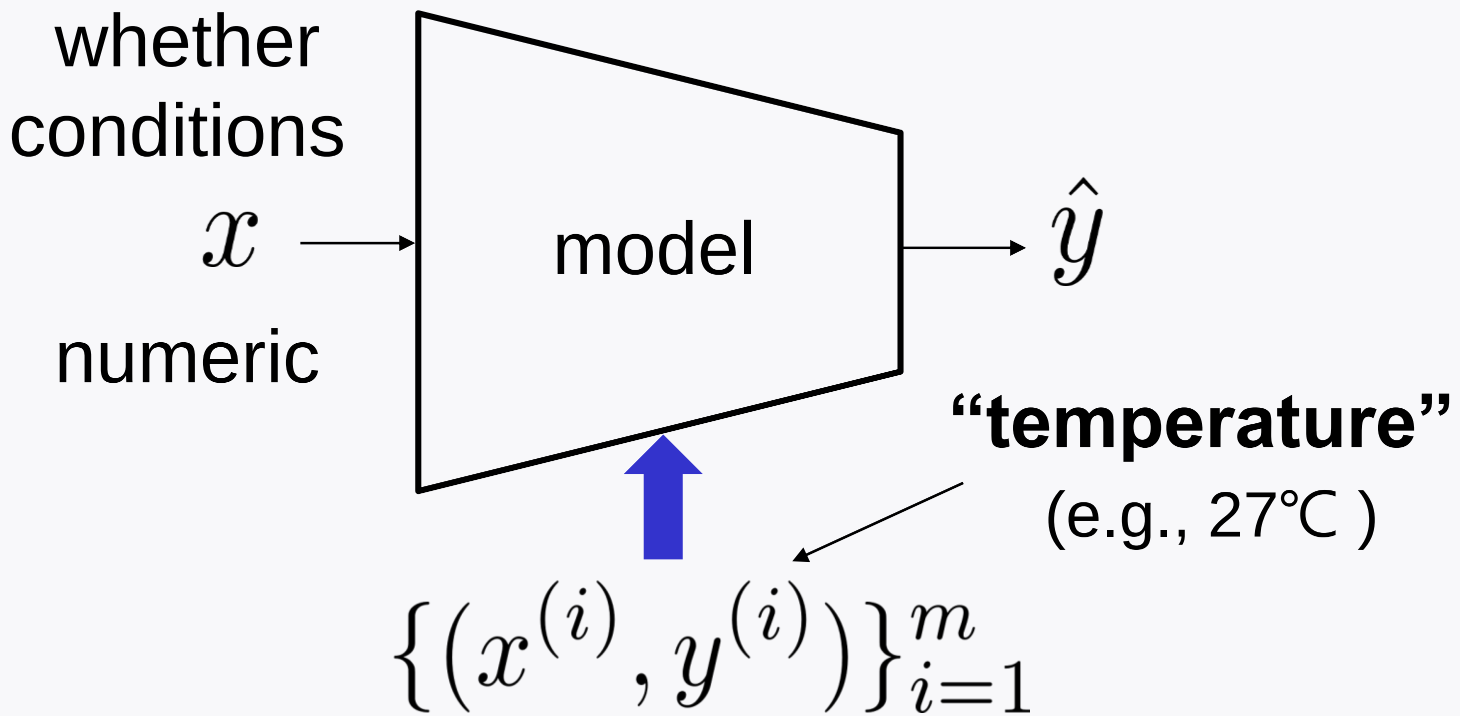
Mini-project #2

Practice Session 22

Changho Suh

January 30, 2024

Recap: Weather prediction



Recap: Data load & preprocessing

```
import pandas as pd

data = pd.read_csv('jena_climate_2009_2016.csv')

wv = data['wv (m/s)']
wv_missing_idx = (wv == -9999.00)
wv_mean = wv[~wv_missing_idx].mean()
wv[wv_missing_idx] = wv_mean

max_wv = data['max. wv (m/s)']
missing_idx = (max_wv == -9999.00)
max_wv_mean = max_wv[~missing_idx].mean()
max_wv[missing_idx] = max_wv_mean

data.pop('Date Time')

data = data[0::6]
```

$m = 70,092$

Recap: Normalization & splitting

```
features = data
labels = data[['T (degC)']]
```

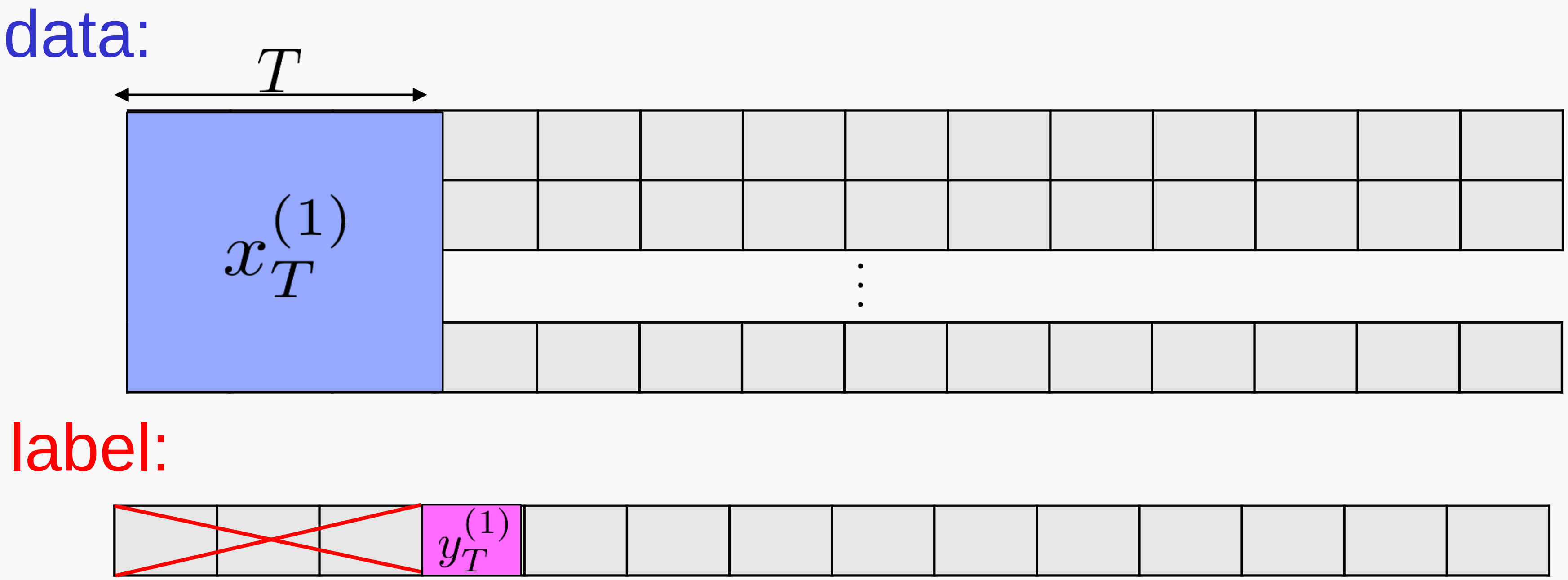
```
from sklearn.preprocessing import StandardScaler

std_scaler = StandardScaler()
features = std_scaler.fit_transform(features)
```

```
from sklearn.model_selection import train_test_split
```

```
X_rest, X_test, y_rest, y_test = train_test_split(features,
                                                  labels,
                                                  test_size=0.1,
                                                  shuffle=False)
X_train, X_val, y_train, y_val = train_test_split(X_rest,
                                                  y_rest,
                                                  test_size=2/9,
                                                  shuffle=False)
```

Time series data generation




```
from tensorflow.keras.preprocessing import timeseries_dataset_from_array

dataset_train = timeseries_dataset_from_array(X_train[:-T],
                                              y_train[T:],
                                              sequence_length = T,
                                              sequence_stride = 1,
                                              batch_size = batch_size,
                                              shuffle = True)
```

Model

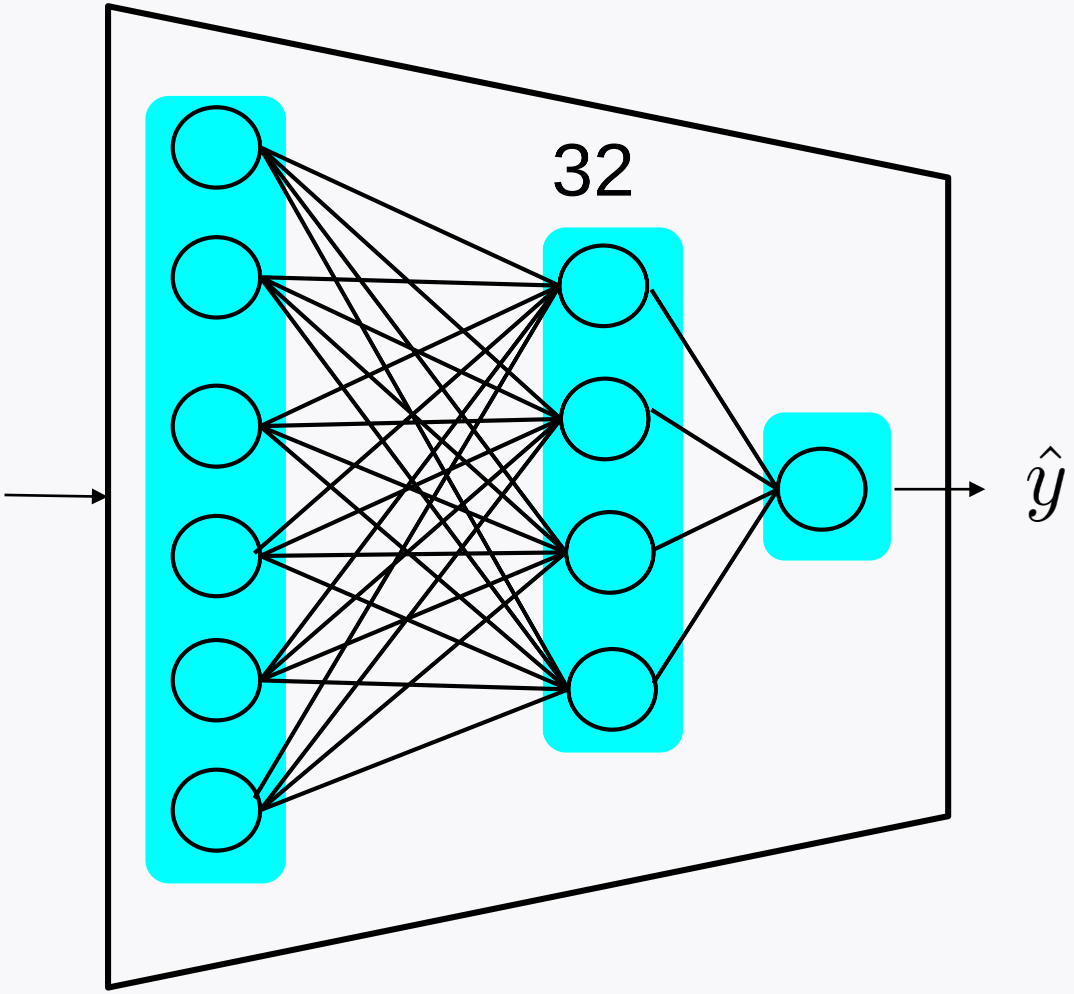
Will implement:

2-layer DNN

2-layer LSTM

DNN architecture

input size:
14xT



DNN architecture: Keras model

```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input,Flatten,Dense
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import LearningRateScheduler,EarlyStopping
```

```
inputs = Input(shape=(T,14))
x = Flatten()(inputs)
x = Dense(units=32,activation='relu')(x)
outputs = Dense(units=1)(x)
dnn_model = Model(inputs=inputs,outputs=outputs)
```

DNN architecture: Keras model

```
dnn_model.summary()
```

```
Model: "model_1"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 24, 14)]          0

 flatten_1 (Flatten)         (None, 336)               0

 dense_2 (Dense)             (None, 32)                10784

 dense_3 (Dense)             (None, 1)                 33

=================================================================
Total params: 10,817
Trainable params: 10,817
Non-trainable params: 0

_________________________________________________________________
```

Early stopping & learning rate decaying

```
from tensorflow.keras.callbacks import EarlyStopping

es_callback = EarlyStopping(monitor='val_loss',patience=10)

from tensorflow.keras.callbacks import LearningRateScheduler

def scheduler(epoch,lr):
    if epoch in [10,20,30]: lr = 0.1*lr
    return lr
lrs_callback = LearningRateScheduler(scheduler)
```

Adam optimizer & compile

```
from tensorflow.keras.optimizers import Adam

opt = Adam(learning_rate=0.001,
           beta_1 = 0.9,
           beta_2 = 0.999)
```

```
from tensorflow.keras.metrics import RootMeanSquaredError
dnn_model.compile(loss='mean_squared_error',
                  metrics=RootMeanSquaredError(),
                  optimizer=opt)
```

Training

```
hist= dnn_model.fit(dataset_train,
                    epochs=30,
                    validation_data=dataset_val,
                    callbacks=[es_callback,lrs_callback])
```

```
Epoch 1/30
3064/3064 [==============================] - 3s 895us/step - loss: 5.3481 - root_mean_squared_error: 2.3126 - val_loss: 1.0391
- val_root_mean_squared_error: 1.0193 - lr: 0.0010
Epoch 2/30
3064/3064 [==============================] - 2s 764us/step - loss: 0.9471 - root_mean_squared_error: 0.9732 - val_loss: 1.2296
- val_root_mean_squared_error: 1.1089 - lr: 0.0010
Epoch 3/30
3064/3064 [==============================] - 2s 758us/step - loss: 0.7985 - root_mean_squared_error: 0.8936 - val_loss: 0.8527
- val_root_mean_squared_error: 0.9234 - lr: 0.0010
Epoch 4/30
3064/3064 [==============================] - 2s 775us/step - loss: 0.7408 - root_mean_squared_error: 0.8607 - val_loss: 0.6701
- val_root_mean_squared_error: 0.8186 - lr: 0.0010
```

:

```
Epoch 27/30
3064/3064 [==============================] - 2s 801us/step - loss: 0.4876 - root_mean_squared_error: 0.6983 - val_loss: 0.5134
- val_root_mean_squared_error: 0.7166 - lr: 1.0000e-05
Epoch 28/30
3064/3064 [==============================] - 2s 806us/step - loss: 0.4874 - root_mean_squared_error: 0.6982 - val_loss: 0.5132
- val_root_mean_squared_error: 0.7164 - lr: 1.0000e-05
Epoch 29/30
3064/3064 [==============================] - 3s 814us/step - loss: 0.4873 - root_mean_squared_error: 0.6981 - val_loss: 0.5130
- val_root_mean_squared_error: 0.7163 - lr: 1.0000e-05
Epoch 30/30
3064/3064 [==============================] - 2s 800us/step - loss: 0.4872 - root_mean_squared_error: 0.6980 - val_loss: 0.5130
- val_root_mean_squared_error: 0.7162 - lr: 1.0000e-05
```

Performance measure

Will use another measure instead of RMSE:

Root-mean-square error (RMSE):

$$\sqrt{\frac{1}{m_{\text{test}}}\sum_{i=1}^{m_{\text{test}}}\|y^{(i)}-\hat{y}^{(i)}\|^2}$$

Normalized RMSE:

$$\frac{\text{RMSE}}{\boxed{\sqrt{\frac{1}{m_{\text{test}}}\sum_{i=1}^{m_{\text{test}}}\|y^{(i)}-\mu\|^2}}} \longleftarrow \sigma_{\text{test}}$$

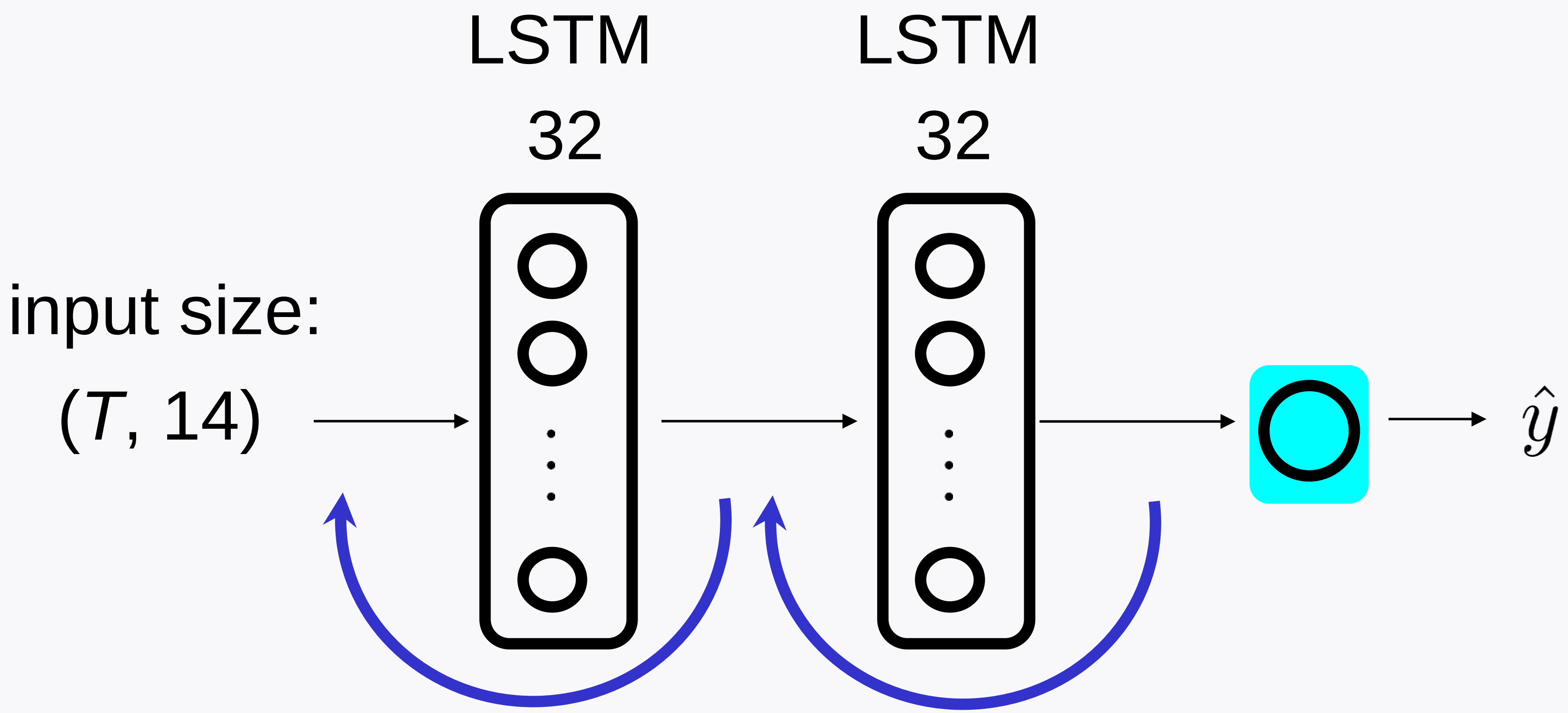
Evaluation: Normalized RMSE

```
eval_rmse = dnn_model.evaluate(dataset_test)[1]
eval_nrmse = eval_rmse/y_test.std()
print(eval_nrmse)
```

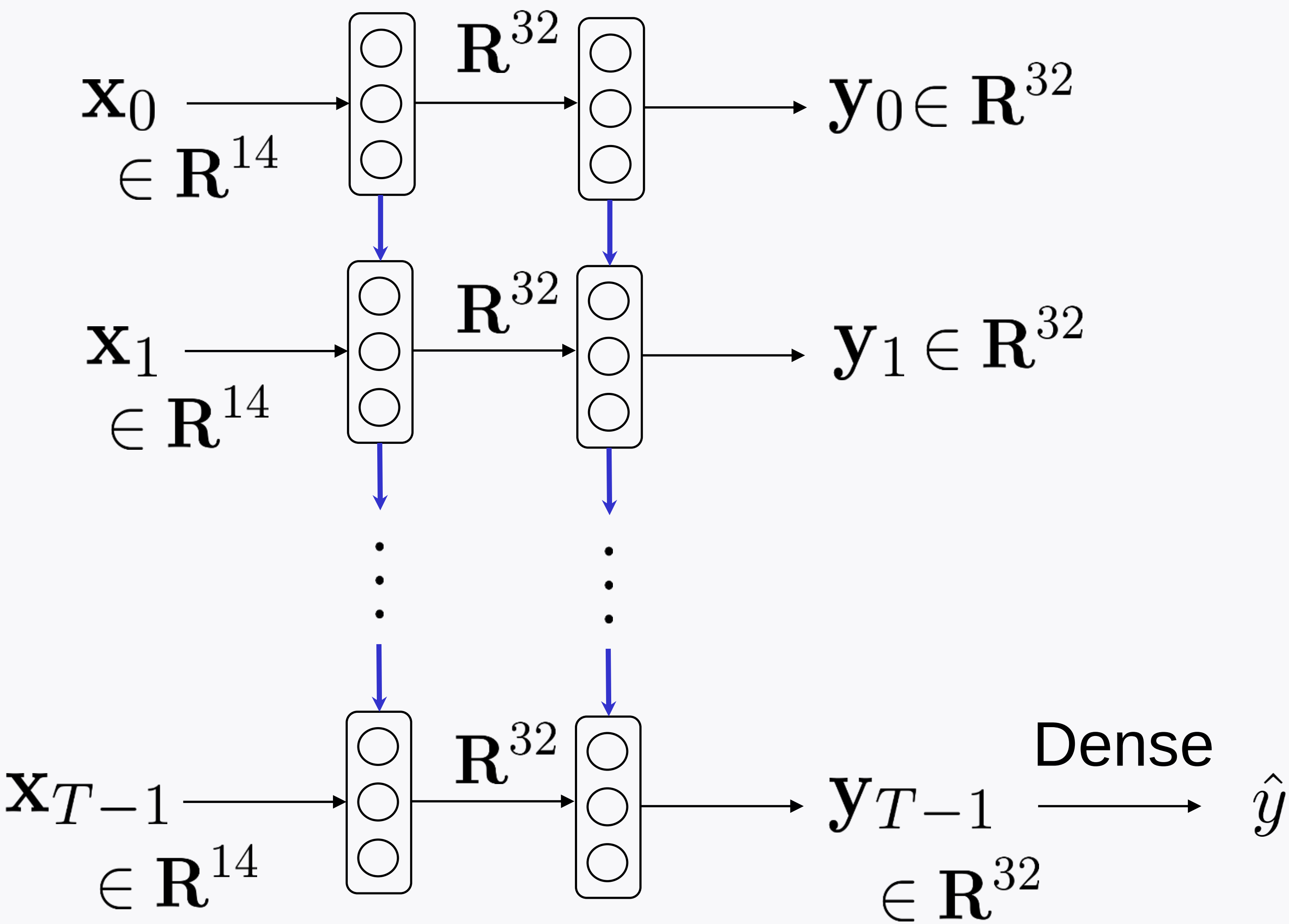
```
436/436 [==============================] - 1s 2ms/step - loss: 0.5136 - root_mean_squared_error: 0.7167
T (degC)    0.091312
dtype: float64
```

normalized RMSE

RNN architecture



Unrolled version



RNN architecture: Keras model

```
from tensorflow.keras.layers import Input
from tensorflow.keras.layers import LSTM
from tensorflow.keras.layers import Dense
from tensorflow.keras.models import Model

inputs = Input(shape=(T,14))

x = LSTM(units=32,return_sequences=True)(inputs)
x = LSTM(units=32)(x)
```

activate all the output sequences

By default: return_sequences=False

```
outputs=Dense(units=1)(x)

rnn_model = Model(inputs=inputs, outputs=outputs)
```

RNN architecture: Keras model

```
rnn_model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 24, 14)]          0

 lstm (LSTM)                 (None, 24, 32)            6016

 lstm_1 (LSTM)               (None, 32)                8320

 dense (Dense)               (None, 1)                 33

=================================================================
Total params: 14,369
Trainable params: 14,369
Non-trainable params: 0
_________________________________________________________________
```

Early stopping & learning rate decay

```
from tensorflow.keras.callbacks import EarlyStopping

es_callback = EarlyStopping(monitor='val_loss',patience=10)

from tensorflow.keras.callbacks import LearningRateScheduler

def scheduler(epoch,lr):
    if epoch in [10,20,30]: lr = 0.1*lr
    return lr
lrs_callback = LearningRateScheduler(scheduler)
```

Adam optimizer & compile

```
from tensorflow.keras.optimizers import Adam

opt = Adam(learning_rate=0.001,
           beta_1 = 0.9,
           beta_2 = 0.999)
```

```
from tensorflow.keras.metrics import RootMeanSquaredError

rnn_model.compile(loss='mean_squared_error',
                  metrics=RootMeanSquaredError(),
                  optimizer = opt)
```

Training

```
history = rnn_model.fit(dataset_train,
                        epochs=30,
                        validation_data=dataset_val,
                        callbacks=[es_callback,lrs_callback])
```

```
Epoch 1/30
3064/3064 [==============================] - 25s 7ms/step - loss: 6.9035 - root_mean_squared_error: 2.6275 - val_loss: 0.7795 - val_root_mean_squared_error: 0.8829 - lr: 0.0010
Epoch 2/30
3064/3064 [==============================] - 23s 7ms/step - loss: 0.6391 - root_mean_squared_error: 0.7994 - val_loss: 0.5613 - val_root_mean_squared_error: 0.7492 - lr: 0.0010
Epoch 3/30
3064/3064 [==============================] - 24s 8ms/step - loss: 0.5728 - root_mean_squared_error: 0.7568 - val_loss: 0.5412 - val_root_mean_squared_error: 0.7356 - lr: 0.0010
Epoch 4/30
3064/3064 [==============================] - 24s 8ms/step - loss: 0.5520 - root_mean_squared_error: 0.7430 - val_loss: 0.5379 - val_root_mean_squared_error: 0.7335 - lr: 0.0010
```

:

```
Epoch 27/30
3064/3064 [==============================] - 24s 8ms/step - loss: 0.4336 - root_mean_squared_error: 0.6585 - val_loss: 0.4471 - val_root_mean_squared_error: 0.6687 - lr: 1.0000e-05
Epoch 28/30
3064/3064 [==============================] - 24s 8ms/step - loss: 0.4335 - root_mean_squared_error: 0.6584 - val_loss: 0.4471 - val_root_mean_squared_error: 0.6687 - lr: 1.0000e-05
Epoch 29/30
3064/3064 [==============================] - 24s 8ms/step - loss: 0.4334 - root_mean_squared_error: 0.6583 - val_loss: 0.4472 - val_root_mean_squared_error: 0.6687 - lr: 1.0000e-05
Epoch 30/30
3064/3064 [==============================] - 25s 8ms/step - loss: 0.4333 - root_mean_squared_error: 0.6583 - val_loss: 0.4471 - val_root_mean_squared_error: 0.6687 - lr: 1.0000e-05
```

Evaluation: Normalized RMSE

```
eval_rmse = rnn_model.evaluate(dataset_test)[1]
eval_nrmse = eval_rmse/y_test.std()
print(eval_nrmse)
```

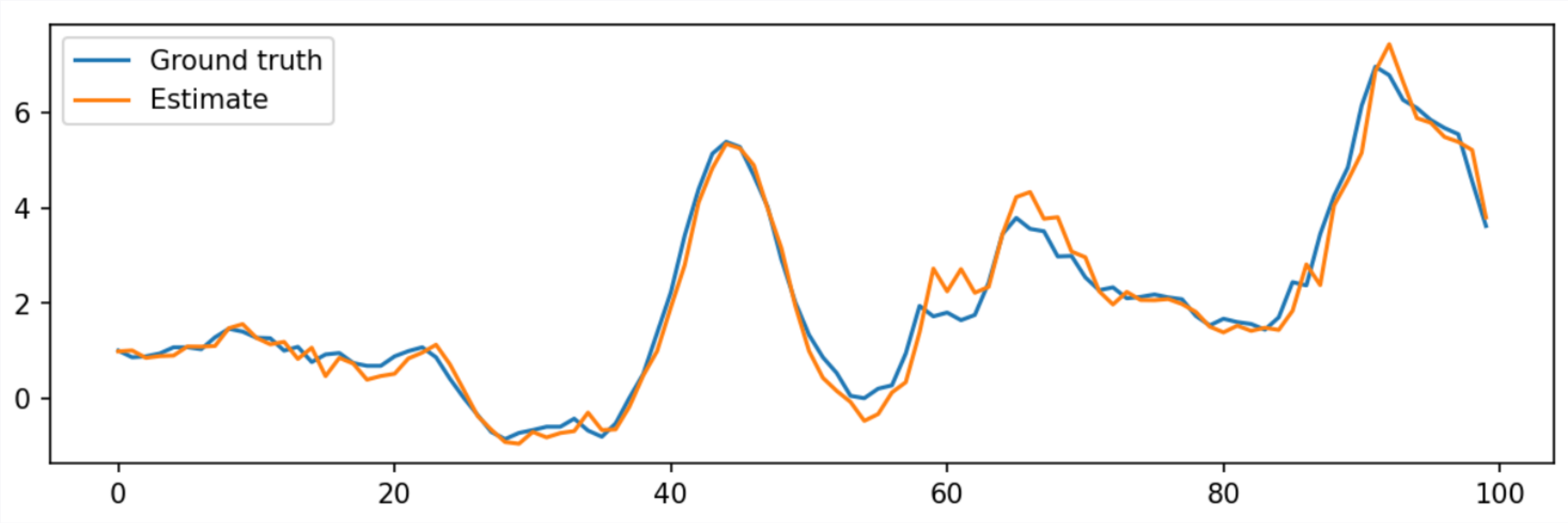
```
436/436 [==============================] - 1s 3ms/step - loss: 0.4410 - root_mean_squared_error: 0.664
T (degC)    0.084609
dtype: float64
```

normalized RMSE

Ground truth vs. estimate

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10,3), dpi=150)
plt.plot(y_test[T:100+T].values)
estimated = rnn_model.predict(dataset_test)
plt.plot(estimated[:100])
plt.legend(['Ground truth', 'Estimate'])
```



Look ahead

Will learn how to write a proposal.