Machine Learning for Sensory Signals
Homework # 1

1. Show that
(a) ∂(x Ax) = (A + A T)x

Machine learning Sensory Signals

Homework # 1

1. Show that

(a) $\frac{\partial}{\partial x} x^T A x = (A + A^T)x$

Proof :- let x be a $n \times 1$, and A be $n \times n$,

and α be the scalar resulting from product

of

$$\alpha = x^T A x$$

and it's given by.

$$\alpha = \sum_{j=1}^{n} \sum_{i=1}^{n} a_{ij} x_i x_j$$

Differentiating wrt K^{th} element of "x" we ha

$$\frac{\partial \alpha}{\partial x_K} = \sum_{j=1}^{n} a_{Kj} x_j + \sum_{i=1}^{n} a_{ij} x_j$$

for all $K = 1, 2, \ldots, n$, and consequently.

$$\frac{\partial \alpha}{\partial x} = x^T A^T + x^T A = x^T (A^T + A)$$

(b)∂tr(AB) = B T∂A

(b) $\frac{\partial}{\partial A} tr(AB) = B^T$

Proof :-

$$tr\,AB = tr \begin{bmatrix} \leftarrow \vec{a}_1 \rightarrow \\ \leftarrow \vec{a}_2 \rightarrow \\ \vdots \\ \leftarrow \vec{a}_n \rightarrow \end{bmatrix} \begin{bmatrix} \uparrow & \uparrow & \uparrow & & \uparrow \\ \vec{b}_1 & \vec{b}_2 & b_3 & \cdots & b_n \\ \downarrow & \downarrow & & & \downarrow \end{bmatrix}$$

$$= tr \begin{bmatrix} \vec{a}_1^T \vec{b}_1 & \vec{a}_1^T \vec{b}_2 & \cdots & \vec{a}_1^T \vec{b}_n \\ \vec{a}_2^T \vec{b}_1 & \vec{a}_2^T \vec{b}_2 & \cdots & \vec{a}_2^T \vec{b}_n \\ \vdots & & \ddots & \vdots \\ \vec{a}_n^T \vec{b}_1 & \cdots & & \vec{a}_n^T \vec{b}_n \end{bmatrix}$$

$$= \sum_{i=1}^{m} a_{1i} b_{i1} + \sum_{i=1}^{m} a_{2i} b_{i2} + \cdots \sum_{i=1}^{m} a_{ni} b_{in}$$

$$\frac{\partial tr\,AB}{\partial A} \Rightarrow \frac{\partial tr\,AB}{\partial a_{ij}}$$

$$= b_{ji}$$

$$= B^T$$

Hence Proved.

3. What are the different approaches to Machine learning in terms of classification settings. Enumerate the difference between Generative modeling and discriminative modeling.

Ans =

Different approaches to Machine learning in terms of classification settings:

1. Generative Modeling
    ->Estimating the likelihood and prior density seperately

2. Discriminative Modeling
    -> Directly Modeling the class posterior probability

3. Discriminant Function
    -> No probabilistic interpretation
    -> Fitting the cost function

Difference between generative and Discriminative Modelling:

1. A generative model learns the joint probability distribution p(x,y) and a discriminative model learns the conditional probability distribution p(y|x), read as "the probability of y given x".

2. The distribution p(y|x) is the natural distribution for classifying a given example x into a class y, which is why algorithms that model this directly are called discriminative algorithms.

Whereas in Generative algorithms model p(x,y), which can be tranformed into p(y|x) by applying Bayes rule and then used for classification. However, the distribution p(x,y) can also be used for other purposes. For example you could use p(x,y) to generate likely (x,y) pairs, generation of new data.

3. **Example:**
A generative algorithm models how the data was generated in order to categorize a signal. It asks the question: based on my generation assumptions, which category is most likely to generate this signal?

A discriminative algorithm does not care about how the data was generated, it simply categorizes a given signal.

For a two class classification problem :-

for generative ~~Models~~ approaches, under general assumptions, the posterior probability of class C_1 can be written as

Logistic sigmoid acting on linear function of the feature vector ϕ, so that.

$$p(C_1|\phi) = y(\phi) = \sigma(w^T \phi)$$

with

$$p(C_2|\phi) = 1 - p(C_1|\phi)$$

Here, $\sigma(\cdot)$ is the logistic sigmoid function.

The above model is known as logistic regression Model.

For a M dimensional feature space ϕ, this Model has M adjustable parameters.

We ~~know~~, can now use Max. Likelihood to determine the parameters of the logistic regression Model.

Derivative of logistic sigmoid function, which can be expressed as.

$$\frac{d\sigma}{da} = \sigma(1-\sigma)$$

For a data set $\{\phi_n, t_n\}$, where $t_n \in \{0, 1\}$ and $\phi_n = \phi(x_n)$, with $n = 1, \ldots, N$,

Likelihood can be written as.

$$p(t|w) = \prod_{n=1}^{N} y_n^{t_n} \{1 - y_n\}^{1-t_n}$$

Where

$t = (t_1, \ldots, t_N)^t$ and $y_n = p(C_1|\phi_n)$.

Error function can be defined as.

negative logarithm of likelihood.

which gives the cross entropy error in the form

$$E(w) = -\ln p(t|w) = -\sum_{n=1}^{N} \{t_n \ln y_n + (1-t_n) \ln(1-y_n)\}$$

where $y_n = \sigma(a_n)$, $a_n = w^T \phi_n$,

Taking gradient of the error function wrt w, we obtain.

$$\nabla E(w) = \sum_{n=1}^{N} (y_n - t_n) \phi_n$$

Multiclass Logistic Regression

For a large class of distributions, the posterior probabilities are given by a softmax transformation of linear functions of the feature variable,

$$p(C_k|\phi) = y_k(\phi) = \frac{\exp(a_k)}{\sum \exp(a_j)}$$

$$a_k = w_k^T \phi$$

Here we use Maximum likelihood to determine the parameters $\{w_k\}$ of this model directly.

Derivatives of y_k wrt all the activation a_j.

$$\frac{\partial y_k}{\partial a_j} = y_k(I_{kj} - y_j)$$

$I_{kj} \Rightarrow$ elements of identity Matrix.

Likelihood function :-

using 1 of k coding scheme, in which target vector t_n for a feature vector ϕ_n belonging to class C_k is binary vector, with all elements zero except for element k.

The likelihood function is then given by.

$$p(T|w_1 \dots w_K) = \prod_{n=1}^{N} \prod_{K=1}^{K} p(C_K|\phi_n)^{t_{nK}}$$

$$= \prod_{n=1}^{N} \prod_{K=1}^{K} t_{nK} y_{nK}.$$

where $y_{nK} = y_K(\phi_n)$

T is a $N \times K$ matrix of target variables with elements t_{nK}.

Taking negative logarithms then gives.

$$E(w_1 \dots w_K)$$

$$= -\ln p(T|w_1 \dots w_K)$$

$$= -\sum_{n=1}^{N} \sum_{K=1}^{K} t_{nK} \ln y_{nK}$$

which is known as Cross entropy function.

Now take the gradient of error function wrt the of the parameter vector w_j.

The derivative of softmax function,

$$\nabla_{w_j} E(w_1 \ldots w_k) = \sum_{n=1}^{N} (y_{nj} - t_{nj}) \phi_n$$

Logistic_regression_train_val_test_mnist_keras

March 7, 2018

1 Logistic Regression on MNIST

In [1]:
```
import six.moves.cPickle as pickle
import numpy as np
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.datasets import mnist
from keras.utils import np_utils
from keras import optimizers
```

```
Using TensorFlow backend.
```

In [2]:
```
def build_logistic_model(input_dim, output_dim):
    model = Sequential()
    model.add(Dense(output_dim, input_dim=input_dim, activation='softmax'))
    return model
```

In [3]:
```
batch_size = 128
nb_classes = 10
nb_epoch = 15
input_dim = 784
```

In [4]:
```
# the data, shuffled and split between train and test sets
(X_train, y_train), (X_test, y_test) = mnist.load_data()
print X_train.shape
print y_train.shape
```

```
(60000, 28, 28)
(60000,)
```

In [5]:
```
#converting each image into a single row vector
X_train = X_train.reshape(60000, input_dim)
X_test = X_test.reshape(10000, input_dim)
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train /= 255
X_test /= 255
```

In [6]:
```
# convert class vectors to binary class matrices
Y_train = np_utils.to_categorical(y_train, nb_classes)


#we stack the the images and there corresponding labels together.

X_total_train=np.hstack((X_train,Y_train))
print X_total_train.shape

# We make sure that the data is shuffled properly so that we have a uniformly distribute
np.random.shuffle(X_total_train)

#Again split the data into image vectors and their corresponding labels
X_train = X_total_train [:60000,:784]
Y_train = X_total_train [:60000,784:794]
print X_train.shape
print Y_train.shape
```

```
(60000, 794)
(60000, 784)
(60000, 10)
```

In [7]:
```
#Dividing the training set (X_train) of 60,000 images into a training set(X_train_t) of
#and a validation set(X_train_v) of 10,000
X_train_t = X_train[: 50000,:]
X_train_v = X_train[ 50000:60000,:]
print X_train_t.shape
print X_train_v.shape
print(X_train_t.shape[0], 'train samples')
print(X_train_v.shape[0], 'validation samples')
```

```
(50000, 784)
(10000, 784)
(50000, 'train samples')
(10000, 'validation samples')
```

In [8]:
```
# The corresponding labels are also diivded into training and validation set
Y_train_t = Y_train[: 50000,:]
Y_train_v = Y_train[50000:60000,:]
print Y_train_t.shape
print Y_train_v.shape

# convert class vectors to binary class matrices for test labels
Y_test = np_utils.to_categorical(y_test, nb_classes)
print(Y_test.shape[0], 'Test samples')
```

```
(50000, 10)
(10000, 10)
```

```
(10000, 'Test samples')
```

In [9]:
```
model = build_logistic_model(input_dim, nb_classes)

        model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 10)                7850
=================================================================
Total params: 7,850
Trainable params: 7,850
Non-trainable params: 0
_________________________________________________________________
```

In [10]:
```
# compile the model
         #setting the learning rate to be 0.01
         sgd = optimizers.SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
         model.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['accuracy'])

         #using training and validation data for 15 epochs
         history = model.fit(X_train_t, Y_train_t,
                             batch_size=batch_size, nb_epoch=nb_epoch,shuffle=True,
                             verbose=1, validation_data=(X_train_v, Y_train_v))

         #evaluating the model on test set
         score = model.evaluate(X_test, Y_test, verbose=0)
```

```
/usr/local/lib/python2.7/dist-packages/keras/models.py:942: UserWarning: The `nb_epoch` argument
  warnings.warn('The `nb_epoch` argument in `fit` '


Train on 50000 samples, validate on 10000 samples
Epoch 1/15
50000/50000 [==============================] - 1s 21us/step - loss: 1.3573 - acc: 0.6646 - val_l
Epoch 2/15
50000/50000 [==============================] - 1s 16us/step - loss: 0.7720 - acc: 0.8307 - val_l
Epoch 3/15
50000/50000 [==============================] - 1s 16us/step - loss: 0.6263 - acc: 0.8544 - val_l
Epoch 4/15
50000/50000 [==============================] - 1s 16us/step - loss: 0.5564 - acc: 0.8647 - val_l
Epoch 5/15
50000/50000 [==============================] - 1s 19us/step - loss: 0.5141 - acc: 0.8711 - val_l
Epoch 6/15
50000/50000 [==============================] - 1s 17us/step - loss: 0.4850 - acc: 0.8762 - val_l
Epoch 7/15
```

```
50000/50000 [==============================] - 1s 16us/step - loss: 0.4636 - acc: 0.8798 - val_l
Epoch 8/15
50000/50000 [==============================] - 1s 17us/step - loss: 0.4470 - acc: 0.8829 - val_l
Epoch 9/15
50000/50000 [==============================] - 1s 17us/step - loss: 0.4336 - acc: 0.8855 - val_l
Epoch 10/15
50000/50000 [==============================] - 1s 16us/step - loss: 0.4225 - acc: 0.8875 - val_l
Epoch 11/15
50000/50000 [==============================] - 1s 16us/step - loss: 0.4131 - acc: 0.8893 - val_l
Epoch 12/15
50000/50000 [==============================] - 1s 16us/step - loss: 0.4051 - acc: 0.8917 - val_l
Epoch 13/15
50000/50000 [==============================] - 1s 16us/step - loss: 0.3980 - acc: 0.8928 - val_l
Epoch 14/15
50000/50000 [==============================] - 1s 17us/step - loss: 0.3918 - acc: 0.8941 - val_l
Epoch 15/15
50000/50000 [==============================] - 1s 16us/step - loss: 0.3863 - acc: 0.8954 - val_l
```

```
In [11]: print('Test score:', score[0])
         print('Test accuracy:', score[1])

('Test score:', 0.3645026022195816)
('Test accuracy:', 0.9024)
```

2 comparing the validation and test accuracies for different set of Learning rate and Batch Size

Learning Rate	Val Acc	Test Acc
0.001	0.9144	0.9197
0.01	0.9141	0.9210
0.05	1.9153	0.9216
0.1	0.9170	0.9214

Batch Size	Val Acc	Test Acc
1	0.9139	0.9207
32	0.9206	0.9261
128	0.9208	0.926
1024	0.9208	0.926