Giacomo Romanini
Mid-Term Exam Intelligent Systems

In a big city, congestion happens in the morning or evening. As a result, the duration of travel to or from work is extended and still varies by about 30%. It can be observed that in the rain, congestion increases (travel time is on average X% longer) and in the event of snow, it increases even longer (travel time on average Y% longer). We would like to predict such situations each mornin one-two ~~hours~~ hour before the congestion. What kind of intelligent technology and how could it be used here?

1. The problem consists in calculating the estimated average travel time stretching in one or two hours from the current moment. This value changes based on the starting time and the weather, while other parameters, such as location or transpor mean are not considered or can be ignored. This also means that we cannot rely on data of the current state of the traffic, but only on historic data.

Supposing the given values (30%, X%, Y%) are averages, in order to get a more precise solution (compared to a probabilistic calculation), there is the necessity to calculate the correlation between the intensity of the causes and their effects.

In particular:
1. How much is time affected from the starting time? wich kind of distribution does the probability follow?
2. How much does the intensity of the rain or snow impact the final time?

Without the data used to calculate the given statistics, we can either create a very simple probabilistic formula (estimating the probability of these events as gaussian distributions or similar), or create our own model which will train and adjust as the time passes, whose efficiency can be ~~either~~ then checked by its correlation to the given values.
As the problem is about calculating estimated time, and not weather forecasting, we are assuming that weather type, intensity and timing are already given as input information.

Tasks:

1. If possible: require the data used to get the statistical values. This would give us the distribution of the probabilities. as well as same initial data to train our algorithm on.

2. Identify the best model
As a prediction model, it ~~need~~ will need to have a strong final correlation to the given percentages. There are no Temporal dependencies that would affect our output, so our model won't need memory.

3. Create the model
Create the matematical model and code it accordingly

4. Retrieve the data.
Retrieve any past data or start collecting and recording new one. The data required is a lot, so it could be needed to use various gathering strategies.

5. Train the model
Based on the acquired data, train the model to adjust its weights and give the most accurate solution

~~6.~~ 6. Test the model
Use part of the data, as well as the given percentages to check the model accuracy.

2. We don't require any long or short term memory, as there ~~are~~ is no indication in the formulation of the problem, so for ~~this reason~~ the same inputs we are expecting the same output. No internal state is required. For this reason I would use a Feed forward Neural Network for our prediction.

Some of the most popular FNN algorithms used for predictions are:

1. Backpropagation (Gradient Descent):
Back propagation is the fundamental algorithm for training neural networks. It uses gradient descent to minimize the error between predicted and actual outputs by adjusting the weights of the networks.
As show by:
Tsong-Lin Lee,
Back-propagation neural network for long term tidal predictions
Ocean Engineering, Volume 31, Issue 2, 2004
and
Tsong-Lin Lee,
Back-propagation neural network for ~~short~~ the prediction of the short-term storm surge in Taichung harbor, Taiwan
Engineering Applications of Artificial Intelligence, Volume 21, Issue 1, 2008

backpropagation has a great scalability factor both in terms of size of data processed than in applications.

2. Mini-Batch Gradient Descent
Mini-batch gradient descent combines the efficency of SGD (Stochastic Gradient Descent) with the stability of batch gradient descent by updating the model parameters using a small batch of training samples.
Pro: balances efficiency and stability of training
Cons: Requires a large number of data.

As shown by Wu Lizhen, Zhao Yifan, Wang Gang, Hao Xiaohong, in A novel short-term load forecasting method based on mini-batch stochastic gradient descent regression model,
Electric Power Systems Research, Volume 211, 2022
MBGD can be used to elaborate a short term prediction based on data which have input and output size similar to ours (5 parameters in input and 1 in output), but still requires a lot of data.

3. Levenberg-Marquardt:
Levenberg-Marquardt is a specialized optimization algorithm commonly used for training neural networks with a small to medium number of parameters. It combines aspects of gradient descent and the Gauss-Newton method
In Sadig Mammadli,
Financial time series prediction using artificial neural network based on Levenberg-Marquardt algorithm,
Procedia Computer Science, Volume 120, 2017,
It is shown how this algorithm can be applied for a prediction strategy, with a limited number of input parameters and few output results

3. Let's review our necessities:
We have three main inputs: time, probability of rain, and probability of snow. We can assume that both of them have a limited number of parameters, so the numeric inputs will be a limited number.
We can also assume that time will probably have a Gaussian-like distribution effect on the output, as there will be more people leaving around the critical hours, as well as some of the numeric parameters of the weather inputs (e.g. snow intensity)

Also, we don't know how our training data will be composed or retrieved.

With these considerations, my strategy choice is the Levenberg-Marquardt algorithm.

The back propagation (Gradient Descent) algorithm is the least efficient of the three, and its performance can be further degraded by a large dataset.
The MBGD algorithm, while converging faster than the normal Gradient Descent, relies heavily on mini-batch sets of data, which, while it's true that the focus is on a specific time period, introduces the probability of biased results, as there will probably also be more data collected during those critical hours.
The LM Algorithm is highly related to the gaussian model, which has a strong correlation with our inputs, is efficent with a small number of input parameters and relies on the entire dataset for its training.

normalized from 0 to 1440

4 Inputs

1. Time of predicted leaving (from 0 to 1440, indicating the minute of leaving, ~~we could normalize it from 0 to 1~~)

2. Probability of rain at that time (From 0 to 1, where 1 is 100%)

3. Eventual intensity of rain at that time (from 0 to 1, where 1 is the maximum)

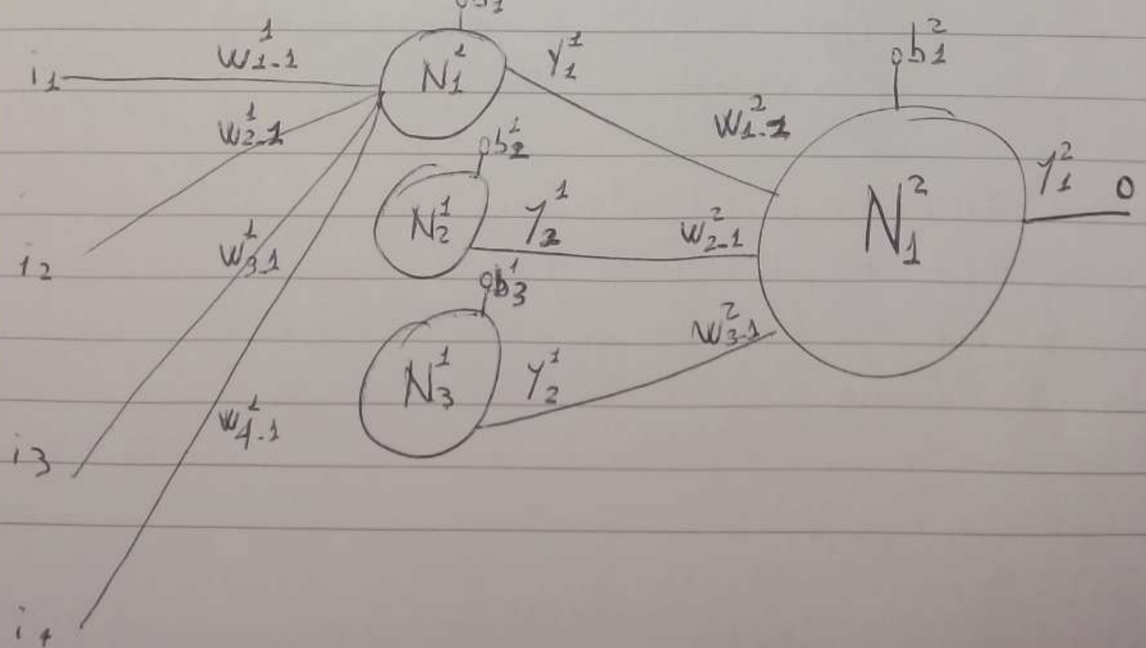
4. Temperature at that time (from 0 to 1, where 0 is -20 and 1 is 40 degrees celcius)

Output

1. Percentage of how long the estimated time would increase (From 0 to 1, where 1 is 100%)

Having one ~~input~~ output and few inputs I think one hidden Layer can be enought, adding layers would just consist in repeating the same process for each additional layer.

Imaging a network with a three-neuron hidden layer:



Assuming a linear activation function for the output layer and a Gaussian activation function for the ~~input lay~~ hidden layer the output formula would be:

$$o = w^2_{1-1} \cdot y^1_1 + w^2_{2-1} \cdot y^1_2 + w^2_{3-1} \cdot y^1_3 + b^2_1$$

where, for each hidden layer's neuron N

$$y^1_N = e^{-\left(\left(w^1_{1-N} \cdot i_1 + w^1_{2-N} \cdot i_2 + w^1_{3-N} \cdot i_3 + w^1_{4-N} \cdot i_4 + b^1_N\right)^2\right)}$$

Assuming we have the data to train the algorithm.

the training would go as follow, in MATLAB code:

~~for iterations~~
~~calculate outputs as described before~~
~~delta = ... e - y where y is the desired output~~

With
w1 = matrix of weights in the first layer, including biases on the first row
W2 = " second layer, "
X = inputs, with the first row set to 1 to fit for biases
Y = desired output

```
net = fitnet(3,'trainlm');
net.layers{1}.transferFcn = 'gaussian';

net.train(net, X, Y)
```