Artificial Neural Networks

Prof. Dr. Sen Cheng

Dec 2, 2019

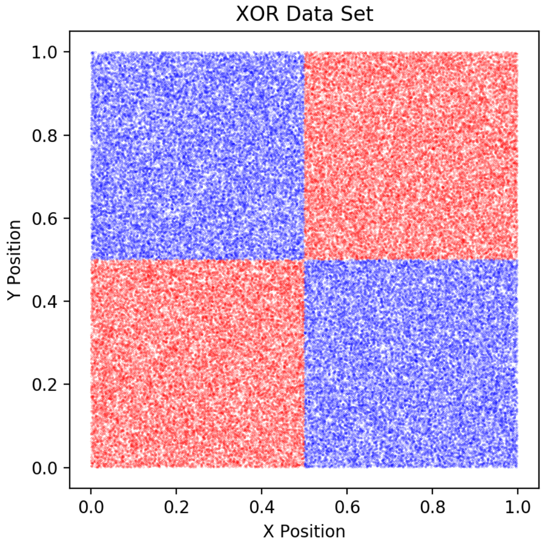
Problem Set 9: Multi-Layer Perceptron

Tutors: Sandhiya Vijayabaskaran (sandhiya.vijayabaskaran@rub.de), Nicolas Diekmann (nicolas.diekmann@rub.de)

1. **Backpropagation algorithm**

   A skeleton of a class that models a multi-layer perceptron (MLP) with one hidden layer is provided in *prob_mlp.py*. The *main* function of the provided code already loads the data set and splits it into training and test sets (Note: While you are actively developing and testing your code, use a smaller training set to speed things up.)

   (a) Implement the function *train* that trains the MLP's weights with backpropagation using stochastic gradient descent.



   The first two columns of the data set contain x and y position of the data point while the third column represents the class identiy. Class identities are coded with $+1$ and -1. Use the following parameters for your network:

   - Hyperbolic tangent as activation function for hidden and output layers.
   - Use the squared error (SE) as loss function (Note: for a classification problem, one would normally use the cross-entropy loss, but here we will stick to the squared error as it is easier to compute.)
   - Hidden layer with 64 units. Also add a bias unit to the hidden layer.

   (b) Train the MLP on the provided XOR data set (*xor.npy*) using a learning rate of 0.005.

   (c) Implement a *predict* function in the MLP class. This function should compute the mean squared error (MSE) on the test set and predict the class labels to compute the misclassified points (using 0 as the threshold).

(d) Repeat the training for one epoch 100 times (reinitializing the weights in between runs).

   i. Record the MSE for each run and compute the mean and standard deviation of the error across all the runs.

   ii. Also record the misclassified points for each run using the test function. This will be useful for Problem 3.

2. **Xavier initialization**

   The MLP class provided initializes weights by drawing them from a normal distribution with zero mean and standard deviation of 1. Extend the class so that weights can be optionally initialized using Xavier initialization. Test the implementation identically to Problem 1, changing only the initialization type. Plot the mean and SD of the error together with those recorded from Problem 1. Which initialization performs better? Is that expected?

3. **Misclassifications**

   (a) Count the misclassifications that you recorded from Problems 1. and 2. in separate 100x100 bins and make heatplots. How does the choice of weight initialization method affect the distribution of the MLP's misclassifications?

   (b) Instead of the hyperbolic tangent use ReLU activations for the hidden layer and repeat the previous tasks. Also change the number of hidden units to 20. How does the MSE for the weight initialization methods compare to when using the hyperbolic tangent as activation function? How does the distribution of the MLP's misclassifications differ qualitatively?

4. **Training ANN in Keras**

   The above tasks can also be solved using standard machine learning libraries. For example, you could use `sklearn.neural_network.MLPClassifier` to construct an identical architecture. We will concentrate on using `keras` in this exercise. The file *prob_keras.py* contains code which creates an MLP identical to Problem 1. The code is briefly explained below :

   - We will use the Sequential model API, which enables the creation of a stack of layers (from `keras.models` import `Sequential`)
   - Layers are added to a keras model using the method `model.add`. The type of layer we will use is the *Dense* layer, which means that each unit is connected to *all units* in the previous layer, as in the case of an MLP. The input layer is specified directly as an argument, `input_dim` in the hidden layer. All layers in keras are available from `keras.layers`
   - The activation function is specified using *Activation* layers in the code. It is also possible to use the argument `activation` directly instead. We must also specify the optimizer to use for training (SGD) which can be imported from `keras.optimizers`.
   - Finally, before training, the model must be configured for training by calling `model.compile`. The model is trained for one epoch using `model.fit`.
   - Once the model has been trained, we can evaluate the model's performance on the test set. In the code provided, we have used `test_on_batch`. You can also use the `evaluate` function to do this.

   (a) Modify the above code to directly perform classification: the dataset provided uses class labels -1 and +1. Modify the class labels of the training set by using a one-hot encoding instead. The built-in function `keras.utils.to_categorical` can be used to accomplish this. Next, change the output layer to have one neuron per class, and use a softmax activation to compute the class probabilities. Repeat the training using the cross-entropy loss instead of MSE.

   (b) Evaluate the loss and predict the classes on the test set. Also compute the MSE and plot it as before. Compare it to the MSE in assignment 1. Why don't we compare the cross-entropy loss directly with the MSE?

Solutions

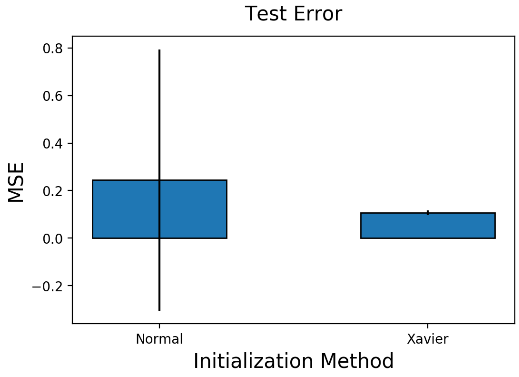
The full code solution for problems $1-3$ is available in the file *sol_mlp.py*.

1. **Backpropagation algorithm**

   The train function should implement the forward and backward pass using Algorithm 9.1 from the lecture notes. The prediction function computes the output using the forward pass and the mean squared error.

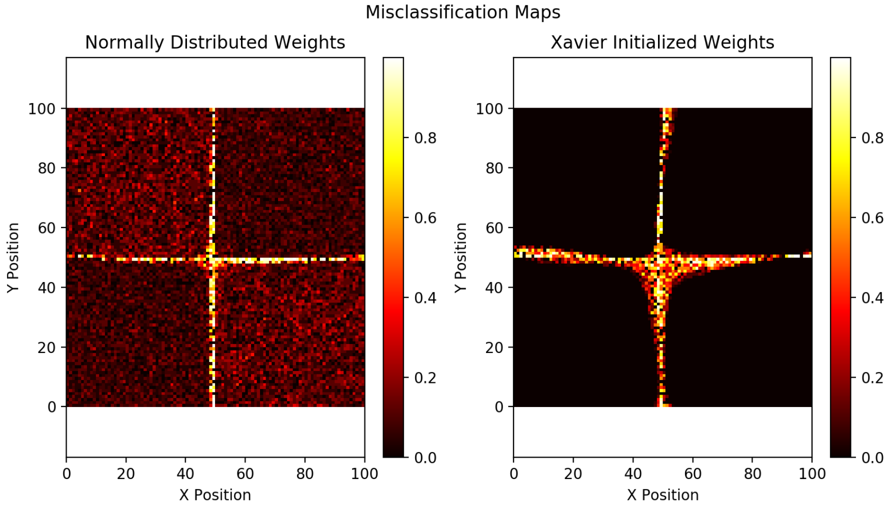
2. **Xavier initialization**

   Weights drawn from a normal distribution with unit variance lead to a large MSE with high variance. If Xavier initialization is used, the MSE is small with low variance.

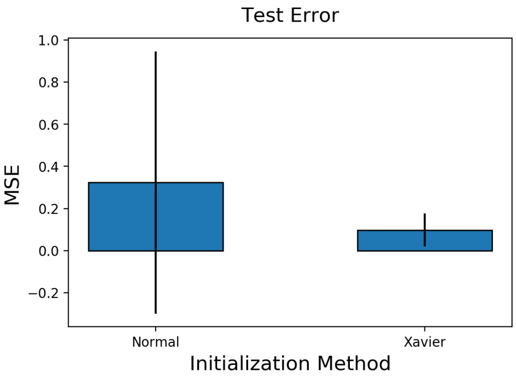


3. **Misclassifications**

   (a) The distribution of misclassified points using different weight initialization schemes. For both initialization schemes, misclassifications mostly occur at the class boundaries. If weights are normally distributed, the MLP is not able to properly separate the data in some runs, which is reflected by off-boundary misclassifications covering the misclassification map (resembling background noise). When using Xavier initialization, the off-boundary misclassifications disappear almost completely, showing the superiority of Xavier initialization.

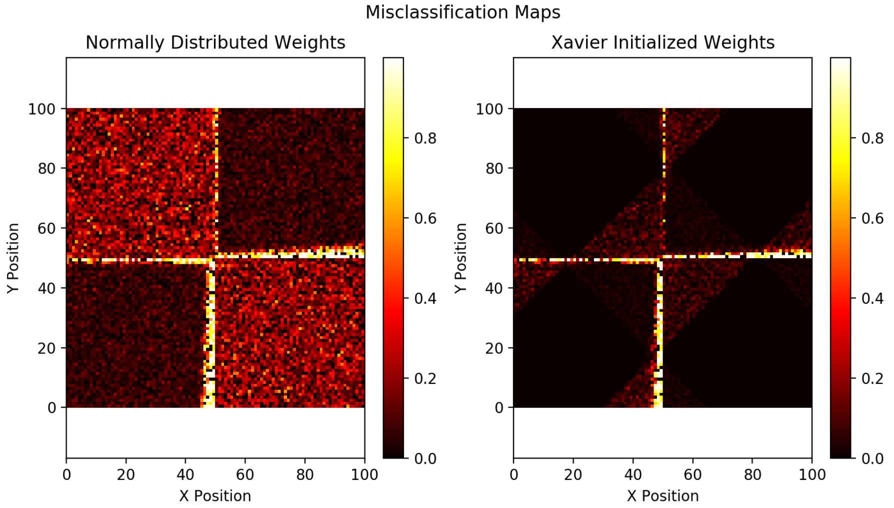


   (b) Prediction errors using ReLU activations.



Compared to using the hyperbolic tangent as activation function, the MSE is similiar for weights initalized in the standard way and slightly lower for Xavier-initialized weights. However, the variance of of MSE when using Xavier-initialized weights is higher.

Distribution of misclassified points using ReLU activation.



Misclassifications also occur mostly at the class boundaries. Off-boundary misclassifications behave similarly compared to when using the hyperbolic tangent activation. Noteworthy is the linear pattern of off-boundary misclassifications produced by the ReLU activation function (you will find a pattern reminiscent of curves when using the hyperbolic tangent)

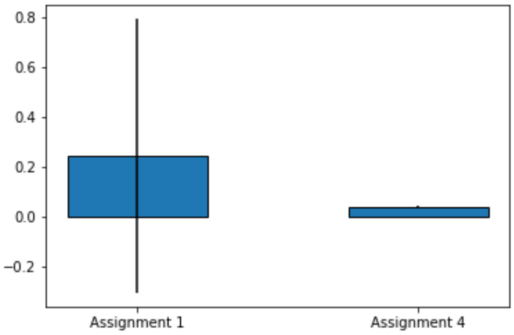
4. **Training ANN in Keras**

(a)

```
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.optimizers import SGD
from keras.utils import to_categorical
from keras import backend
import numpy as np

#load and split dataset
dataset = np.load('xor.npy')
X, y = dataset[:,:2], dataset[:,2]
y = np.where(y==−1,0,1) #convert class labels to 0 & 1
bias = np.ones((X.shape[0],1))
X = np.concatenate((bias, X), axis=1)
y_enc= to_categorical(y)
X_train, X_test, y_train, y_test = X[:80000], X[80000:], y_enc[:80000], y_enc[80000:]
```

```

#create the network
mse = []
for run in range(100) :
    model = Sequential()
    model.add(Dense(64,input_dim=3))
    model.add(Activation('tanh'))
    model.add(Dense(2))
    model.add(Activation('softmax'))

    #add optimizer, compile and train
    sgd = SGD(lr=0.005)
    model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['mse'])

    #evaluate loss and predict classes on test set
    model.fit(X_train, y_train, batch_size=1, epochs=1)
    loss = model.evaluate(X_test,y_test)
    mse.append(loss[1])

    backend.clear_session()

mse_mean = np.mean(mse)
mse_std = np.std(mse)
```

(b) We do not compare MSE with the Cross-entropy loss as it is not possible to directly compare the values of two different loss functions which are computed differently. Instead, we compute the MSE as an additional metric in order to compare it with the MSE from Problem 1.


Assignment 1
Assignment 4