CS689: Machine Learning - Fall 2019

Homework 5

Deep Chakraborty, 31262057

1. (*30 points*) Data Set Preparation:

a. (*10 pts*) To process the data into data cases used for training, I did the following steps:

1. Partition the users into train, val, test sets.
2. For each user, use a window length randomly chosen between 1 and 30 and slide it over the the sequence of observations (excluding timestamp and labels) with a stride of 1. The observations within each window form the examples $\mathbf{x}$.
3. Given the last observation's timestamp within each window, randomly pick the label set for an observation withing 60 minutes from this timestamp. The timestamp for this observation forms the forecast input timestamp t and the label set forms the target $\mathbf{y}$.
4. The examples $\mathbf{x}$ were rescaled between -1 and $+1$.

b. (*10 pts*) I partitioned the data into training, validation and test sets according to users. From the 26 users in the dataset, I used 20 users ($\sim 76\%$) for train, and 3 users ($\sim 12\%$) each for val and test. The reasons were:

1. To assess the generalizability of the model that learns from the users in the training set to the unseen users in the val/test sets.
2. To fine-tune hyperparameters for the models through cross-validation on the held out validation set of unseen users.
3. To report the final performance of the best models on the completely unseen test data.

Moreover, I decided to partition the data on the basis of users to **avoid data leakage** from validation and test sets during training.

c. (*10 pts*) I decided to use all the features in the dataset. This is to allow the deep networks that I trained to automatically discover any correlation between features and exploit that for the forecasting performance. However, to suppress the effect of the missing features, I first computed a binary mask from each time step to mark the missing and observed feature dimensions, and then imputed the missing features with the mean of that feature computed from the training set. These binary masks are input as an additional feature alongside the sensor readings. (more description about this in the modelling section).

2. (*30 points*) Modeling and Learning:

a. (*15 pts*) I model our multivariate time series with D-dimensional feature variables of length T as $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_T) \in \mathbb{R}^{TxD}$, where for each timestamp $t \in \{1, 2, ..., T\}$, $\mathbf{x}_t \in \mathbb{R}^D$ represents the observations at timestamp t, and x_t^d denotes the d^th feature value of $\mathbf{x}_t$.

Approach for dealing with missing data

Since my primary models are recurrent neural networks, I had to account for two problems: (a) Missing data imputation, and (b) Irregular time gaps between successive observations. Inspired by [1], I impute the missing values with the feature mean from the training set and suppress their effect by computing a binary masking vector marking feature presence/absence. To deal with the irregular time gaps, at each timestep, I compute the time elapsed since the last observation for each feature and feed this vector of *time intervals* along with *masking* vectors as additional features at each time step.
Concretely, let s_t be the timestamp at the t^{th} observation (assuming $s_1 = 0$). The *masking* vector $m_t \in \{0,1\}^D$ denotes which features are missing at time step t, and the time interval vector $\delta_t \in \mathbb{R}^D$ records the time elapsed for each feature X_t^d since its last observation. Thus we have:

$$m_t^d = \begin{cases} 1, & \text{if } x_t^d \text{ is observed} \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

$$\delta_t^d = \begin{cases} s_t - s_{t-1} + \delta_{t-1}^d, & t > 1, m_{t-1}^d = 0 \\ s_t - s_{t-1} & t > 1, m_{t-1}^d = 1 \\ 0, & t = 1 \end{cases} \tag{2}$$

The missing feature values x_t^d are imputed with the mean of the feature across the training set, $x_t^d = m_t^d x_t^d + (1 - m_t^d)\tilde{x}_t^d$ where $\tilde{x}_t^d$ is th mean calculated from observed features x_t^d across the training set.
The missing label observations at each time step, $\mathbf{L}_t = \{l_{1t}, l_{2t}, ..., l_{5t}\}$ where each label l_{kt} corresponds to one of the 5 activities are interpolated from the nearest observed activity label either backward or forward in time.
Here, we are interested in time series forecasting, where we predict labels $L_{nt} = \{l_{1_t}^{(n)}, l_{2_t}^{(n)}, ..., l_{5_t}^{(n)}\}$ at a timestamp t, , given dataset $\mathcal{D} = \{\mathbf{X}_n, \mathbf{\Delta}_n, \mathbf{M}_n, \mathbf{L}_n\}$, where $\mathbf{X}_n = [\mathbf{x}_1^{(n)}, ..., \mathbf{x}_{T_n}^{(n)}]$, $\mathbf{\Delta}_n = [\delta_1^{(n)}, ..., \delta_{T_n}^{(n)}]$, $\mathbf{M}_n = [\mathbf{m}_1^{(n)}, ..., \mathbf{m}_{T_n}^{(n)}]$ and $\mathbf{L}_{nt} = [l_{1_1}^{(n)}, ..., l_{5_{T_n}}^{(n)}]$.

Machine learning models

I tried 3 different models for the forecasting problem: (1) Logistic Regression, (2) Deep Averaging Neural Network, and (3) Gated Recurrent Neural Network. The details are as follows:

Logistic Regression: For establishing a baseline, I trained 5 different logistic regression models (with L1 regularization to facilitate automatic feature selection) for the 5 labels using training data $\mathcal{D} = \{\tilde{\mathbf{X}}_n, \tilde{\mathbf{\Delta}}_n, \tilde{\mathbf{M}}_n$ where $\sim$ denotes the values of each of the input features averaged over the sequence length, in order to convert the dataset into a fixed length feature vectors as required by logistic regression. Given the forecast timestamp t converted to a time interval vector $\delta_t^{(n)}$ which is concatenated to the averaged features to indicate the time at which the forecast should be made, these 5 models give the probability of 5 activity labels. Since the input is an average over the sequence length, this model can accept any length sequence at test time, and the forecast timestamp is encoded as a time interval vector since the last time stamp.

[1] https://www.nature.com/articles/s41598-018-24271-9

Deep Averaging Neural Network Inspired by [2], I use a multi layer fully connected neural network which takes as input $\{\tilde{\mathbf{X}}_n, \tilde{\mathbf{\Delta}}_n, \tilde{\mathbf{M}}_n$ which is again the input features averaged over the sequence length for any given example. The change in the inputs to this model compared to Logistic Regression includes passing the activity labels $\tilde{\mathbf{L}}_n\}$ at each input time step through an embedding layer that transforms the binary labels into dense vectors of size 32. These dense vectors are then flattened and concatenated with the rest of the features at each time step before taking the average over the sequence length. The neural network consists of 2 hidden layers of size (512, 128) and an output sigmoid layer of size 5. Again, given the forecast time interval vector $\delta_t^{(n)}$ which is concatenated to the averaged features, the output layer predicts the probability of each activity. Once again, since the input is an average over the sequence length, this model can accept any length sequence at test time, and the forecast timestamp is encoded as a time interval vector since the last time stamp.

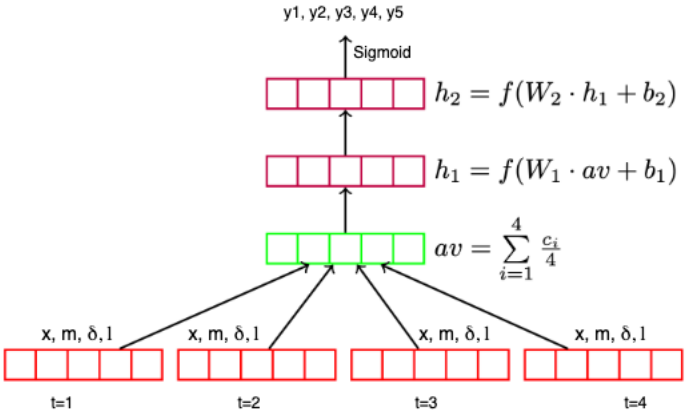


Figure 1: Deep Averaging Neural Network

Recurrent Neural Network with Gated Recurrent Units For my advanced model, I implemented an Encoder using a gated recurrent unit (GRU) RNN, as GRUs can accept any sequence length input and share parameters over all time steps making it the natural choice for our problem. GRU also has the advantage over RNN of persisting hidden state information over long sequences and avoiding vanishing gradients. Our encoder (2 layer GRU with 128 hidden dimension) takes as input $\{\mathbf{X}_n, \mathbf{\Delta}_n, \mathbf{M}_n\}$ and $\mathbf{L}_n$ passed through an embedding layer and concatenated with the features at each time step. The output of the GRU at the last time step (encoding information across the sequence) is then concatenated with $\delta_t^{(n)}$ (the forecast time interval vector) and given to a dense layer (size 32) to predict the probability of each of the 5 activities at forecast timestamp t through a 5-node sigmoid output layer. At training time, I batched the varying length sequences into buckets of size 1-30 to avoid padding the inputs. At test time, the RNN can naturally handle a sequence of any length and forecast horizon is determined by the time interval vector.

b. (*15 pts*) Learning procedure:

1. **Logistic Regression:** I used the sklearn implementation with L1 penalty, learned using L-BFGS optimizer.

[2] https://www.aclweb.org/anthology/P15-1162/

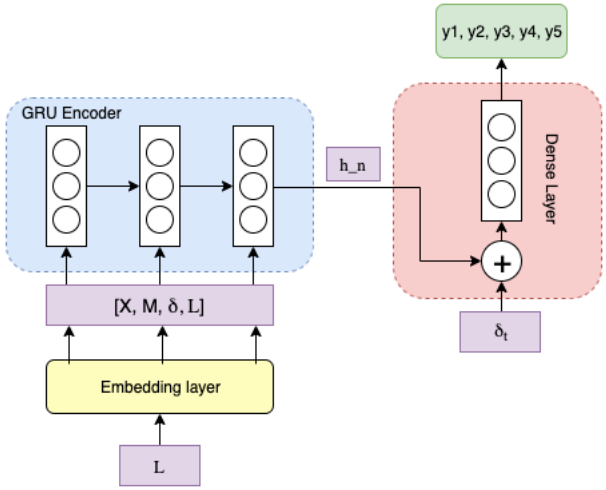


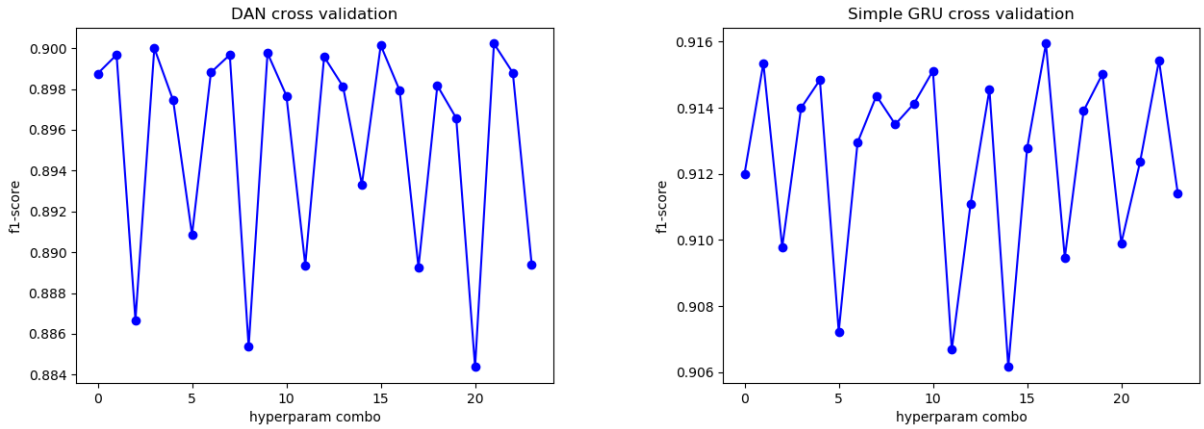
Figure 2: Gated Recurrent Neural Network

2. **Deep Averaging Neural Net:** I used pytorch nn.Module to implement the model. RRM optimization framework was used to learn the model using the Adam optimizer with learning rate 0.0001 and weight decay 0.0001 for 5 epochs, and binary cross entropy loss averaged over all 5 labels and data cases. Convergence was asserted using early stopping as validation F1 score stopped improving.
3. **Gated Recurrent Neural Net:** I used pytorch nn.Module with GRU and Embdedding layers to implement the model. RRM optimization framework was used to learn the model using the Adam optimizer with learning rate 0.001 and weight decay 0.0001 for 3 epochs, and binary cross entropy loss averaged over all 5 labels and data cases. Convergence was asserted using early stopping as validation F1 score stopped improving.

3. (*35 points*) **Experiment and Results:** In this question, you will describe your experiments and results.

a. (*10 pts*) I used F1 score on the held-out validation set as the criteria for model selection, since the ratio of positive to negative labels in the dataset is very skewed. First, I found the optimal hyperparameters for each model using a grid search over the different hyperparams (Such as embedding size, hidden size of GRU cells and no. of recurrent layers, number of hidden units and fully connected layers, learning rate and dropout) and training each model using the above data and applicable hyperparams. The model with the best validation F1 score is chosen as the best model from each approach.
The total number of examples in my dataset is: (train:75986, val:14587, test:6784). For the neural network models, I used a custom DataLoader and sampler that bucketed the varying length input sequence of observations into buckets of sequence length 1-30 (to reflect the test time conditions) and generated batches of size 64 from these buckets. This was done in order to prevent the need for padding variable length inputs to the RNN. Notion of generalization in my model is how well forecasts are made for observations of unseen users in the held-out validation set.

b. (*10 pts*) For the neural network models, F1 scores on validation set during hyperparameter grid search are given in the following figures.
The best validation results are summarized in the following table:



(a) (b)

Figure 3: (a) grid search on Deep Averaging Network, (b) grid search on GRU RNN.

Model	LYING_DOWN	SITTING	FIX_walking	TALKING	OR_standing	Mean F1 score
LR	0.6992	0.5190	0.9315	0.9219	0.8919	0.7927
DAN	0.9544	0.8040	0.9319	0.9417	0.9105	0.9085
GRU	0.9538	0.8228	0.9368	0.9465	0.9123	**0.9145**

Table 1: Validation F1 scores for different labels using different models

Using the best model for GRU on the test set, the mean F1 score was 0.9145.
c. (*15 pts*) Code.

4. (*5 points*) Recurrent neural networks outperformed all fixed input size based models such as fully connected nets and logistic regression. However, even the best performing model didn't do too much better than naive guessing of label from the last observation. My initial best model was densely searching for forecast labels in the next 60 mins, and therefore it took quite a bit of memory, and i had to solve it by randomly picking a label in the next 60 mins. If I had more time, I would try an attention based encoder-decoder GRU model.