Dataset:

Japan wiki, has a spike; grouped by time window; 5 – 10 – 15 min ; with 10min used for tuning

(t, t – 1, … t- n) to predict (t + 2)

The prediction is done as we close window t, therefore if we predict moment t+1 and most requests are in the beginning of the interval the system would not have time to react.

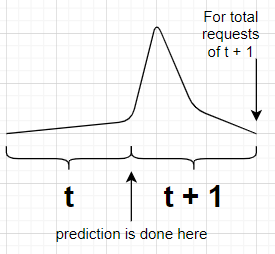


Illustration of scenario when not leaving buffer window

Baseline

1.The naïve approach: predict that the workload won’t change, so nr requests at moment t+2 will be nr requests at moment t.

See reports/baseline – for plots and measurements

2. A classic approach: ARIMA

See arima/params for choice of params

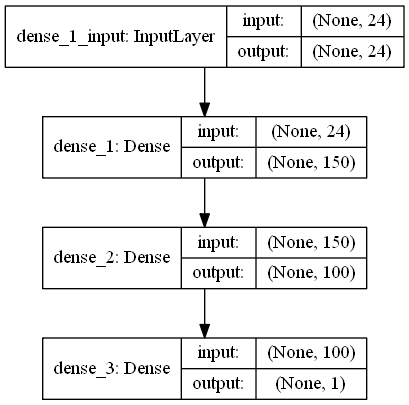
See arima/centralized for results

Parameter tuning

Methodology : This is the validation phase ; so we use K-Fold cross validation and average the results; K-Fold splits the dataset into k parts, performs training on k-1 and validation on the one left out; we use 3 folds for this exp

1. MLP –
   1. Set the baseline mlp model

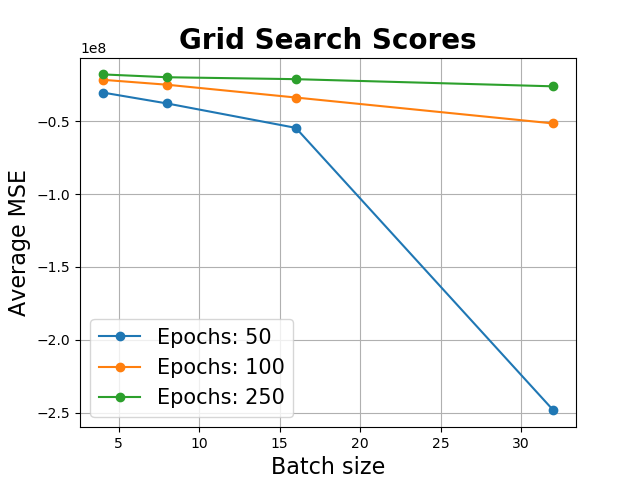
After manually trying some configurations, set a baseline MLP model, with n = 24



* 1. Epoch + batch size

Performed a grid search for selecting an optimal epoch no and batch size

Batch size should ideally be a power of 2 (to fit into GPU architectures, some exp were ran on google colab which offers this feature). Lower batch size is more accurate but higher gives more training speed. As expected the best MSE is obtained for the lowest batch\_size(4) however it does not drop significantly at 8, regardless of epochs no. The selection of epoch no is again a tradeoff between speed and acc. We see a smaller no of epochs(50) performs poorly, while the difference between 100 and 250 is not that great, meaning that we can get a good approximation of a model using a batch size of 100.



* 1. Optimizer

Best: -19291753.008883 using {'batch\_size': 8, 'epochs': 100, 'optimizer': 'Adadelta'}

-24734800.666075 (3219248.710688) with: {'batch\_size': 8, 'epochs': 100, 'optimizer': 'RMSprop'}

-190604349.171842 (36507454.158356) with: {'batch\_size': 8, 'epochs': 100, 'optimizer': 'Adagrad'}

-19291753.008883 (3536821.212706) with: {'batch\_size': 8, 'epochs': 100, 'optimizer': 'Adadelta'}

-25828119.384562 (2841690.892304) with: {'batch\_size': 8, 'epochs': 100, 'optimizer': 'Adam'}

-29578706.323891 (1428493.494790) with: {'batch\_size': 8, 'epochs': 100, 'optimizer': 'Adamax'}

-20400557.558779 (4422621.888017) with: {'batch\_size': 8, 'epochs': 100, 'optimizer': 'Nadam'}

1.4. activation function

Best: -19571788.407985 using {'activation': 'relu', 'batch\_size': 8, 'epochs': 100}

-4875804739.406558 (225550845.887195) with: {'activation': 'softmax', 'batch\_size': 8, 'epochs': 100}

-20314197.128252 (3515510.240613) with: {'activation': 'softplus', 'batch\_size': 8, 'epochs': 100}

-4034049993.191782 (206573705.233292) with: {'activation': 'softsign', 'batch\_size': 8, 'epochs': 100}

-19571788.407985 (3139867.544099) with: {'activation': 'relu', 'batch\_size': 8, 'epochs': 100}

-4175933232.044314 (214993464.805943) with: {'activation': 'tanh', 'batch\_size': 8, 'epochs': 100}

-3055609656.033742 (195436185.770978) with: {'activation': 'sigmoid', 'batch\_size': 8, 'epochs': 100}

-4584848514.738154 (216786308.421556) with: {'activation': 'hard\_sigmoid', 'batch\_size': 8, 'epochs': 100}

-20661311.356256 (3845006.383268) with: {'activation': 'linear', 'batch\_size': 8, 'epochs': 100}

1.5. Topology

1.5.1 Dropout layers

4,150-100,10594400.008,2492.349,4.499

8,150-100,10475304.863,2315.771,3.831

16,150-100,12986216.805,2355.314,3.268

24,150-100,12895060.549,2535.667,3.682

32,150-100,14818853.305,2803.265,4.380

Generally, we only need to implement regularization when our network is at risk of overfitting. This can happen if a network is too big, if you train for too long, or if you don’t have enough data. – does not really improve in my case

TODO

Final comparison:

Prediction interval 5, 10, 15

MSE, MAE, MAPE

MLP, CNN, CNN\_LSTM, ARIMA, BASELINE

Prediction overhead comparison :

Ts arima vs dl model