Recurrent Neural Networks

Machine Learning for Behavioral Data

April 11, 2022

Today's Topic

Week	Lecture/Lab
1	Introduction
2	Data Exploration
3	Regression
4	Classification
5	Model Evaluation
6	Knowledge Tracing
7	Knowledge Tracing
8	Time Series Prediction

Supervised learning on time series:

- **Probabilistic graphical models**
- **Neural networks: LSTM, GRU, etc.**

Getting ready for today’s lecture…

- **If not done yet**: clone the repository containing the Jupyter notebook and data for today’s lecture into Google Colab
- Join us on Kahoot for the Easter quiz!

Easter quiz about the past

www.kahoot.it

Enter the game pin!

Win a chocolate Easter bunny!

Today's Topic

Week	Lecture/Lab
1	Introduction
2	Data Exploration
3	Regression
4	Classification
5	Model Evaluation
6	Knowledge Tracing
7	Knowledge Tracing
8	**Time Series Prediction**

Supervised learning on time series:

- Probabilistic graphical models
- Neural networks: LSTM, GRU, etc.

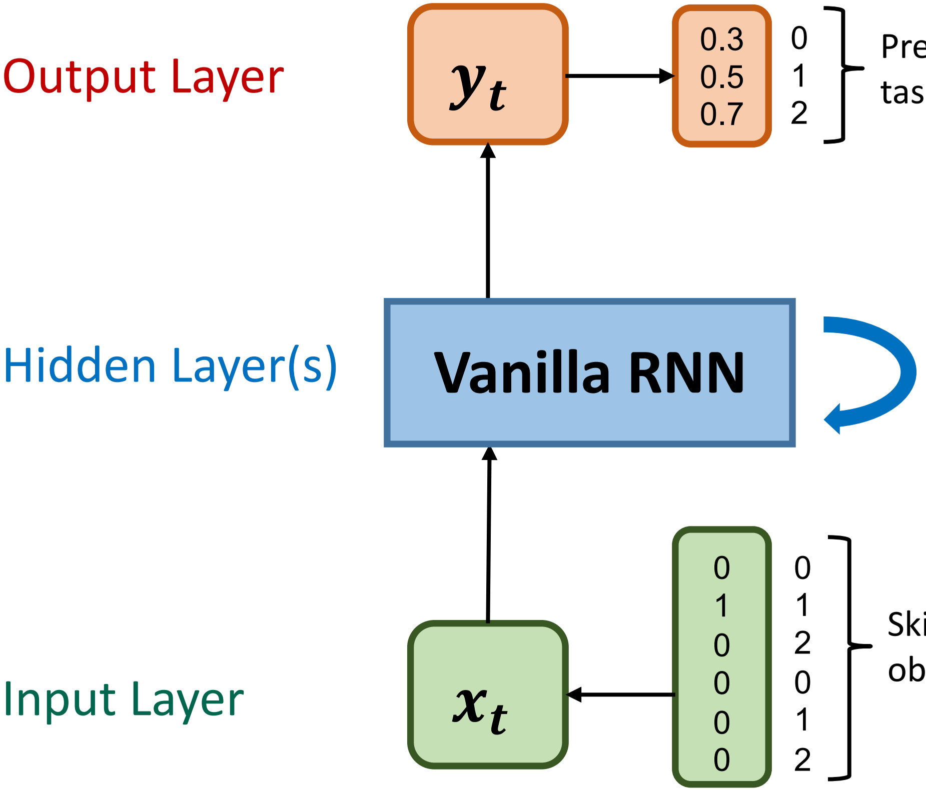
Today – Recurrent Neural Networks

- Parameters and hyperparameter tuning
- Different architectures
- Different tasks:
  - “Many-to-many” versus “Many-to-one”
  - Classification versus Regression

Today – Recurrent Neural Networks

- **Parameters and hyperparameter tuning**
- Different architectures
- Different tasks:
  - “Many-to-many” versus “Many-to-one”
  - Classification versus Regression

Deep Knowledge Tracing - revisited

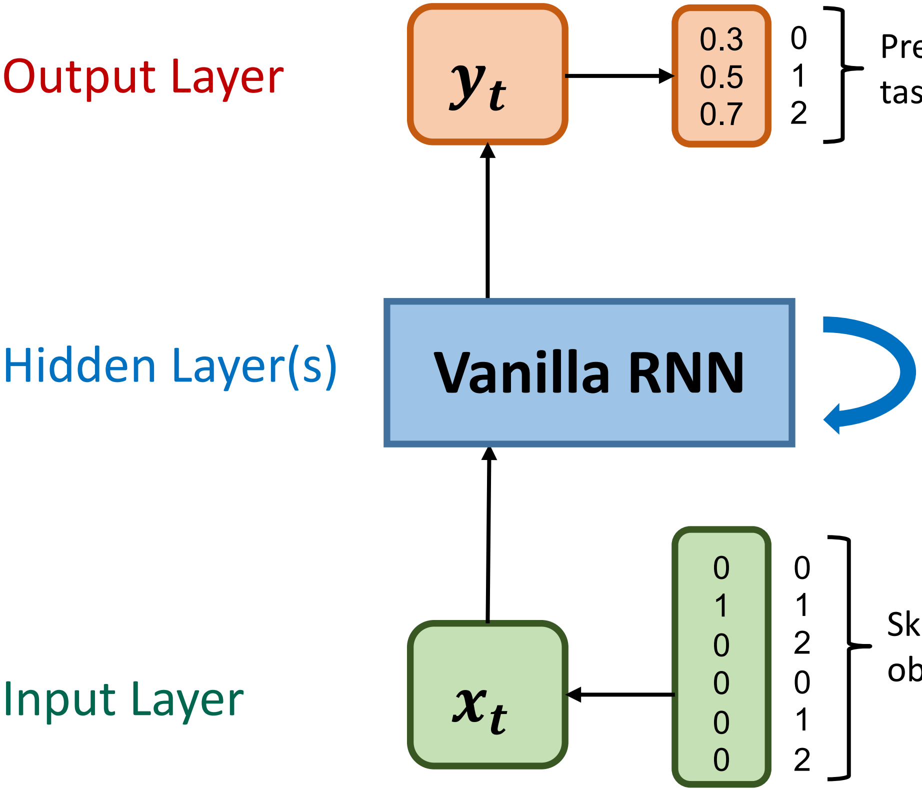


Predict probability that the student will solve a task correctly at time t for each skill or item

Skills or items: one-hot encoded observation o_{t-1} at time step $t-1$

[*Deep Knowledge Tracing*, Piech et al., NIPS 2015]

Deep Knowledge Tracing - revisited



Predict probability that the student will solve a task correctly at time t for each skill or item

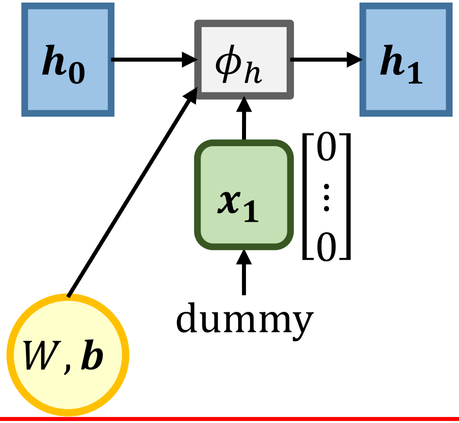
$$\boldsymbol{y_t} = \sigma(W_{yh}\boldsymbol{h_t} + \boldsymbol{b_y})$$

$$\boldsymbol{h_t} = \tanh(W_{hx}\boldsymbol{x}_t + W_{hh}\boldsymbol{h_{t-1}} + \boldsymbol{b_h})$$

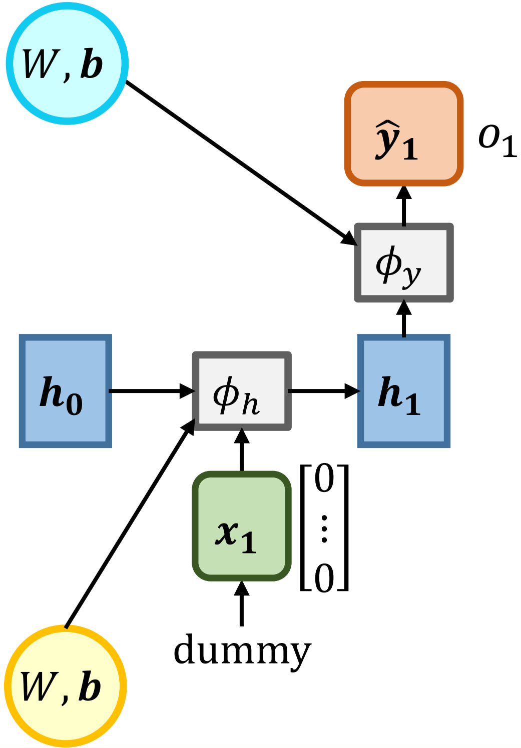
Skills or items: one-hot encoded observation o_{t-1} at time step $t-1$

[*Deep Knowledge Tracing*, Piech et al., NIPS 2015]

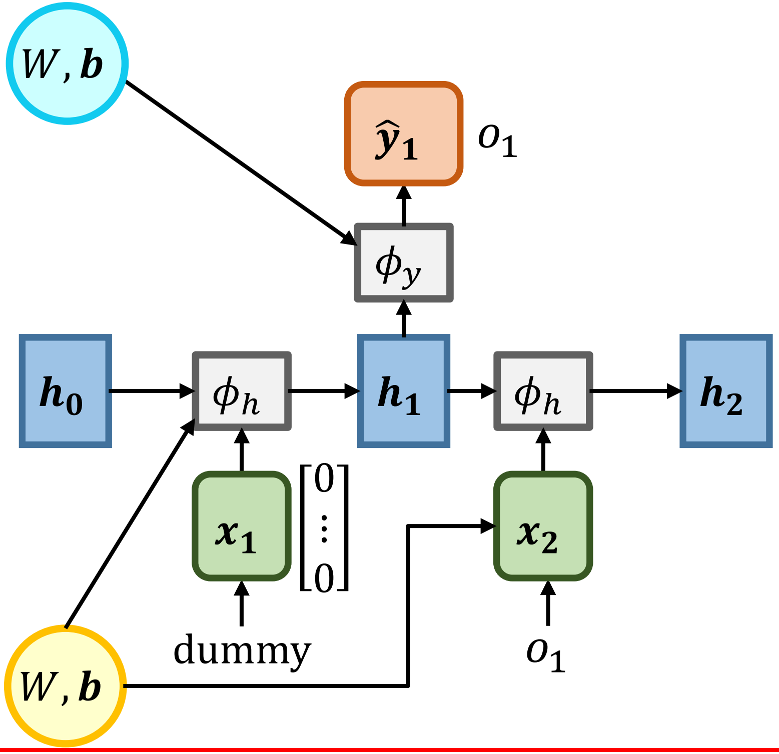
Computational Graph - revisited



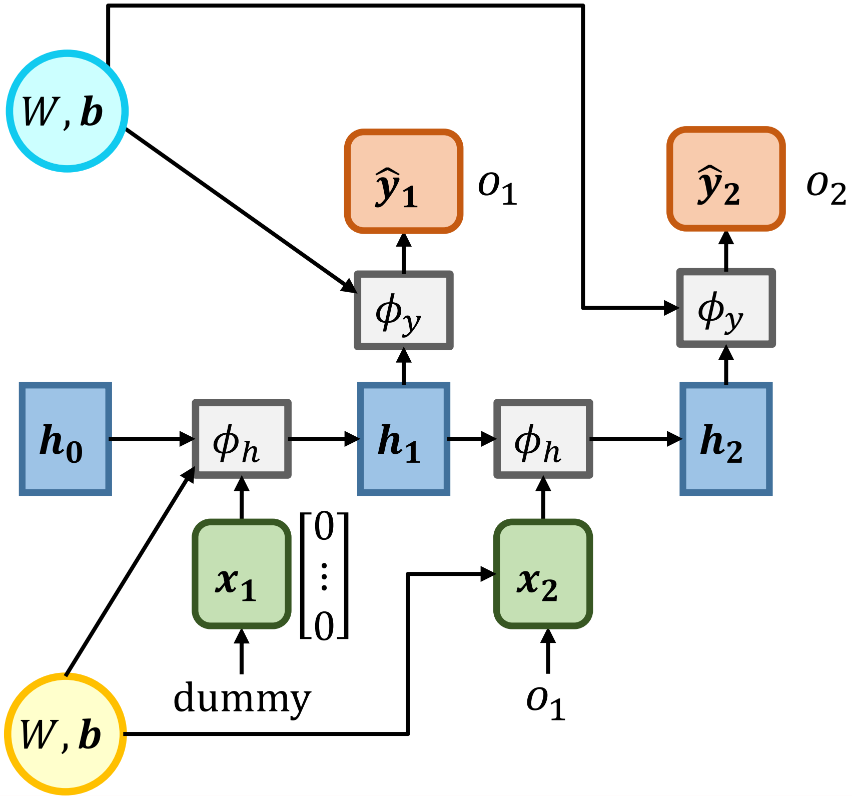
Computational Graph - revisited



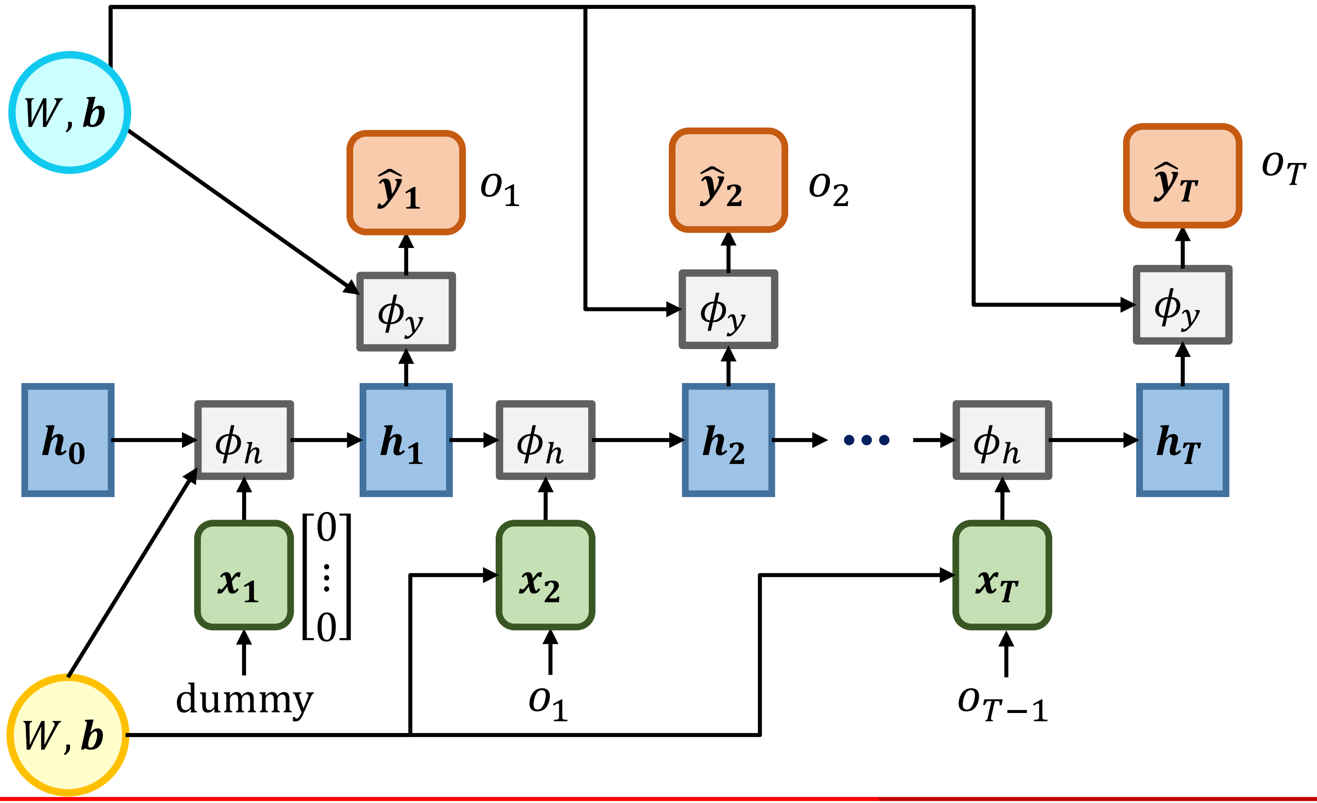
Computational Graph - revisited



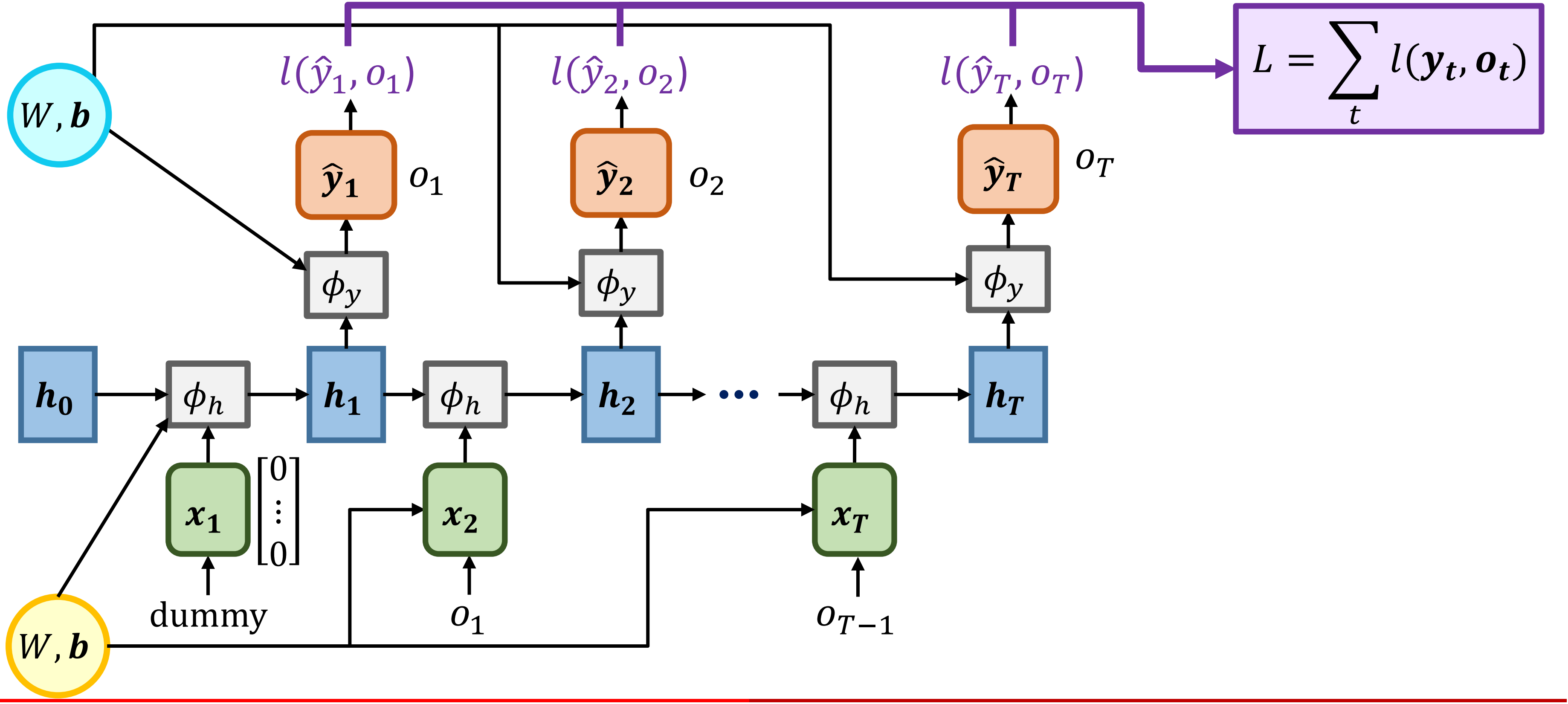
Computational Graph - revisited



Computational Graph - revisited



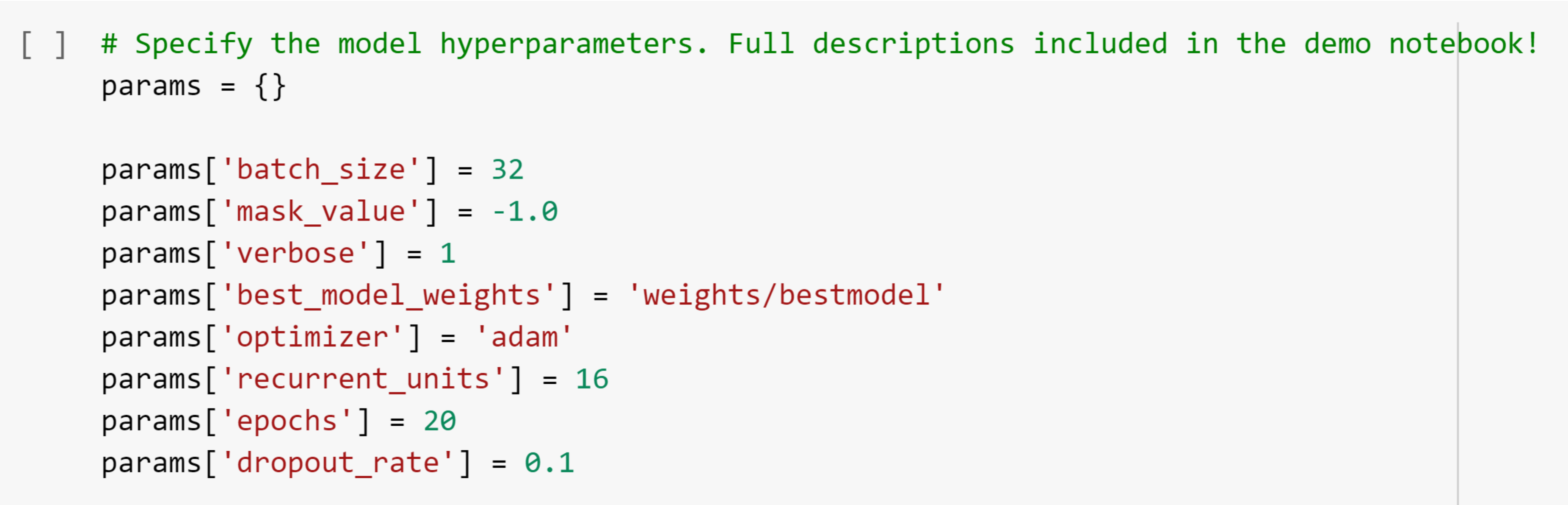
Computational Graph - revisited



Training and Prediction using DKT

- Training: gradient descent
- Prediction: compute inference in the network (see computational graph)

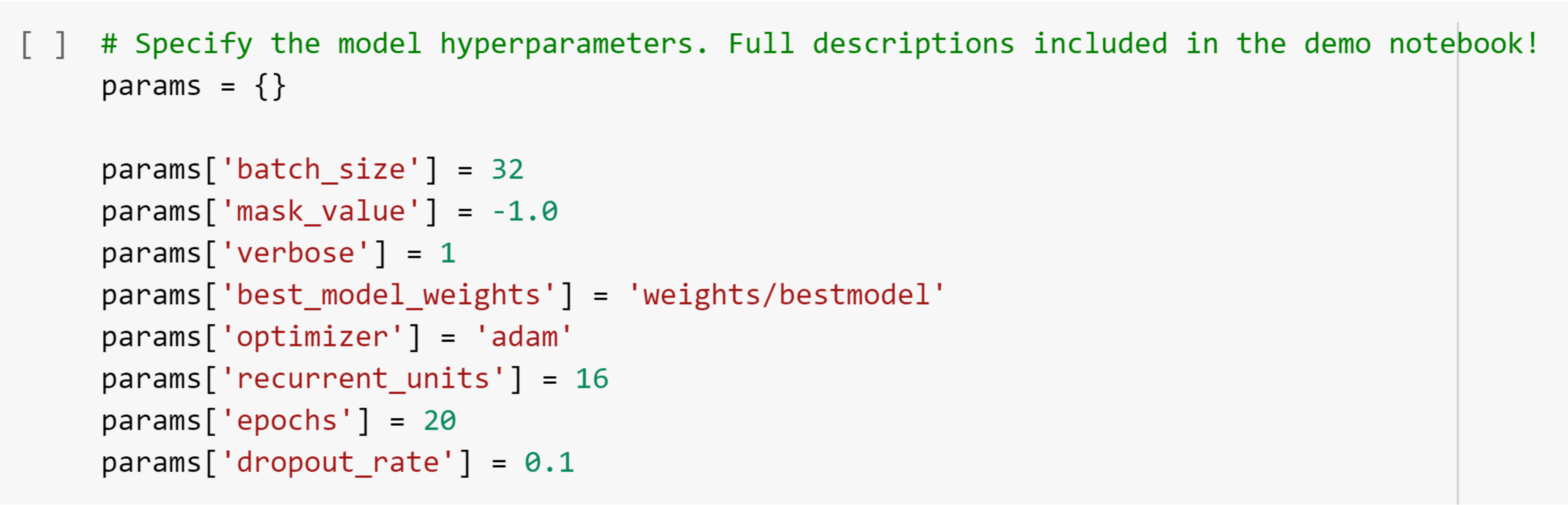
RNNs – Specifying Parameters



```
# Specify the model hyperparameters. Full descriptions included in the demo notebook!
params = {}

params['batch_size'] = 32
params['mask_value'] = -1.0
params['verbose'] = 1
params['best_model_weights'] = 'weights/bestmodel'
params['optimizer'] = 'adam'
params['recurrent_units'] = 16
params['epochs'] = 20
params['dropout_rate'] = 0.1
```

RNNs – Tuning hyperparameters



```python
# Specify the model hyperparameters. Full descriptions included in the demo notebook!
params = {}

params['batch_size'] = 32
params['mask_value'] = -1.0
params['verbose'] = 1
params['best_model_weights'] = 'weights/bestmodel'
params['optimizer'] = 'adam'
params['recurrent_units'] = 16
params['epochs'] = 20
params['dropout_rate'] = 0.1
```

RNNs – Tuning hyperparameters

- Optimal number of epochs can be found using callbacks
- Other parameters can be tuned using for example:
  a) Train-Validation-Test split
  b) Train-Test split, using a k-fold cross validation on the training data to determine the optimal parameters

Today – Recurrent Neural Networks

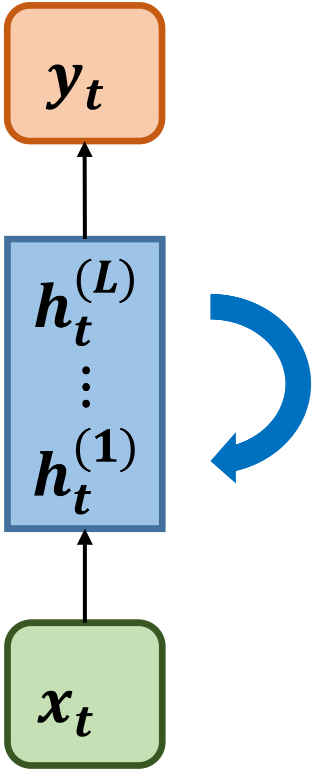
- Parameters and hyperparameter tuning
- **Different architectures**
- Different tasks:
  - “Many-to-many” versus “Many-to-one”
  - Classification versus Regression

Recurrent Neural Network

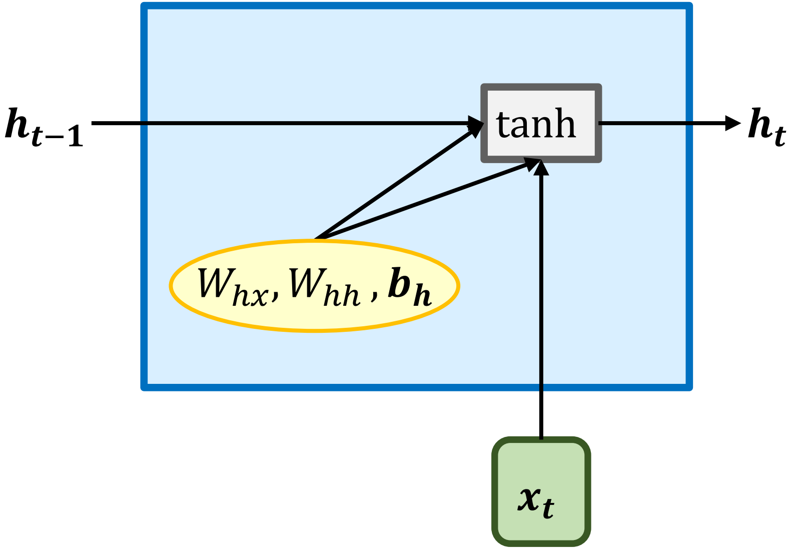
Output Layer

Hidden Layer(s)

Input Layer

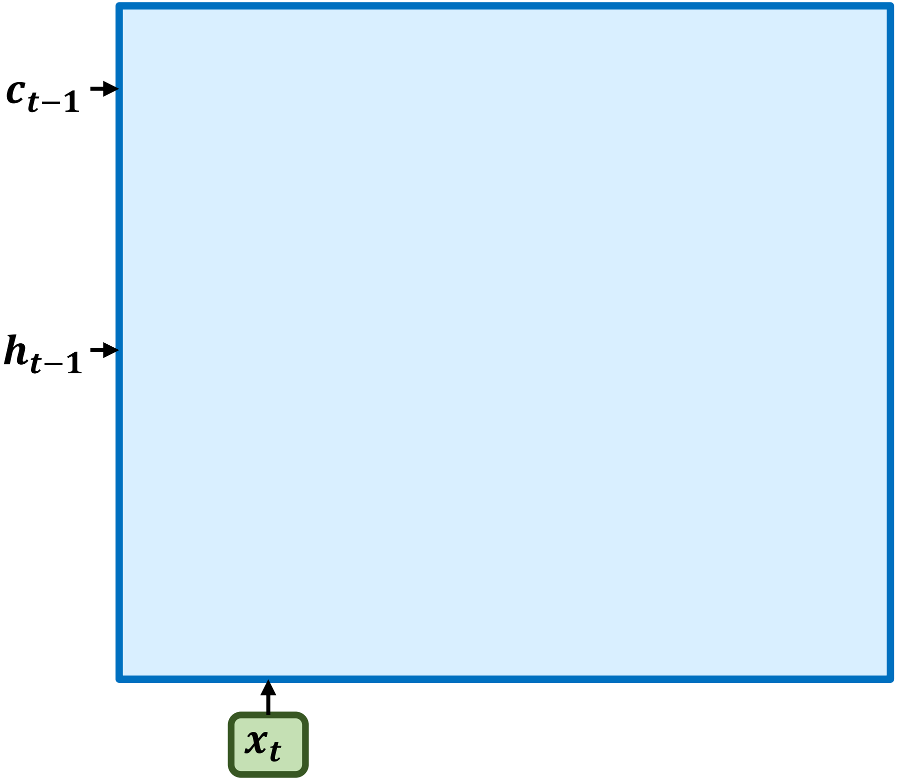


Vanilla RNN - revisited



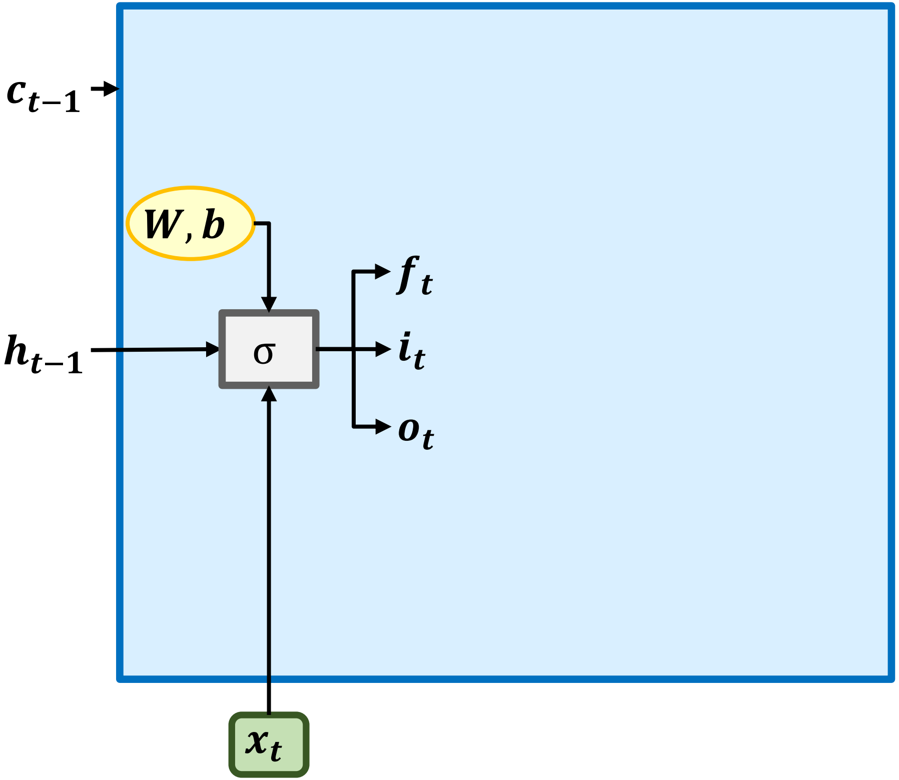
$$\boldsymbol{h_t} = \tanh(W_{hx}\boldsymbol{x}_t + W_{hh}\boldsymbol{h_{t-1}} + \boldsymbol{b_h})$$

Long-Short Term Memory Network (LSTM)



- Two states:
  - Hidden state $\boldsymbol{h_{t-1}}$
  - Cell state $\boldsymbol{c_{t-1}}$

Long-Short Term Memory Network (LSTM)



① Updating the gates:

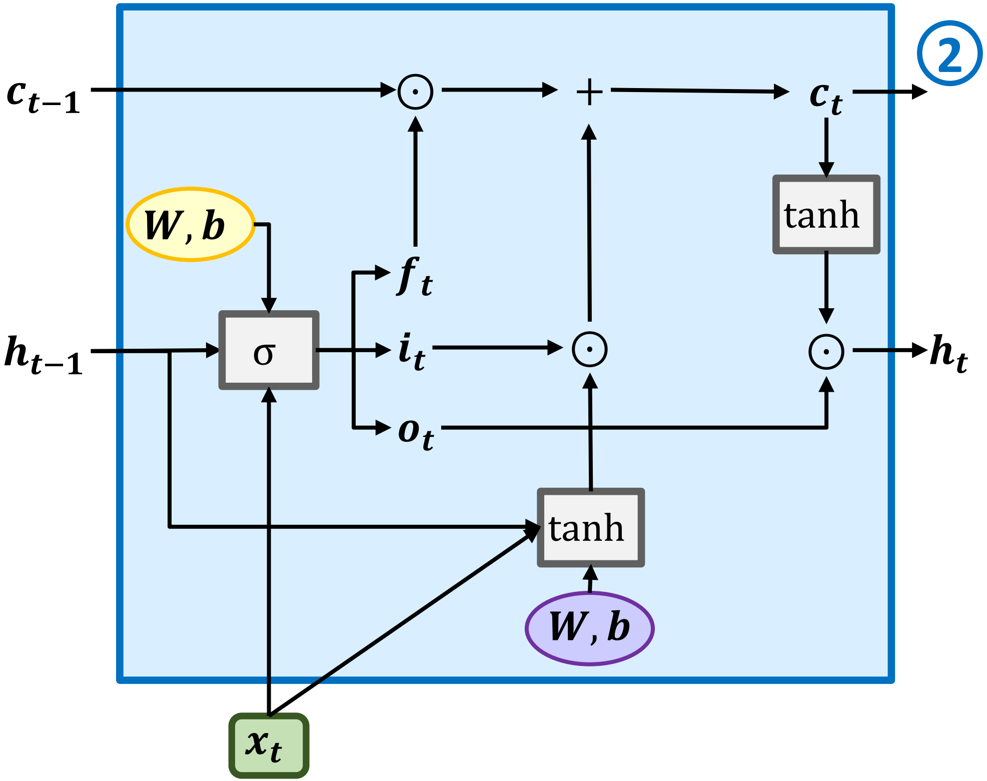
- $\boldsymbol{f}$ forget gate: whether to erase cell
- $\boldsymbol{i}$ input gate: whether to write to cell
- $\boldsymbol{o}$ output gate: how much to reveal cell

$$\boldsymbol{f}_t = \sigma\left(W_{fx}\boldsymbol{x_t} + W_{fh}\boldsymbol{h_{t-1}} + \boldsymbol{b_f}\right)$$

$$\boldsymbol{i_t} = \sigma(W_{ix}\boldsymbol{x_t} + W_{ih}\boldsymbol{h_{t-1}} + \boldsymbol{b_i})$$

$$\boldsymbol{o_t} = \sigma(W_{ox}\boldsymbol{x_t} + W_{oh}\boldsymbol{h_{t-1}} + \boldsymbol{b_o})$$

Long-Short Term Memory Network (LSTM)

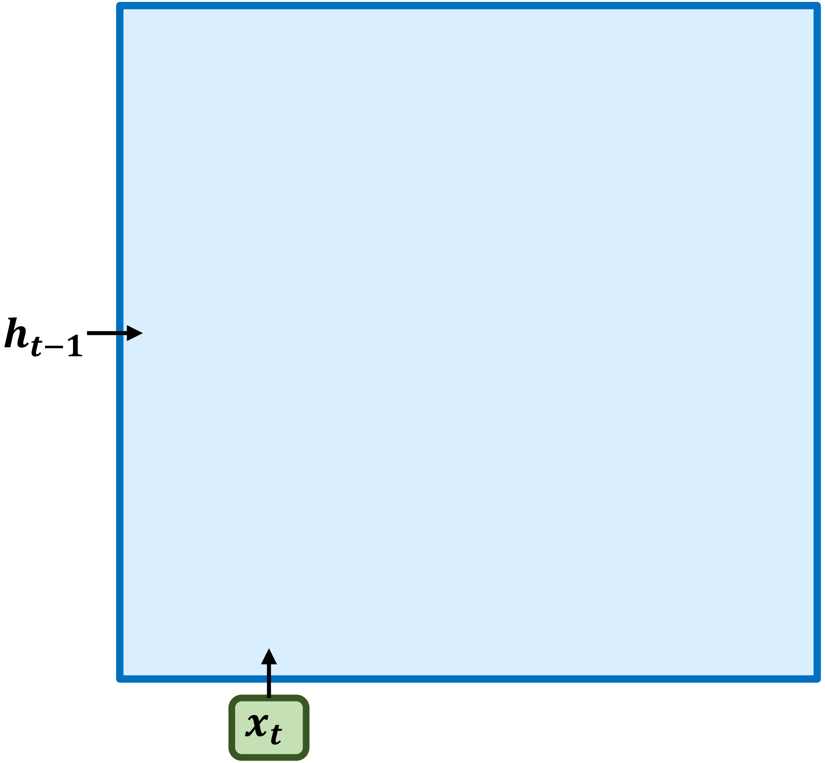


② Updating the states:

$$\boldsymbol{c}_t = \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \tanh(W_{cx}\boldsymbol{x}_t + W_{ch}\boldsymbol{h}_{t-1} + \boldsymbol{b}_c)$$

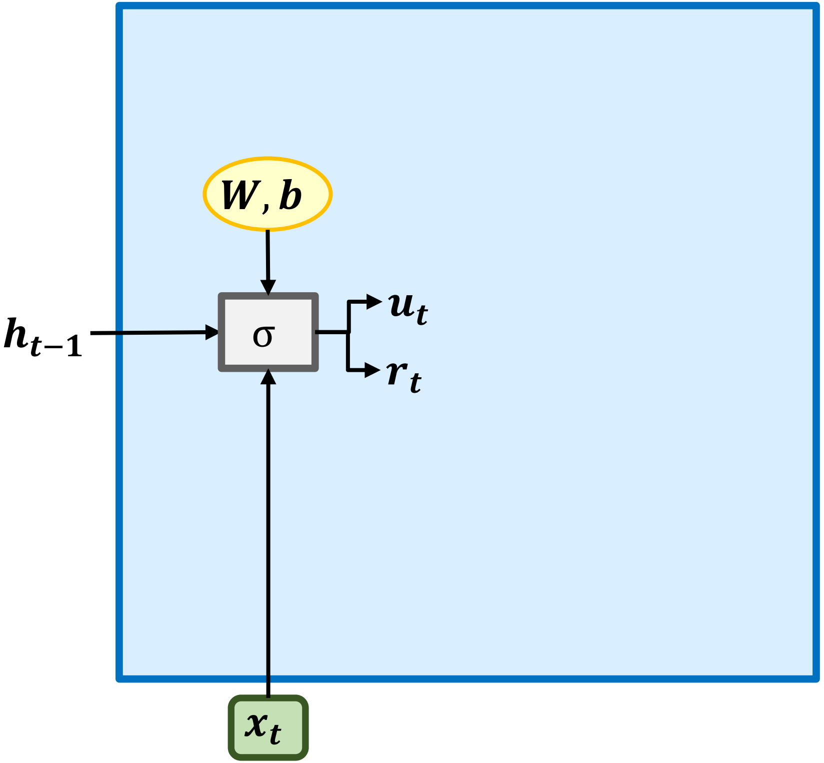
$$\boldsymbol{h}_t = \boldsymbol{o}_t \odot \tanh(\boldsymbol{C}_t)$$

Gated Recurrent Units (GRU)



- Only one state (got rid of cell):
  - Hidden state h_{t-1}

Gated Recurrent Units (GRU)



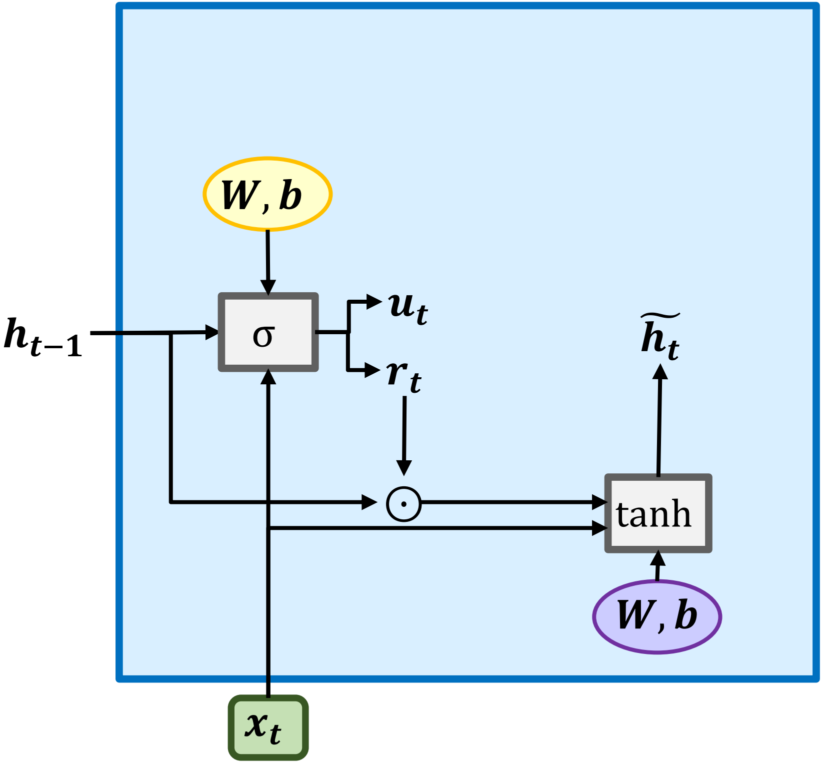
1 Updating the gates:

- $\boldsymbol{r}$ reset gate: how much of the previous state to remember
- $\boldsymbol{u}$ update gate: how much of the new state is just a copy of the old state

$$\boldsymbol{r}_t = \sigma(W_{rx}\boldsymbol{x}_t + \ W_{th}\boldsymbol{h}_{t-1} + \boldsymbol{b}_r)$$

$$\boldsymbol{u}_t = \sigma(W_{ux}\boldsymbol{x}_t + \ W_{uh}\boldsymbol{h}_{t-1} + \boldsymbol{b}_u)$$

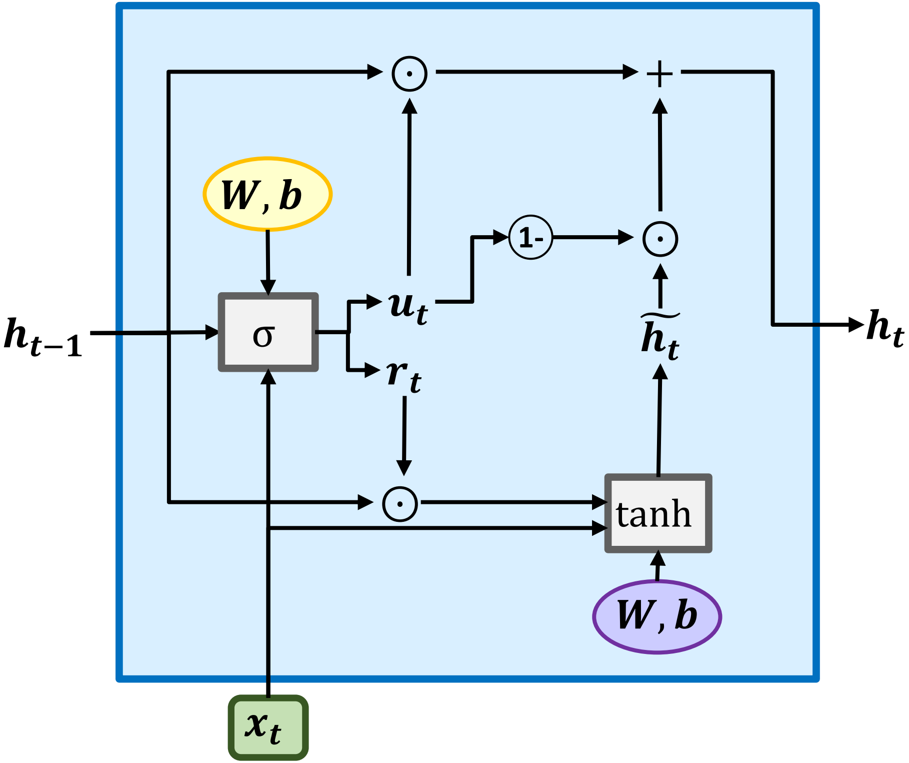
Gated Recurrent Units (GRU)


$\boldsymbol{W}, \boldsymbol{b}$
$\boldsymbol{h_{t-1}}$
σ
$\boldsymbol{u_t}$
$\boldsymbol{r_t}$
$\widetilde{\boldsymbol{h_t}}$
⊙
tanh
$\boldsymbol{W}, \boldsymbol{b}$
$\boldsymbol{x_t}$

② Get candidate hidden state:

$$\widetilde{\boldsymbol{h_t}} = \tanh(W_{hx}\boldsymbol{x_t} + W_{ht}(r_t \odot \boldsymbol{h_{t-1}}) + \boldsymbol{b_h})$$

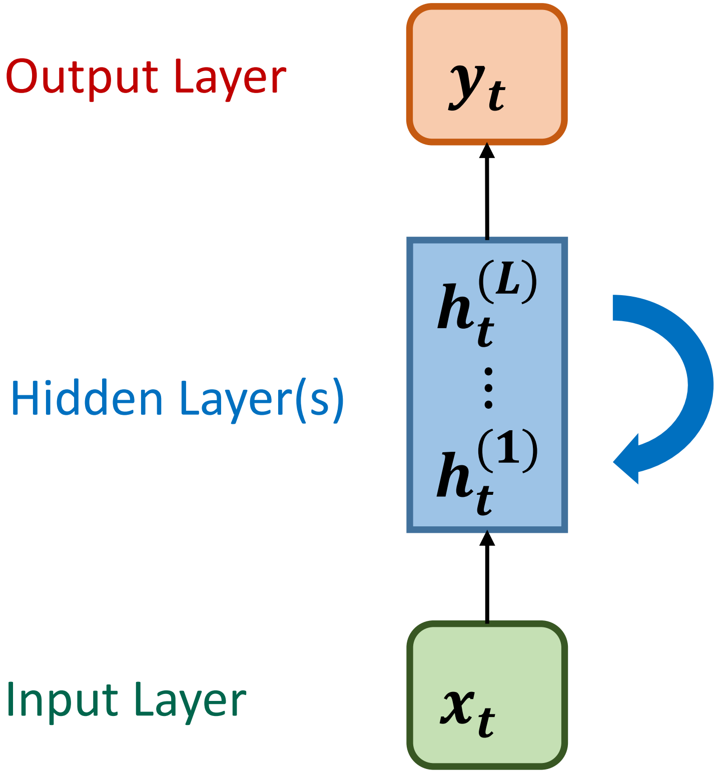
Gated Recurrent Units (GRU)



③ Updating the state:

$$h_t = u_t \odot h_{t-1} + (1 - u_t) \odot \widetilde{h_t}$$

Same input/output – different architectures



- Vanilla RNN
- LSTM
- GRU

Today – Recurrent Neural Networks

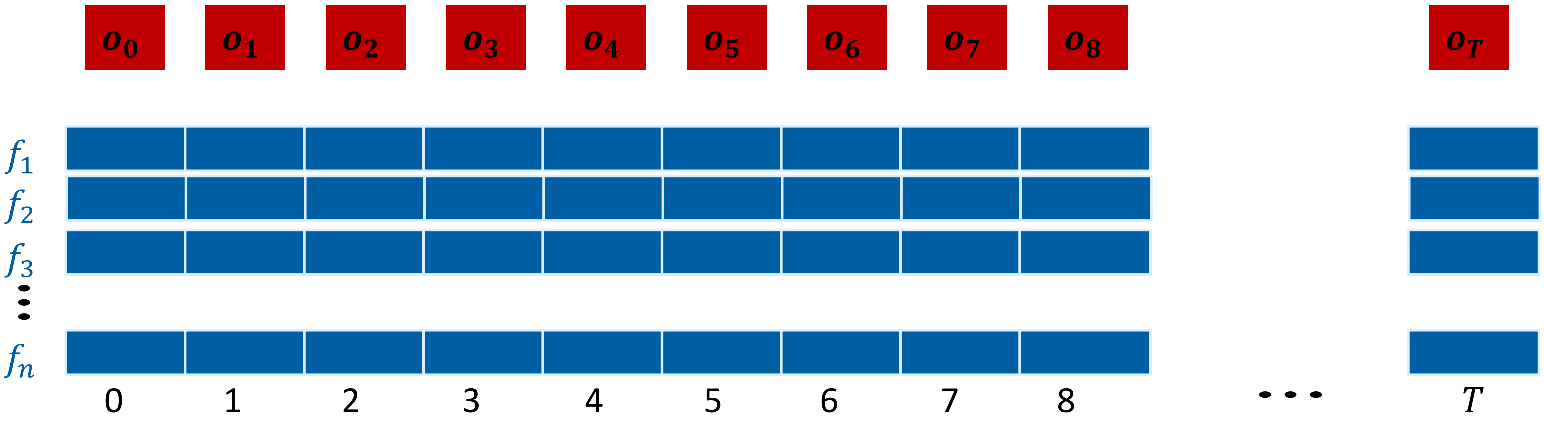
- Parameters and hyperparameter tuning
- Different architectures
- **Different tasks**:
  - **“Many-to-many” versus “Many-to-one”**
  - Classification versus Regression

Today – Recurrent Neural Networks

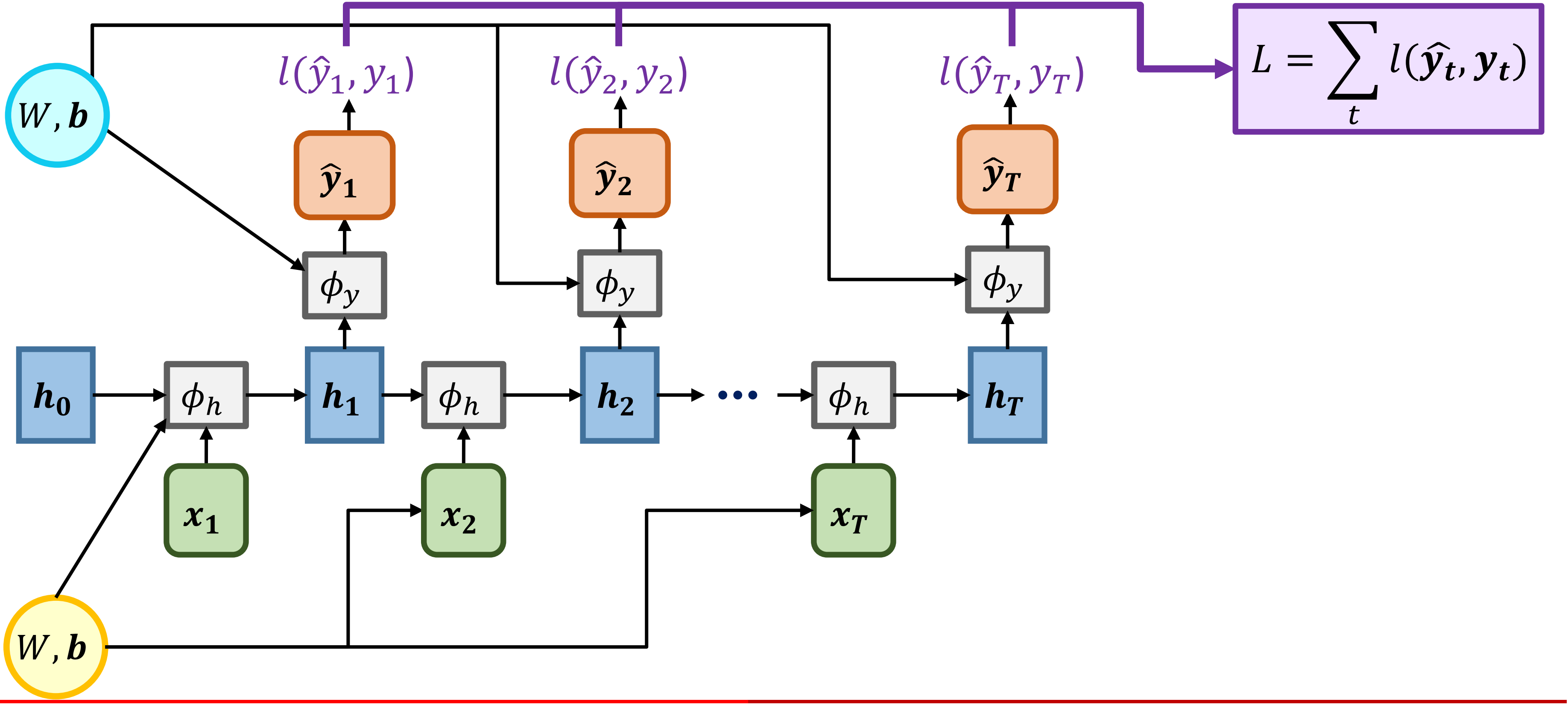
- Parameters and hyperparameter tuning
- Different architectures
- **Different tasks**:
  - **“Many-to-many” versus “Many-to-one”**
  - Classification versus Regression

Many-to-many aka the Tracing Task

- Prediction of a target variable o_t at each time step t

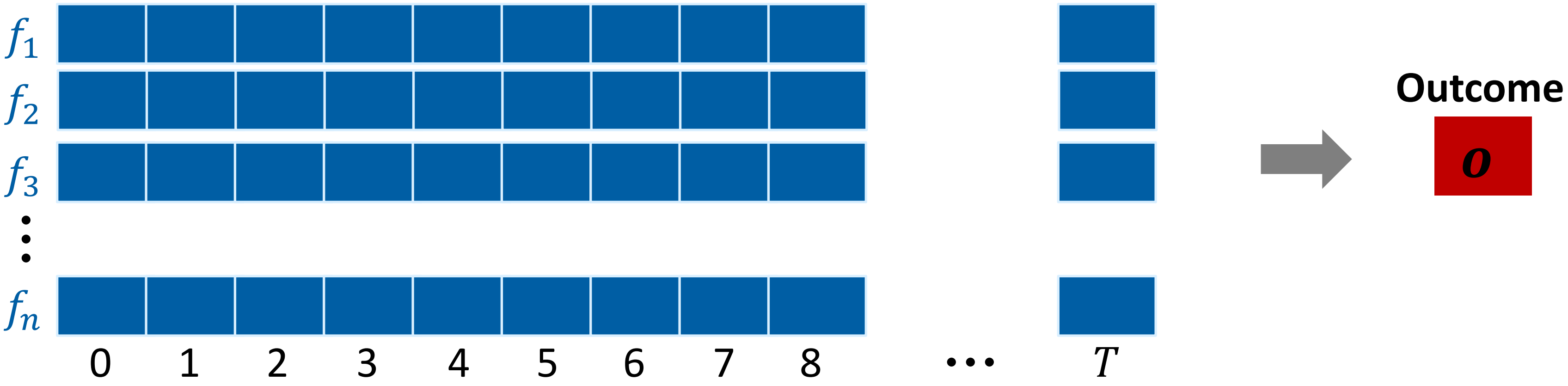


Computational Graph – Many-to-many

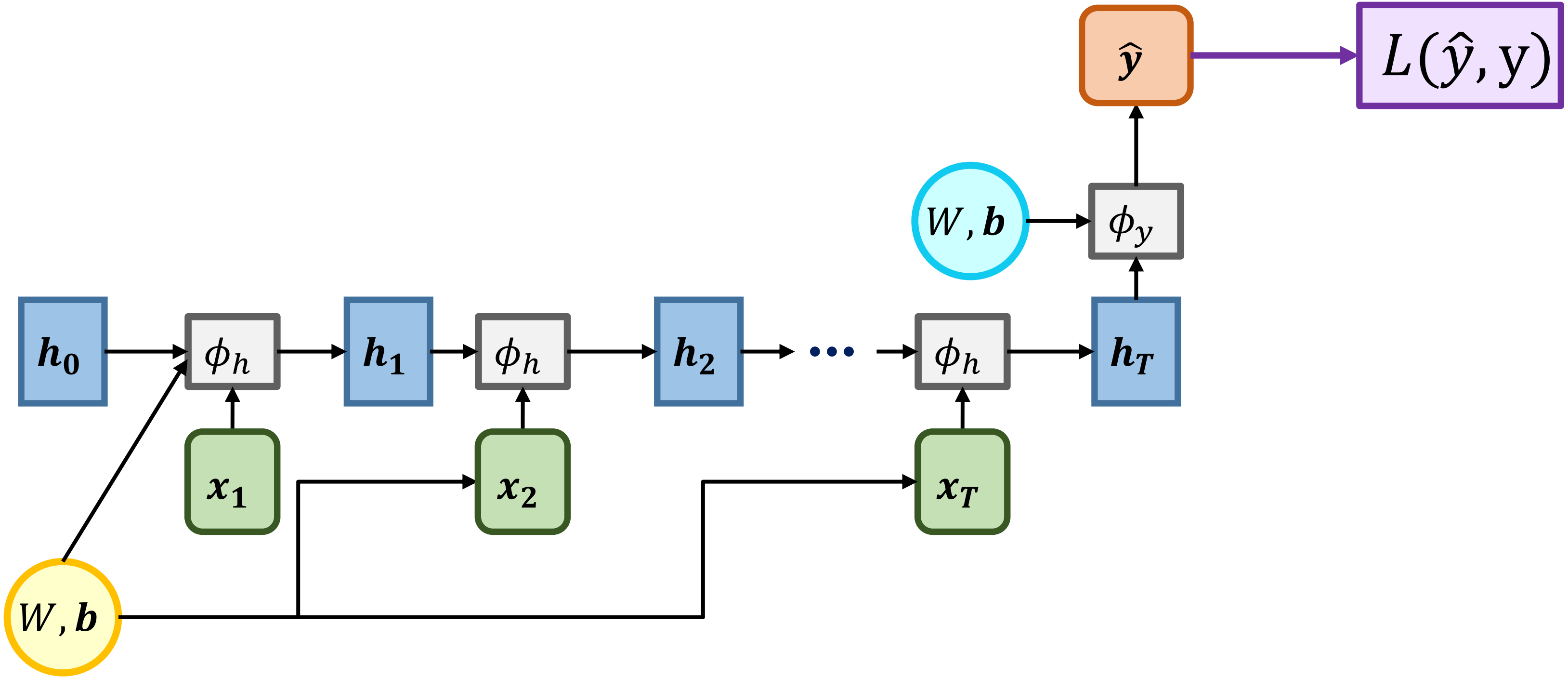


Many-to-one aka the Time-Series Prediction Task

- Prediction of a target variable $\boldsymbol{o}$ after $t \leq T$ time steps, where T is the total number of time steps



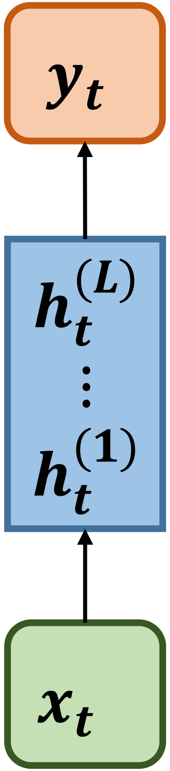
Computational Graph – Many-to-one



Today – Recurrent Neural Networks

- Parameters and hyperparameter tuning
- Different architectures
- **Different tasks**:
  - “Many-to-many” versus “Many-to-one”
  - **Classification versus Regression**

Classification vs. Regression



Output Layer

$\boldsymbol{y_t}$

Hidden Layer(s)

$\boldsymbol{h_t^{(L)}}$

$\vdots$

$\boldsymbol{h_t^{(1)}}$

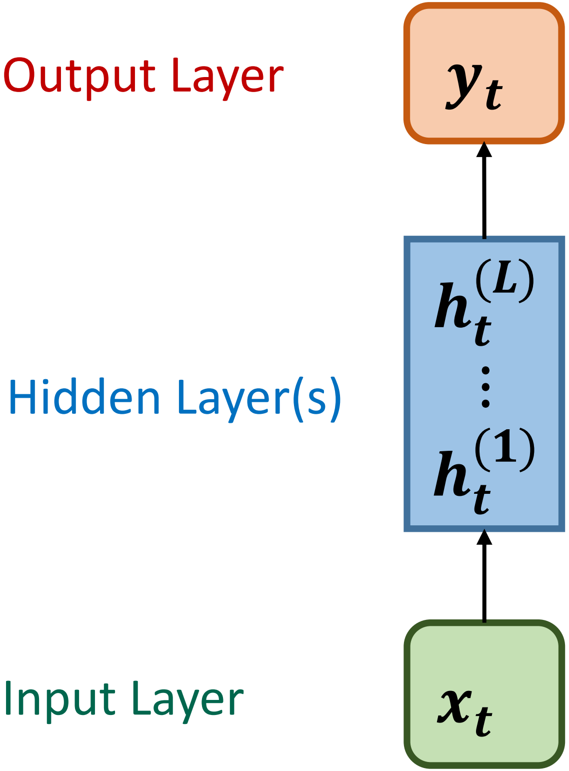
Input Layer

$\boldsymbol{x_t}$

$$\boldsymbol{y_t} = \phi_y(W_{yh}\boldsymbol{h}_t^L + \boldsymbol{b}^{(y)})$$

$$\boxed{L(\widehat{\boldsymbol{y_t}}, \boldsymbol{y_t})}$$

Classification vs. Regression: Output Layer



Output Layer

$\boldsymbol{y_t}$

Hidden Layer(s)

$\boldsymbol{h_t^{(L)}}$

$\vdots$

$\boldsymbol{h_t^{(1)}}$

Input Layer

$\boldsymbol{x_t}$

$$\boldsymbol{y_t} = \phi_y(W_{yh}\boldsymbol{h}_t^L + \boldsymbol{b}^{(y)})$$

$$\boxed{L(\widehat{\boldsymbol{y_t}}, \boldsymbol{y_t})}$$

Classification

Regression

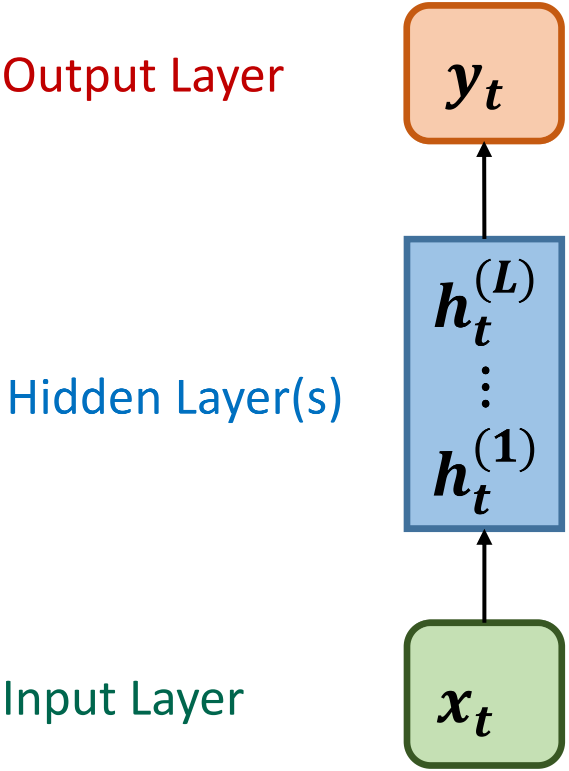
Sigmoid activation:

$$\boldsymbol{y_t} = \sigma(W_{yh}\boldsymbol{h}_t^L + \boldsymbol{b}^{(y)})$$

Linear activation:

$$\boldsymbol{y_t} = W_{yh}\boldsymbol{h}_t^L + \boldsymbol{b}^{(y)}$$

Classification vs. Regression: Training Loss



$$\boldsymbol{y_t} = \phi_y(W_{yh}\boldsymbol{h}_t^L + \boldsymbol{b}^{(y)})$$

$L(\widehat{\boldsymbol{y_t}}, \boldsymbol{y_t})$

Classification

- Binary crossentropy
- Categorical crossentropy

Regression

Mean squared error

Your Turn

- Given:
  - Data from a MOOC
  - An LSTM for predicting quiz performance of a student for every week of the course (tracing task)
- Your Task:
  1) Adjust the create_model function in order to predict pass/fail after 5 weeks of the course (time series prediction task) and send us the binary accuracy + AUC
     - Hint 1: return_sequences=False
     - Hint 2: what does TimeDistributed(…) do?
  2) Tune hyperparameters of your choice and send us binary accuracy and AUC

Summary

- Parameters and hyperparameter tuning
- Different architectures
- Different tasks:
  - “Many-to-many” versus “Many-to-one”
  - Classification versus Regression

Lab Session on April 13

- First part (8.15-9.00): hands-on tutorial on RNNs
- Second part (starting at 9.15): office hours for questions regarding the project