**First Question**

Let us suppose there are c categories and categories have p1, p2 ….,pc products

Since the purchases by customer across categories is independent, we will model the probability of consumer purchases in each category c as individual binomial distribution separately. We will have one binomial distribution for each of the categories.

So, for  *P*(*Iict* =1) we will have c binomial probability distribution

Now since the consumer can buy only one product within each of the categories and there are multiple products with in each category, we will model this as multinomial distribution. We will have one such multinomial distribution for each of the category.

So, for  *P*(*Cit* =*j*|*Iict* =1) we will again have c multinomial distribution.

We can use normal distribution to model the amount of product j brought *P*(*Qijt* |*Iict* =1⋀*Cit* =*j*). From economics, we choose ln(Q) as the output variable to predict rather than Q. If the item can be brought in integer units, we can also model this as Poisson distribution. Poisson distribution is used to model counts of rare events. This holds true for customer purchasing at a shopping mall as the customer will only buy small number of products out of the large number of products available to buy. Also, in Poisson regression we have link function g(y) = ln(y). Since, in our case we are dealing integer units, using Poisson regression is advisable

The factors influencing the purchase of category c are:

1. Seasonal factors: Our purchases are driven by what time of year we are purchasing . We buy more ice-creams during summers, purchase warm just before the winter begins, we buy more candies, sweets, chocolates, snacks during Christmas, Halloween or other major festivals. People tend to purchase differently even within a month. We buy differently during the start of the month than at the middle of the month and differently again at the end of month (salaried customers have more expendable income at the beginning of the month. Also we shop differently during weekends and weekdays.
2. Apart from advertisements, any other promotions such as discount, free gifts offered on the product
3. Where’s the store located. The buying pattern changes as per the region customer lives in . If the store is located in upscale region of the city, more people are like to buy more luxurious categories such as branded watches.
4. Competition in the product space. One way to characterize competition is competitor’s product price in the same category. While building the model, you can take ratio of your price to the competitor price.
5. Where in the store is the item placed. Products placed on the shelfs which are at eye level will tend to have more visibility and hence more sales. Also, alongside which other category, product was placed. Breads should be placed near jams or vice versa.
6. Numeric and weighted distribution of the product. Is my product present in most of the relevant stores, so the chances of product being sold are more.
7. How many different sizes product is available in. Some users may buy bigger size and some small. If there are multiple sizes, we can cater diverse consumer base
8. Macro-economic factors such as GDP, CPI inflation, population. This gives an idea about how much expendable income every consumer has.
9. The customer related attributes such as income group, age group, gender,

**Model Selection**

While selecting modeling technique to build the regressor, we need to consider the following aspects:

1. Approximates the training data as closely as possible.
2. Generalizable for unseen future data (Avoids overfitting).
3. Has good runtime performance i.e. it takes less time to run inference and occupies small disk size
4. The aim of modeling exercise. If the sole objective of model building is to predict the output where machine learning model act as black box, then we try to fit a model which gives the best possible performance on test data under given cost constraints. However, if the objective of model building is not just predicting the outcome but also interpretability and determining which features have an impact on the output, then we select model which is interpretable and less complex

Based on elementary data exploration, in order to have good performance on training as well as test data the model we build needs to tackle:

1. Data which is time dependent
2. Plausibility of non-linear interactions between the features and output variable and also the correlation between output variables (propensity of customer buying an item) since the behavior of consumer shopping varies with time.
3. Missing data. There are few cases where the product is not purchased by any of the customer.
4. We will need to impute the data for such cases.

**Model Selection**

The training data in prediction problem is panel data. We have 2000 unique customers and for each there are 40 unique product categories. We can either have every single customer as one individual time series or we take a combination of customer and product categories as one single individual i.e. we will have 2000\*40 = 80,000 individual time series.

In case of considering one customer as a single single individual, we would be having multiple observations (as there are multiple product categories) for every time step in the time series If we are using panel data-based models, they can handle multi observation per time point in.

However, we are going to use LSTM based neural networks, which will need only one observation per time step, so we are going to use combination of customer and product categories to segregate data into individual time series. So we will have one time series for each combination of customer and product id.

Even for LSTM We can treat every customer I as one single individual leading to one time series for every customer. For LSTM, we can’t have multiple rows for same time period for same individual. So will need to one hot encode the product categories and model output will be multidimensional. This is problematic as model can’t scale to new product categories. This is in exact opposite to our motivation of using LSTM autoencoder based prediction network.

Only if the customer *i* buys a product *j* during week *w,* we will have the corresponding row in the data.Looking at the data it is very clear that a particular customer does not buy product during all the weeks as there will be no row if the customer does not purchase product during a particular week. So as first step we add a new column “is\_purchased” to the data and setting it equal to 1. This column indicates whether customer *i* has purchased product *j* or not during week *w*.

Now we need to have one row for each customer *i* buying each product *j* during each week *w.* In other words, we need one for every unique combination of customer, product and week. So, we need data with 2000\*40\*49 = 3,920,000 rows. We do these by first creating dataframe for all possible combination.

Once, we have this data-frame, we the merge with the previous data to get the values of price, advertised, *is\_purchased* columns. Result of the merge operation will give you *is\_purchased* columns with ‘Nan’ values if the customer does not buy that product during that week. So, for all ‘Nan’ values we set them equal to zero.

We can either proceed by building standard classification model choosing either a random forest or support vector machine based algorithm and not taking into time series nature of the data. However, since the individual observations are no longer independent, such classifiers we will not give the correct picture and it does not capture information about the impact of previous purchases.

So, we need to model the output as set of two types of features. One set of features capture the time invariant impact such as price, advertisement etc. and other set of features which capture the dependency between the current output and previous time period outputs.

*X* = features whose impact is time invariant e.g. price, etc.

*:* H is function which takes data from previous time steps and extracts features capturing the dependency of current output on previous data

One option to capture the impact of previous purchase (or no purchases) could be using lag variables for *is\_purchased* and use random-forest or svm classifier. However, using lag variables as a linear combination with time invariant, is oversimplification and does not capture the all the complex non interaction between the current purchase and previous purchases.

One point to note here is that Yt refers to observation from single time series. In our case, we have multiple such time series. We need to build separate models for each time series as the model built on one time series will not generalize to other time series. This is infeasible from both training and inference production pipeline perspective, given that we have 80000 time series (2000 customers \* 40 products) and majority of them would be sparse as the customer purchases only few of the categories in a given week.

So, we need an approach which can take the sequential nature of the output variable into account and captures the underlying non-linear interactions and at the same time can generalize across different time series i.e. different customers and different products. LSTM networks are exactly the kind of solution we are looking. i.e. LSTM networks learns *H(Yt-1, Yt-2, …, Yt-n)*  or extracts features from the previous output*.* LSTM is our feature extractor.

But how de use LSTM network as feature extractor.

We use model architecture as given the paper by Uber for forecasting the daily rides in different cities. The model architecture has the following advantages:

1. Efficiently extracts features from outputs of previous time steps; thereby increasing the model accuracy.
2. Is the model generazible to new time series i.e. new customers/products. We just have to provide customer and product specific attributes as input to the prediction module

In this paper, they represent model architecture consisting of:

1). LSTM based autoencoder for extracting useful and representative embeddings from sequence of previous time steps which can acts features for prediction of current time output. The idea is to train an encoder that extracts the representative features from a time series, in the sense that a decoder can reconstruct the input time series from the encoded space. At test time, the quality of encoding of each sample will provide insight on how close it is to the training set.

2) The encoded output is then concatenated with features from current time step and then pass through predictor module which is a series of dense layers.

**How the overall model works?**

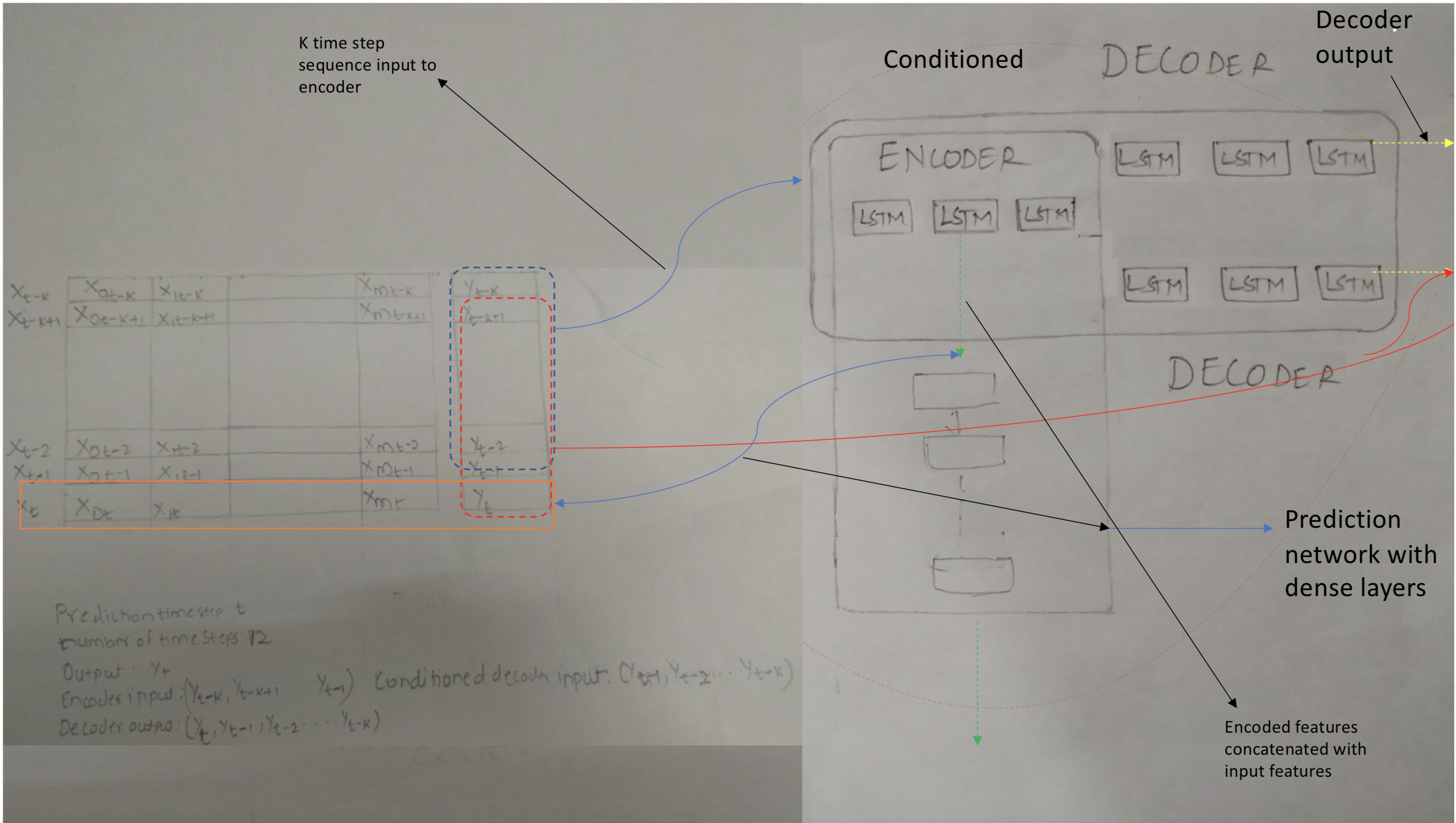
Assume we time series with m input features. Xt = [X1t, X2t …, Xmt] is the input row and Yt is the 1-d output at time point t. At time step, the input to the autoencoder is sequence input and output of previous k time steps ({Xt-1, Xt-2 …., Xt-k}, (Yt-1, Yt-2 ….Yt-k)). Essentially the input is of the (1, k, 1) where 1 stands for one row of observation, k denotes number of time steps in the sequence, 1 is the number of features in the sequence.

During training, we chose decoder output to be to one time step head({Xt, Xt-1 ….Xt-k+1 }, { Yt, Yt-1 ….Yt-k-1 }) rather than using the same encoder input sequence of steps ({Xt-1, Xt-2 ….Xt-k}, (Yt-1, Yt-2 ….Yt-k)). In order to predict the next few frames correctly, the model needs information about which objects and background are present and how they are moving so that the motion can be extrapolated. The hidden state coming out from the encoder will try to capture this information. Therefore, this state can be seen as a representation of the input sequence. On the other hand, if the output is same as input, the encoder might learn an identity function which does not extract useful features for the prediction network.

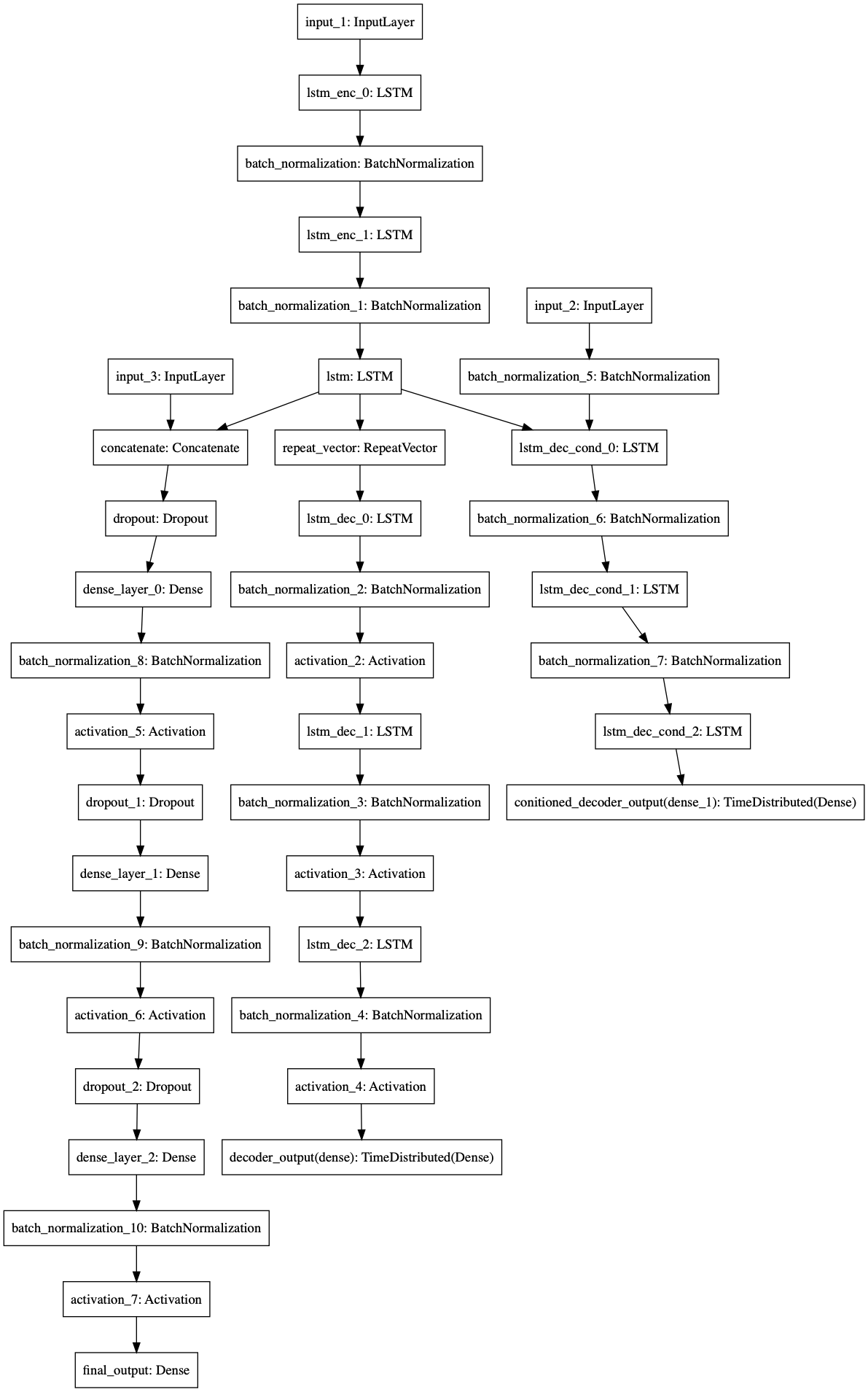
The paper also suggests that we conditioned the decoder by providing the input to decoder network which is same as the output except it’s 1 time step behind the decoder output i.e. the decoder output at time is same as decoder input at time t+1.

However, the problem with autoencoder which outputs k time step ahead of input and is conditioned is that it might be biased towards the short term correlation between the output and past few time steps just before output. The decoder conditioned on previous few time steps will easily pick these correlations and hence the gradient will quickly become zero, thereby preventing it to learn long term dependencies.

To overcome the challenges with both the approaches we will use a composite model as shown below:



Block diagram of our model architecture:



This composite model tries to overcome the shortcomings that each model suffers on its own.

For our case we have k = 12, based on the data exploration and under the assumption that seasonality in customer purchase is of 3 months period.

Other option would be using panel data regression (using either fixed, random or fixed effect). However, this is not regression problem but classification problem.

**Data preparation and Feature Engineering:**

Preparing data is equally important in any modelling exercise. Looking at the data, we have observations only if the customer purchases. If for a given week and product, there’s no observation it is implicit that the customer has not purchased the specific product during that specific week. We need to explicitly have a column which indicates that the customer has purchased a item in a particular or not.

So, after loading the given file “train.csv”. Let’s, call this as ‘*data’* we do the following steps:

1. We then add new column ‘is\_purchased’ to this dataframe and assign every row for this column a value of 1. This indicates the customer has purchased this item.
2. First create a dataframe with columns *‘Customer\_id’, ‘Product\_id’, ‘week\_no’* such that all possible combination of customers, product and week are captured. The resulting dataframe will have 2000\*40\*49 = 3,920,000 rows. Let’s call this as ‘data\_unstacked’
3. Then we merge the big data frame ‘data\_unsatcked’ with the initial dataframe ‘data’ to get price and advertised column.
4. Then we merge the ‘data\_unstacked’ dataframe to get ‘is\_purchased’ column from the ‘data’ data frame
5. We assign missing values of is\_purchased as 0 because the reason this value is missing is that customer has indeed not made the purchase for that item and during that week.
6. We also assign missing values of advertised column as 0, using the above reasoning.
7. For price, we impute the missing values of price based on the price of that product in the week before.

Although we have mixture of numerical variables and categorical variables as predictor, there’s no need to do any scaling because we have Batch Normalization layer in the network which automatically scales the batch of data.

**Feature Engineering:**

For many non-deep learning methods, we extract features by binning, taking ratios and doing PCA, kernel transformations etc. and then feed the resulting features to machine learning model for training.

In our case, we are relying on our LSTM auto encoder to extract complex features and pass the encoded cell states as feature to our forecaster/classifier (forecaster/classifer is also a neural network).

**Overfitting:**

I held out the data for week No. 47 (48th week) as validation data and data for week no. 48(49th week) as test set to do our final performance check.

To ensure that our model does not over fit and has good performance on future data, we use Batch Normalization in LSTM layers and Batch Normalization and dropout in dense layess in the predictor module.

While using Batch Normalization we need to ensure we have batch normalization layer before activation or else the impact of batch normalization is not there.

**Future work**

As I spent majority of the time debugging a trivial problem, I was not able to do Hyperparameter tuning for the model architecture. To improve the model performance, we need to do Hyperparameter tuning on:

1. Number of time steps in the lstm sequence. We have currently chosen LSTM sequence of 12 on basis of elementary data exploration and intuition. For optimal model performance we need to try with different values of time steps
2. Learning rate
3. Number of neurons and number of LSTM and dense layers
4. Number of neurons in LSTM and dense layers
5. Dropout value. But the model performance is fairly robust on

The future work would involve having an error estimate for every point rather than an overall error estimate. This could help us to decide which predictions to discard and which to use.

Another important area of improvement is data generator which generates batch wise data for the model to train on, needs to be implemented in multiprocessing mode. Current implementation of fit\_generator method allows for multiprocessing for data generation but the chances are that we process the same batch of data at every step of an epoch since the data is generated simultaneously and we do not have any indexing information. Solution to this is using sequence data generator which overcomes this problem.