AI Lab - Machine Learning, DNN for regression

Alessandro Farinelli

Thanks to Alberto Castellini, Davide Corsi, Luca Marzari and Celeste Veronese for help with slides and code

University of Verona
Department of Computer Science

May 2024

Start Your Working Environment

Update your repository to download the new lesson

Important: do a backup copy of your working directory to make sure you avoid any issue

```
> cd AI_Lab
> git commit -a -m "a message describing the commit"
> git pull
> conda activate ai-lab
> conda install scikit-learn pandas seaborn keras tensorflow "<--IMPORTANT"
> jupyter notebook
```

To open the assignment navigate with your browser to: ML/ML_DNN_2_problem.ipynb

Dataset Description

Impact of traffic and meteorological values (temperature, wind) on air pollution (NO2).

- 500 observations (rows), sub-sample of data collected by Norwegian Public Roads Administration.
- Between October 2001 and August 2003, in Alnabru, Oslo, Norway.
- Response (column 1), hourly values of logarithm of NO2 concentration (particles) [lno2]
- Predictors (column 2 to 8):
    1. logarithm of number of cars per hour [lc]
    2. temperature 2 meters above ground (degree C) [t2]
    3. wind speed (meters/second) [ws]
    4. temperature difference between 25 and 2 meters above ground level (degree C) [td25]
    5. wind direction (degrees between 0 and 360) [wd]
    6. hours of day [hd]
    7. number of days (starting from October 1, 2001) [nd]
- Available from `http://lib.stat.cmu.edu/datasets/`, submitted by Magne Aldrin (`magne.aldrin@nr.no`). [28/Jul/04]

Dataset available in ML/NO2.csv

lno2	lc	t2	ws	td25	wd	hd	dn
3.71844	7.6912	9.2	4.8	-0.1	74.4	20	600
3.10009	7.69894	6.4	3.5	-0.3	56	14	196
3.31419	4.81218	-3.7	0.9	-0.1	281.3	4	513
4.38826	6.95177	-7.2	1.7	1.2	74	23	143
4.3464	7.51806	-1.3	2.6	-0.1	65	11	115
4.16044	7.67183	2.6	1.6	0.3	224.2	19	527
4.01277	5.52545	-7.9	1.6	0.3	211.9	5	502
2.15176	4.68213	-4.1	3.8	-0.1	63.1	4	453
3.157	7.15618	-12.7	5.2	-0.1	64.5	12	462
2.37955	4.74493	-1.6	3	0.4	58.3	3	554

Useful libraries/API for DNN

- Keras `https://keras.io/`
  - a high-level neural networks API written in Python
  - capable of running on top of TensorFlow and other libraries
  - supports convolutional and recurrent NN
  - run seamlessly on CPU and GPU
  - great for fast prototyping
- TensorFlow `https://www.tensorflow.org/`
  - an end-to-end open source platform for machine learning
  - comprehensive, flexible ecosystem of tools, libraries and community resources
  - A tool for easily build and deploy ML powered applications
- PyTorch `https://pytorch.org/`
  - PyTorch is an open source machine learning framework that accelerates the path from research prototyping to production deployment.
  - Great for deep customization and hence research
- Colab `https://colab.research.google.com/`
  - free Jupyter notebook environment that requires no setup and runs entirely in the cloud.

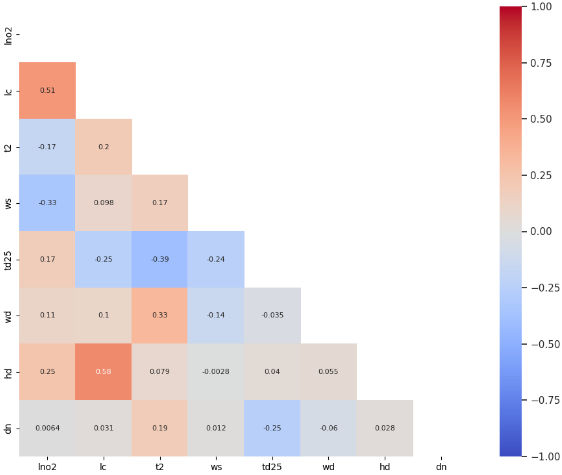
Steps

- consider the datase on NO2, infer concentration of NO2 using other features as predictors [Regression]
- use a DNN
  1. load and visualize data
  2. standardize data
  3. divide dataset in train, test, validation
  4. create the DNN model
  5. train the model
  6. test the DNN model
  7. evaluate the trained model (loss, accuracy, RMSE, NRMSE, absolute error)

Loading and visualizing data

```
# Load data
df = pd.read_csv('NO2.csv', index_col=False)

# Descriptive statistics summary
df.describe()

# Correlation matrix
corrmat = df.corr()

# Generate a mask for the upper triangle
matrix = np.triu(corrmat)
f, ax = plt.subplots(figsize=(12, 9))
sns.set(font_scale=1)
sns.heatmap(corrmat, vmin=-1, vmax=1, center= 0, square=True, annot=True,
 annot_kws={'size': 8}, mask=matrix, fmt='.2g', cmap= 'coolwarm')

plt.show()
```

Correlation matrix



Standardize data

```
# Standardizing data
sc= MinMaxScaler(feature_range=(-1,1))

for var in features:
    if(var != 'lno2'):
        df[var] = sc.fit_transform(df[var].values.reshape(-1, 1))

#NumPy representation of the data frame (removing labels)
df = df.to_numpy() #df=df.values
```

Divide dataset in train, test, validation

```
X = ... #all rows, column 1 to 7 (features 2 to 8), insert code here
y = ... #all rows, first column

seed = 7
np.random.seed(seed)

# split dataset in 75% for traininig and 25% for testing (500 -> 375,125)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,
 random_state=seed)

# split training in 70% for traininig and 30% for validating (375 -> 300,75)
.... #insert code here
```

Create the DNN model

```
# create model
model = Sequential()
model.add(Dense(10, input_dim=X_train.shape[1], activation='relu'))
model.add(Dense(30, activation='relu'))
model.add(Dense(40, activation='relu'))
model.add(Dense(1))
# Compile model
model.compile(optimizer ='adam', loss = 'mean_squared_error', metrics =[metrics.m
```

Generating a sequential model

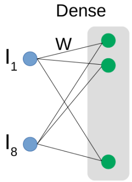
```
# create model
model = Sequential()
```

- Sequential model: linear stack of layer
- it can be created by:
  1. passing a list of layer instances to the constructor
  2. adding layers to the model, after the creation, using the .add() method

Adding the input layer

```
model.add(Dense(10, input_dim=X_train.shape[1], activation='relu'))
```

- The model needs to know what input shape it should expect
- The first layer in a Sequential model (and only the first, because following layers can do automatic shape inference) needs to receive information about its input shape
- Dense: implements the operation: output = activation(dot(input, kernel) + bias)
  - activation is the element-wise activation function
  - kernel is a weights matrix
  - bias is a bias vector

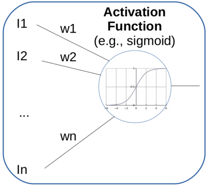


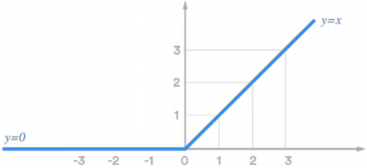
Activation function

```
model.add(Dense(10, input_dim=X_train.shape[1], activation='relu'))
```

Available activation functions

- sigmoid
- hard sigmoid
- softmax
- tanh (hyperbolic tangent)
- *ReLU*: Rectified Linear Unit
  - $max(sum_j(l_j * w_j + b_j), 0)$ (element-wise max)

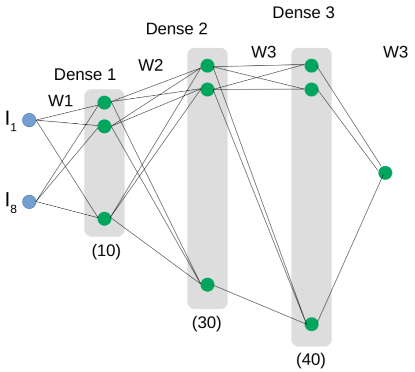




Adding internal layers

```
model.add(Dense(30, activation='relu'))
model.add(Dense(40, activation='relu'))
model.add(Dense(1))
```

Following layer can do *automatic shape inference* (no need to specify input dimension)



Compiling the model

```
model.compile(optimizer ='adam', loss = 'mean_squared_error', metrics =[metrics.m
```

Compilation: configuration of the training process

- *Optimizer*: (e.g., adam, see `https://arxiv.org/abs/1412.6980v8`): an algorithm for first-order gradient-based optimization of stochastic objective functions
- *Loss function*: the objective that the model will try to minimize (e.g., Root Mean Squared Error between real and estimated output value)
- A list of metrics: used to judge the performance of your model in validation (e.g., accuracy, mean absolute error)

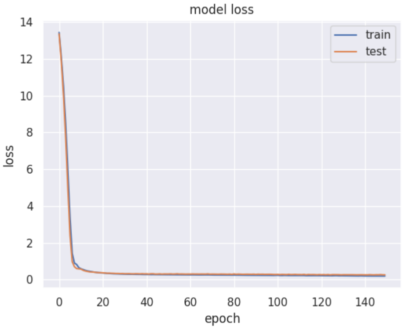
Training

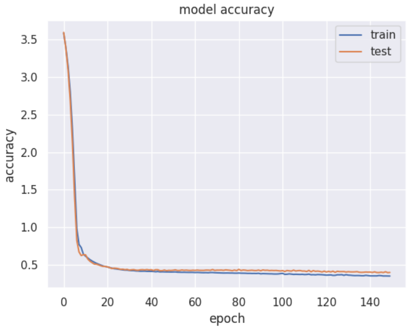
```
history = model.fit(X_train, y_train, validation_data=(X_val, y_val),
 epochs=150, batch_size=32)
```

- Keras models are trained on *Numpy arrays* of input data and labels (see previous slide on data loading)
- *validation_data*: data on which to evaluate the loss and any model metrics at the end of each epoch
- *epochs*: number of iterations of the training phase
- *batch_size*: number of samples per gradient update (default: 32)

Monitoring the training process

```
Epoch 1/150
10/10 [==============================] - 2s 105ms/step - loss: 12.5007 - mean_absolute_error: 3.4395 - val_loss: 11.6196 - val_mean_absolute_error: 3.3384
Epoch 2/150
10/10 [==============================] - 0s 23ms/step - loss: 10.7223 - mean_absolute_error: 3.1686 - val_loss: 9.6738 - val_mean_absolute_error: 3.0289
Epoch 3/150
10/10 [==============================] - 0s 22ms/step - loss: 9.2778 - mean_absolute_error: 2.9226 - val_loss: 7.3082 - val_mean_absolute_error: 2.5995
Epoch 4/150
10/10 [==============================] - 0s 36ms/step - loss: 7.0036 - mean_absolute_error: 2.4657 - val_loss: 4.7041 - val_mean_absolute_error: 2.0096
Epoch 5/150
10/10 [==============================] - 0s 34ms/step - loss: 4.8271 - mean_absolute_error: 1.9417 - val_loss: 2.4164 - val_mean_absolute_error: 1.3244
Epoch 6/150
10/10 [==============================] - 0s 42ms/step - loss: 2.5945 - mean_absolute_error: 1.3383 - val_loss: 1.3055 - val_mean_absolute_error: 0.9303
Epoch 7/150
10/10 [==============================] - 0s 36ms/step - loss: 2.0313 - mean_absolute_error: 1.1693 - val_loss: 1.1355 - val_mean_absolute_error: 0.8625
Epoch 8/150
```



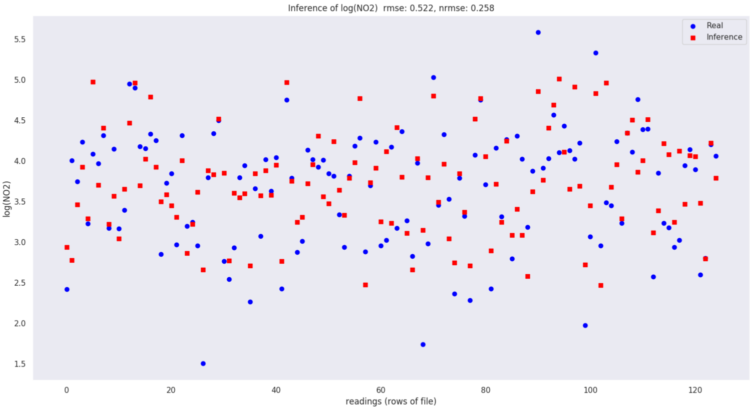


`model.summary()`

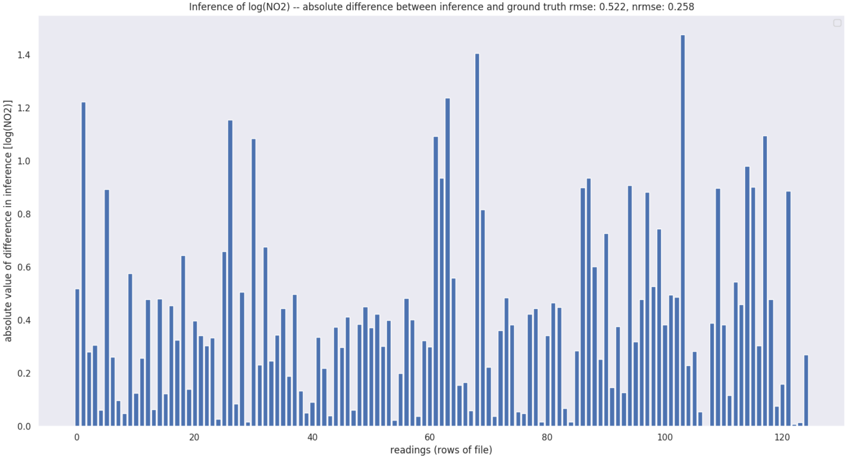
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 10)                80
_________________________________________________________________
dense_1 (Dense)              (None, 30)                330
_________________________________________________________________
dense_2 (Dense)              (None, 40)                1240
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 41
=================================================================
Total params: 1,691
Trainable params: 1,691
Non-trainable params: 0
```

Inference on new data

```
pred = model.predict(X_test) #compute the prediction
rmse = RMSE(y_test, pred) #evaluate the RMSE: value should be in [0.5,0.6]
nrmse = NRMSE(y_test, pred) #evalute the NRMSE: value should be in [0.2,0.3]
```



Inference on new data, visualizing the asbolute error



Assignment

- perform this data analysis:
  1. load the dataset (NO2.csv) and visualise data correlation
  2. standardize data
  3. divide dataset in train, test, validation
  4. create the DNN model outlined above [model large]
  5. train the model
  6. test the DNN model
  7. evaluate the trained model (loss, accuracy, RMSE, NRMSE, absolute error)
- Repeat the analysis by using the following models (hidden layers):
  1. 1 layers containing 3 neurons [model tiny]
  2. 1 layer containing 10 neurons [model small]
  3. 2 layers containing respectively 10 and 30 neurons [model medium]