LOGISTIC REGRESSION

Edit

I. OBJECTIVE:

We use different equations depending on the number of output classes. With 2 classes, we will use binomial cross entropy for loss and more than 2 classes involves using the cross-entropy with softmax.

Regression	Binomial	Multinomial (n>2)
$\hat{y}$	$\hat{y} = \frac{1}{1+e^{-\theta X}}$	$\hat{y} = \frac{e^{\theta_y X}}{\sum e^{\theta X}}$
cross entropy	$J(\theta) = -\frac{1}{n}\sum_i^n [y_i log(\hat{y}_i) + (1-y_i) log(1-\hat{y}_i)]$	$J(\theta) = -\sum_i y_i ln(\hat{y}_i)$
forward pass	$[NX1] = [NXD][DX1]$	$[NXC] = [NXD][DXC]$
response	$y = 0$ or 1	$y =$ one-hot-encoded

II. MULTINOMIAL CROSS ENTROPY:

Example of how to calculate the cross-entropy loss for a 3 class problem.

$$J(\theta) = -\sum_i y_i ln(\hat{y}_i)$$

Ex:

Computed ($\hat{y}$)	Targets (y)
[0.3, 0.3, 0.4]	[0, 0, 1]

$$J(\theta) = -\sum_i^n [0 * ln(0.3) + 0 * ln(0.3) + 1 * ln(0.4)] = -ln(0.4)$$

III. BACKPROPAGATION:

With the multinomial cross entropy, you can see that we only keep the loss contribution from the correct class. Usually, with neural nets, this will be case if our ouputs are sparse (just 1 true class). Therefore, we can rewrite our loss into just a sum(-log(y_hat)) where y_hat will just be the probability of the correct class. We just replace y_i (true y) with 1 and for the probabilities for the other classes, doesn't matter because their y_i is 0. This is referred to as negative log likelihood.

Note: The y^i below is just the probability for the correct class, so the loss is only affected by the probability of the correct class. However, the gradients DO take into account the probabilities for the other classes. This allows us to change all the weights towards minimizing the loss function. Also all the ys below are y_hats.

$$J(\theta) = -\sum_i ln(\hat{y}_i)$$

Negative Log Likelihood

$$\frac{\partial J}{\partial W_j} = \frac{\partial J}{\partial y}\frac{\partial y}{\partial W_j} = -\frac{1}{y}\frac{\partial y}{\partial W_j} = -\frac{1}{\frac{e^{W_y X}}{\sum e^{XW}}}\frac{\sum e^{XW} e^{W_y X} 0 - e^{W_y X} e^{W_j X} X}{(\sum e^{XW})^2} = \frac{X e^{W_j X}}{\sum e^{XW}} = XP$$

$$\frac{\partial J}{\partial W_y} = \frac{\partial J}{\partial y}\frac{\partial y}{\partial W_y} = -\frac{1}{y}\frac{\partial y}{\partial W_y} = -\frac{1}{\frac{e^{W_y X}}{\sum e^{XW}}}\frac{\sum e^{XW} e^{W_y X} X - e^{W_y X} e^{W_y X} X}{(\sum e^{XW})^2} = \frac{1}{P}(XP - XP^2) = X(P-1)$$

$$W_j = W_j - \alpha \frac{\partial J}{}$$

- SEARCH -

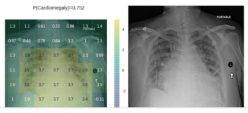
Search …

- FOLLOW ME ON TWITTER -

Tweets by @GokuMohandas

Goku Mohandas
@GokuMohandas

Your model may be performing really well by incorrectly focusing on confounding features (extraneous influencers in the data that aren't accounted for). Check out @johnrzech's post where x-ray stickers unintentionally influenced the classifications: medium.com/@jrzech/what-a…



Embed View on Twitter

- RECENT POSTS -

Update on Embeddings (Spring 2017)

Exploring Sparsity in Recurrent Neural Networks

Question Answering from Unstructured Text by Retrieval and Comprehension

Overcoming Catastrophic Forgetting in Neural Networks

Opening the Black Box of Deep Neural Networks via Information

$$W_i = W_i - \alpha \frac{\partial}{\partial W_i}$$

IV. SOFTMAX CLASSIFIER IMPLEMENTATION:

Naive Implementation:

```
loss = 0.0
dW = np.zeros_like(W)
num_train = X.shape[0]
num_classes = W.shape[1]

for i in xrange(num_train):

    scores = X[i, :].dot(W)
    scores -= np.max(scores)
    normalized_scores = np.exp(scores) / np.sum(np.exp(sco

    for j in xrange(num_classes):

        if j == y[i]:
            loss += -np.log( normalized_scores[j] )
            dW[:, j] += (normalized_scores[j] - 1.0) * X[i
        else:
            dW[:, j] += (normalized_scores[j] - 0.0) * X[i

loss /= num_train
dW /= num_train
loss += 0.5 * reg * np.sum(W * W)
dW += reg * W
```

Vectorized Implementation:

```
loss = 0.0
dW = np.zeros_like(W)
num_train = X.shape[0]
num_classes = W.shape[1]

scores = np.exp(X.dot(W))
normalized_scores = (scores.T / np.sum(scores, axis=1)).T

ground_truth = np.zeros_like(normalized_scores)
ground_truth[range(num_train), y] = 1.0 # correct class

loss = np.sum(np.sum(-np.log(normalized_scores[range(num_t
dW = X.T.dot(normalized_scores - ground_truth)

loss /= num_train
dW /= num_train
loss += 0.5 * reg * np.sum(W * W)
dW += reg * W
```

V. CODE BREAKDOWN:

We will be using **load_data()** to load the MNIST data if needed and separate into train and test sets.

```
import tensorflow as tf
import numpy as np
import input_data

class parameters():

    def __init__(self):
        self.LEARNING_RATE = 0.05
        self.NUM_EPOCHS = 500
        self.DISPLAY_STEP = 1 # epoch

def load_data():
    mnist = input_data.read_data_sets("MNIST_data/", one_h
    trainX, trainY, testX, testY = mnist.train.images, mni
    return trainX, trainY, testX, testY
```

We will create a model and use softmax cross-entropy with logits as our loss function. And **step()** will do the training/validation steps for us. Not the **forward_only** argument. When we are validating on the test set, we do not want to train on this set, so we will not do any operations involving the optimizer.

```
def create_model(sess, learning_rate):
    tf_model = model(learning_rate)
    sess.run(tf.initialize_all_variables())
    return tf_model

class model(object):

    def __init__(self, learning_rate):

        # Placeholders
        self.X = tf.placeholder("float", [None, 784])
        self.y = tf.placeholder("float", [None, 10])

        # Weights
        with tf.variable_scope('weights'):
            W = tf.Variable(tf.random_normal([784, 10], st

        self.logits = tf.matmul(self.X, W)
        self.cost = tf.reduce_mean(tf.nn.softmax_cross_ent
        self.optimizer = tf.train.GradientDescentOptimizer

        # Prediction
        self.prediction = tf.argmax(self.logits, 1)

```

```
    def step(self, sess, batch_X, batch_y, forward_only=Tr

        input_feed = {self.X: batch_X, self.y: batch_y}
        if not forward_only:
            output_feed = [self.prediction,
                           self.cost,
                           self.optimizer]
        else:
            output_feed = [self.cost]

        outputs = sess.run(output_feed, input_feed)

        if not forward_only:
            return outputs[0], outputs[1], outputs[2]
        else:
            return outputs[0]
```

We will train the model using the entire train/test batches at each step.

```
def train(FLAGS):

    with tf.Session() as sess:

        model = create_model(sess, FLAGS.LEARNING_RATE)
        trainX, trainY, testX, testY = load_data()

        for epoch_num in range(FLAGS.NUM_EPOCHS):
            prediction, training_loss, _ = model.step(sess

            # Display
            if epoch_num%FLAGS.DISPLAY_STEP == 0:
                print "EPOCH %i: \n Training loss: %.3f, T
                    epoch_num, training_loss, model.step(s

if __name__== '__main__':
    FLAGS = parameters()
    train(FLAGS)
```

VI. RAW CODE:

GitHub Repo (Updating all repos, will be back up soon!)

Posted in: *Regression*
Tagged: *Tutorials*

← *Linear Regression* *Vanilla Neural Network* →

3 THOUGHTS ON "LOGISTIC REGRESSION"

FIDELA January 30, 2018 at 5:41 pm EDIT REPLY →

___123___Logistic Regression – The Neural Perspectives___123___

★ Like

FRIGG February 4, 2018 at 3:45 pm EDIT REPLY →

___123___Logistic Regression – The Neural Perspectives___123___

★ Like

KATIA BUSER February 5, 2018 at 7:52 pm EDIT REPLY →

They would not assume it's joyous if it concerned a planet
burning and all of us getting stumped flat by a planet
now, would they? We consider the number '5' as making use of to acceptable groups
of any entities in anyway-to five fishes, 5 kids, 5
apples, five days https://math-problem-solver.com/ .
60 minute response time or its free.

★ Like

LEAVE A REPLY

Enter your comment here...