Name Entity Recognition using Neural Networks

Data Preparation:

Data has been taken from AirBnB Homestays listing available on Kaggle. From the file, column "Description" is filtered out and then preprocessed to clean unknown characters and punctuations. Later, these sentences are converted into stream of tokens and passed to Standford NER function to label (Ground Truth) the dataset. Dataset prepared thus has total 114684 rows which is then divided into Train and Test data in 80:20 ratio.

Ground Truth:

Labels assigned by Stanford NER library are then given integer IDs as follows:
"0": "person", "1": "location", "2": "property", "3": "facility", "4": "organization", "5": "Misc"

Network Details:

After several input values are received, a many-to-one model creates one output value. The internal state with each input value is accumulated before a final value is generated. In this case, a stream of tokens is given as input to the model and a single output that is the NER Tag is returned. A many-to-one sequence model with word embedding layer is used for NER classification.

The output will be a probability vector having shape (1,6) as we have 6 classes. A pre-trained 50-dimensional GloVe embeddings is used, each word's GloVe representation is used, and average is taken. To get labels in suitable format for training a softmax classifier, lets convert Y from its current shape (m,1) into a "one-hot representation" (m,6). Thus, equations required during forward pass and cross-entropy loss are as follow:

$$z^{(i)} = W.\,avg^{(i)} + b$$

$$a^{(i)} = softmax(z^{(i)})$$

$$\mathcal{L}^{(i)} = -\sum_{k=0}^{n_y-1} Y_{oh,k}^{(i)} * log(a_k^{(i)})$$

Model parameters W and b are initialized using Xavier initialization. Optimizer function used to update parameters and minimize loss is Adam gradient descent. Adam optimizer helps in converging the loss with minimum epochs over the time.

```
v_w = v_w + dW**2
v_b = v_b + db**2

# Update parameters with Adam Gradient Descent
W = W - (learning_rate/np.sqrt(v_w + eps)) * dW
b = b - (learning_rate/np.sqrt(v_b + eps)) * db
```

Network Parameters:

Layers = 2 (Word embedding as hidden layer)

Activation Function = Softmax()

Learning Rate = 0.03

Epochs = 100

Window Size = 3 [-1, +1] (sliding window)

```
Training set:
Accuracy: 0.948516902944
Test set:
Accuracy: 0.947702749739
```

```
In [27]: X_my_sentences = np.array(["Queen", "Anne", "Hill", "is", "just", "few", "minutes", "from", "the", "Seattle", "center"])
         Y_my_labels = np.array([[5],[0],[5],[5],[5],[5],[5],[5],[5],[1],[5]])

         pred = predict(X_my_sentences, Y_my_labels , W, b, word_to_vec_map)
         print_predictions(X_my_sentences, pred)

         Accuracy: 0.818181818182

         Queen Misc
         Anne person
         Hill person
         is Misc
         just Misc
         few Misc
         minutes Misc
         from Misc
         the Misc
         Seattle Misc
         center Misc
```

Conclusion:

The model averages the word vectors in the window and forecasts the middle word label. Some tokens are incorrectly marked during pre-processing. The model usually works as the word sequence is the same as the trained data. A change in window-size and optimizer function may help model predictions.