AI VIETNAM
All-in-One Course

RNN/LSTM for Sequence and Time-Series Data

Quang-Vinh Dinh
Ph.D. in Computer Science

Year 2023

Outline

- RNN in PyTorch
- RNNs for Time-Series Data
- RNNs for IMDB dataset
- From RNN to LSTM
- LSTM Applications

Embedding Layer

Increase space dimentions

index	word
0	[UNK]
1	[pad]
2	ai
3	a
4	are
5	cs
6	is
7	learning

We are learning AI → 0, 4, 7, 2, 1

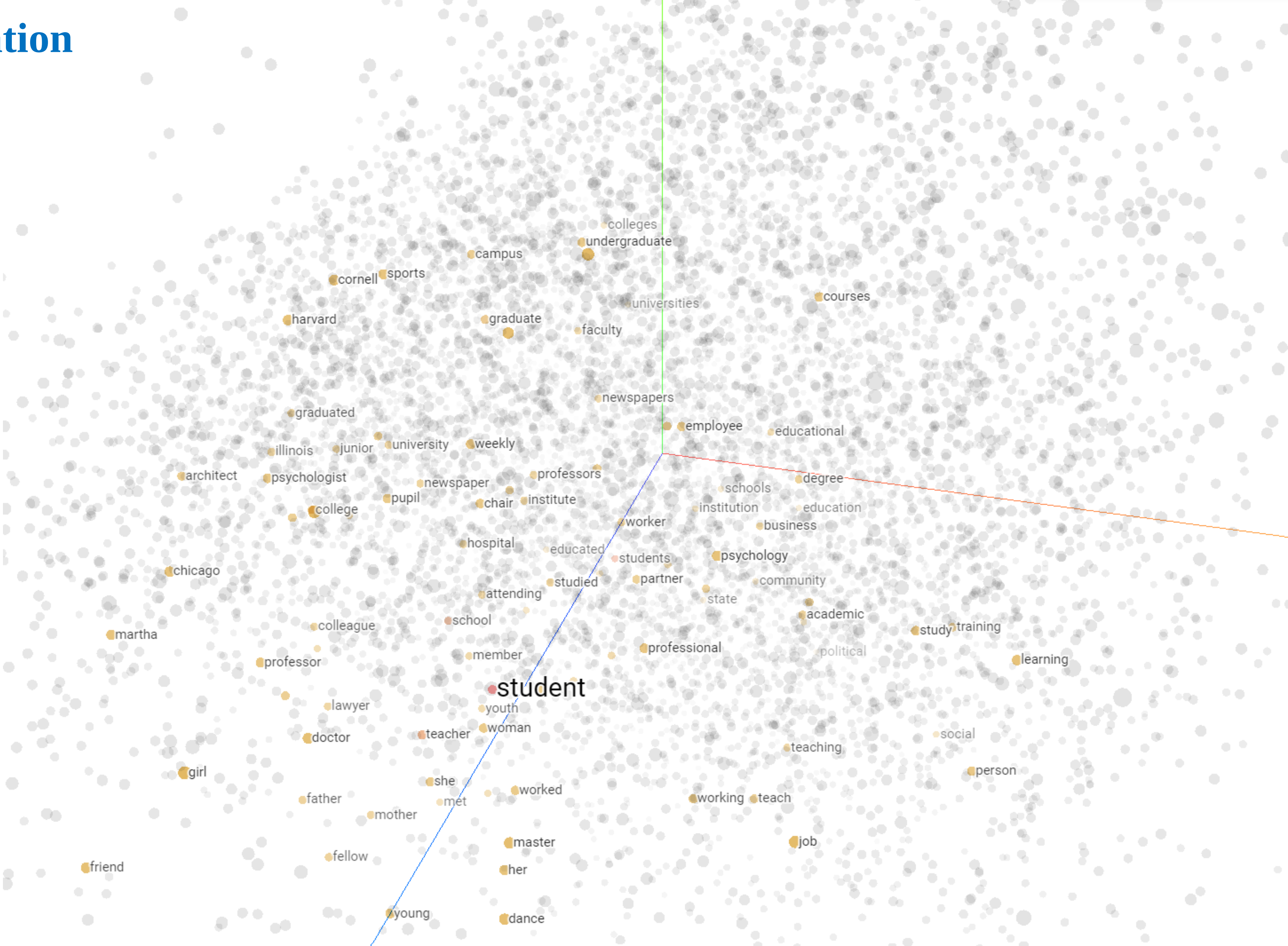
be updated

```
Parameter containing:
tensor([[-0.1882,  0.5530,  1.6267,  0.7013],
        [ 1.7840, -0.8278, -0.2701,  1.3586],
        [ 1.0281, -1.9094,  0.3182,  0.4211],
        [-1.3083, -0.0987,  0.7647, -0.3680],
        [ 0.2293,  1.3255,  0.1318,  2.0501],
        [ 0.4058, -0.6624, -0.8745,  0.7203],
        [ 0.5582,  0.0786, -0.6817,  0.6902],
        [ 0.4309, -1.3067, -0.8823,  1.5977]],
```

After one update

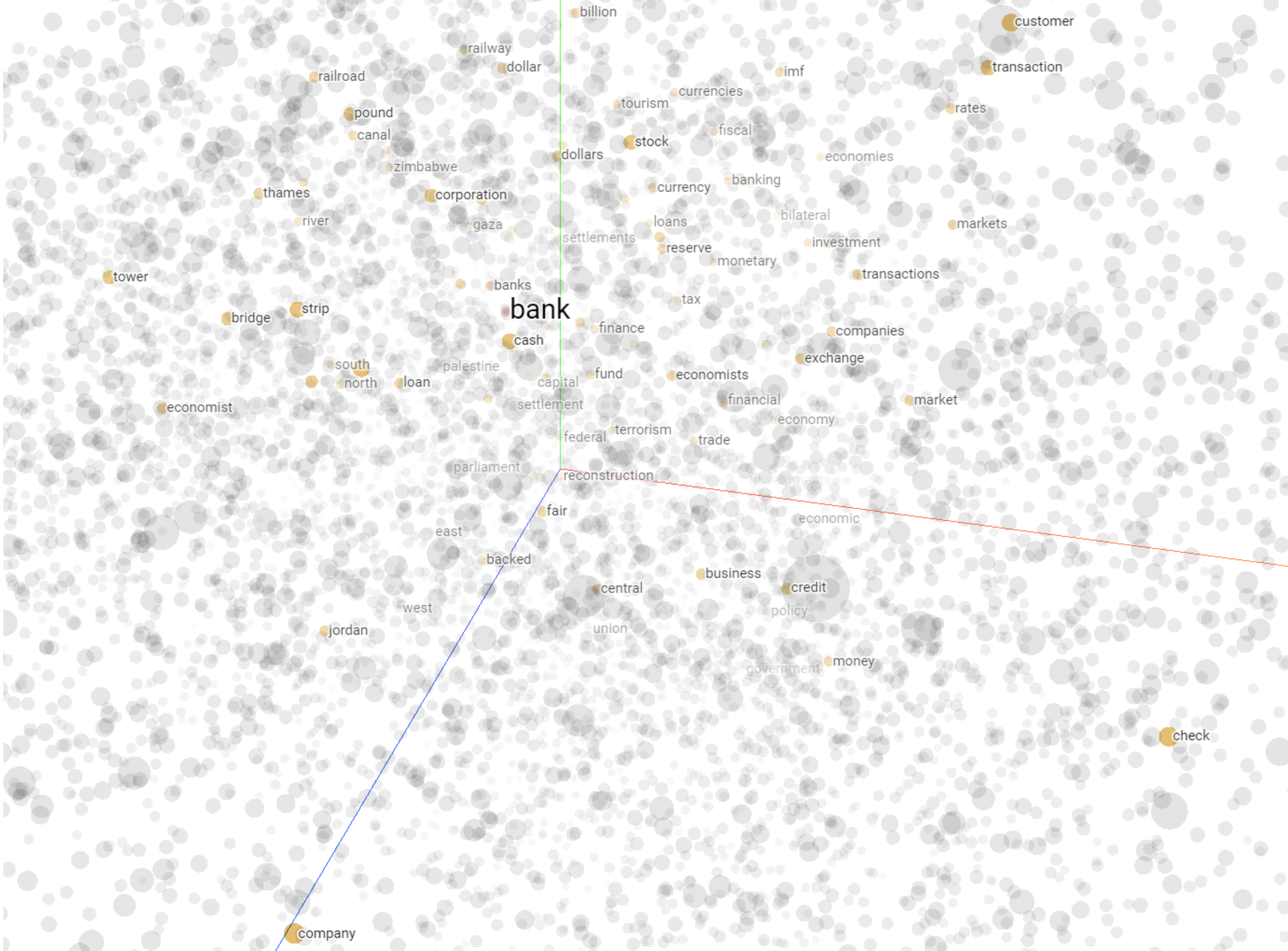
```
Parameter containing:
tensor([[-0.1872,  0.5540,  1.6277,  0.7023],
        [ 1.7830, -0.8268, -0.2711,  1.3576],
        [ 1.0291, -1.9084,  0.3192,  0.4201],
        [-1.3083, -0.0987,  0.7647, -0.3680],
        [ 0.2303,  1.3245,  0.1308,  2.0511],
        [ 0.4058, -0.6624, -0.8745,  0.7203],
        [ 0.5582,  0.0786, -0.6817,  0.6902],
        [ 0.4299, -1.3077, -0.8833,  1.5967]],
```

Embedding visualization



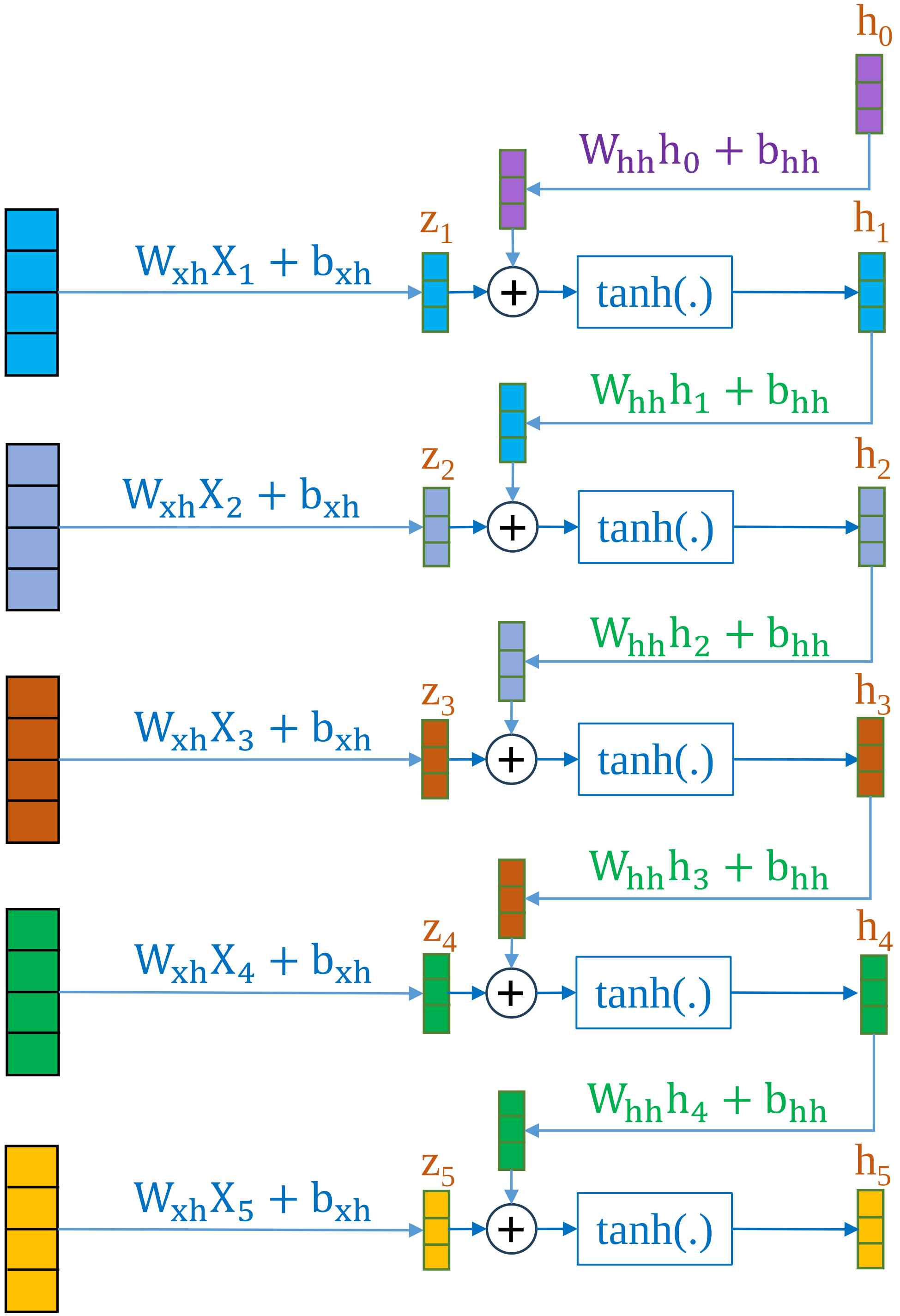
https://projector.tensorflow.org/

Embedding visualization



https://projector.tensorflow.org/

RNN



$h_0 = \mathbf{0} \qquad b_{hh} = \mathbf{0}$

$$h_1 = \tanh(W_{xh}X_1 + b_{xh} + W_{hh}h_0 + b_{hh})$$

$$h_2 = \tanh(W_{xh}X_2 + b_{xh} + W_{hh}h_1 + b_{hh})$$

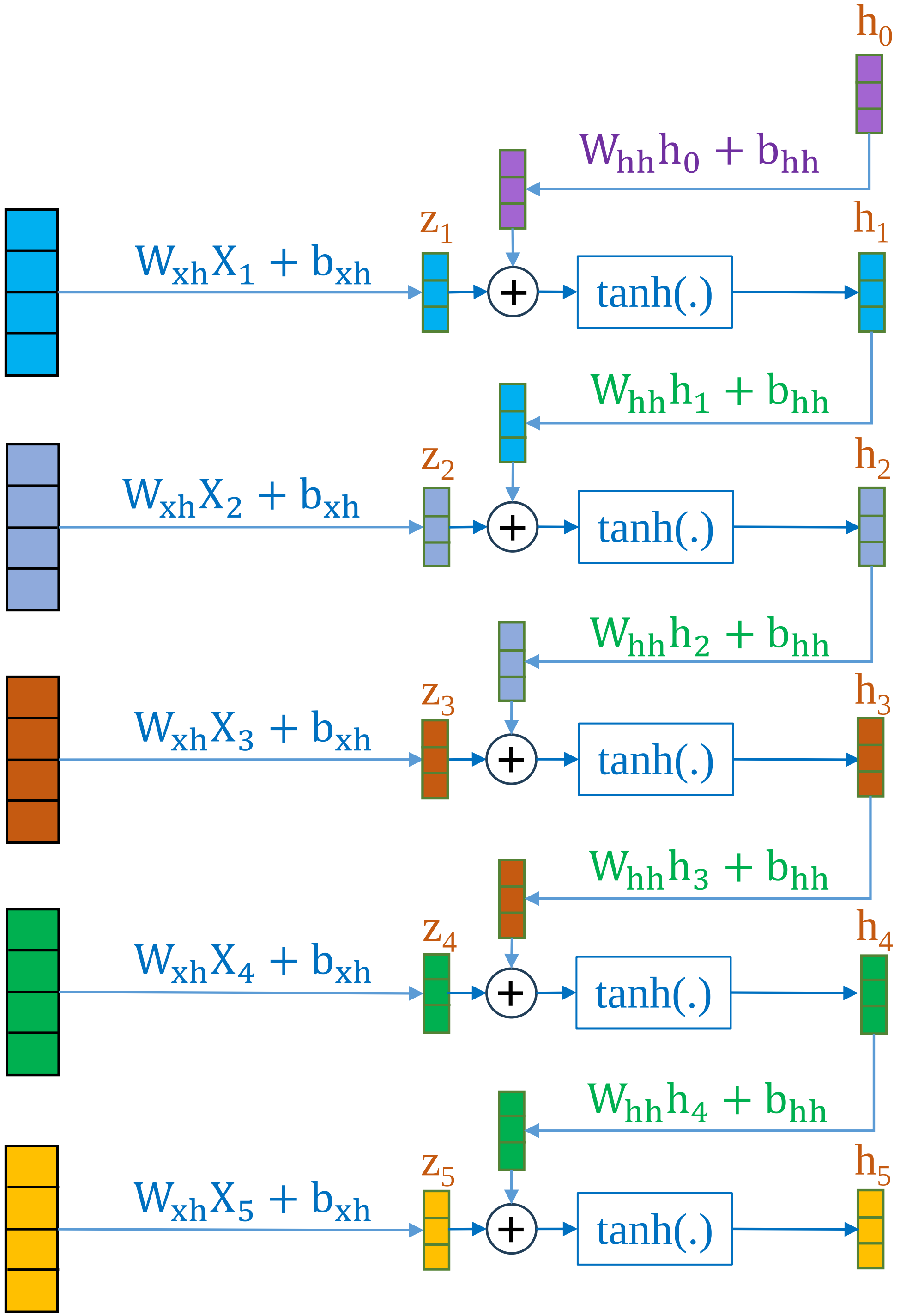
$$h_3 = \tanh(W_{xh}X_3 + b_{xh} + W_{hh}h_2 + b_{hh})$$

$$h_4 = \tanh(W_{xh}X_4 + b_{xh} + W_{hh}h_3 + b_{hh})$$

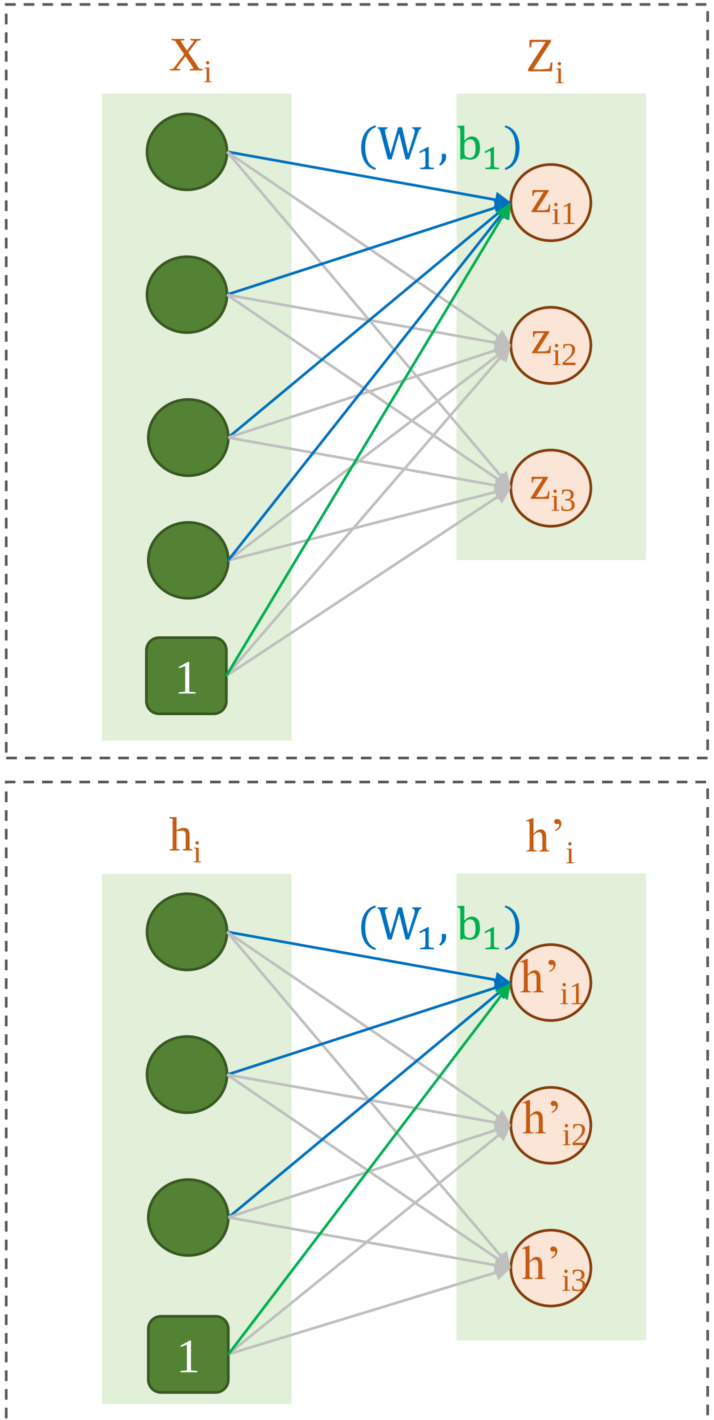
$$h_5 = \tanh(W_{xh}X_5 + b_{xh} + W_{hh}h_4 + b_{hh})$$

$$\Rightarrow h_t = \tanh\left(W_{xh}X_t + b_{xh} + W_{hh}h_{(t-1)} + b_{hh}\right)$$

RNN



Discussion



Stack of RNNs

❖ Recurrent Neural Networks (RNNs)

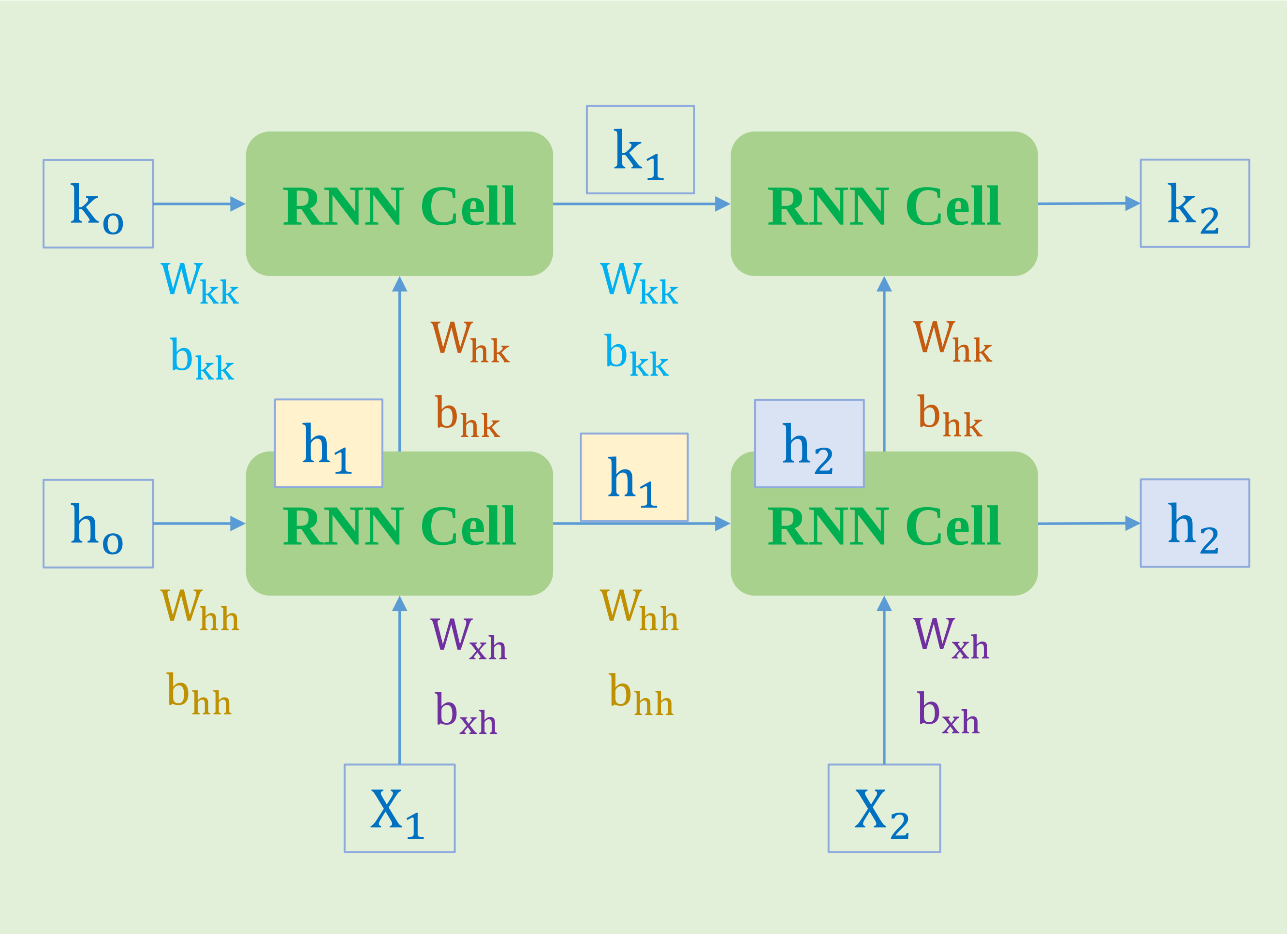
❖ Two layers

$$k_1 = \tanh(W_{hk}h_1 + b_{hk} + W_{kk}k_0 + b_{kk})$$

$$k_2 = \tanh(W_{hk}h_2 + b_{hk} + W_{kk}k_1 + b_{kk})$$

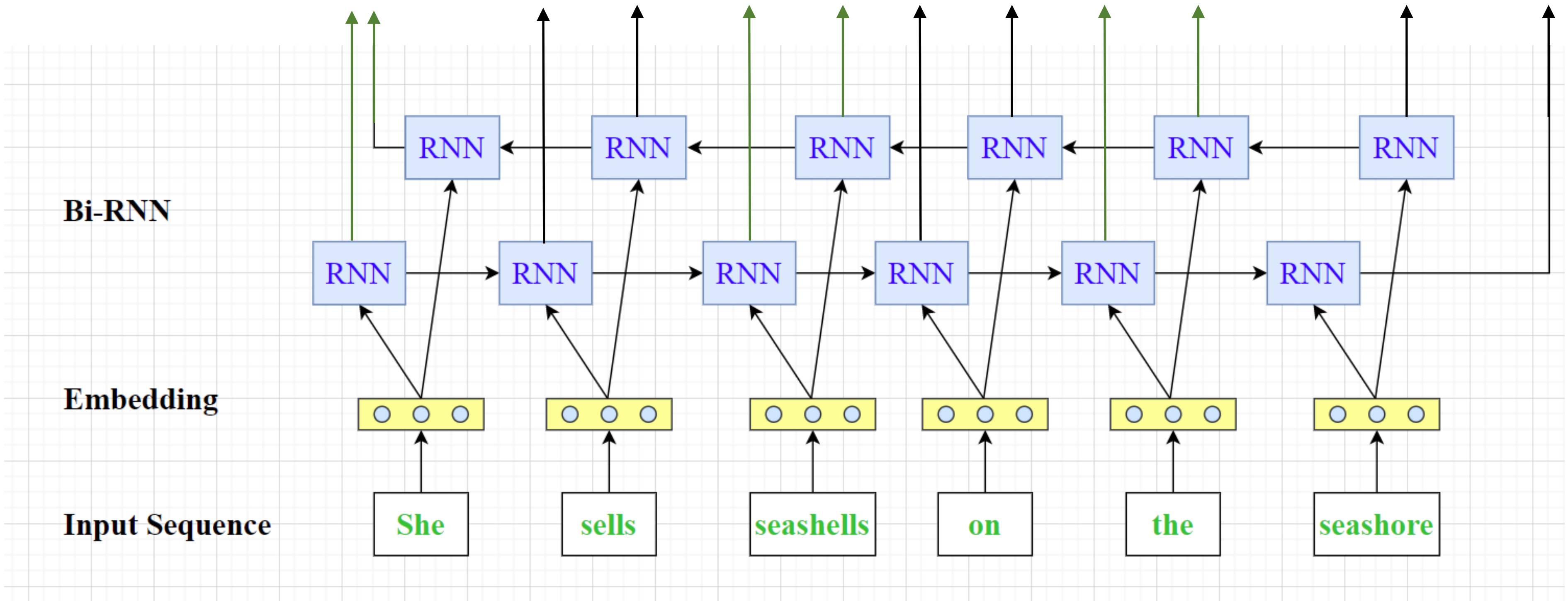
$$h_1 = \tanh(W_{xh}X_1 + b_{xh} + W_{hh}h_0 + b_{hh})$$

$$h_2 = \tanh(W_{xh}X_2 + b_{xh} + W_{hh}h_1 + b_{hh})$$



RNNs

❖ Bidirectional RNNs

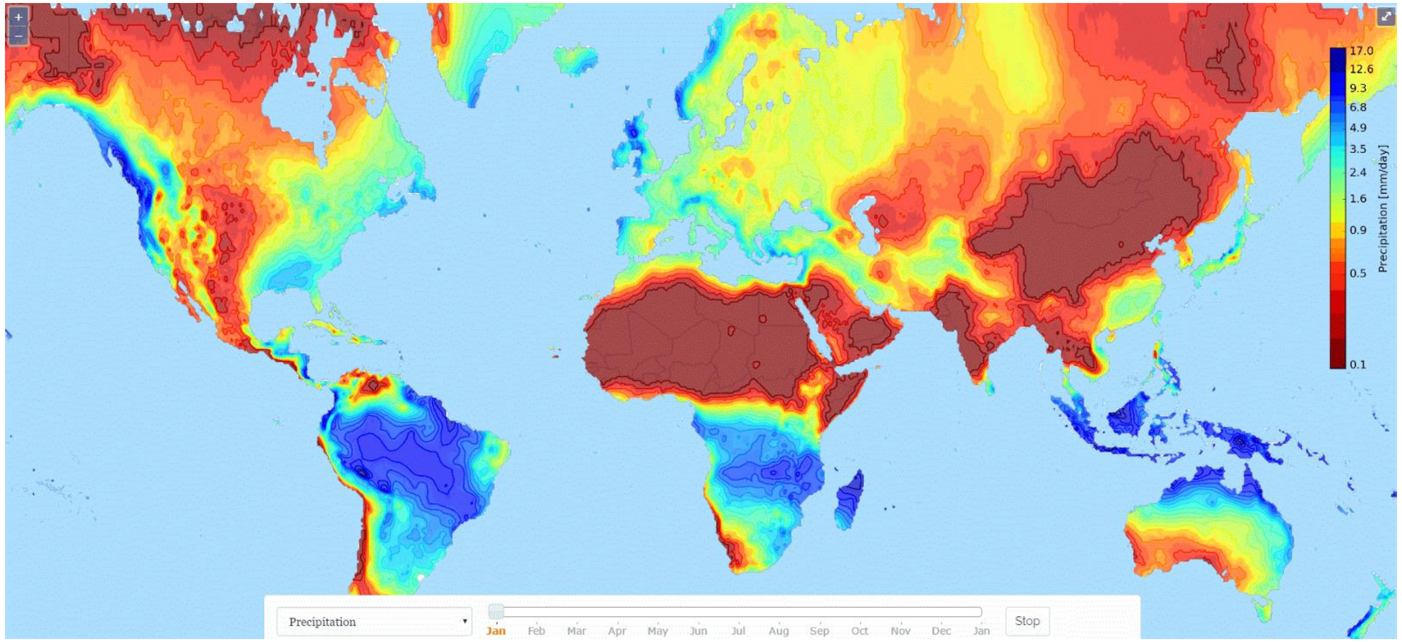


Outline

- RNN in PyTorch
- RNNs for Time-Series Data
- RNNs for IMDB dataset
- From RNN to LSTM
- LSTM Applications

Weather Forecasting

❖ Introduction



Predict future temperature in weather forecasting

Weather Forecasting

❖ Introduction

Problem Statement: Given temperature from the previous 5 hours (including the current one), predict temperature of the next 1 hour.

Hour	**13:00**	**14:00**	**15:00**	**16:00**	**17:00**	**18:00**	**19:00**	**20:00**
Condition								
Temperature	**32**	**31**	**31**	**30**	**29**	**26**	**25**	

Time-series Data

Temperature forecasting

Date	Summary	Precip Type	Temperature (C)	Apparent Temperature (C)	Humidity	Wind Speed (km/h)	Wind Bearing (degrees)	Visibility (km)
2006-04-01 00	Partly Cloudy	rain	9.472222222	7.388888889	0.89	14.1197	251	15.8263
2006-04-01 01	Partly Cloudy	rain	9.355555556	7.227777778	0.86	14.2646	259	15.8263
2006-04-01 02	Mostly Cloudy	rain	9.377777778	9.377777778	0.89	3.9284	204	14.9569
2006-04-01 03	Partly Cloudy	rain	8.288888889	5.944444444	0.83	14.1036	269	15.8263
2006-04-01 04	Mostly Cloudy	rain	8.755555556	6.977777778	0.83	11.0446	259	15.8263
2006-04-01 05	Partly Cloudy	rain	9.222222222	7.111111111	0.85	13.9587	258	14.9569
2006-04-01 06	Partly Cloudy	rain	7.733333333	5.522222222	0.95	12.3648	259	9.982
2006-04-01 07	Partly Cloudy	rain	8.772222222	6.527777778	0.89	14.1519	260	9.982
2006-04-01 08	Partly Cloudy	rain	10.82222222	10.82222222	0.82	11.3183	259	9.982
2006-04-01 09	Partly Cloudy	rain	13.77222222	13.77222222	0.72	12.5258	279	9.982
2006-04-01 10	Partly Cloudy	rain	16.01666667	16.01666667	0.67	17.5651	290	11.2056
2006-04-01 11	Partly Cloudy	rain	17.14444444	17.14444444	0.54	19.7869	316	11.4471
2006-04-01 12	Partly Cloudy	rain	17.8	17.8	0.55	21.9443	281	11.27
2006-04-01 13	Partly Cloudy	rain	17.33333333	17.33333333	0.51	20.6885	289	11.27
2006-04-01 14	Partly Cloudy	rain	18.87777778	18.87777778	0.47	15.3755	262	11.4471
2006-04-01 15	Partly Cloudy	rain	18.91111111	18.91111111	0.46	10.4006	288	11.27
2006-04-01 16	Partly Cloudy	rain	15.38888889	15.38888889	0.6	14.4095	251	11.27
2006-04-01 17	Mostly Cloudy	rain	15.55	15.55	0.63	11.1573	230	11.4471
2006-04-01 18	Mostly Cloudy	rain	14.25555556	14.25555556	0.69	8.5169	163	11.2056
2006-04-01 19	Mostly Cloudy	rain	13.14444444	13.14444444	0.7	7.6314	139	11.2056
2006-04-01 20	Mostly Cloudy	rain	11.55	11.55	0.77	7.3899	147	11.0285
2006-04-01 21	Mostly Cloudy	rain	11.18333333	11.18333333	0.76	4.9266	160	9.982
2006-04-01 22	Partly Cloudy	rain	10.11666667	10.11666667	0.79	6.6493	163	15.8263
2006-04-01 23	Mostly Cloudy	rain	10.2	10.2	0.77	3.9284	152	14.9569
2006-04-10 00	Partly Cloudy	rain	10.42222222	10.42222222	0.62	16.9855	150	15.8263
2006-04-10 01	Partly Cloudy	rain	9.911111111	7.566666667	0.66	17.2109	149	15.8263
2006-04-10 02	Mostly Cloudy	rain	11.18333333	11.18333333	0.8	10.8192	163	14.9569
2006-04-10 03	Partly Cloudy	rain	7.155555556	5.044444444	0.79	11.0768	180	15.8263
2006-04-10 04	Partly Cloudy	rain	6.111111111	4.816666667	0.82	6.6493	161	15.8263

Weather Forecasting

❖ Introduction

Time	Temperature (C)
2006-04-01 00:00:00.000 +0200	9.472222
2006-04-01 01:00:00.000 +0200	9.355556
2006-04-01 02:00:00.000 +0200	9.377778
2006-04-01 03:00:00.000 +0200	8.288889
2006-04-01 04:00:00.000 +0200	8.755556
2006-04-01 05:00:00.000 +0200	9.222222

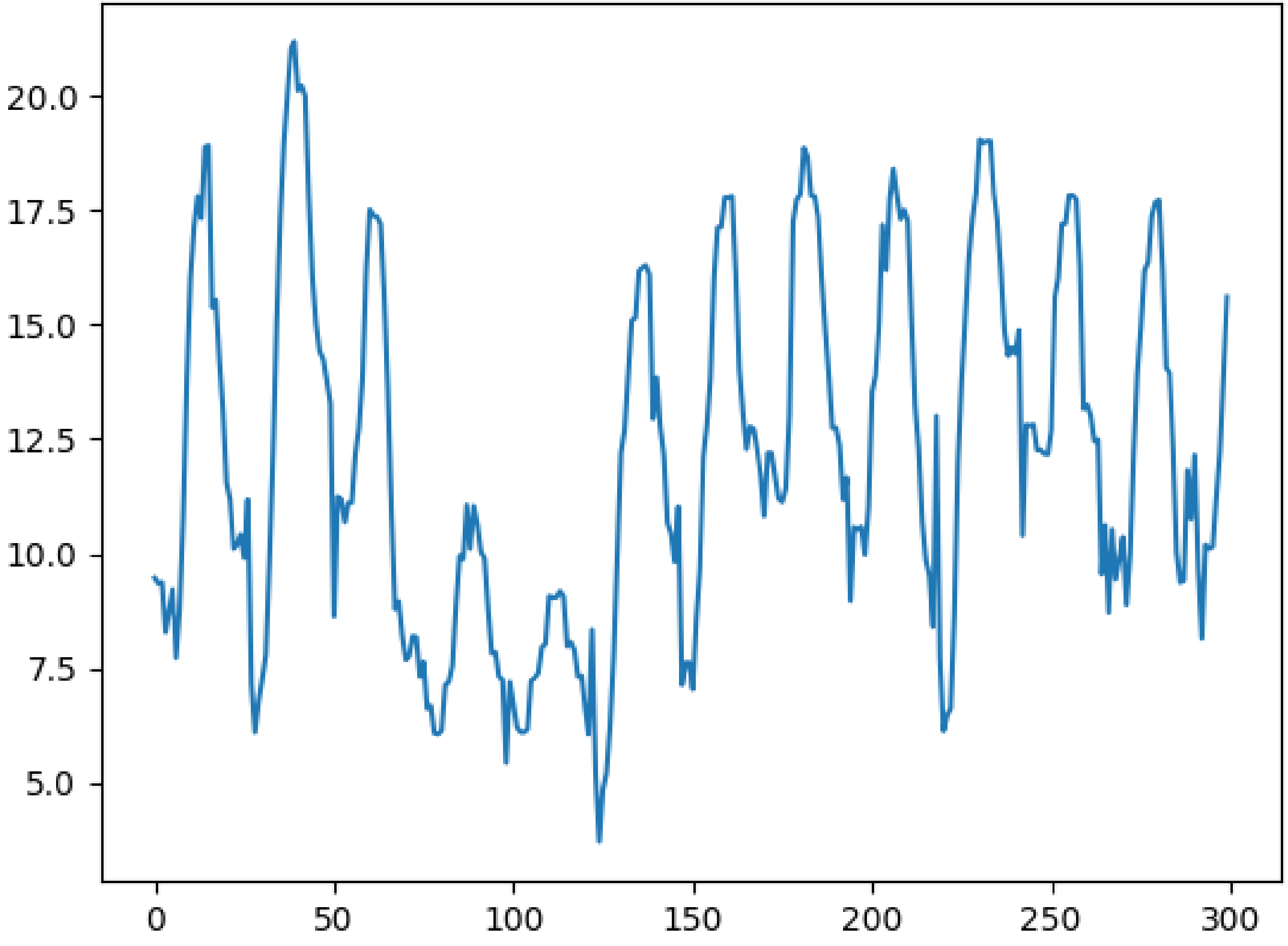
X

y

Temperature forecasting datatable

Time-series Data

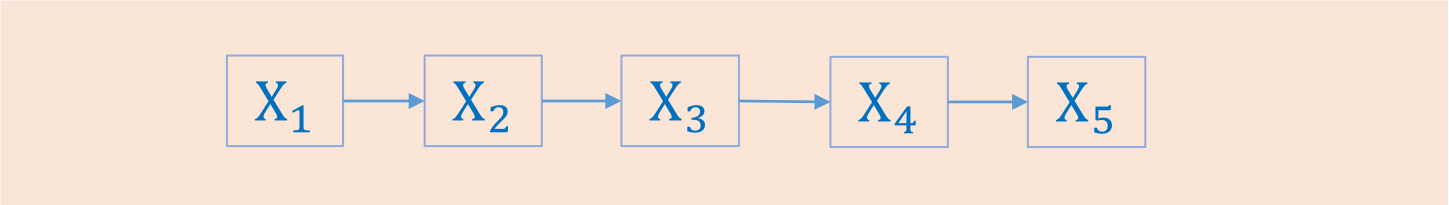
Temperature forecasting

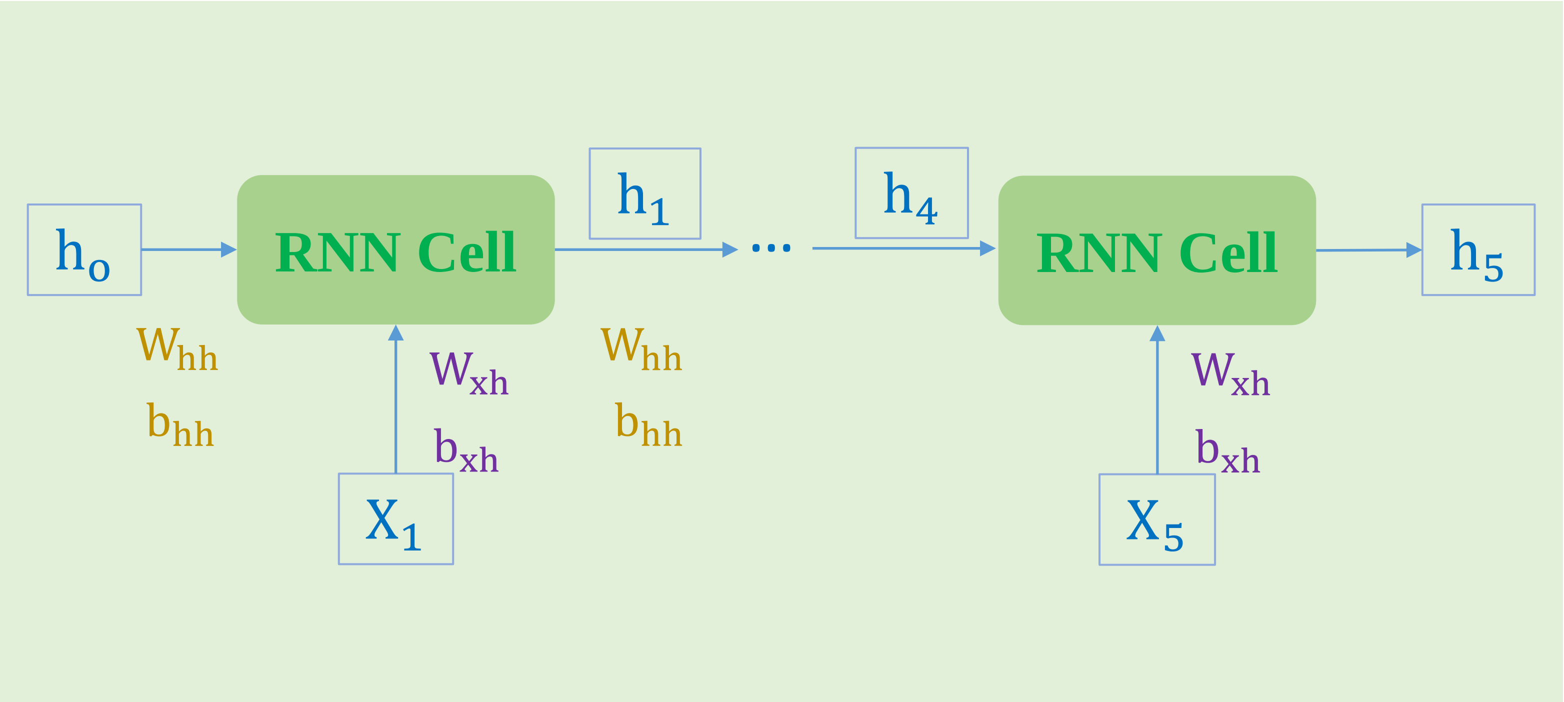


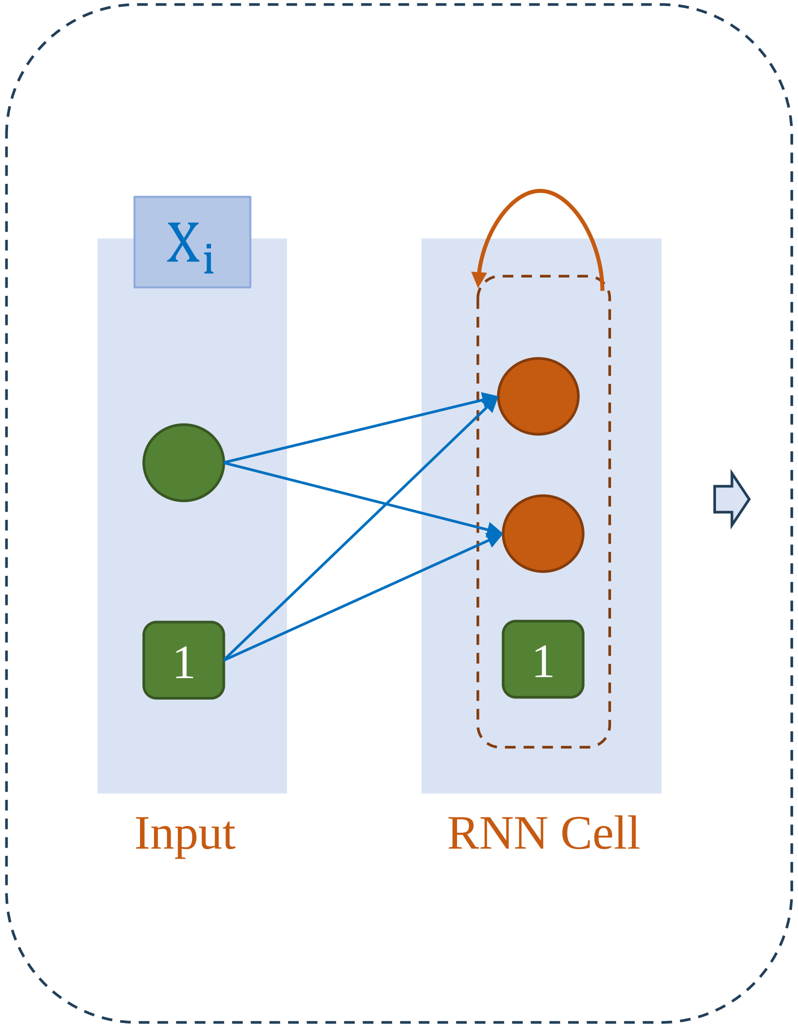
Date	Temperature (C)
2006-04-01 00	9.472222222
2006-04-01 01	9.355555556
2006-04-01 02	9.377777778
2006-04-01 03	8.288888889
2006-04-01 04	8.755555556
2006-04-01 05	9.222222222
2006-04-01 06	7.733333333
2006-04-01 07	8.772222222
2006-04-01 08	10.82222222
2006-04-01 09	13.77222222
2006-04-01 10	16.01666667
2006-04-01 11	17.14444444
2006-04-01 12	17.8
2006-04-01 13	17.33333333
2006-04-01 14	18.87777778
2006-04-01 15	18.91111111
2006-04-01 16	15.38888889
2006-04-01 17	15.55
2006-04-01 18	14.25555556
2006-04-01 19	13.14444444
2006-04-01 20	11.55
2006-04-01 21	11.18333333
2006-04-01 22	10.11666667
2006-04-01 23	10.2

Time-series Data

Temperature forecasting







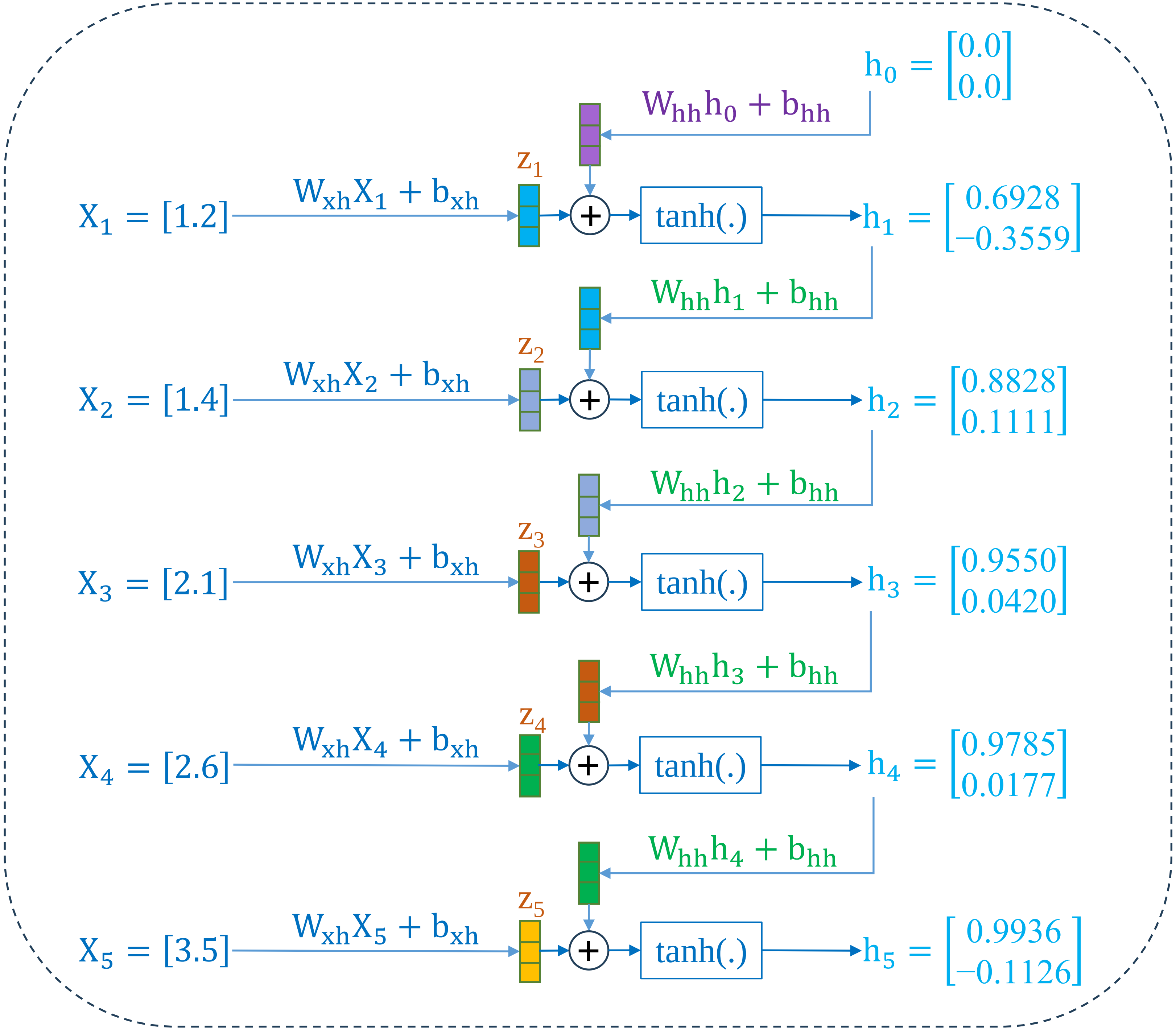
Example

$$W_{xh} = \begin{bmatrix} 0.6584 \\ -0.1671 \end{bmatrix}$$

$$b_{xh} = \begin{bmatrix} -0.5966 \\ 0.0945 \end{bmatrix}$$

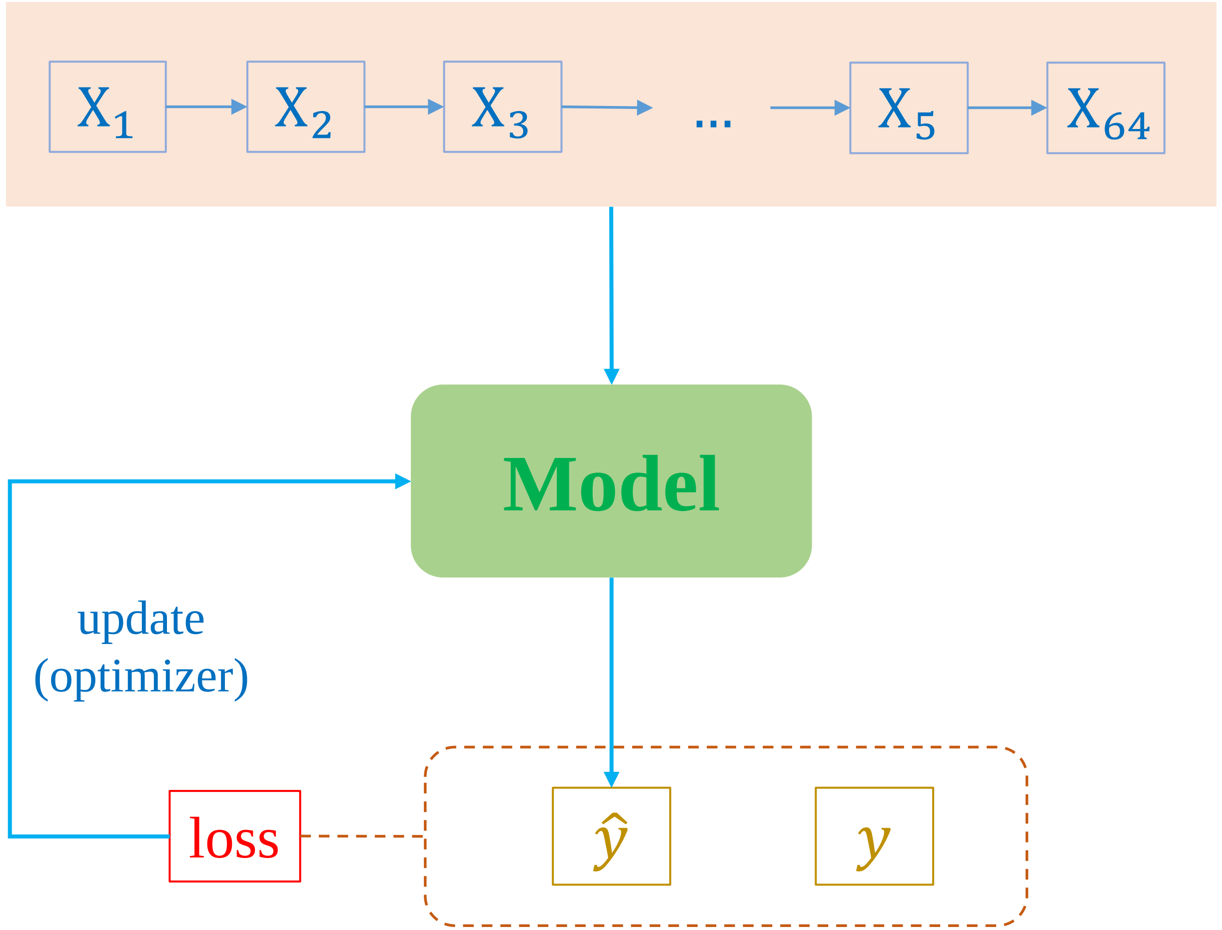
$$W_{hh} = \begin{bmatrix} 0.5147 & -0.1310 \\ 0.6606 & -0.1671 \end{bmatrix}$$

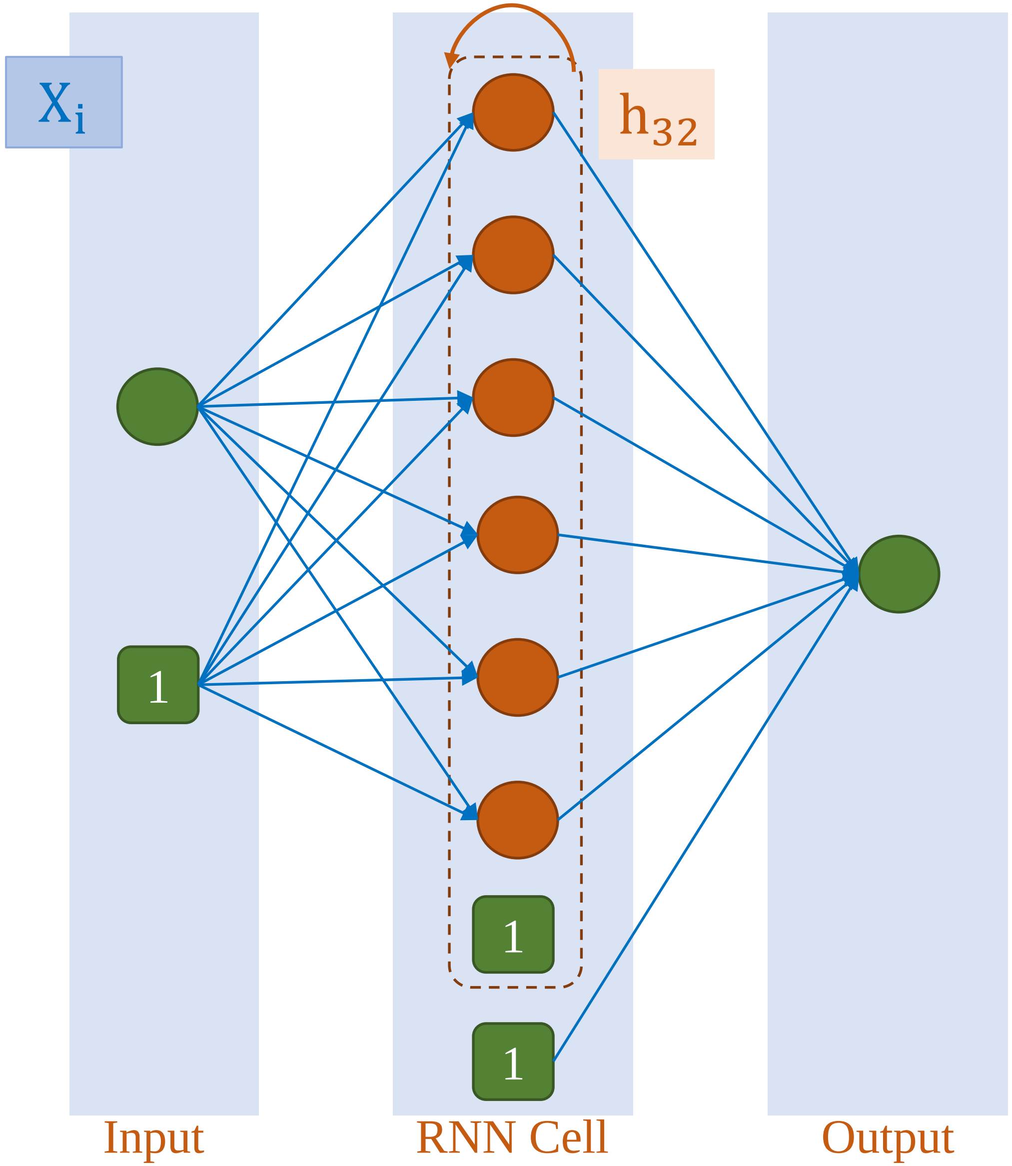
$$b_{hh} = \begin{bmatrix} 0.6599 \\ -0.2662 \end{bmatrix}$$



Implementation

Temperature forecasting





Implementation

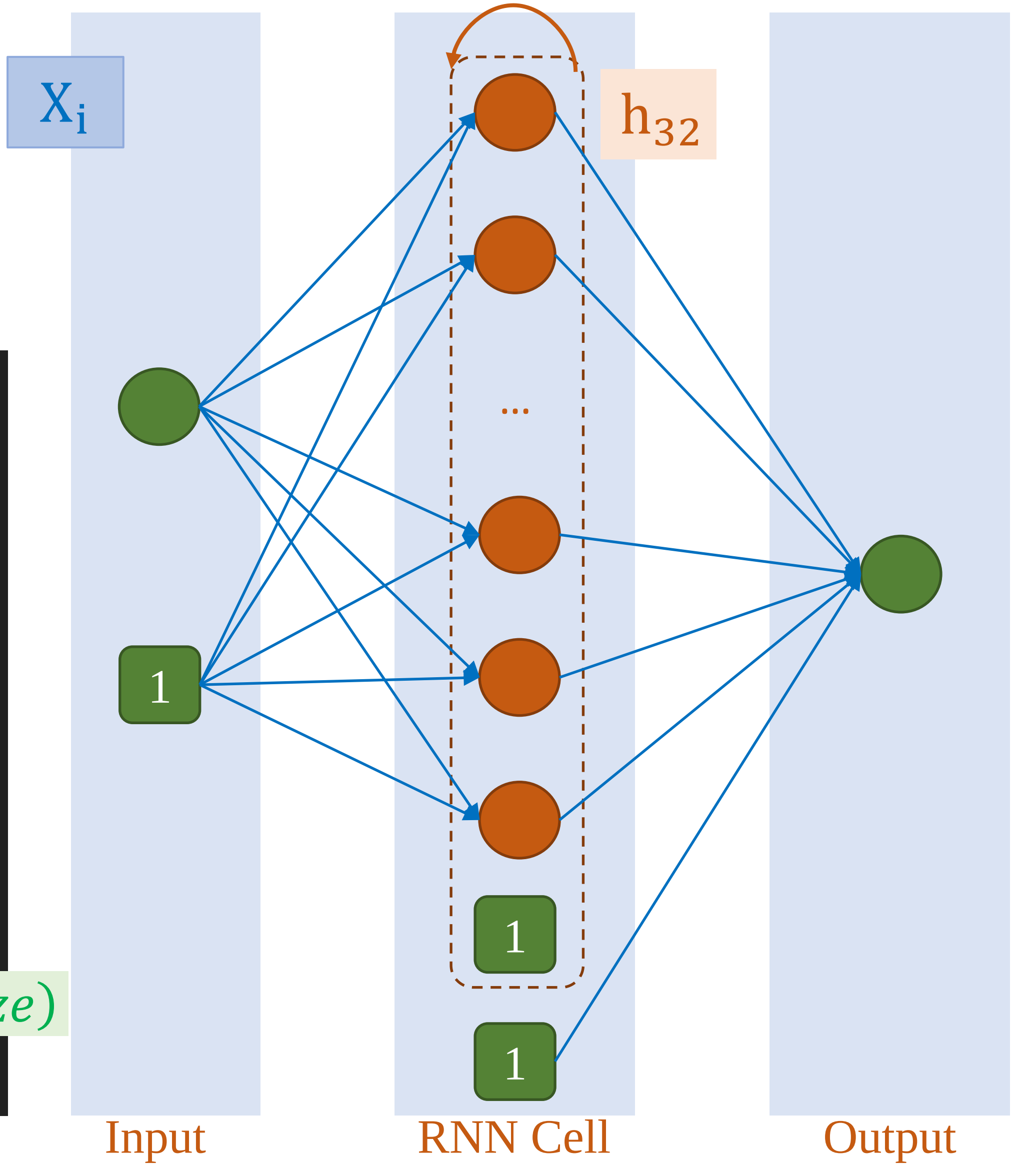
Back to Temperature forecasting

sequence_length = 64   embed_dim = 1

output_dim = 1   hidden_dim = 32

```
class RNNModel(nn.Module):
    def __init__(self, hidden_dim, output_dim):
        super(RNNModel, self).__init__()
        self.rnn = nn.RNN(1, hidden_dim, batch_first=True)
        self.fc = nn.Linear(hidden_dim, output_dim)

    def forward(self, x):
        output_rnn, hidden_rnn = self.rnn(x)
        last_hidden = hidden_rnn[-1,:,:]
        output = self.fc(last_hidden)
        return output

model = RNNModel(hidden_dim=32, output_dim=1)
```

$(num\_rnn\_layers, N, hidden\_size)$



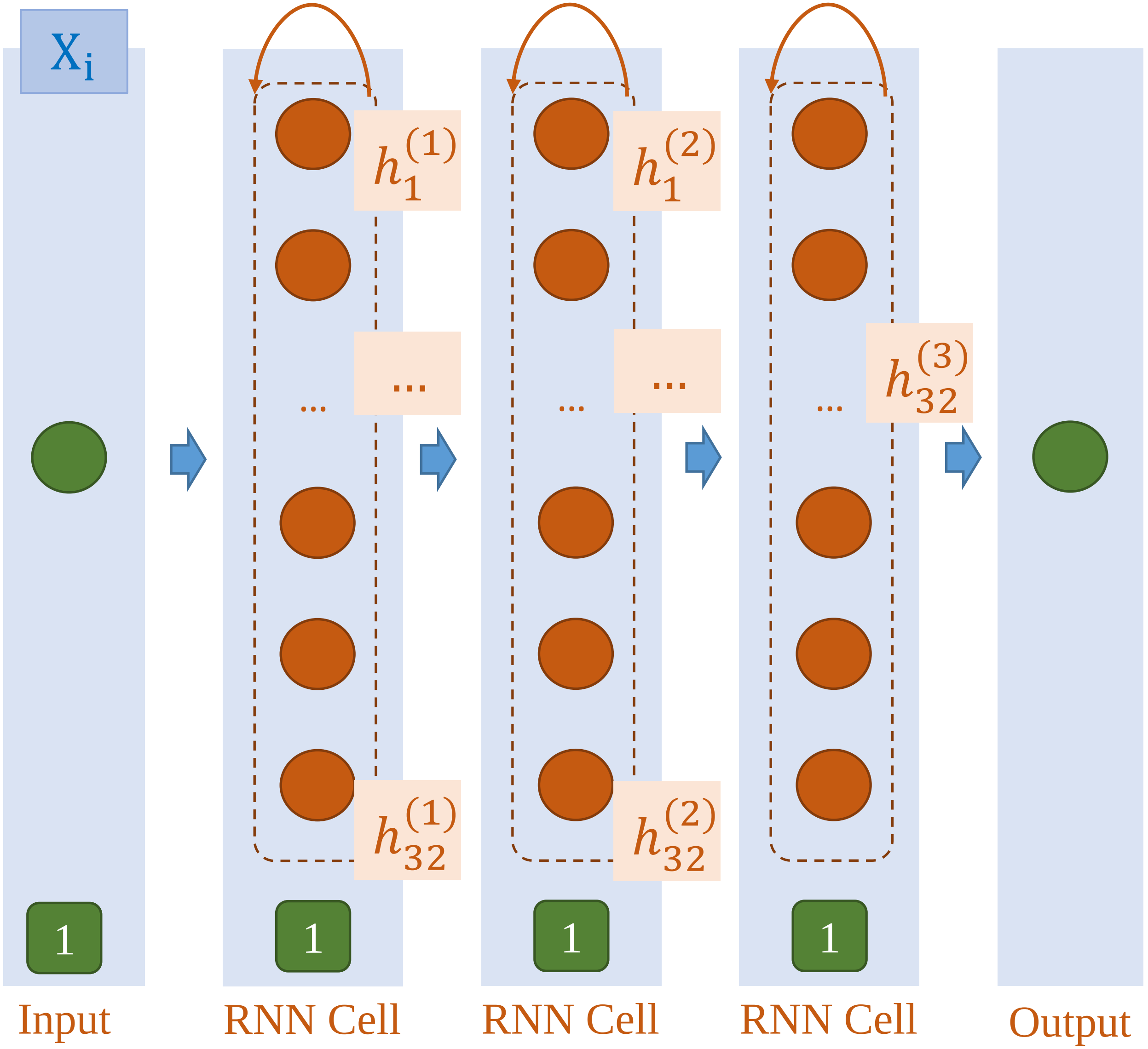
Implementation

Stack of three RNNs

sequence_length = 64 embed_dim = 1

output_dim = 1 hidden_dim = 32

```
class RNNModel(nn.Module):
    def __init__(self, hidden_dim, output_dim):
        super(RNNModel, self).__init__()
        self.rnn = nn.RNN(1, hidden_dim,
                          num_layers=3,
                          batch_first=True)
        self.fc = nn.Linear(hidden_dim, output_dim)

    def forward(self, x):
        output_rnn, hidden_rnn = self.rnn(x)
        last_hidden = hidden_rnn[-1,:,:]
        output = self.fc(last_hidden)
        return output

model = RNNModel(hidden_dim=32, output_dim=1)
```



Implementation

Data preparation

```python
def prepare_data(data, lag, ahead, train_ratio, batch_size):
    # Create sequences
    X, y = create_sequences(data, lag, ahead)

    # Flatten all the features of a sample for RNN
    X = X.reshape(X.shape[0], -1, 1)

    # Split the data
    train_size = int(len(X) * train_ratio)
    X_train, X_test = X[:train_size], X[train_size:]
    y_train, y_test = y[:train_size], y[train_size:]

    # Convert to PyTorch tensors
    X_train_tensor = torch.tensor(X_train, dtype=torch.float32)
    y_train_tensor = torch.tensor(y_train, dtype=torch.float32)
    X_test_tensor = torch.tensor(X_test, dtype=torch.float32)
    y_test_tensor = torch.tensor(y_test, dtype=torch.float32)

    # ...
```

(N, L, H_{in}) when `batch_first=True`

```python
def create_sequences(data, lag, ahead):
    X, y = [], []
    for i in range(len(data) - lag - ahead + 1):
        X.append(data[i:(i + lag)])
        y.append(data[(i + lag):(i + lag + ahead)])
    return np.array(X), np.array(y)
```

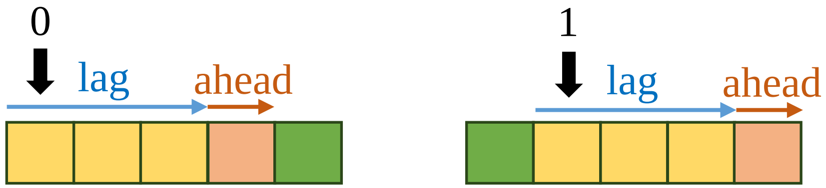
```python
X, y = create_sequences(data, lag, ahead)
print(X.shape)
print(y.shape)
```

```
(96389, 64)
(96389, 1)
```

train_data_length = 5

lag = sequence_length = 3

ahead = 1



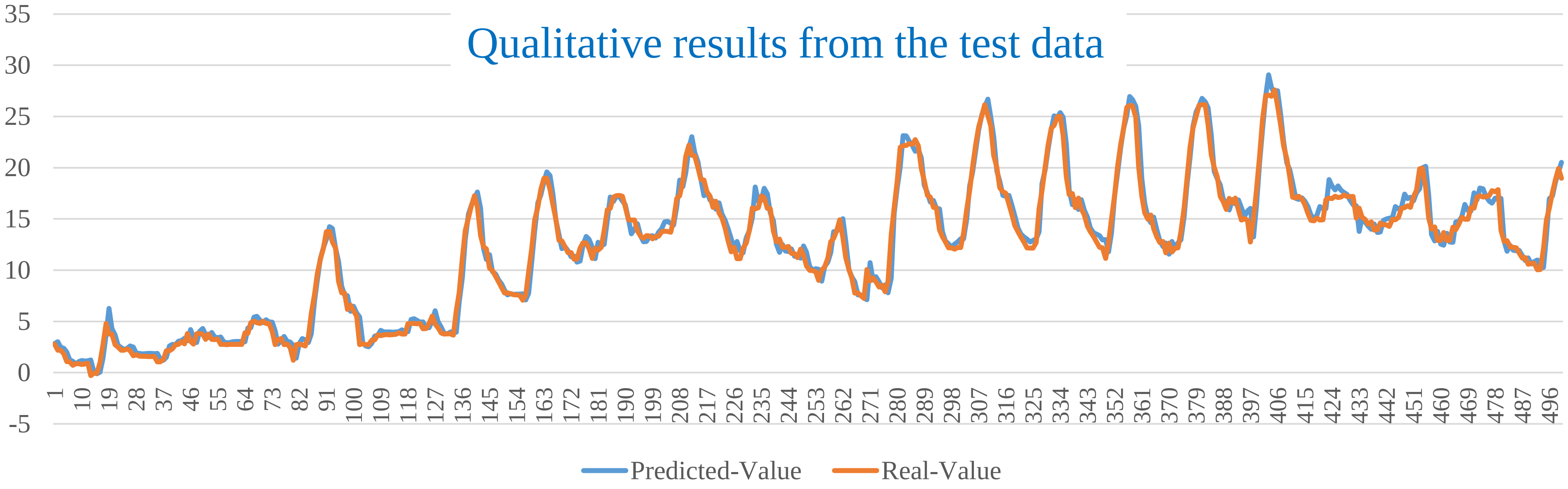
range(5-3-1+1) = range(2) → 0, 1

Implementation

Train

```python
criterion = nn.MSELoss()
optimizer = Adam(model.parameters(),
                 lr=lr)
```

```python
def train_model(model, criterion, optimizer, train_loader, num_epochs):
    for epoch in range(num_epochs):
        model.train()
        for i, (sequences, labels) in enumerate(train_loader):
            sequences, labels = sequences.to(device), labels.to(device)
            # Forward pass
            outputs = model(sequences)
            loss = criterion(outputs, labels)
            # Backward and optimize
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
    return model, losses
```



R2 Score: 0.984
MAE: 0.71
MSE: 1.23

Common Metrics for TS Data

y	11.18	11.66	8.97

$\hat{y}$	-0.33	-0.61	1.35

$y - \hat{y}$	11.51	12.27	7.62

$(y - \hat{y})^2$	132.48	150.55	50.06

$$\bar{y} = \frac{1}{N}\sum_{i=1}^{N} y_i$$

$$\bar{y} = \frac{1}{3}(11.18 + 11.66 + 8.97)$$
$$= 10.6$$

$y - \bar{y}$	0.58	1.06	-1.63

$(y - \bar{y})^2$	0.336	1.123	2.656

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

$$MSE = \frac{1}{3}(132.48 + 150.55 + 50.06)$$
$$= 111.03$$

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|$$

$$MAE = \frac{1}{3}(|11.51| + |12.27| + |7.62|)$$
$$= 10.46$$

$$R^2 = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y}_i)^2}$$

$$R^2 = 1 - \frac{132.48 + 150.55 + 50.06}{0.3364 + 1.123 + 2.6569}$$
$$= -79.91$$

Outline

- RNN in PyTorch
- RNNs for Time-Series Data
- RNNs for IMDB dataset
- From RNN to LSTM
- LSTM Applications

Text Classification

❖ IMDB dataset

- 50,000 movie review for sentiment analysis
- Consist of: + 25,000 movie review for training
  + 25,000 movie review for testing
- Label: positive – negative

“A wonderful little production. The filming technique is very unassuming- very old-time-BBC fashion and gives a comforting, and sometimes discomforting, sense of realism to the entire piece…..”	positive
“This show was an amazing, fresh & innovative idea in the 70's when it first aired. The first 7 or 8 years were brilliant, but things dropped off after that. By 1990, the show was not really funny anymore, and it's continued its decline further to the complete waste of time it is today….”	negative
“I thought this was a wonderful way to spend time on a too hot summer weekend, sitting in the air conditioned theater and watching a light-hearted comedy. The plot is simplistic, but the dialogue is witty and the characters are likable (even the well bread suspected serial killer)….”	positive
“BTW Carver gets a very annoying sidekick who makes you wanna shoot him the first three minutes he's on screen.”	negative

Text Classification

❖ **IMDB dataset**

- 50,000 movie review for sentiment analysis
- Consist of: + 25,000 movie review for training
  + 25,000 movie review for testing
- Label: positive – negative

```
from datasets import load_dataset

imdb = load_dataset("imdb")
train_data, test_data = imdb['train'], imdb['test']
```

```
print(train_data.shape)
print(test_data.shape)
```

```
(25000, 2)
(25000, 2)
```

```
tokenizer = get_tokenizer("basic_english")
vocab_size = 20000

def yield_tokens(data_iter):
    for data in data_iter:
        yield tokenizer(data["text"])

vocab = build_vocab_from_iterator(yield_tokens(train_data),
                                  min_freq = 3,
                                  max_tokens=vocab_size,
                                  specials=["<pad>", "<s>", "<unk>"])
vocab.set_default_index(vocab["<unk>"])
```

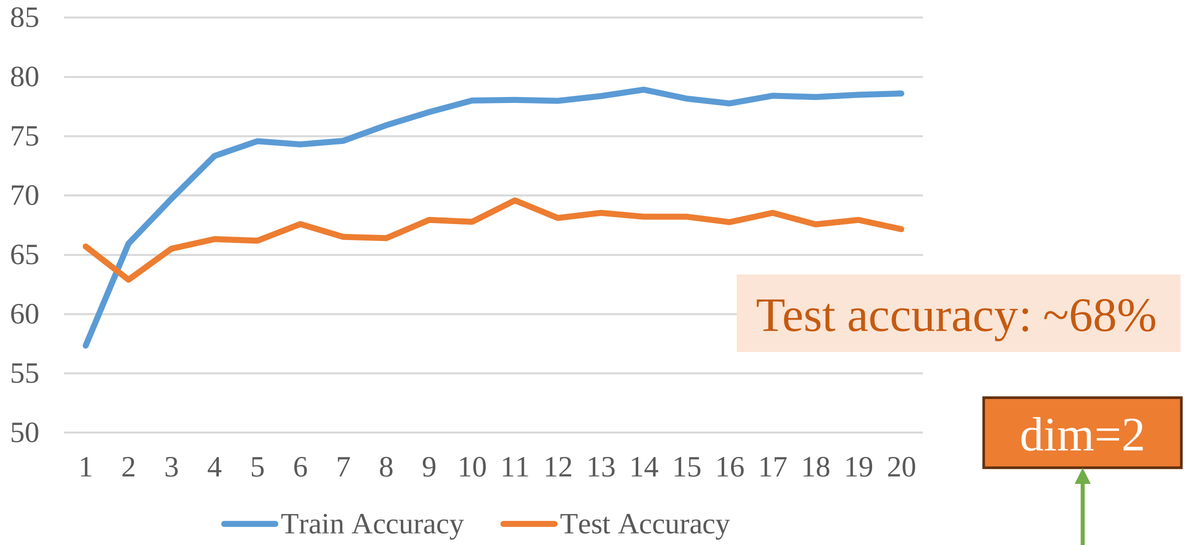
```
print(train_data[0]['text'])
```

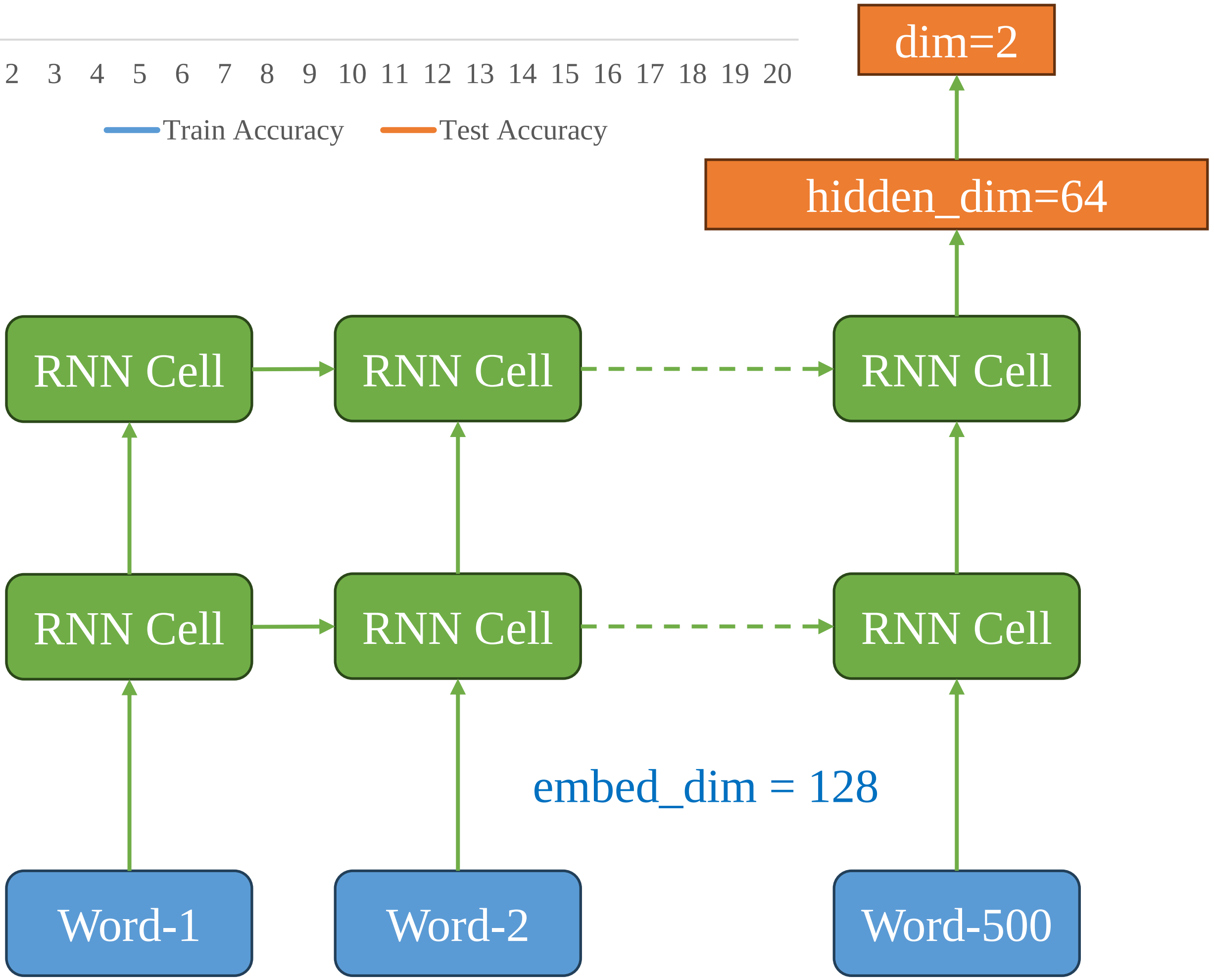
```
I rented I AM CURIOUS-YELLOW from my video st
heard that at first it was seized by U.S. cus
sial" I really had to see this for myself.<br
everything she can about life. In particular
ought about certain political issues such as
denizens of Stockholm about their opinions or
me about I AM CURIOUS-YELLOW is that 40 years
n, even then it's not shot like some cheaply
le in Swedish cinema. Even Ingmar Bergman, ar
nd the filmmakers for the fact that any sex s
o be shown in pornographic theaters in Americ
ntended) of Swedish cinema. But really, this
```

```
print(train_data[0]['label'])
```

```
0
```

Using RNN






```python
class TextClsModel(nn.Module):
    def __init__(self, vocab_size, emb_dim,
                 hidden_dim, num_layers):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, emb_dim)
        self.rnn = nn.RNN(emb_dim, hidden_dim,
                          num_layers = num_layers,
                          batch_first = True)
        self.fc = nn.Linear(hidden_dim, 2)

    def forward(self, x):
        x = self.embedding(x)
        _, hidden = self.rnn(x)
        last_hidden = hidden[-1,:,:]
        x = self.fc(last_hidden)
        return x
```

```
===============================================================
Layer (type:depth-idx)                    Output Shape
===============================================================
├─Embedding: 1-1                          [-1, 500, 128]
├─RNN: 1-2                                [-1, 500, 64]
├─Linear: 1-3                             [-1, 2]
===============================================================
```

Outline

- RNN in PyTorch
- RNNs for Time-Series Data
- RNNs for IMDB dataset
- From RNN to LSTM
- LSTM Applications

LSTM

❖ Construction

$$h_0 = \mathbf{0} \qquad b_{hh} = \mathbf{0}$$

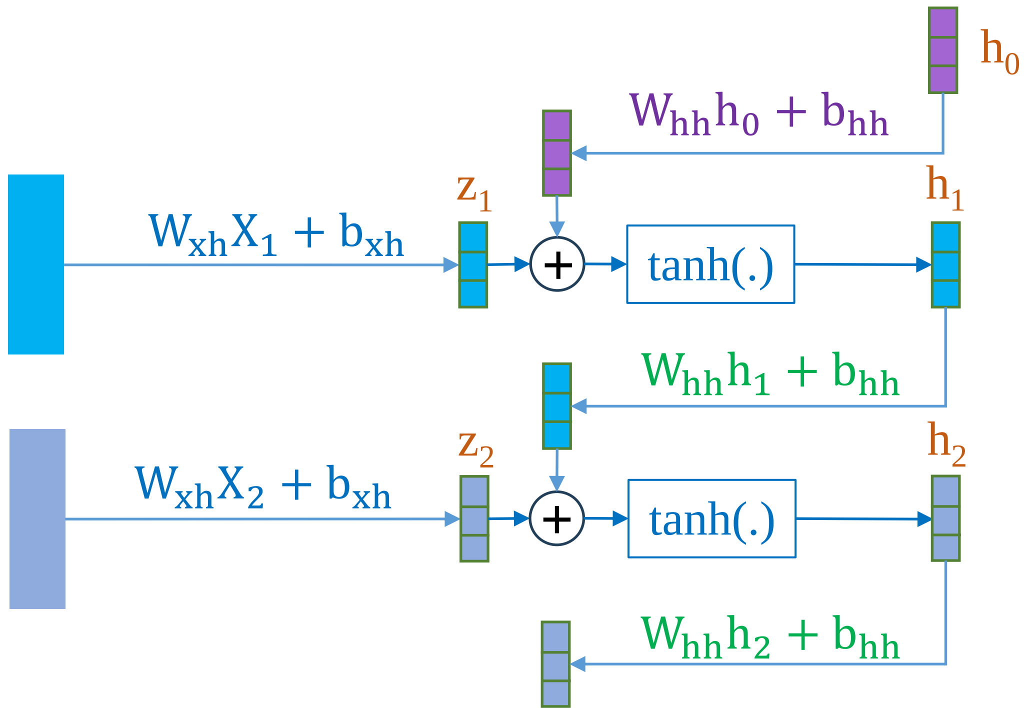
$$h_1 = \tanh(W_{xh}X_1 + b_{xh} + W_{hh}h_0 + b_{hh})$$

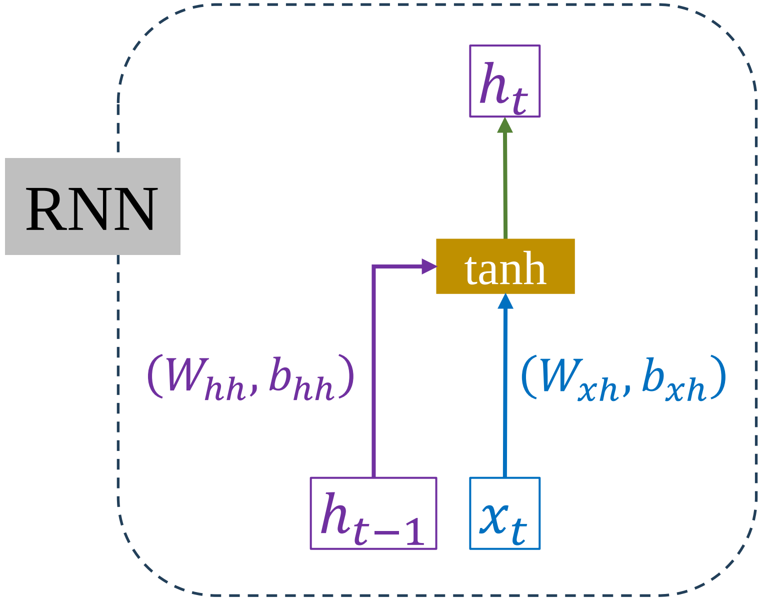
$$h_2 = \tanh(W_{xh}X_2 + b_{xh} + W_{hh}h_1 + b_{hh})$$

$$h_3 = \tanh(W_{xh}X_3 + b_{xh} + W_{hh}h_2 + b_{hh})$$

$$h_4 = \tanh(W_{xh}X_4 + b_{xh} + W_{hh}h_3 + b_{hh})$$

$$h_5 = \tanh(W_{xh}X_5 + b_{xh} + W_{hh}h_4 + b_{hh})$$



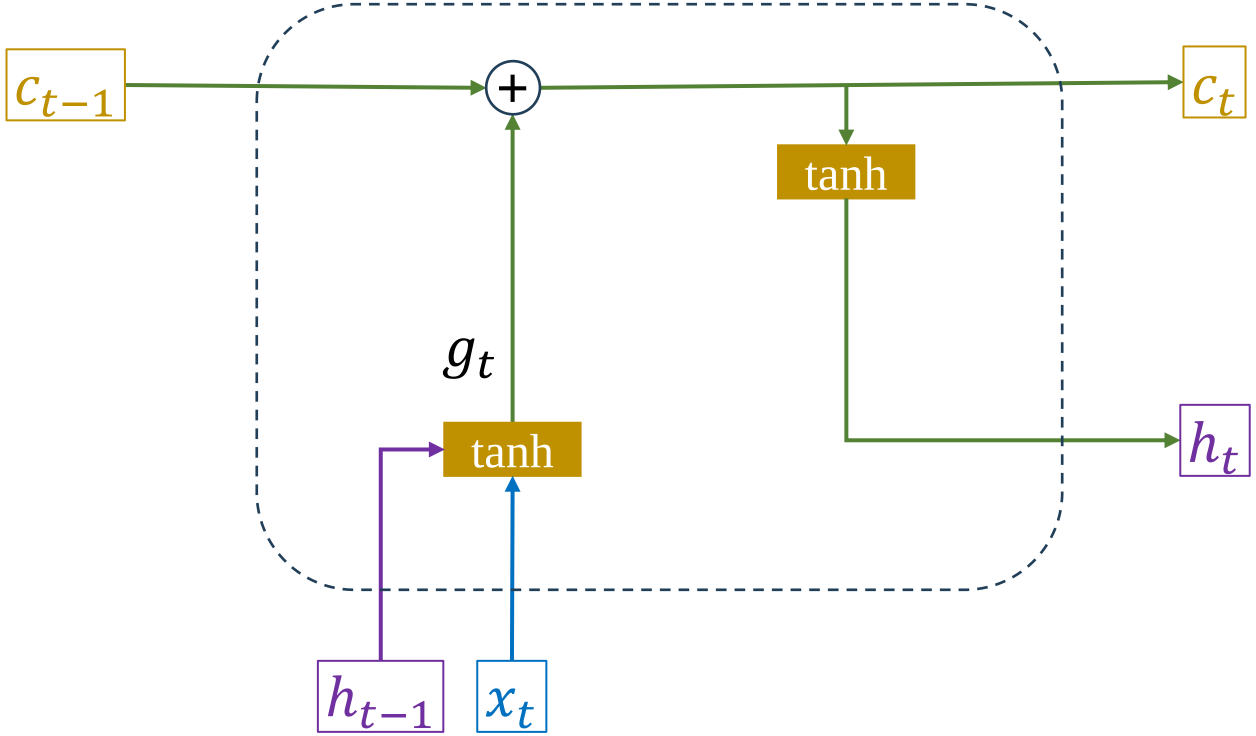


$$h_t = \tanh\left(W_{xh}X_t + b_{xh} + W_{hh}h_{(t-1)} + b_{hh}\right)$$

LSTM

❖ Construction

$h_0 = \mathbf{0}$ $\quad b_{..} = \mathbf{0}$ $\quad c_0 = \mathbf{0}$



$$g_t = \tanh\left(W_{ig}x_t + b_{ig} + W_{hg}h_{(t-1)} + b_{hg}\right)$$

$$c_t = g_t + c_{t-1}$$

$$h_t = \tanh(c_t)$$

➡ $c_1 = g_1$

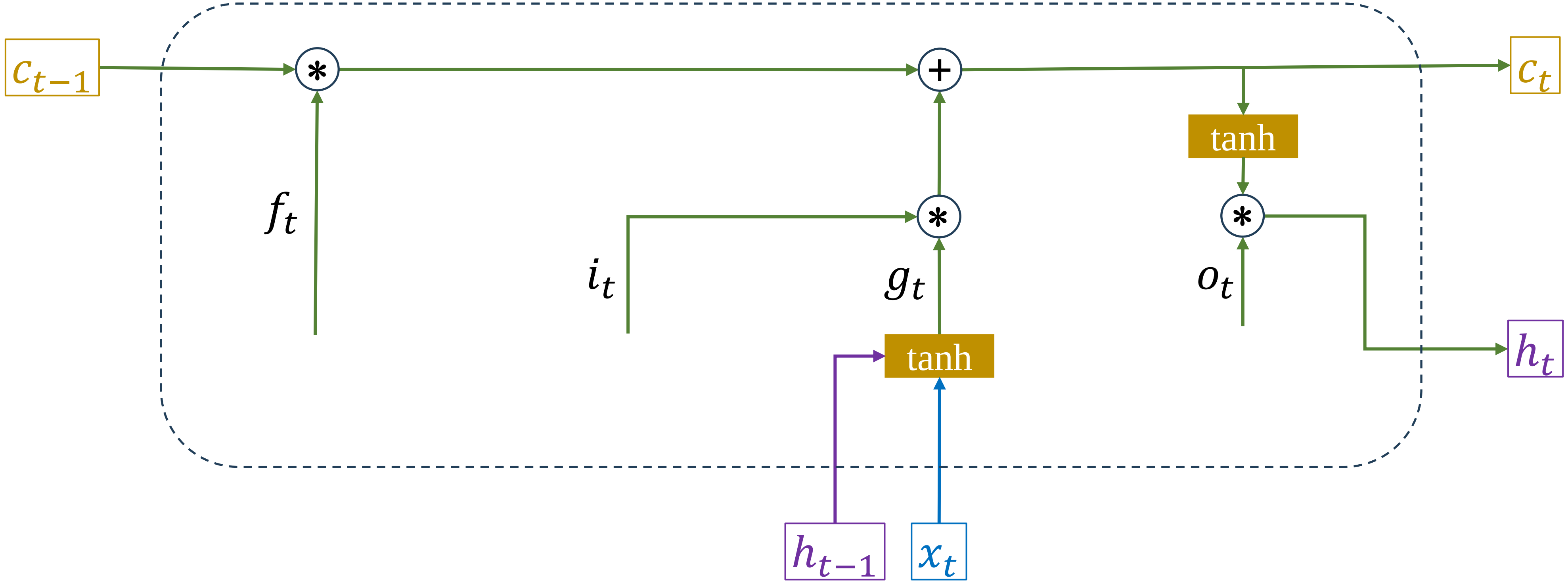
LSTM

❖ Construction

$$i_t = \sigma(\mathrm{W_{ii}x_t} + \mathrm{b_{ii}} + \mathrm{W_{hi}h_{(t-1)}} + \mathrm{b_{hi}})$$
$$f_t = \sigma(\mathrm{W_{if}x_t} + \mathrm{b_{if}} + \mathrm{W_{hf}h_{(t-1)}} + \mathrm{b_{hf}})$$
$$o_t = \sigma(\mathrm{W_{io}x_t} + \mathrm{b_{io}} + \mathrm{W_{ho}h_{(t-1)}} + \mathrm{b_{ho}})$$

$\mathrm{h_0} = \mathbf{0}$

$\mathrm{b_{..}} = \mathbf{0}$ $\quad$ $\mathrm{c_0} = \mathbf{0}$



LSTM

❖ Construction

$\mathrm{h}_0 = \mathbf{0}$

$\mathrm{b}_{..} = \mathbf{0}$ $\quad$ $\mathrm{c}_0 = \mathbf{0}$

$$i_t = \sigma\left(\mathrm{W_{ii}x_t + b_{ii} + W_{hi}h_{(t-1)} + b_{hi}}\right)$$

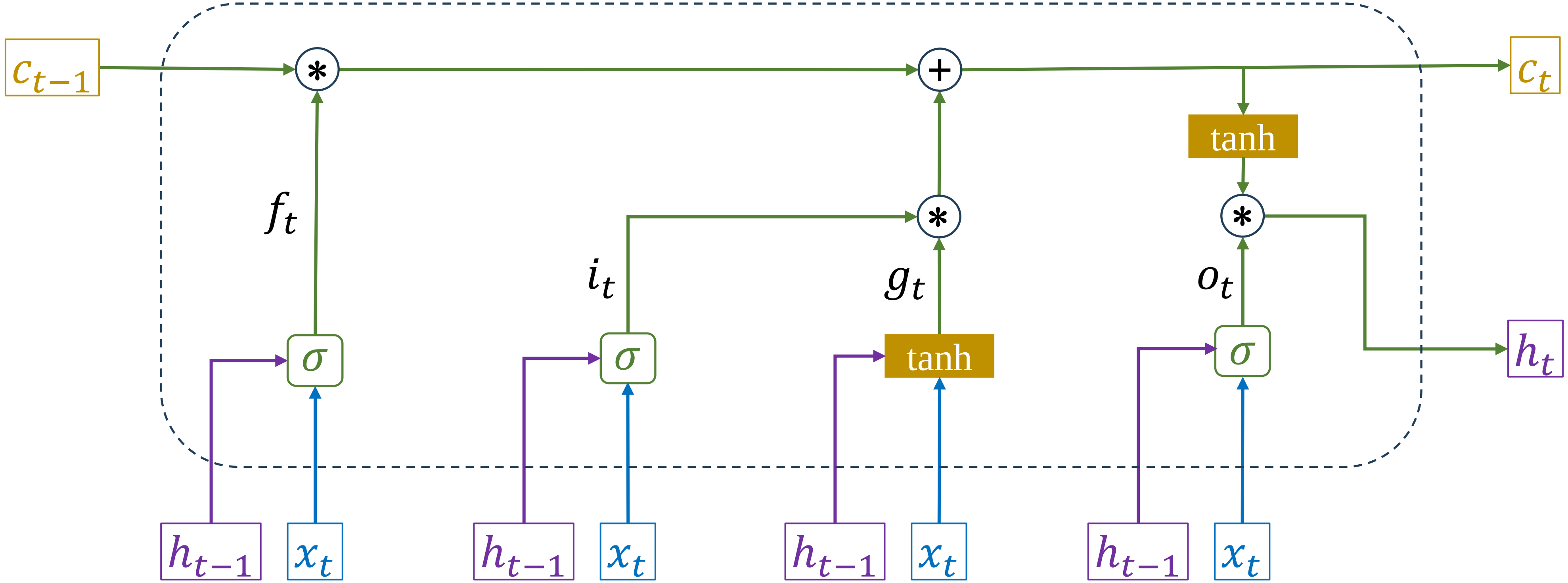
$$f_t = \sigma\left(\mathrm{W_{if}x_t + b_{if} + W_{hf}h_{(t-1)} + b_{hf}}\right)$$

$$o_t = \sigma\left(\mathrm{W_{io}x_t + b_{io} + W_{ho}h_{(t-1)} + b_{ho}}\right)$$

$$g_t = \tanh\left(\mathrm{W_{ig}x_t + b_{ig} + W_{hg}h_{(t-1)} + b_{hg}}\right)$$

$$c_t = f_t \odot g_t + i_t \odot c_{t-1}$$

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



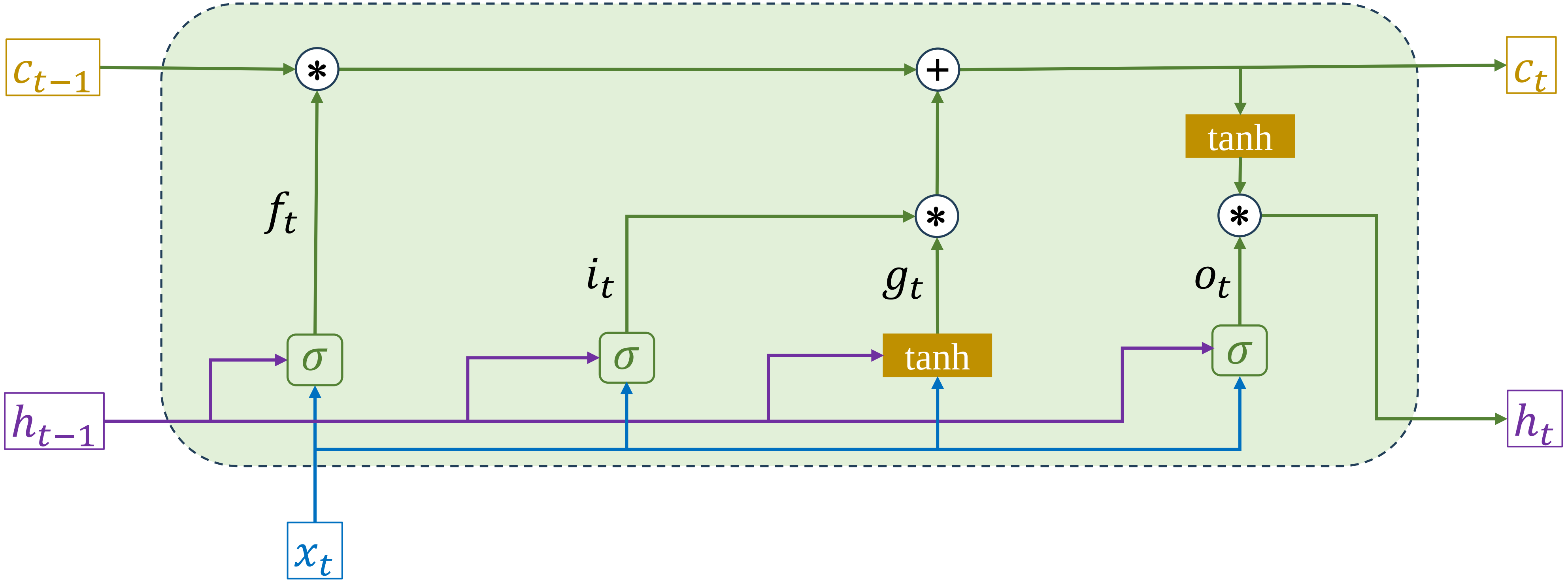
LSTM

❖ **Construction**

$h_0 = \mathbf{0}$

$b_{..} = \mathbf{0}$ $\quad c_0 = \mathbf{0}$

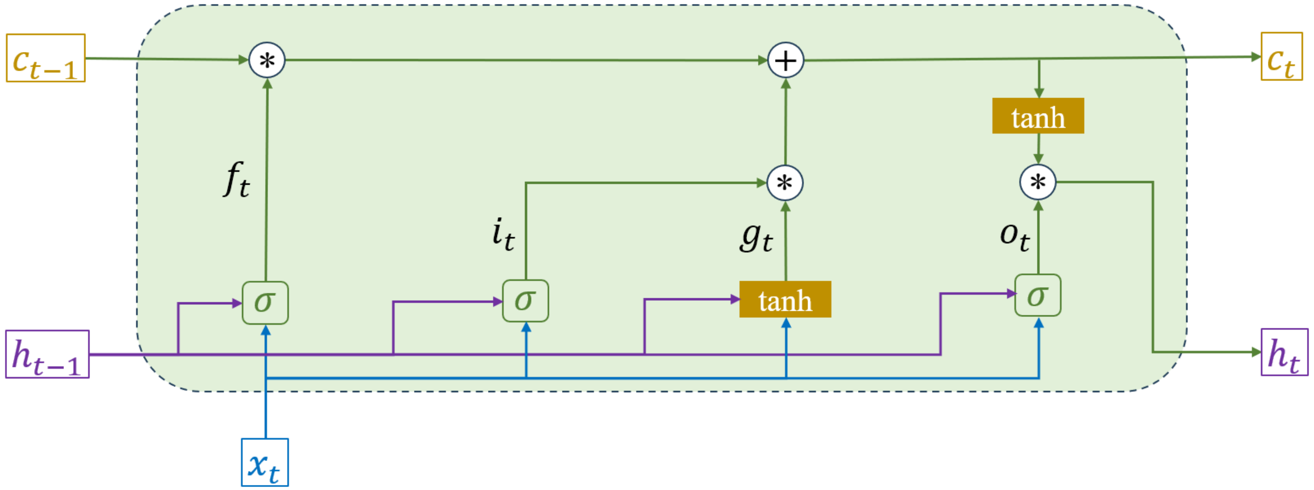
$$i_t = \sigma(\mathrm{W_{ii}x_t} + \mathrm{b_{ii}} + \mathrm{W_{hi}h_{(t-1)}} + \mathrm{b_{hi}})$$
$$f_t = \sigma(\mathrm{W_{if}x_t} + \mathrm{b_{if}} + \mathrm{W_{hf}h_{(t-1)}} + \mathrm{b_{hf}})$$
$$o_t = \sigma(\mathrm{W_{io}x_t} + \mathrm{b_{io}} + \mathrm{W_{ho}h_{(t-1)}} + \mathrm{b_{ho}})$$
$$g_t = \tanh(\mathrm{W_{ig}x_t} + \mathrm{b_{ig}} + \mathrm{W_{hg}h_{(t-1)}} + \mathrm{b_{hg}})$$
$$c_t = f_t \odot g_t + i_t \odot c_{t-1}$$
$$h_t = o_t \odot \tanh(c_t)$$

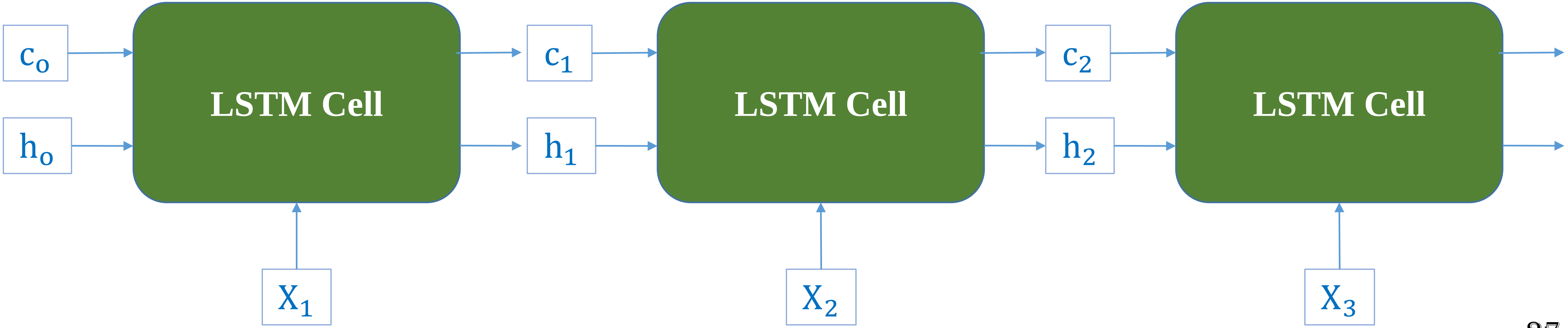


Text Deep Models

❖ Long short-term memory

```
lstm = nn.LSTM(emb_dim,
               hidden_dim,
               num_layers,
               batch_first=True)
```



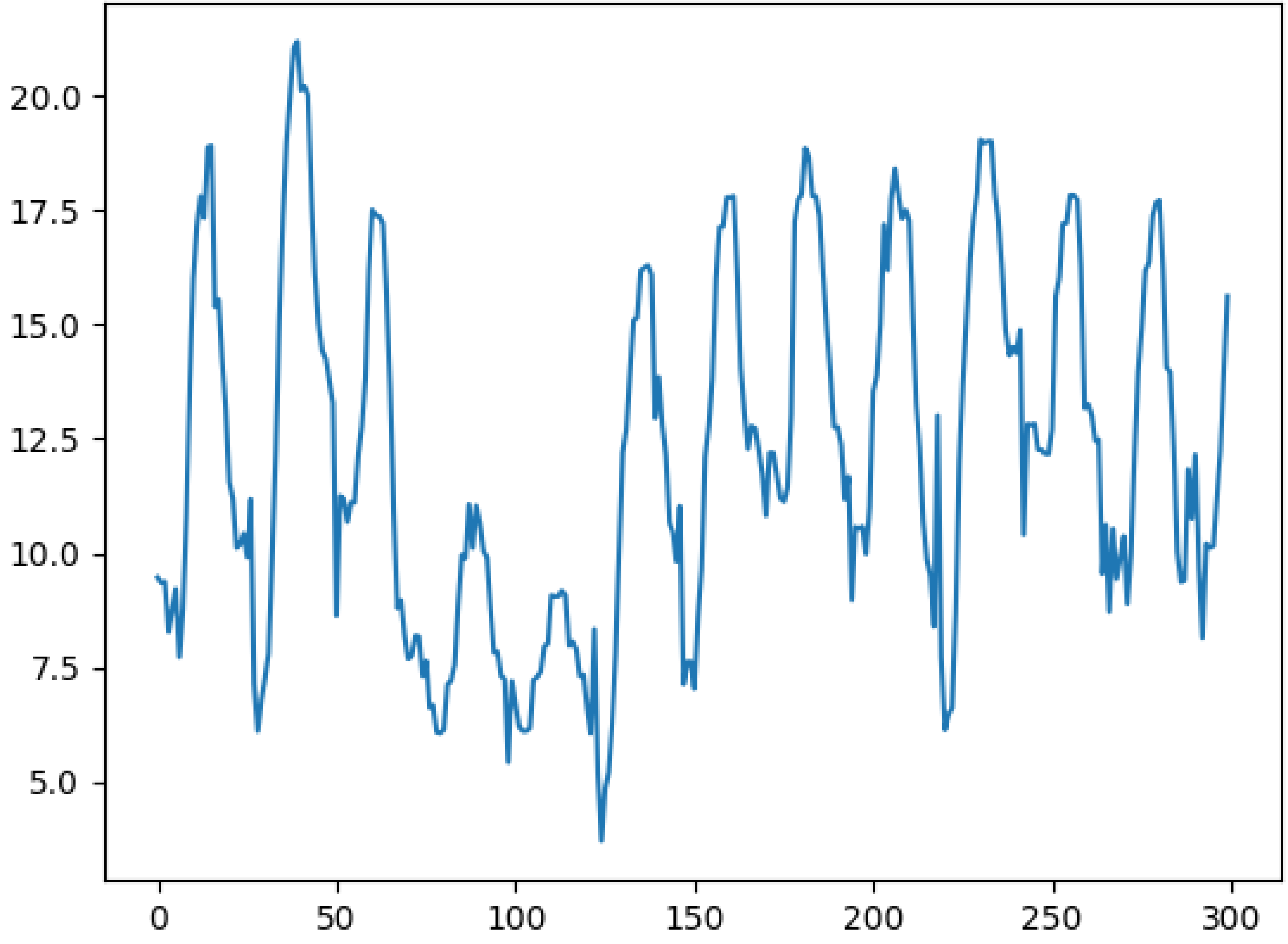


Outline

- RNN in PyTorch
- RNNs for Time-Series Data
- RNNs for IMDB dataset
- From RNN to LSTM
- LSTM Applications

Time-series Data

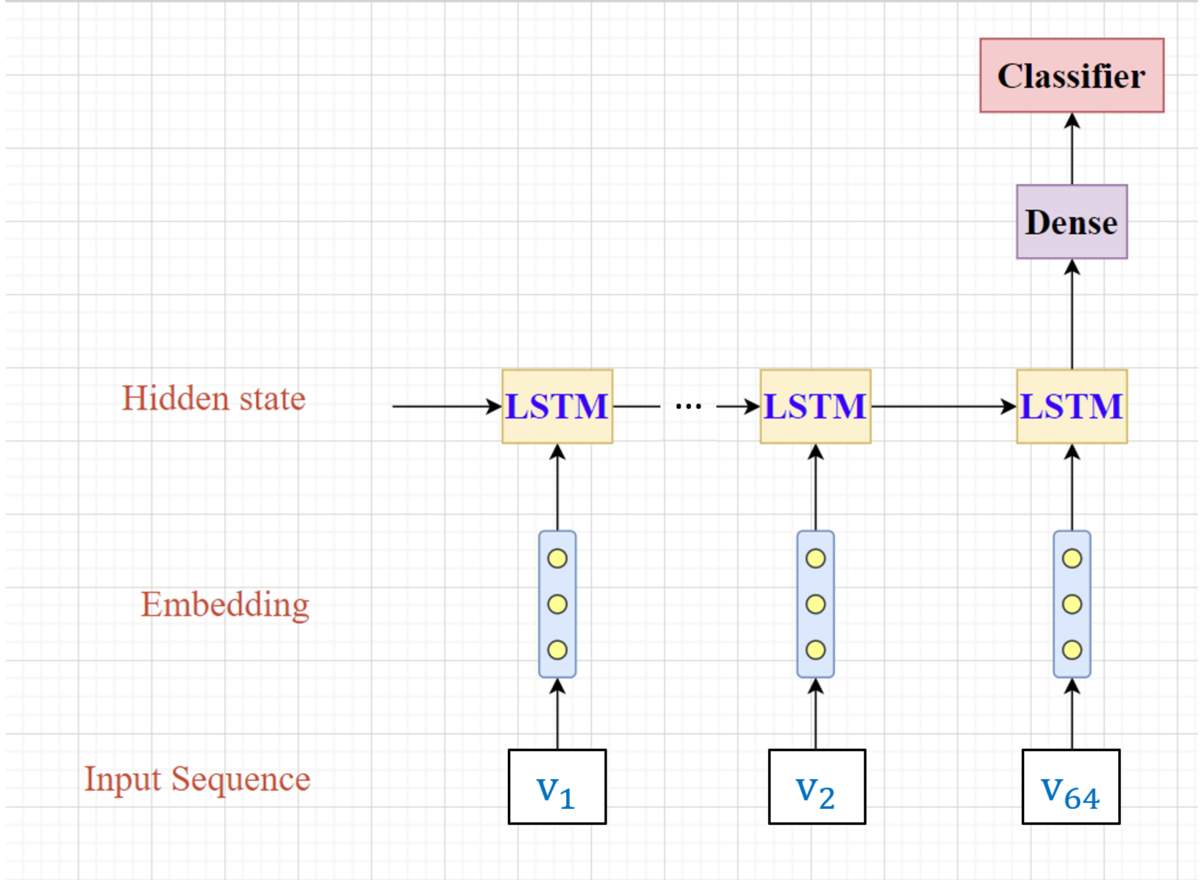
Temperature forecasting

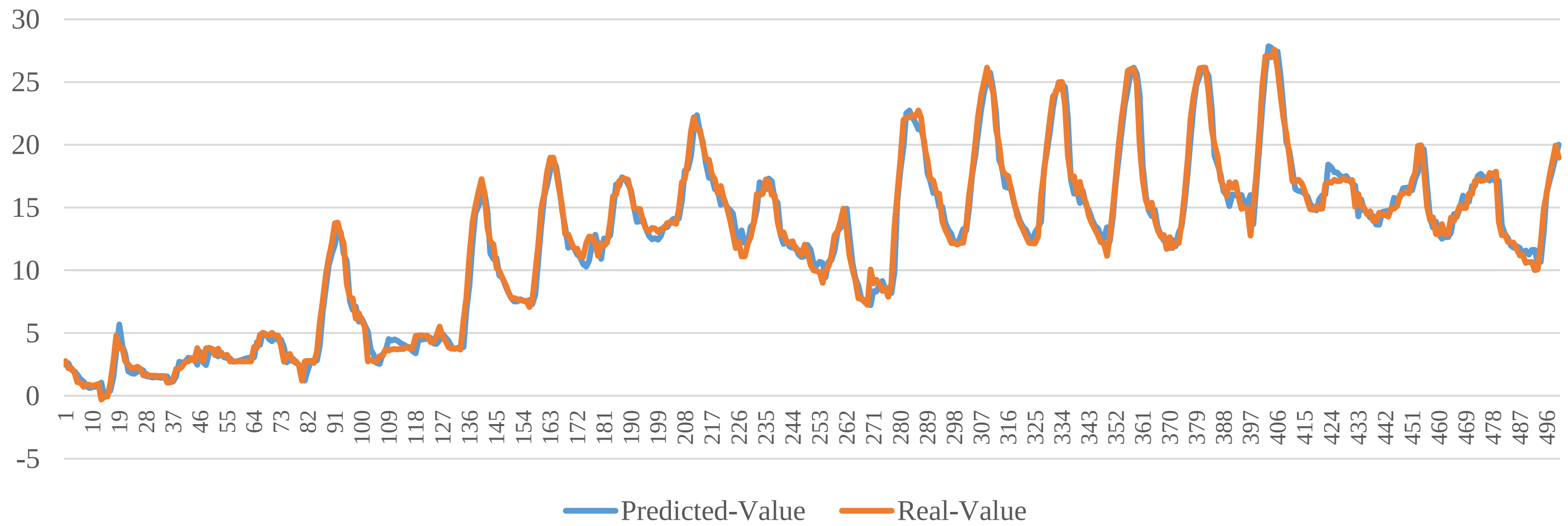


Date	Temperature (C)
2006-04-01 00	9.472222222
2006-04-01 01	9.355555556
2006-04-01 02	9.377777778
2006-04-01 03	8.288888889
2006-04-01 04	8.755555556
2006-04-01 05	9.222222222
2006-04-01 06	7.733333333
2006-04-01 07	8.772222222
2006-04-01 08	10.82222222
2006-04-01 09	13.77222222
2006-04-01 10	16.01666667
2006-04-01 11	17.14444444
2006-04-01 12	17.8
2006-04-01 13	17.33333333
2006-04-01 14	18.87777778
2006-04-01 15	18.91111111
2006-04-01 16	15.38888889
2006-04-01 17	15.55
2006-04-01 18	14.25555556
2006-04-01 19	13.14444444
2006-04-01 20	11.55
2006-04-01 21	11.18333333
2006-04-01 22	10.11666667
2006-04-01 23	10.2
2006-04-10 00	10.42222222
2006-04-10 01	9.911111111
2006-04-10 02	11.18333333
2006-04-10 03	7.155555556
2006-04-10 04	6.111111111

❖ LSTM

```
class LSTMModel(nn.Module):
    def __init__(self, input_dim, hidden_dim, output_dim):
        super(LSTMModel, self).__init__()
        self.lstm = nn.LSTM(1, hidden_dim, batch_first=True)
        self.fc = nn.Linear(hidden_dim, output_dim)

    def forward(self, x):
        output_lstm, (hidden_lstm, cell_lstm) = self.lstm(x)
        last_hidden = hidden_lstm[-1,:,:]
        output = self.fc(last_hidden)
        return output
```



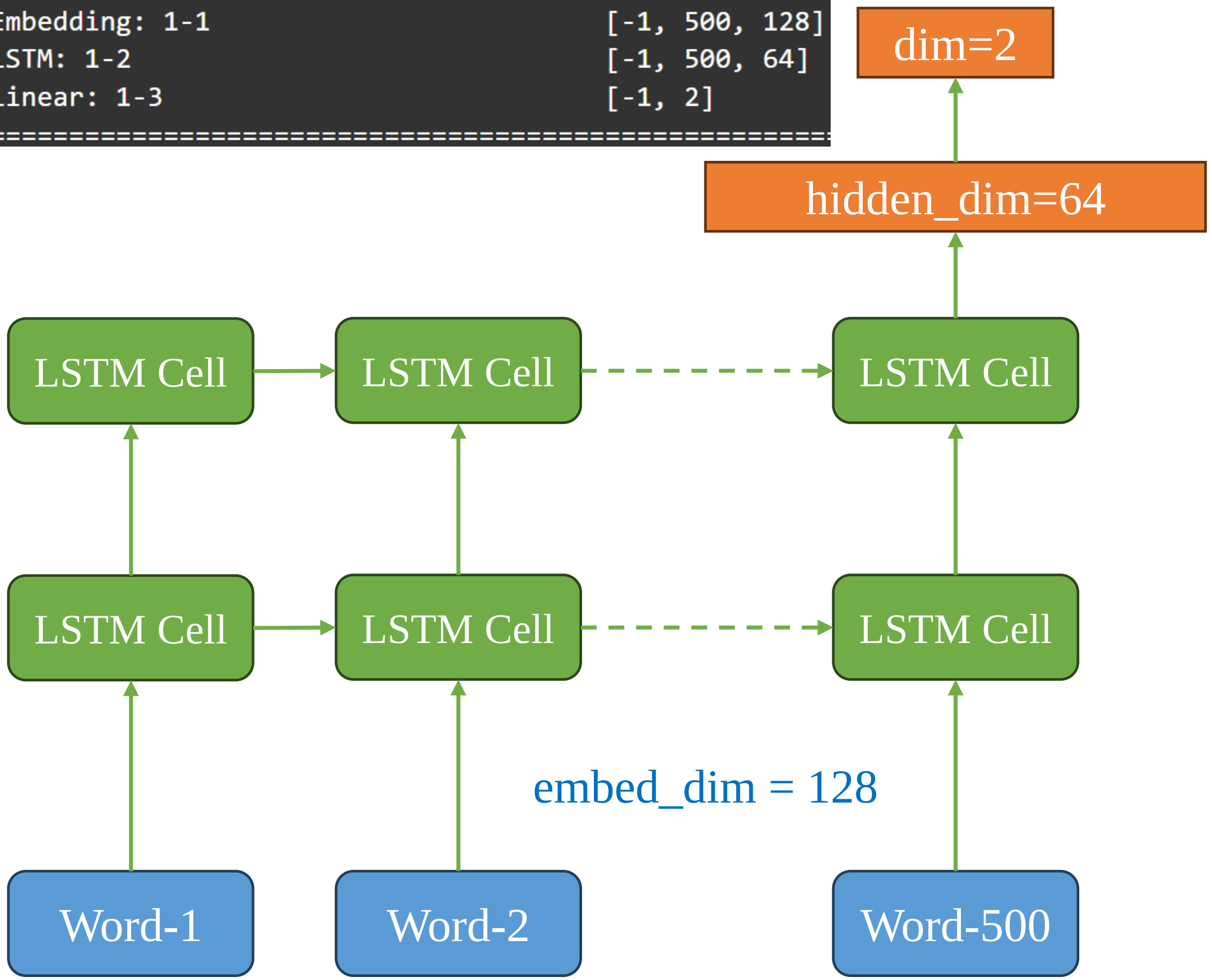


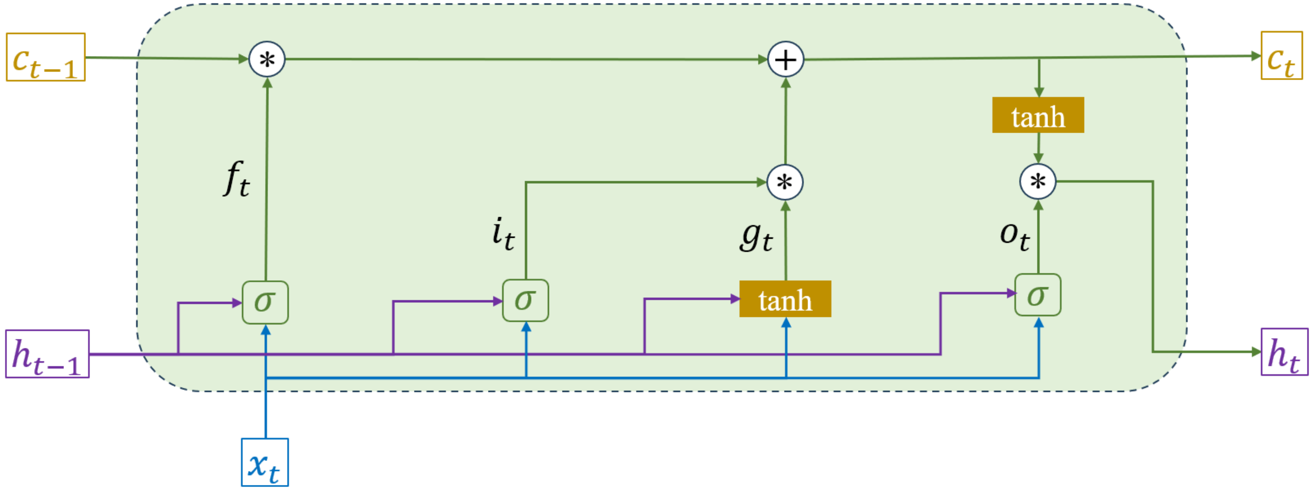
R2 Score: 0.986
MAE: 0.67
MSE: 1.06

Text Deep Models

Long short-term memory

```
================================================================
Layer (type:depth-idx)                   Output Shape
================================================================
├─Embedding: 1-1                         [-1, 500, 128]
├─LSTM: 1-2                              [-1, 500, 64]
├─Linear: 1-3                            [-1, 2]
================================================================
```

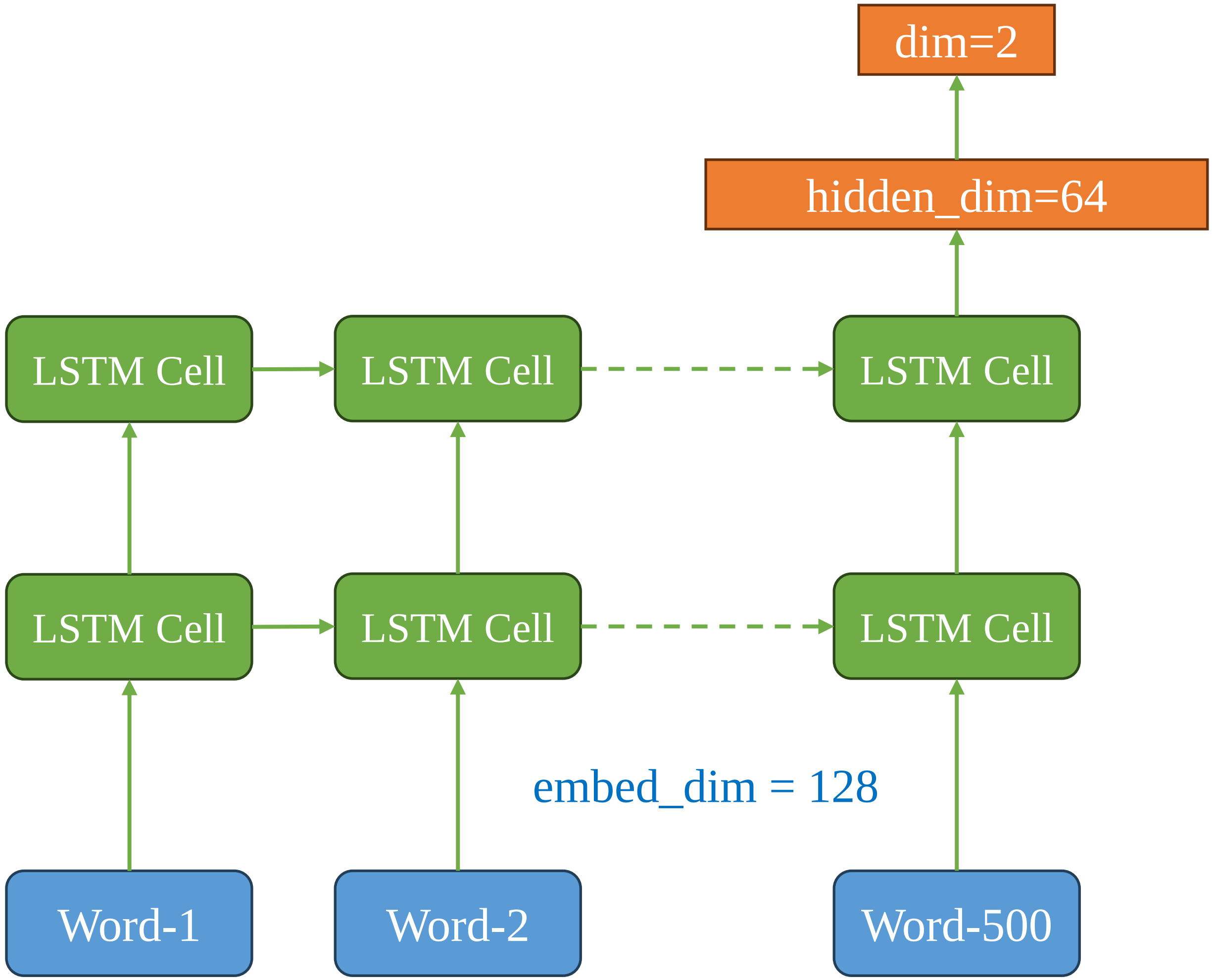


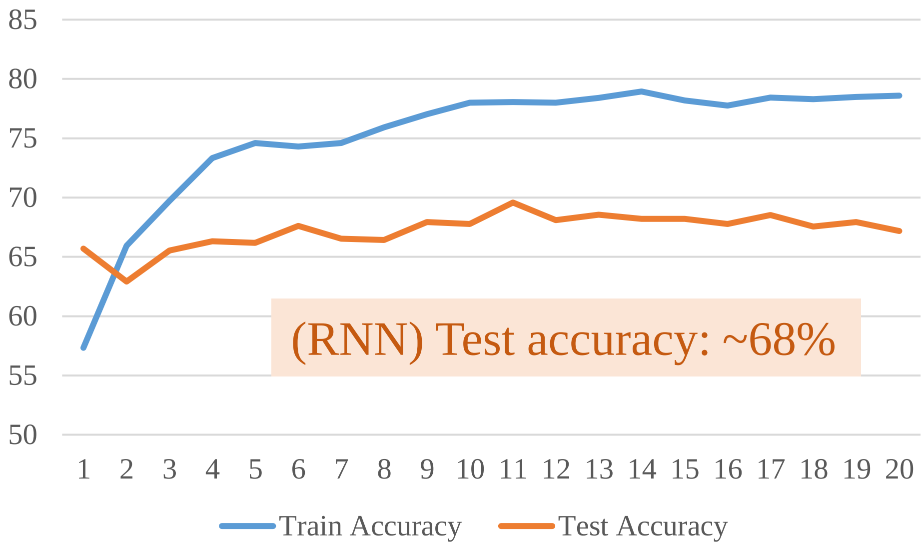


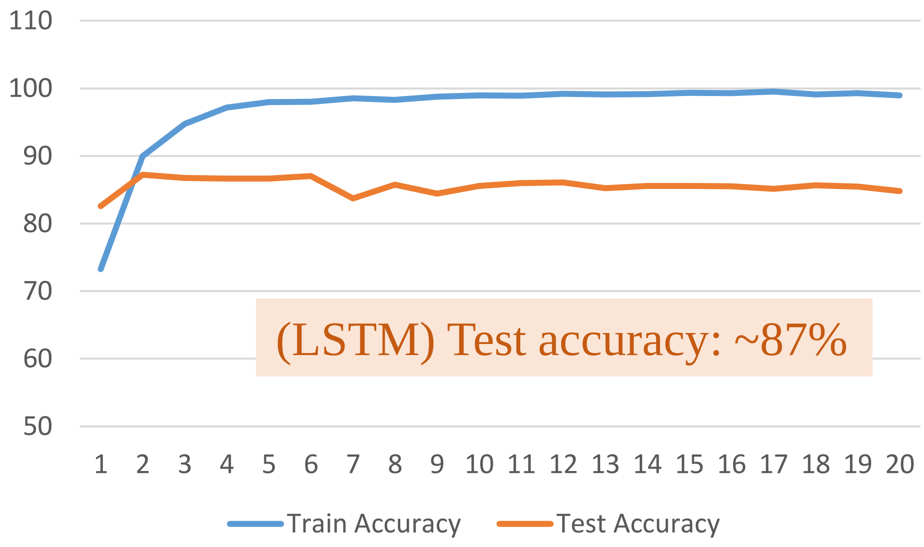

```python
class TextClsModel(nn.Module):
    def __init__(self, vocab_size, emb_dim,
                 hidden_dim, num_layers):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size,
                                      emb_dim)
        self.lstm = nn.LSTM(emb_dim,
                            hidden_dim,
                            num_layers = num_layers,
                            batch_first=True)
        self.fc = nn.Linear(hidden_dim, 2)

    def forward(self, x):
        x = self.embedding(x)
        _, (hidden, _) = self.lstm(x)
        last_hidden = hidden[-1,:,:]
        x = self.fc(last_hidden)
        return x
```

Text Deep Models

Long short-term memory

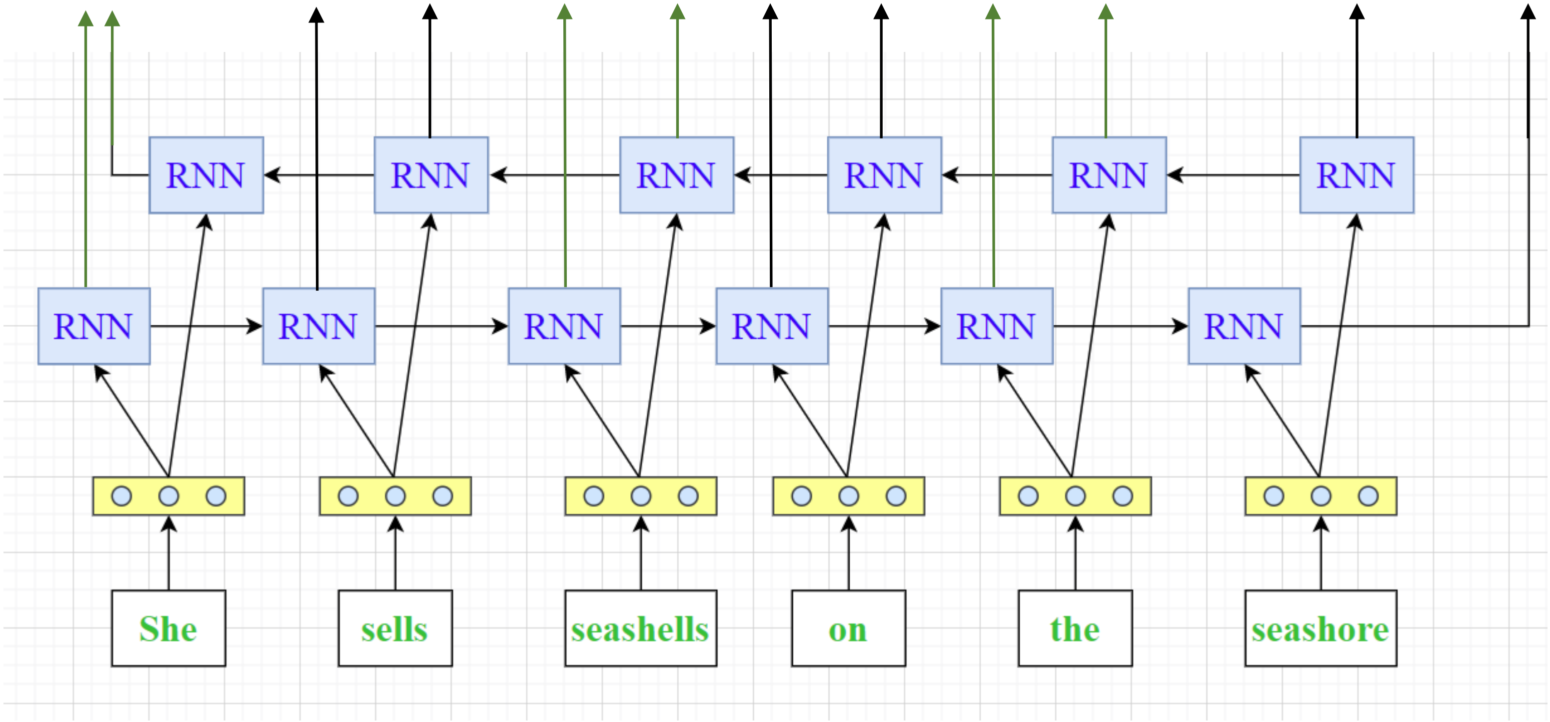






Text Deep Models

❖ Bidirectional RNN/LSTM/GRU



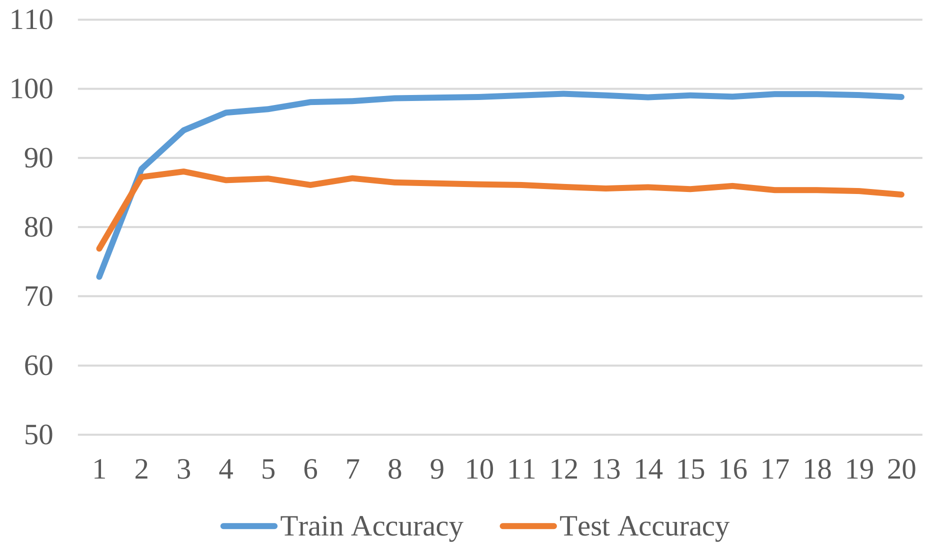

```python
class TextClsModel(nn.Module):
    def __init__(self, vocab_size, emb_dim,
                 hidden_dim, num_layers):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size,
                                      emb_dim)
        self.rnn = nn.RNN(emb_dim, hidden_dim,
                          num_layers = 2,
                          bidirectional = True,
                          batch_first = True)
        self.fc = nn.Linear(hidden_dim, 2)
    def forward(self, x):
        x = self.embedding(x)
        _, hidden = self.rnn(x)
        last_hidden = hidden[-1,:,:]
        x = self.fc(last_hidden)
        return x
```

Text Deep Models

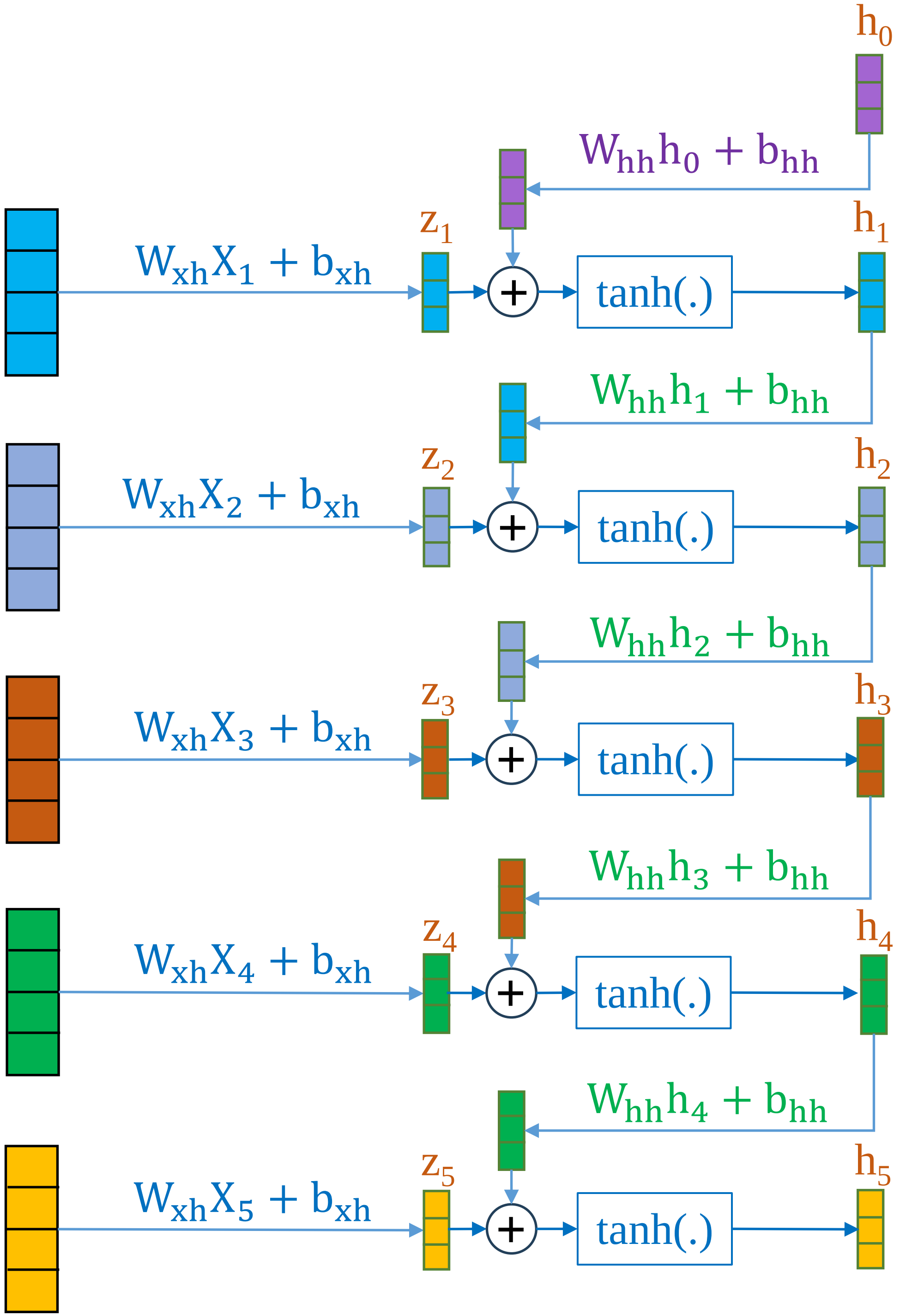
❖ Bidirectional RNN/LSTM

```
class TextClsModel(nn.Module):
    def __init__(self, vocab_size, emb_dim,
                 hidden_dim, num_layers):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size,
                                      emb_dim)
        self.lstm = nn.LSTM(emb_dim, hidden_dim,
                            num_layers = 2,
                            bidirectional = True,
                            batch_first = True)
        self.fc = nn.Linear(hidden_dim, 2)
    def forward(self, x):
        x = self.embedding(x)
        _, (hidden, _) = self.lstm(x)
        last_hidden = hidden[-1,:,:]
        x = self.fc(last_hidden)
        return x
```

(LSTM) Test accuracy: ~88%



RNN



$h_0 = \mathbf{0} \qquad b_{hh} = \mathbf{0}$

$$h_1 = \tanh(W_{xh}X_1 + b_{xh} + W_{hh}h_0 + b_{hh})$$

$$h_2 = \tanh(W_{xh}X_2 + b_{xh} + W_{hh}h_1 + b_{hh})$$

$$h_3 = \tanh(W_{xh}X_3 + b_{xh} + W_{hh}h_2 + b_{hh})$$

$$h_4 = \tanh(W_{xh}X_4 + b_{xh} + W_{hh}h_3 + b_{hh})$$

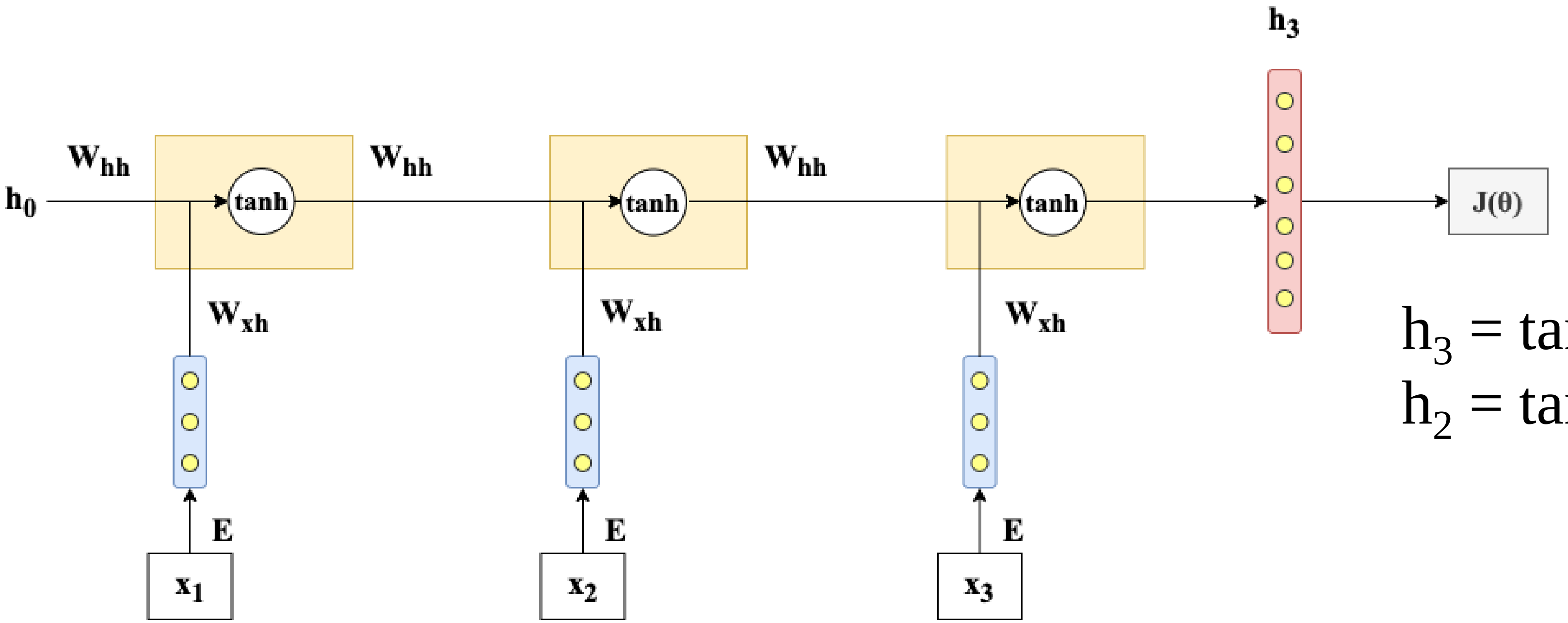
$$h_5 = \tanh(W_{xh}X_5 + b_{xh} + W_{hh}h_4 + b_{hh})$$

$$\Rightarrow h_t = \tanh\left(W_{xh}X_t + b_{xh} + W_{hh}h_{(t-1)} + b_{hh}\right)$$

Text Deep Models

❖ Recurrent Neural Networks (RNN) - Classification

Backpropagation



Loss: $J(\theta)$

Compute: $\frac{\partial J}{\partial W_{xh}}$

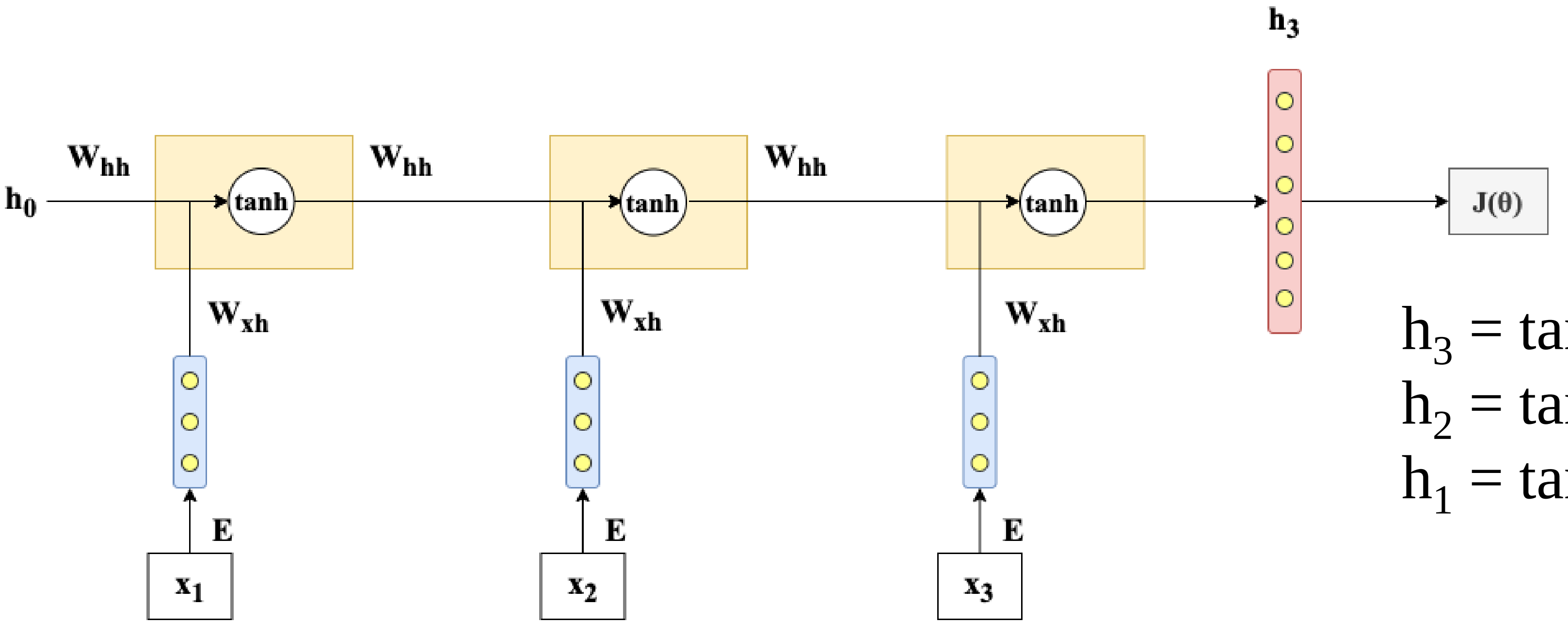
$$h_3 = \tanh(W_{hh}h_2 + b_{hh} + W_{xh}x_3 + b_{xh})$$
$$h_2 = \tanh(W_{hh}h_1 + b_{hh} + W_{xh}x_2 + b_{xh})$$

$$\frac{\partial J}{\partial W_{xh}} = \frac{\partial J}{\partial h_3}\frac{\partial h_3}{\partial W_{xh}} + \frac{\partial J}{\partial h_3}\frac{\partial h_3}{\partial h_2}$$

Text Deep Models

❖ Recurrent Neural Networks (RNN) - Classification

Backpropagation



Loss: $J(\theta)$

Compute: $\frac{\partial J}{\partial W_{xh}}$

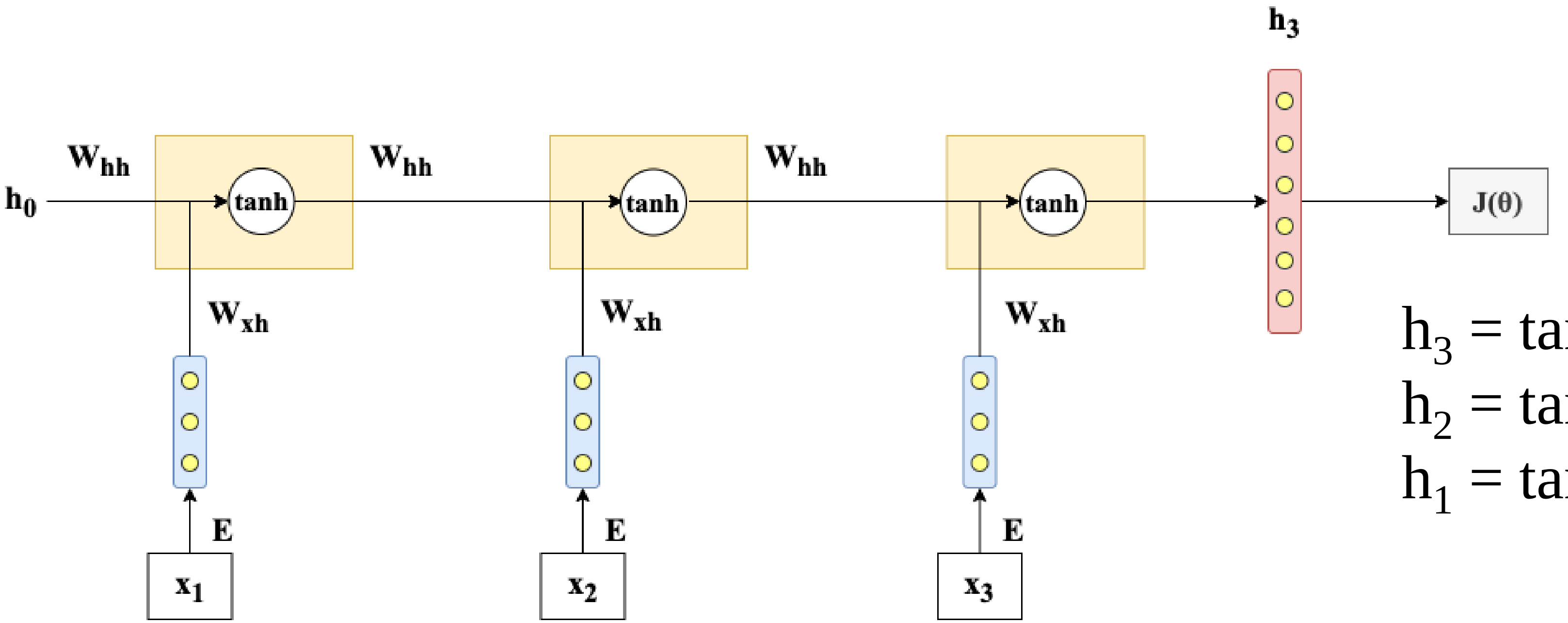
$$h_3 = \tanh(W_{hh}h_2 + b_{hh} + W_{xh}x_3 + b_{xh})$$
$$h_2 = \tanh(W_{hh}h_1 + b_{hh} + W_{xh}x_2 + b_{xh})$$
$$h_1 = \tanh(W_{hh}h_0 + b_{hh} + W_{xh}x_1 + b_{xh})$$

$$\frac{\partial J}{\partial W_{xh}} = \frac{\partial J}{\partial h_3}\frac{\partial h_3}{\partial W_{xh}} + \frac{\partial J}{\partial h_3}\frac{\partial h_3}{\partial h_2}\left(\frac{\partial h_2}{\partial W_{xh}} + \frac{\partial h_2}{\partial h_1}\right) = \frac{\partial J}{\partial h_3}\frac{\partial h_3}{\partial W_{xh}} + \frac{\partial J}{\partial h_3}\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial W_{xh}} + \frac{\partial J}{\partial h_3}\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial h_1}$$

Text Deep Models

❖ Recurrent Neural Networks (RNN) - Classification

Backpropagation



Loss: $J(\theta)$

Compute: $\frac{\partial J}{\partial W_{xh}}$

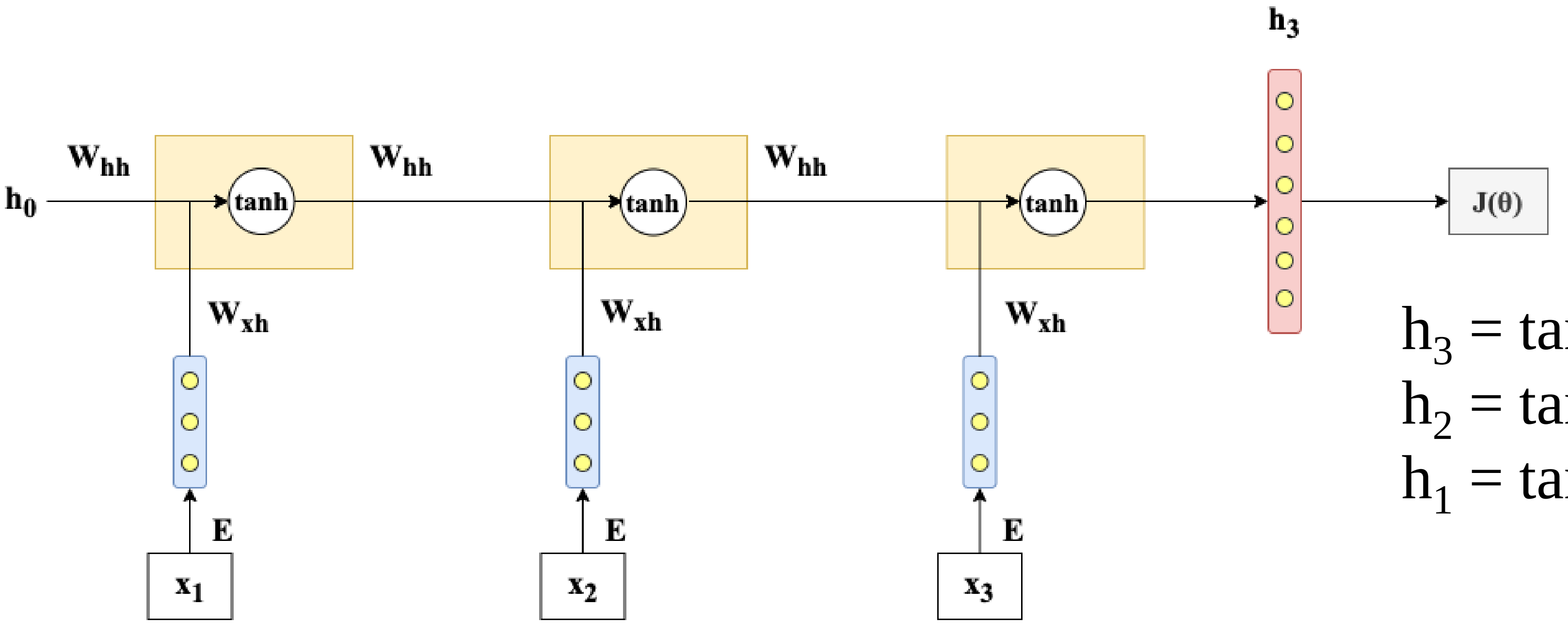
$$h_3 = \tanh(W_{hh}h_2 + b_{hh} + W_{xh}x_3 + b_{xh})$$
$$h_2 = \tanh(W_{hh}h_1 + b_{hh} + W_{xh}x_2 + b_{xh})$$
$$h_1 = \tanh(W_{hh}h_0 + b_{hh} + W_{xh}x_1 + b_{xh})$$

$$\frac{\partial J}{\partial W_{xh}} = \frac{\partial J}{\partial h_3}\frac{\partial h_3}{\partial W_{xh}} + \frac{\partial J}{\partial h_3}\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial W_{xh}} + \frac{\partial J}{\partial h_3}\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial h_1}\frac{\partial h_1}{\partial W_{xh}}$$

Text Deep Models

❖ Recurrent Neural Networks (RNN) - Classification

Backpropagation



Loss: $J(\theta)$

Compute: $\frac{\partial J}{\partial W_{xh}}$

$$h_3 = \tanh\,(W_{hh}h_2 + b_{hh} + W_{xh}x_3 + b_{xh})$$
$$h_2 = \tanh\,(W_{hh}h_1 + b_{hh} + W_{xh}x_2 + b_{xh})$$
$$h_1 = \tanh\,(W_{hh}h_0 + b_{hh} + W_{xh}x_1 + b_{xh})$$

$$\frac{\partial J}{\partial W_{xh}} = \sum_{k=1}^{T} \frac{\partial J}{\partial h_T} \underbrace{\frac{\partial h_T}{\partial h_k}}_{\frac{\partial h_T}{\partial h_{T-1}} \frac{\partial h_{T-1}}{\partial h_{T-2}} \cdots \frac{\partial h_{k+2}}{\partial h_{k+1}} \frac{\partial h_{k+1}}{\partial h_k}} \frac{\partial h_k}{\partial W_{xh}}$$