THE HONG KONG POLYTECHNIC UNIVERSITY
香港理工大學

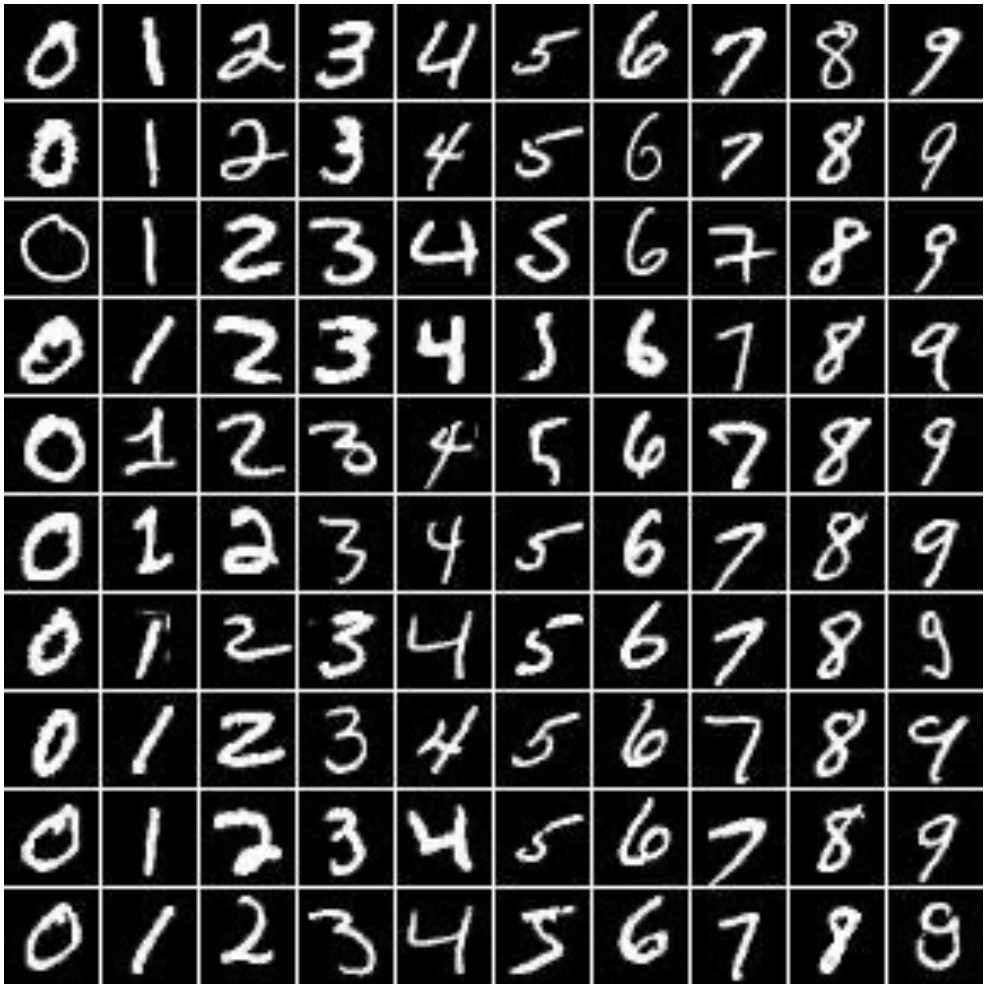
COMP4434 Big Data Analytics

Lab 8
Recurrent Neural Networks

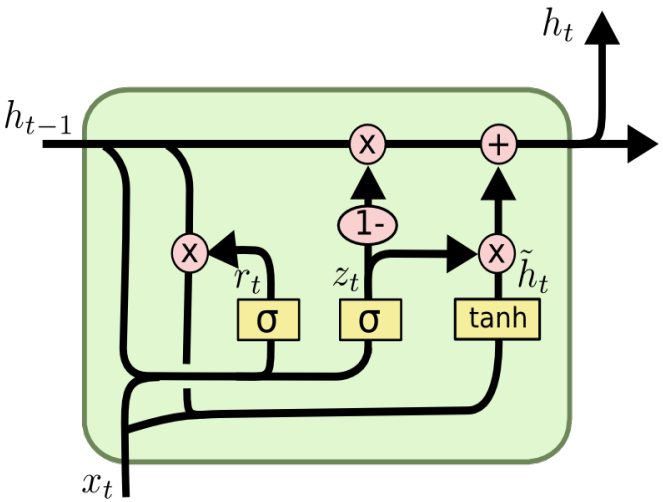
HUANG Xiao
xiaohuang@comp.polyu.edu.hk

MNIST dataset

- The MNIST dataset contains handwritten digits from 0 - 9 (10 classes)
- The dataset has a training set of 60,000 28 x 28 x 1 (height x width x channel) images with 6000 examples per class, and a test set of 10,000 images
- The digits have been size-normalized and centered in a fixed-size image
- It is easy to classify even with simple ML algorithm(99.2% accuray achieved by SVM)



Gated Recurrent Unit (GRU)

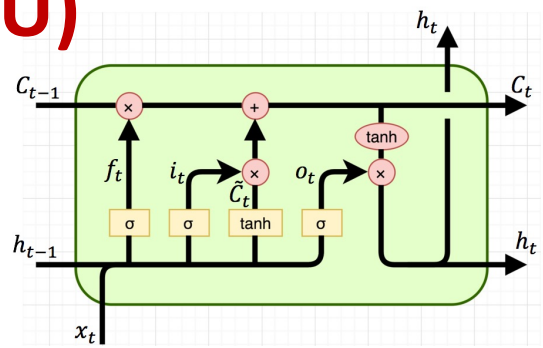


$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$



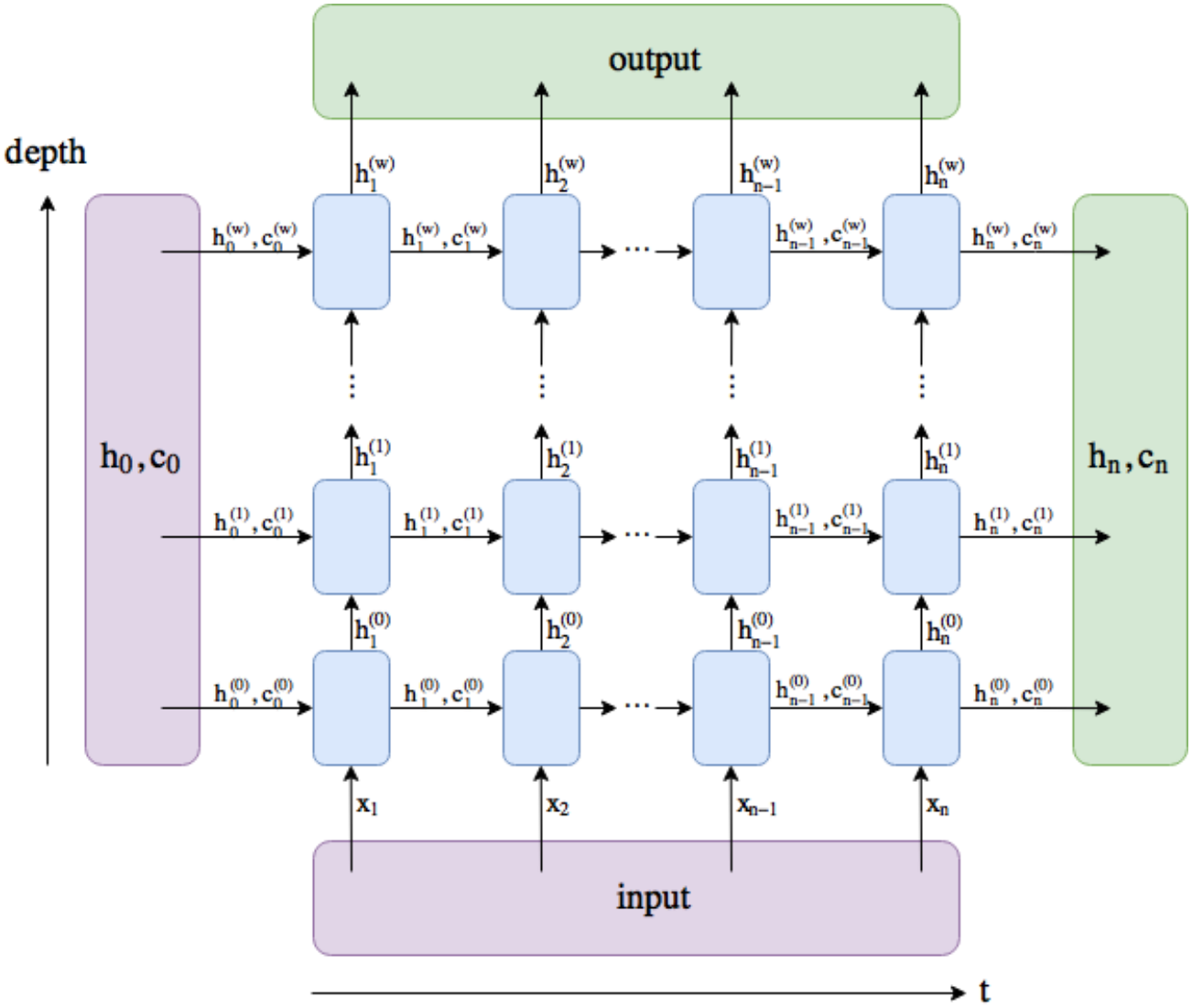
- CLASS torch.nn.GRU(self, input_size, hidden_size, num_layers=1, bias=True, batch_first=False, dropout=0.0, bidirectional=False, device=None, dtype=None)
  - **input_size** – The number of expected features in the input x
  - **hidden_size** – The number of features in the hidden state h
  - **num_layers** – Number of recurrent layers. E.g., setting num_layers=2 would mean stacking two GRUs together to form a stacked GRU, with the second GRU taking in outputs of the first GRU and computing the final results. Default: 1

Input of nn.GRU

- **input**: tensor of shape (L, H_{in}) for unbatched input, (L, N, H_{in}) when `batch_first=False` or (N, L, H_{in}) when `batch_first=True` containing the features of the input sequence.
- **h_0**: tensor of shape $(D * \text{num\_layers}, H_{out})$ or $(D * \text{num\_layers}, N, H_{out})$ containing the initial hidden state for the input sequence. Defaults to zeros if not provided.
- **output**: tensor of shape $(L, D * H_{out})$ for unbatched input, $(L, N, D * H_{out})$ when `batch_first=False` or $(N, L, D * H_{out})$ when `batch_first=True` containing the output features (*h_t*) from the last layer of the GRU, for each *t*. If a `torch.nn.utils.rnn.PackedSequence` has been given as the
- **h_n**: tensor of shape $(D * \text{num\_layers}, H_{out})$ or $(D * \text{num\_layers}, N, H_{out})$ containing the final hidden state for the input sequence.

- N = batch size
- L = sequence length
- H_{in}= input size
- H_{out}= hidden size
- D = 2 if bidirectional=True otherwise 1

num_layers


output
depth
$h_1^{(w)}$
$h_2^{(w)}$
$h_{n-1}^{(w)}$
$h_n^{(w)}$
$h_0^{(w)}, c_0^{(w)}$
$h_1^{(w)}, c_1^{(w)}$
$h_{n-1}^{(w)}, c_{n-1}^{(w)}$
$h_n^{(w)}, c_n^{(w)}$
h_0, c_0
h_n, c_n
$h_1^{(1)}$
$h_2^{(1)}$
$h_{n-1}^{(1)}$
$h_n^{(1)}$
$h_0^{(1)}, c_0^{(1)}$
$h_1^{(1)}, c_1^{(1)}$
$h_{n-1}^{(1)}, c_{n-1}^{(1)}$
$h_n^{(1)}, c_n^{(1)}$
$h_1^{(0)}$
$h_2^{(0)}$
$h_{n-1}^{(0)}$
$h_n^{(0)}$
$h_0^{(0)}, c_0^{(0)}$
$h_1^{(0)}, c_1^{(0)}$
$h_{n-1}^{(0)}, c_{n-1}^{(0)}$
$h_n^{(0)}, c_n^{(0)}$
x_1
x_2
x_{n-1}
x_n
input
t

model.eval() and torch.no_grad()

- model.eval() will notify all your layers that you are in eval mode, that way, batchnorm or dropout layers will work in eval mode instead of training mode.
- torch.no_grad() impacts the autograd engine and deactivate it. It will reduce memory usage and speed up computations but you won't be able to backprop (which you don't want in an eval script).
- model.train() tells your model that you are training the model. This helps inform layers such as Dropout and BatchNorm, which are designed to behave differently during training and evaluation.