Assignment 3: Machine Learning and Protein Decoy Detection

Integrated Workshop

Due Friday, Nov. 10, 2023 at 11:59 PM EST

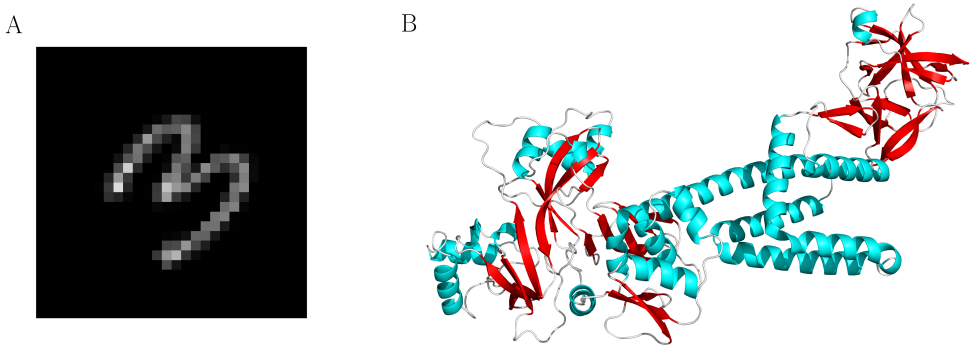

Fig. 1: A. An image of a hand drawn digit from the MNIST database B. Prediction of SARS-CoV-2 protein nsp2 by Google's deep learning research group, DeepMind. Their method is named AlphaFold. How close to the experimental structure do you think this model is?

Matlab Deep Learning Toolbox

Matlab is a commercial product with many developed toolboxes to help you compute whatever you'd like as easily as possible. To find new toolboxes, open Matlab, select the Home tab and then select Add-ons. This will open a new page called the Add-on Explorer. In the search bar, search "deep learning". Select the first search result called "Deep Learning Toolbox" and install. And like that, you're ready to learn, deeply! As a note, if you're interested in pursuing deep learning and data science more generally, most of the work is done in Python. I would recommend learning about TensorFlow or PyTorch. There are many great classes available at Yale about machine learning e.g. CBB 555 Unsupervised Learning for Big Data and CBB 663 Deep Learning Theory and Applications.

Classification: MNIST

In Machine learning, there are common datasets that are used to test new methods as well as illustrate concepts. If you follow any tutorial about image recognition, you will certainly come across MNIST. MNIST is a collection of hand drawn numbers from 0-9 that have been pixelated into 28x28 matrices. It is a great test case for image classification, as it is not obvious how one would code by hand a method to recognize the diversity of hand drawn digits, but it is a straightforward task for machine learning methods.

Let's load the dataset! MNIST is so well known, there is a command to download it from Matlab:

```
% Get Data
    [XTrain,YTrain,anglesTrain] = digitTrain4DArrayData;
    [XTest,YTest,anglesTest] = digitTest4DArrayData;
```

Plot one of the images using imshow(). Note the shape of the data using size(). We have (28, 28, 1, 5000). The third dimension is typically reserved for different RGB values, however, the images are grey-scale so we only have one value.

To prepare the data for a feed-forward neural network, we need to flatten these 2D images into a single row of features. Prepare Xtrain and Xtest to have the shape (784, 5000) so each column is a flattened image. Even in this strange flattened representation, our network will easily recognize MNIST!

One standard way to classify something is to not output the class of the observation, but to output the probabilities that the observation fits into each class. In this way, our network will take in a vector of length 784 and output a vector of length 10, where each entry is the probability that the image is a certain number from 0-9. This type of class representation is called "one-hot" encoding, as we will place a 1 in the index that corresponds to that number, with 0s everywhere else. Use the following example:

```
% One hot encode labels
    B = grp2idx(YTrain);
    Y_train_onehot = zeros(10, num_train);
    for ii = [1:num_train]
        Y_train_onehot(B(ii),ii) = 1;
    end
```

Your data is prepared! Time to learn. Matlab has a number of built-in networks, so let's try one out. Initialize your network:

```
% Initializing a classifier
    net = patternnet(10);
```

View a nice graphic of your network with view(net). You should see a graphical representation of your network. At the far left we can see the input to our network is 784 long and on the far right we can see the output is 10 long. We have one hidden layer that is length 10 and one output layer that is also length 10. There is a visual representation of the multiplication of the weights and the addition of the biases, before it is passed through the activation function, which is depicted graphically. The first activation is a sigmoidal type function. The second activation for the output layer is called a softmax layer. A softmax layer is typically added to the outputs of classifiers, as it scales the output to be a probability. As our outputs are probabilities, we will be using the cross-entropy function to score how similar the predicted and actual probabilities are to each other:

$$H(p,q) = -\sum p(x)\log_{10}(q(x)) \tag{1}$$

where p is the true probability distribution of the image being a number from 0-9 and q is the predicted probability distribution by the network that the image is a number from 0-9 and the sum is over all possible outcomes x=0-9.

Let's train the network!

```
% Train the network
    [net,tr] = train(net, X_train_flat, Y_train_onehot);
```

You'll see a nice interface appear that tells you more about the network and its progress as it trains. It should train rather quickly. Now that it is trained, you can test it:

```
% Test the network
    predicted_labels = net(X_test_flat);
```

The training and testing sets are both random selections from a database of handwritten numbers. When training, Matlab will set a default split of the train data into train/validation in order to determine how long to train the data. The network will update its weights according to the training data, while also seeing how accurate the output is for the validation data without changing the weights. This way, once the training accuracy diverges from the validation, it is assumed the network is overfitting to the training data, and training is stopped. Finally, we test it on data that was not used in the training process to simulate how the trained network would perform on data it has not seen before.

Assignment Part 1: MNIST

Now that we have outlined how to set up, train, and test a neural network for MNIST in Matlab, let's explore the results and see what changes we can make. Prepare answers to the following questions in a document titled: `Assignment3-write-up-part1-LASTNAME-FIRSTNAME.txt`

1. How accurate was your out-of-the-box network? The most straightforward way to quantify this is to loop through your predicted labels, compare to the actual label,

and count the number of times it is correct/incorrect. Each value in the output is the probability that the image is the number of that index. Find the index of the maximum value in the output to find the predicted number. Report the fraction of test images that were classified correctly.

2. Are there trends in the mistakes that the network makes? Loop through the data and count the times it classifies a zero a zero, and a zero a one, a two etc. This is known as the confusion matrix, where the x-axis is the actual class and the y-axis is the class the network chose. You can plot a heatmap of a matrix using imagesc(). Please include the confusion matrix along with some analysis i.e. what was the most common confusion? Visualize some examples of common confusions to see if you can see why the network would make that mistake. What number was easiest to identify? What number was the hardest to identify?

3. Let's try improving the network. There are three parameters you should tune: the number of layers, the size of those layers and the activations of those layers. The number and size of layers is specified by the array you pass to patternnet(). For example, patternnet([25,25]) will produce a network with two hidden layers with length 25. Activations can be changed by calling:

```
net.layers{layer_number}.transferFcn = 'new_activation';
```

The possible activations you have to chose from are:

```
compet - Competitive transfer function.
elliotsig - Elliot sigmoid transfer function.
hardlim - Positive hard limit transfer function.
hardlims - Symmetric hard limit transfer function.
logsig - Logarithmic sigmoid transfer function.
netinv - Inverse transfer function.
poslin - Positive linear transfer function.
purelin - Linear transfer function.
radbas - Radial basis transfer function.
radbasn - Radial basis normalized transfer function.
satlin - Positive saturating linear transfer function.
satlins - Symmetric saturating linear transfer function.
softmax - Soft max transfer function.
tansig - Symmetric sigmoid transfer function.
tribas - Triangular basis transfer function.
```

For example, the following would create a network with two hidden layers with length 25 and change the first activation function into a linear function:

```
net = patternnet([25,25]);
net.layers{1}.transferFcn = 'purelin';
```

Patternnet(10) should find $\tilde{6}0\%$ correct classification in the test set. You should be able to find $> 80\%$. Report your final favorite network you designed and your percent accuracy. Report any observations you make as you change your network.

Regression: Protein Decoy Detection

Now that you've mastered recognizing numbers, its time to move on to recognizing proteins. As we discussed in class, protein decoy detection is an essential task in protein structure prediction that involves predicting how close a computational design is to the experimental structure of that protein, without knowing what the experimental structure is. Along with this assignment, you will find two files: `decoy-train-features.txt` and `decoy-test-features.txt`. These files contain data on predictions submitted to the Critical Assessment of protein Structure Prediction in 2018 (CASP13). If you're interested in learning more about CASP, or downloading more data for yourself, see: www.predictioncenter.org

In the provided files, there are 9 columns. The first column is the name of the prediction. t#### indicates the target protein, ts### indicates the group who submitted the prediction and _# is which prediction attempt number it is. The training and testing data is divided so as to not split up targets. The next 7 columns are different decoy detection scores, all of which are based on different features. They include:

1. Packing: This is the O'Hern group score based on protein core packing.
2. VoroMQA: This score uses Voronoi tessellations as input features.
3. SBROD: This score considers the relative backbone orientation.
4. 3DCNN: This method divides up the 3D structure into voxels, which include atom type, and passes it through a Convolutional Neural Network.
5. ProQ2: This score considers a large number of features including sequence alignment.
6. ProQ3: ProQ3 expands on ProQ2 by adding Rosetta energy terms.
7. Protein Pro: This method runs a residual pooling network on sequence alignment data as well as interpolating the graph representation of the prediction.

The final column is the Global Distance Test (GDT). This is the value that defines how close the prediction is to the crystal structure. It ranges from 0 to 100, where 100 is a perfect backbone alignment. This is the value we want to predict from our features.

First, load the training and testing data features into arrays size 7xn and load GDT into arrays size 1xn. This time, let's use Matlab's built in feed-forward neural network for regression:

```
% Initialize the network
    net = feedforwardnet([layer_numbers]);
```

Train and test the model with the same commands as before. Additionally, the same commands can be used to modify the layer number, size and activations.

Assignment Part 2: Protein Decoy Detection

It's time to start learning! Prepare answers to the following questions in a document titled: `Assignment3-write-up-part2-LASTNAME-FIRSTNAME.txt`

1. Try different architectures for your network. What is the lowest mean-absolute-error you can achieve between predicted GDT and actual GDT in the test set? What was the architecture of your network? I found in my testing a mean absolute error in the test set of $\sim$13 GDT.

2. Plot and include a histogram of the test error. You should have 1,402 samples in your test set. How does it look? Why do you think it looks this way?

3. I'm sorry, but I lied to you... one of the input features I gave you is fake data. Can you figure out which one it is? How did you figure it out? A good data scientist does not necessarily need domain knowledge of the features, but examining your features before trying machine learning is a good practice. For example, how is the data distributed? Are there any correlations between your features? Are there any correlations between your features and the label?

4. Finally, compare the correlations between the input features and the GDT. How does it compare to the correlation between your predicted GDT and the actual GDT? What do you think this tells you about the CASP13 dataset?