Data Analytics
EEE 4774 & 6777

Module 5 - Regression

Recurrent Neural Network (RNN)

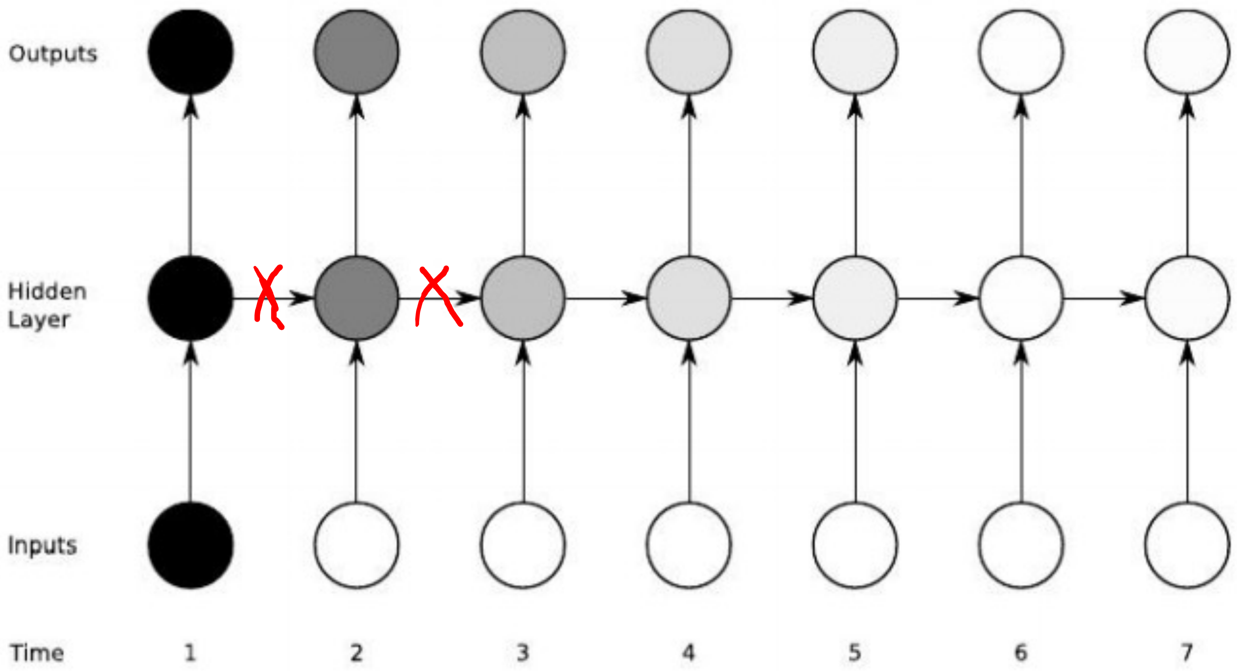
Spring 2022

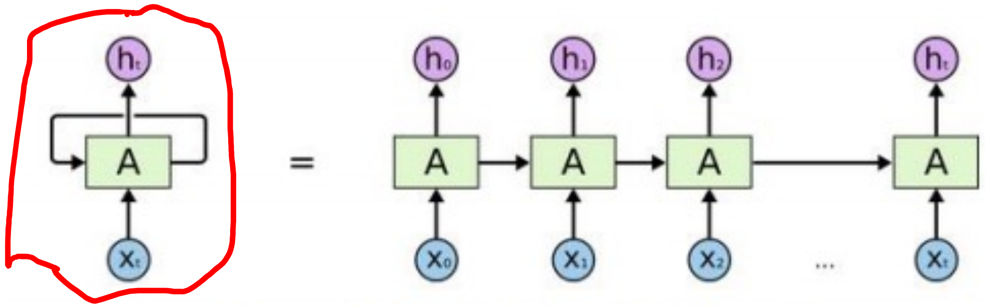
Sequence Learning

- Sequential (time series) data naturally occurs in several domains: speech, language, video, finance, biological signals, etc.
- Temporal dependency between samples unlike standard machine learning
- Classification tasks
- Regression, e.g., time-series forecasting/prediction
- Autoregressive (AR) models
- Recurrent Neural Networks (RNN)

Recurrent Neural Network (RNN)

- Distributed hidden state stores a lot of information about the past efficiently
- Nonlinear dynamics updates the hidden state in a complicated way
- Deterministic, no need to infer hidden state
- Weight sharing among all time steps





An unrolled recurrent neural network.

Vanilla RNN – Forward Pass

- The same as that of an MLP with a single hidden layer
- Except that activations arrive at the hidden layer from both the current external input and the hidden layer activations one step back in time.

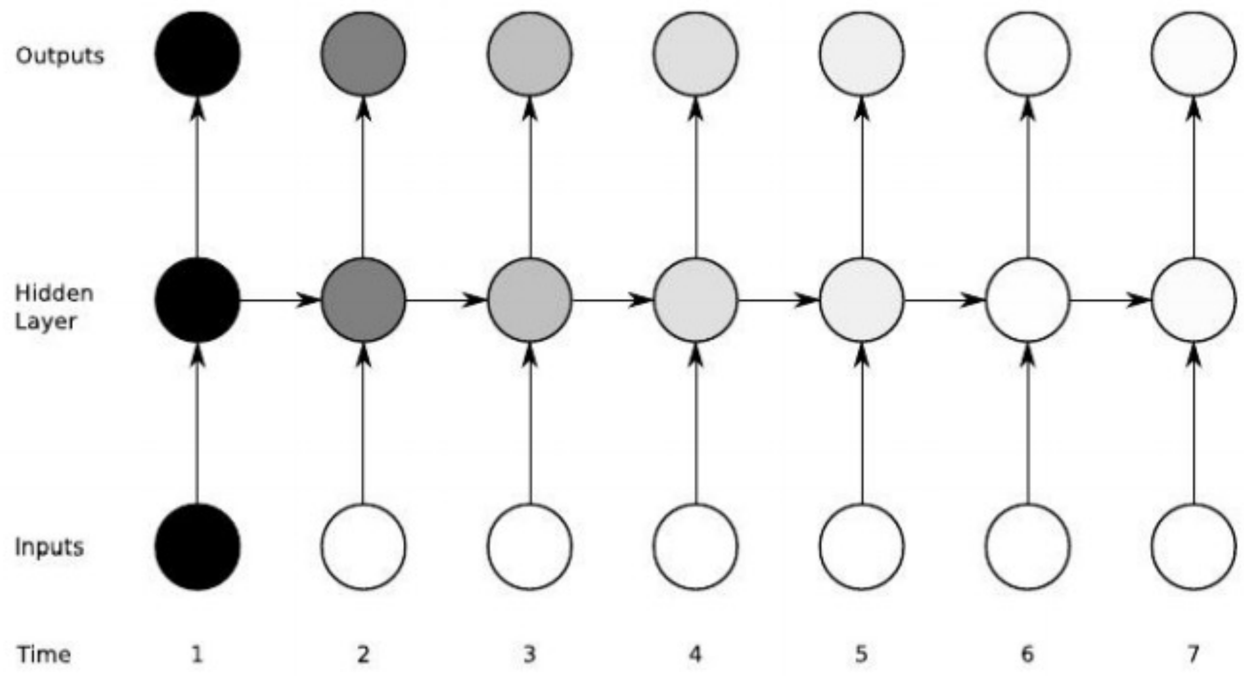
$$a_h^t = \sum_{i=1}^{I} w_{ih} x_i^t + \sum_{h'=1}^{H} w_{h'h} b_{h'}^{t-1}$$

hidden neurons

$$\boxed{b_h^t = \theta_h(a_h^t)}$$

$$a_k^t = \sum_{h=1}^{H} w_{hk} b_h^t$$

output neurons



Vanilla RNN – Backward Pass

- Backpropagation Through Time (BPTT): repeated application of chain rule – extension of standard backpropagation

$$\frac{\partial O}{\partial w_{ij}} = \sum_{t=1}^{T} \frac{\partial O}{\partial a_j^t} \frac{\partial a_j^t}{\partial w_{ij}} = \sum_{t=1}^{T} \delta_j^t b_i^t$$

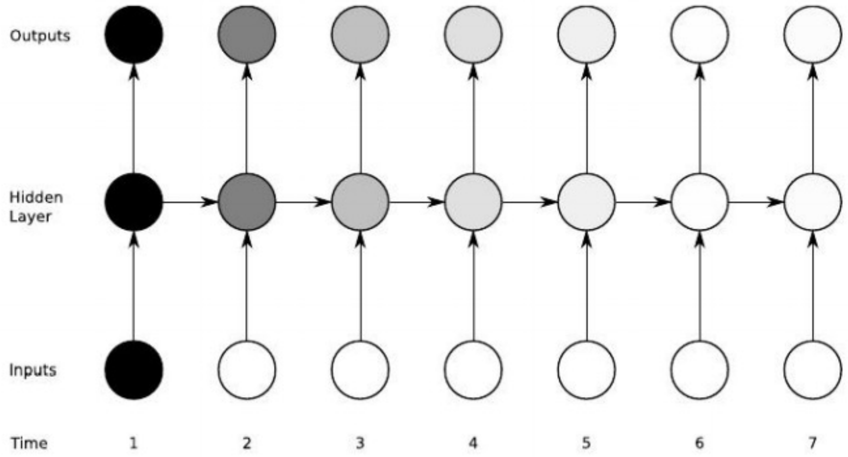
$$\delta_h^t = \theta'(a_h^t) \left(\sum_{k=1}^{K} \delta_k^t w_{hk} + \sum_{h'=1}^{H} \delta_{h'}^{t+1} w_{hh'} \right)$$

$$\delta_j^t \overset{\text{def}}{=} \frac{\partial O}{\partial a_j^t}$$

$$a_h^t = \sum_{i=1}^{I} w_{ih} x_i^t + \sum_{h'=1}^{H} w_{h'h} b_{h'}^{t-1}$$

$$b_h^t = \theta_h(a_h^t)$$

$$a_k^t = \sum_{h=1}^{H} w_{hk} b_h^t$$

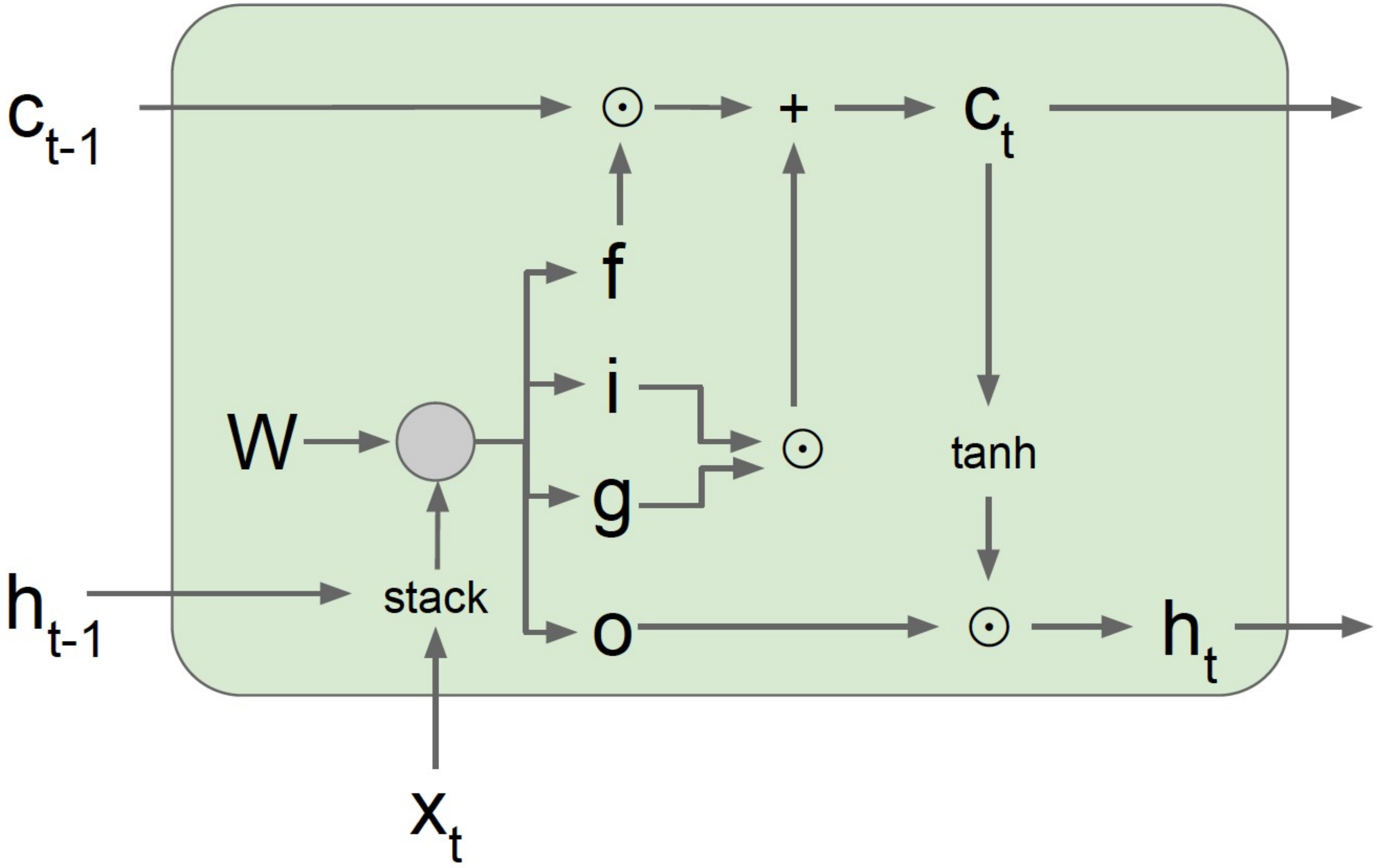


LSTM: Long Short-Term Memory

- Vanilla RNN has vanishing and exploding gradient problems
- The influence of a given input on the output either decays or blows up exponentially
- Exploding is controlled by gradient clipping
- LSTM effectively deals wit the vanishing gradient problem
- It uses memory blocks with input, output, and forget gates to store and access information over long periods of time, hence the name.

Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]



$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

GRU: Gated Feedback Recurrent Neural Networks

- Newer than LSTM
- Very similar to LSTM
- Merges the cell state and hidden state
- Combines the forget and input gates into a single "update gate”
- Computationally more efficient: less parameters, less complex structure
- Use GRU if you want faster compute and less parameters

Implementing LSTM in TensorFlow - Keras

```
import tensorflow as tf
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Dropout
model = Sequential()
model.add(LSTM(50,
activation='sigmoid',return_sequences=True,
input_shape=(n_input_step, n_features)))
model.add(Dense(1))
model.compile(loss='mean_squared_error',
optimizer='adam',metrics=["mean_squared_error"])
model.summary()
```

```
cc = model.fit(station_train_X,
station_train_y, epochs=1000,
batch_size=4, verbose=1, shuffle=False)

yhat = model.predict(station_test_X)
```