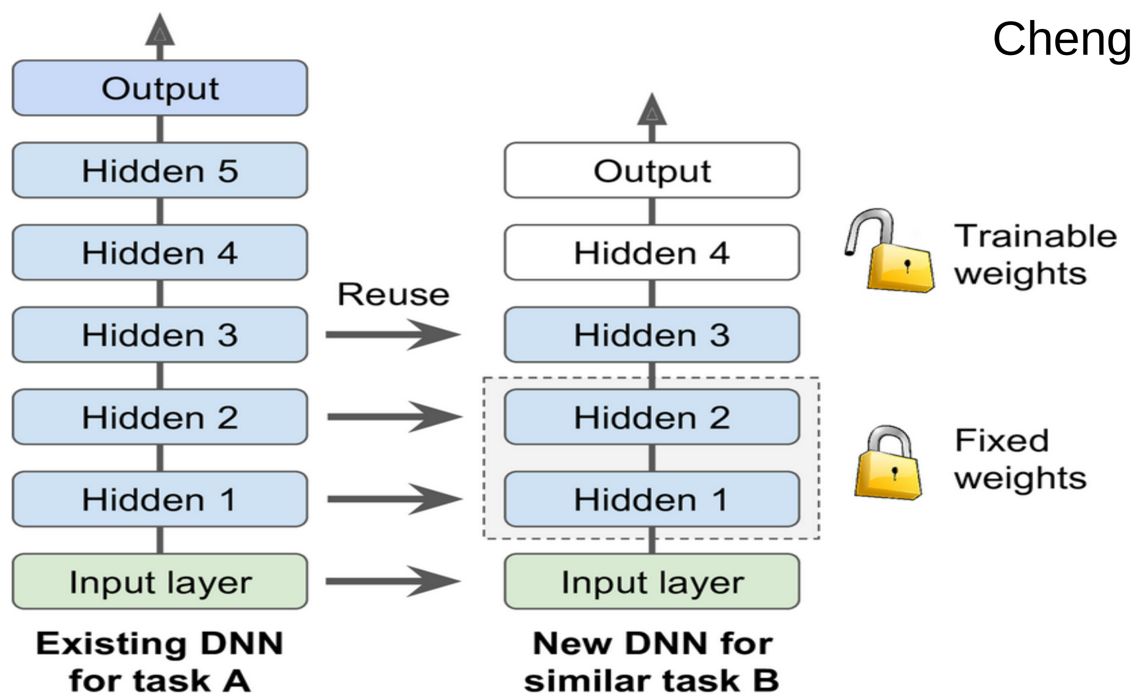
www.LBA.com/library

Some Variations and Extensions



Cheng et al., 2016

Géron, 2017



www.LBA.com/library

NN Training: Basic Elements

- Four basic elements:
  - Data
  - Model
  - Optimization Method
    - Some kind of gradient descent
  - Loss functions
    - Cross entropy: $H(y,p) = -\sum_i y_i \log(p_i)$
    - Mean squared error
- Evaluation
  - Predictive accuracy performance metrics using training and test data

www.LBA.com/library

“Features” (input data)

- NN models do math operations using continuous data
- Input data, or “features,” that are categorical are transformed
- Methods include:
  - Dummy, or “one hot,” encoding
  - “Hashing” : “bucketing” a large number of categories
  - “Embedding” : many categories respresented in a low dim space

www.LBA.com/library

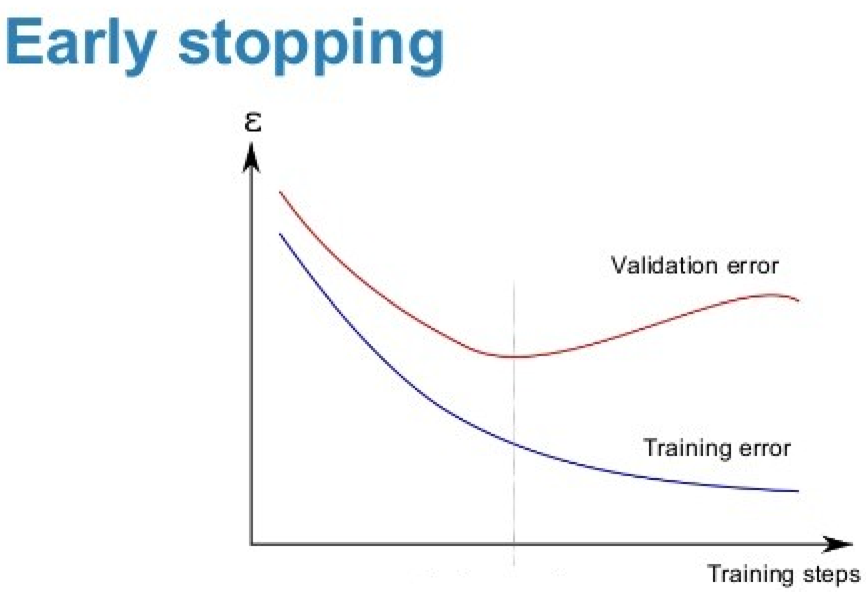
Back Propagation and Gradient Descent

- NN training involves:
  - Calculating the gradient of the loss function with respect to parameters ("weights")
  - Adjusting the weights to decrease the loss function
- "Backprop" is used to calculate the gradient
  - Most loss functions are nonconvex due to NN nonlinearity
- "Gradient descent"
  - Decrease loss in small, "learning rate" steps
  - *Stochastic* GD: use random samples of training data
- Parameter estimates during training can be examined
  - e.g. whether weights are "stuck" at some value like zero

www.LBA.com/library

Regularization

- Goal: reduce generalization error without decreasing training error
  - accept some bias in trade-off to variance reduction
- Methods:
  - Parameter penalties
  - Data augmentation
  - Noise “injection”
  - Multitask learning
  - Parameter sharing
  - Bagging / ensembles
  - Dropout
  - Early stopping



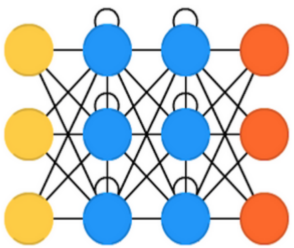
www.LBA.com/library

Performance Assessment

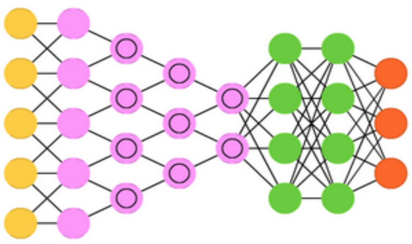
- On both training and test data sets
- For classifiers, the “usual” methods
    - Classification error or accuracy rate
    - Loss
    - LL
    - AUC
    - F1, precision, recall
  - For continuous outcomes, MSE

A Few Notable Special Types

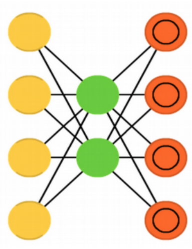
- Recurrent NN
  - text processing, e.g. NER, POS



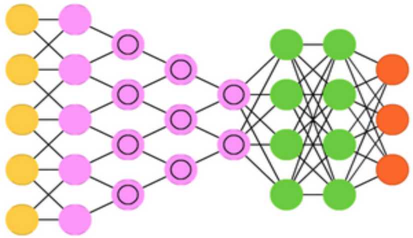
- Convolutional NN
  - image, audio recognition, text processing



- Autoencoder
  - *un*supervised learner
  - data compression

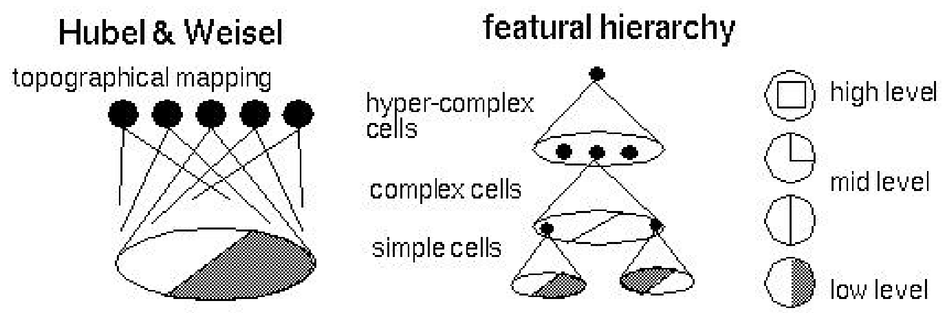


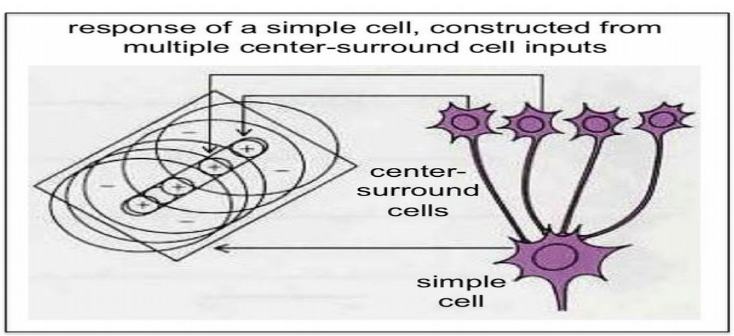
Convolutional NNs

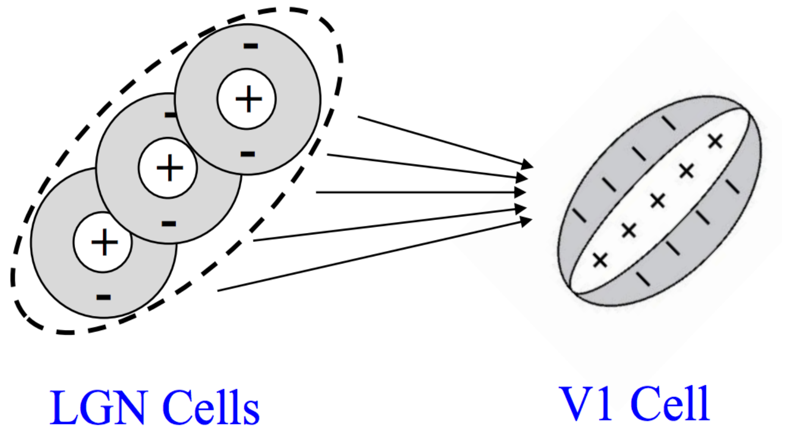


- Grid-like data in one or more dimensions
- Somewhat functionally similar to visual system elements
  - Inspired by Nobel work on mammalian visual processing
- Convolution : applying a kernel to input to produce a “feature map”
- Convolutional layers include both convolution and pooling
- CNN attributes: sparse connections, parameter sharing, equivariance.

Visual Input Processing

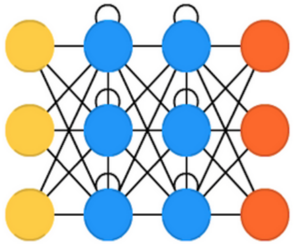




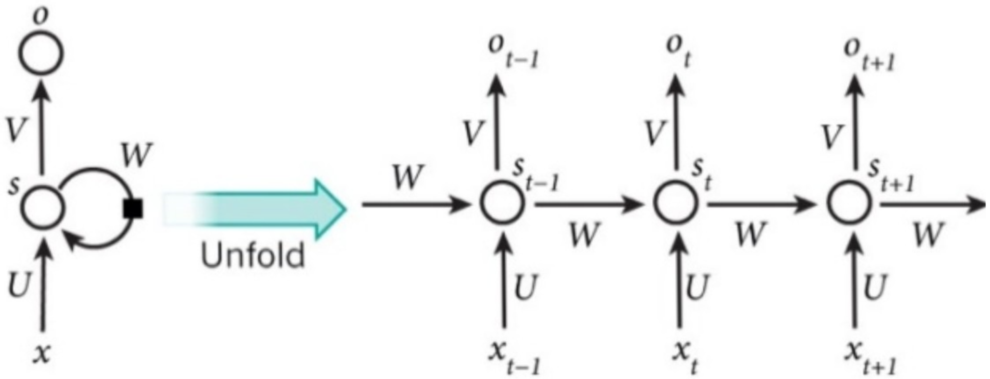


www.LBA.com/library

Recurrent NN's

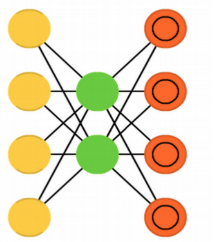


- Sequential data, e.g. for part of speech labeling of text data
- Sequences can vary in lengths
- Sequential dependencies “unfolded” to model state-to-state transitions



- Scalability and generalization facilitated by parameter sharing.

Autoencoders



- Network that reproduces its inputs
- An *unsupervised* learning method
- Two basic components
  - An *encoder* that reduces the dimensionality of inputs
  - A *decoder* that attempts to reproduce inputs from the outputs of the encoder
- Analogous to PCA

www.LBA.com/library

A Growing Host of Libraries and Platforms

- **TensorFlow** http://www.tensorflow.com
- **Theano** http://deeplearning.net/software/theano
- **Keras** https://keras.io/
- **PyTorch** http://pytorch.org
- **Blocks** http://blocks.readthedocs.io/en/latest
- **MxNet** http://mxnet.io
- **Lasagne** http://lasagne.readthedocs.org
- **H_2O** https://www.h2o.ai/

www.LBA.com/library

NN's at Scale

- “Scalability” can be accomplished in scope and in time
- GPU processing
- Using a cluster
- Parameter tying and sharing
- More machine “grunt”
- Care in coding and serializing

A Simple TensorFlow Example

- Tensorflow - machine learning numerical computation library
- Released by Google for public use
- Programmable with Python (also Java, C++, Go; other APIs in development, e.g. for Ruby, Haskell)
- Programming paradigm:
  - define model in graph form
  - execute using optimized C++ code
- Can take advantage of multiple CPUs and GPUs
- Scalable from small platforms to big ones
- Models deployable to diverse environments
- Somewhat of a “learning curve”
- www.tensorflow.org

An Example Application: Predicting Customer Segment Membership

- Customer data from a retailer, the “ZETA” company
- Sample from an enhanced CRM DB
- N=50,000
- Three “Buyer Engagement” segments
  - Low, Med, Hi = 21%, 44%, 35%
- 22 predictor variables from 3rd party suppliers
  - 5 continuous, 17 categorical
  - various degrees of missingness in categoricals
  - mostly inferred or aggregate measures

www.LBA.com/library

A Baseline “Shallow” MNL model

- Using TensorFlow
- 200,000 “training” iterations
- 80%/20% random training/test split

Training Data

```
accuracy:0.581
```

Test Data

```
accuracy: 0.581
```

DNN's for Predicting Engagement

- Two or three hidden layers
- 10 or 15 nodes per hidden layer
- Fully connected MLP
- Categorical feature embedding
- “Leaky ReLU” activation function
- Adaptive gradient optimizer
- “drop-out” probability of 0.10
- 200,000 training iterations
- Classification accuracy calculated for training and test samples

Classification Accuracy: Proportion Correct

Nodes per Hidden Layer	Data	Two Hidden Layers	Three Hidden Layers
10	Training	0.766	0.790
	Test	0.765	0.785
15	Training	0.789	0.789
	Test	0.784	0.785

Resources

Abadi, M., Chen, B, Chen, Z., Davis, A. Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y. and Zheng, X., “TensorFlow: A System for Large-Scale Machine Learning.” arXiv: 1605.08669v2, [es.DC], 31 May 2016.

Carew T, Castellucci V, Kandel E. An analysis of dishabituation and sensitization of the gill-withdrawal reflex in Aplysia. Int J Neurosci 1971;2:79–98.

Cheng, H., Koc, L., Harmsen, J., Skaked, T., Chandra, T., Aradhye, Anderson, G., Corrado, G., Chai, W., Ispir, M., Anil, R., Haque, Z., Hong, L., Jain, V., Liu, x. & Shah, H. (2016) “Wide & Deep Learning for Recommender Systems.” https://research.google.com/pubs/pub45413.html

Géron, A. “Hands-On Machine Learning with Scikit-Learn and TensorFlow.” Sebastopol CA: O'Reilly Media Inc., 2017.

Glorot, X. & Bengio, Y. (2010) “Understanding the difficulty of training deep feedforward neural networks.” Proceedings of the 13 th International Conference on Artificial Intelligence and Statistics (AISTATS) 2010, Chia Laguna Resort, Sardinia, Italy.

Goodfellow, I., Bengio, Y. and Courville, A. “Deep Learning. Cambridge MA: Massachusetts Institute of Technology, 2016.

Papert, Seymour; Minsky, Marvin Lee (1988). Perceptrons: an introduction to computational geometry. Cambridge, Mass: MIT Press. ISBN 0-262-63111-3.

Rumelhart, David E. & McClelland, James L., and PDP Research Group (1987). Parallel Distributed Processing: Explorations in the Microstructure of Cognition. ISBN 9780262680530,

Rumelhart, David E. & McClelland, James L. (1987) Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations. Cambridge Mass: MIT Press ISBN: 9780262291408 http://cognet.mit.edu/book/parallel-distributed-processing

Russell, S. & Norvig, P. “Artificial Intelligence: A Modern Approach, 3rd Ed.” Noida India: Pearson, 2015

Ruder, S. “An Overview of Gradient Descent Optimization Algorithms,” arXiv: 1609.04747v2 [cs.LG] 15 Jun 2017.

Tensorflow “Model Zoo:” https://github.com/tensorflow/models

van Veen, F. “The Neural Network Zoo.” (Sept. 14, 2016) http://www.asimovinstitute.org/neural-network-zoo/

www.LBA.com/library

Try It At Home!

- In an Jupyter Notebook,

  A simple DNN Binary Classifier Predicting Response to a Marketing Campaign
- Python,Tensorflow, scikit-learn, Other bits and pieces
- Bank marketing data available in the UCI Machine Learning Repository
- You'll find the Jupyter Notebook:

  CSP 2018 DNN Binary Classifier Example vx.ipynb

  where 'x' is an integer version number.

  at:

http://www.lba.com/library

- Supplementary Material -

Jupyter Notebook Example:

High Notes
and
Code Snippets

The Notebook:
“A Toe In The Tensorflow DNN Water”

- Python 3 + packages that include Tensorflow and Scikit Learn
- Trains and assesses the predictive accuracy of a two hidden layer binary classifier NN
- Bank marketing data from the UCI machine learning Repository
- Also, a binary logistic regression model for comparison purposes
- Assembled using Continuum.io Anaconda

www.LBA.com/library

The Data

UCI Machine Learning Repository: Bank Marketing Data Set https://archive.ics.uci.edu/ml/datasets/Bank+Marketing#

About Citation Policy Donate a Data Set Contact

UCI

Machine Learning Repository

Center for Machine Learning and Intelligent Systems

Search

Repository Web Google™

View ALL Data Sets

Bank Marketing Data Set

Download: Data Folder, Data Set Description

Abstract: The data is related with direct marketing campaigns (phone calls) of a Portuguese banking institution. The classification goal is to predict if the client will subscribe a term deposit (variable y).

Data Set Characteristics:	Multivariate	**Number of Instances:**	45211	**Area:**	Business
Attribute Characteristics:	Real	**Number of Attributes:**	17	**Date Donated**	2012-02-14
Associated Tasks:	Classification	**Missing Values?**	N/A	**Number of Web Hits:**	321450

Source:

[Moro et al., 2014] S. Moro, P. Cortez and P. Rita. A Data-Driven Approach to Predict the Success of Bank Telemarketing. Decision Support Systems, Elsevier, 62:22-31, June 2014

Packages and “Features”

Required packages are loaded into the Notebook's Python environment. Then, after reading the UCI ML Repository, variables are selected to use as inputs, “features.

```
import pandas as pd
import numpy as np
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score
from sklearn import preprocessing

# The following allows output from multiple statements to come out in a single output cell.
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
```

Avariables are selected from Pandas DataFrame “bankDat” of input data to be used as features. The data are then randomly split into training and test subsets.

```
varsToUse=['age','balance','housing','loan','campaign','previous','y']

campCusts=bankDat[varsToUse]
```

```
Xtrain, Xtest, yTrain, yTest = train_test_split(campCustX, convY,
                                                test_size=0.20, random_state=23)
```

www.LBA.com/library

“Normalization”, Logistic Regression

The training data are normalized, and then the test data are rescaled using the normalization parameters of the test data.

```
Xtrain=Xtrain.astype(float)

scaler = preprocessing.StandardScaler().fit(Xtrain)

# XTrain's columns should be mean = 0, std = 1
Xtrain = scaler.transform(Xtrain)

# Xtest is rescaled based on Xtrain's means and std devs
Xtest = scaler.transform(Xtest.astype(float))
```

A logistic regression model is fit to the training data, and then applied to the test data.

```
from sklearn import linear_model
logReg=linear_model.LogisticRegression()
logMod1=logReg.fit(Xtrain,yTrain)
```

```
training data accuracy 0.882
training data auc 0.688
test data accuracy 0.881
test data auc 0.689
```

www.LBA.com/library

NN: mini-batch generation

A function is used to randomly sample training data to be used in "mini-batches."

```
def get_batch(epoch, ncases, b_ndx, b_size):
    # epoch is the alg iteration, b_ndx is the batch no.
    # b_size is batch size

    ndxs= np.random.randint(ncases,size=b_size)
    X_bat=Xtrain[ndxs]
    y_bat=yTrain[ndxs]
    return X_bat, y_bat
```

NN: parameters

Various parameters are set. ;-)

Weights are initialized.

Tensors are defined to hold input and output data.

```
learn_rate = 0.1
b_size = 100                    # batch size
n_epochs = 100                  # no. of epochs
ncases=Xtrain.shape[0]          #no. of records
n_bats = int(np.ceil(ncases/b_size))  # no. of batches

# Inputs, nodes in hidden layers, classes in the output

n_hid_1 = 4 # 1st layer number of neurons
n_hid_2 = 4 # 2nd layer number of neurons
num_inp = 6 # selected campaign predictors
num_class = 2 # y values, 0 or 1


# tf placeholders for input data

X = tf.placeholder("float32", shape=(None, num_inp),name="X")
y = tf.placeholder("int32", shape=(None),name="y")

# Layer weights & biases

weights = {
    'h1': tf.Variable(tf.random_normal([num_inp, n_hid_1])),
    'h2': tf.Variable(tf.random_normal([n_hid_1, n_hid_2])),
    'out': tf.Variable(tf.random_normal([n_hid_2, num_class]))
     }
biases = {
    'b1': tf.Variable(tf.random_normal([n_hid_1])),
    'b2': tf.Variable(tf.random_normal([n_hid_2])),
    'out': tf.Variable(tf.random_normal([num_class]))
     }
```

NN: network, loss,optimizer, accuracy

Operations between layers and activation functions are specified.

```
def neural_net(x):
    # A two hidden fully connected layers each with 4 neurons, relu activation fcns
    layer_1 = tf.nn.relu(tf.matmul(X, weights['h1']))
    # Hidden fully connected layer with 4 neurons
    layer_2 = tf.nn.relu(tf.matmul(layer_1, weights['h2']))
    # Output fully connected layer with a neuron for each class
    out_layer = tf.matmul(layer_2, weights['out'])
    return out_layer
```

Loss function, optimizer, and accuracy metrics are specified.

```
logits = neural_net(X)

loss_op = tf.reduce_mean(tf.nn.sparse_softmax_cross_entropy_with_logits(
    logits=logits, labels=y))

optimizer = tf.train.AdamOptimizer(learning_rate=learn_rate)
train_op = optimizer.minimize(loss_op)

correct_pred=tf.nn.in_top_k(logits,y,1)
accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32))
```

www.LBA.com/library

Running in Tensorflow Session

A Tensorflow “session” is defined and executed. All variables initialized globally beforehand.

```
with tf.Session() as sess:
    init.run()
    print('epoch ',end='')
    for epoch in range(n_epochs):

        for b_ndx in range(n_bats):
            X_batch, y_batch=get_batch(epoch,ncases, b_ndx, b_size)
            sess.run(train_op, feed_dict={X: X_batch, y: y_batch})
        if epoch % 20 == 0:
            print(epoch, end=', ')

    print('done!\n')
    accTrain=accuracy.eval(feed_dict={X: Xtrain, y: yTrain})
    print('\ntraining data accuracy {:5.3f}'.format(accTrain))
    trainProbs=tf.nn.softmax(logits).eval(feed_dict={X: Xtrain, y: yTrain})
    print('training data auc {:5.3f}'.format(roc_auc_score(yTrain,trainProbs[:,1])))

    accTest=accuracy.eval(feed_dict={X: Xtest, y: yTest})
    print('test data accuracy {:5.3f}'.format(accTest))
    testProbs=tf.nn.softmax(logits).eval(feed_dict={X: Xtrain, y: yTrain})
    print('test data auc {:5.3f}'.format(roc_auc_score(yTrain,testProbs[:,1])))
```

www.LBA.com/library