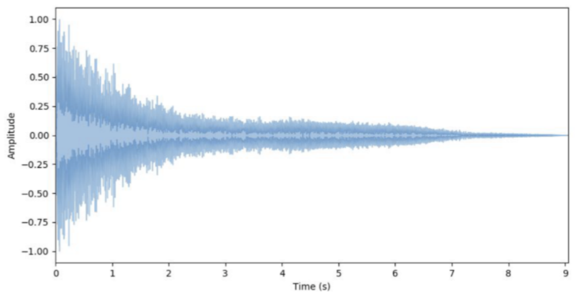
DLAP_5

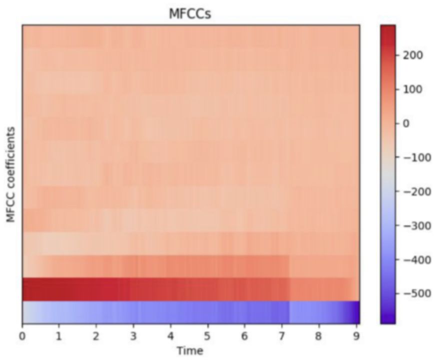
RNN

- Order is important
- Variable length
- Used for sequential data
- Each item is processed in context
- Ideal for audio/music



Waveform - Univatiate time series

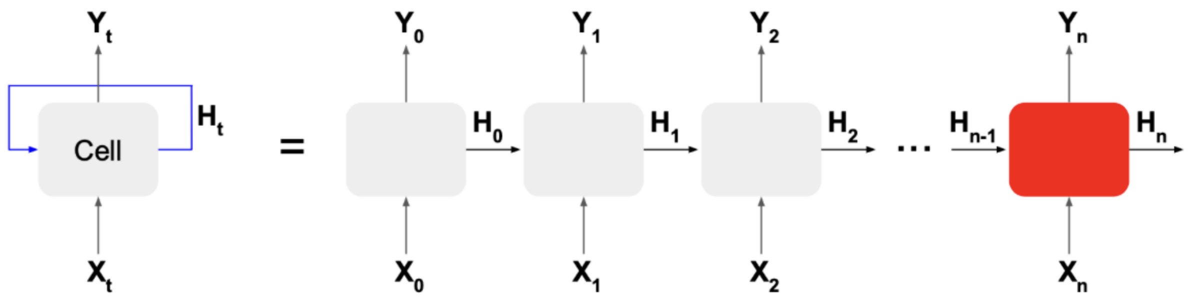
dim = [sr x time, 1]



MFCCs - Multivariate time series

dim = [sr/hop_length x time, MFCCs]

RNN architecture



Memory cell for simple RNN

- Dense layer
- Input = state vector + input data
- Activation function = *tanh*

Tanh 사용이유

- RNN을 구성하는 time step은 deep network를 만들고 이는 역전파 과정에서 gradient vanishing, exploding를 일으킴
- Tanh는 [-1, 1] 사이값을 가지기 때문에 gradient vanishing, exploding 문제를 해결

RNN 문제점

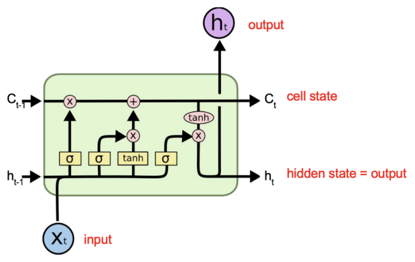
- No long-term memory
- Network can't use info from the distant past
- Can't learn patterns with long dependencies

LSTM

- Special type of RNN
- Can learn long-term patterns
- Detects patterns with 100 steps
- Struggles with 100s/1000s of steps

LSTM Cell

- Contains a simple RNN cell
- Second state vector = cell state = long-term memory
- Forget gate
- Input gate
- Output gate
- Gates work as filters



Cell state

- 적은 계산으로 안정적인 gradients를 가짐
- x → decide what to forget

$$C_t^f = C_{t-1} * f_t$$

$$C_{t-1} = [1, 2, 4]$$
$$f_t = [1, 0, 1]$$

$$\boxed{C_t^f = [1, 2, 4] * [1, 0, 1] = [1, 0, 4]}$$

- + → decide what noew info to remember

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i)$$

$$C_t^{'} = tanh(W_c[h_{t-1}, x_t] + b_C)$$

$$C_t^i = C_t^{'} * i_t$$

$$\boxed{C_t} = \boxed{C_t^f} + \boxed{C_t^i}$$

Output

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * tanh(C_t)$$

RNN-LSTM for music genre classification

```
import json
import numpy as np
from sklearn.model_selection import train_test_split
import tensorflow.keras as keras
import matplotlib.pyplot as plt

DATA_PATH = "../13/data_10.json"


def load_data(data_path):
    """Loads training dataset from json file.

        :param data_path (str): Path to json file containing data
        :return X (ndarray): Inputs
        :return y (ndarray): Targets
    """

    with open(data_path, "r") as fp:
        data = json.load(fp)

    X = np.array(data["mfcc"])
    y = np.array(data["labels"])
    return X, y


def plot_history(history):
    """Plots accuracy/loss for training/validation set as a function of the epochs
```

```
        :param history: Training history of model
        :return:
    """

    fig, axs = plt.subplots(2)

    # create accuracy sublpot
    axs[0].plot(history.history["accuracy"], label="train accuracy")
    axs[0].plot(history.history["val_accuracy"], label="test accuracy")
    axs[0].set_ylabel("Accuracy")
    axs[0].legend(loc="lower right")
    axs[0].set_title("Accuracy eval")

    # create error sublpot
    axs[1].plot(history.history["loss"], label="train error")
    axs[1].plot(history.history["val_loss"], label="test error")
    axs[1].set_ylabel("Error")
    axs[1].set_xlabel("Epoch")
    axs[1].legend(loc="upper right")
    axs[1].set_title("Error eval")

    plt.show()


def prepare_datasets(test_size, validation_size):
    """Loads data and splits it into train, validation and test sets.

    :param test_size (float): Value in [0, 1] indicating percentage of data set to allocate to test split
    :param validation_size (float): Value in [0, 1] indicating percentage of train set to allocate to validation split

    :return X_train (ndarray): Input training set
    :return X_validation (ndarray): Input validation set
    :return X_test (ndarray): Input test set
    :return y_train (ndarray): Target training set
    :return y_validation (ndarray): Target validation set
    :return y_test (ndarray): Target test set
    """

    # load data
    X, y = load_data(DATA_PATH)

    # create train, validation and test split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size)
    X_train, X_validation, y_train, y_validation = train_test_split(X_train, y_train, test_size=validation_size)

    return X_train, X_validation, X_test, y_train, y_validation, y_test


def build_model(input_shape):
    """Generates RNN-LSTM model

    :param input_shape (tuple): Shape of input set
    :return model: RNN-LSTM model
    """

    # build network topology
    model = keras.Sequential()

    # 2 LSTM layers
    model.add(keras.layers.LSTM(64, input_shape=input_shape, return_sequences=True))
    model.add(keras.layers.LSTM(64))

    # dense layer
    model.add(keras.layers.Dense(64, activation='relu'))
    model.add(keras.layers.Dropout(0.3))

    # output layer
    model.add(keras.layers.Dense(10, activation='softmax'))

    return model


if __name__ == "__main__":

    # get train, validation, test splits
    X_train, X_validation, X_test, y_train, y_validation, y_test = prepare_datasets(0.25, 0.2)
```

```
# create network
input_shape = (X_train.shape[1], X_train.shape[2]) # 130, 13
model = build_model(input_shape)

# compile model
optimiser = keras.optimizers.Adam(learning_rate=0.0001)
model.compile(optimizer=optimiser,
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.summary()

# train model
history = model.fit(X_train, y_train, validation_data=(X_validation, y_validation), batch_size=32, epochs=30)

# plot accuracy/error for training and validation
plot_history(history)

# evaluate model on test set
test_loss, test_acc = model.evaluate(X_test, y_test, verbose=2)
print('\nTest accuracy:', test_acc)
```