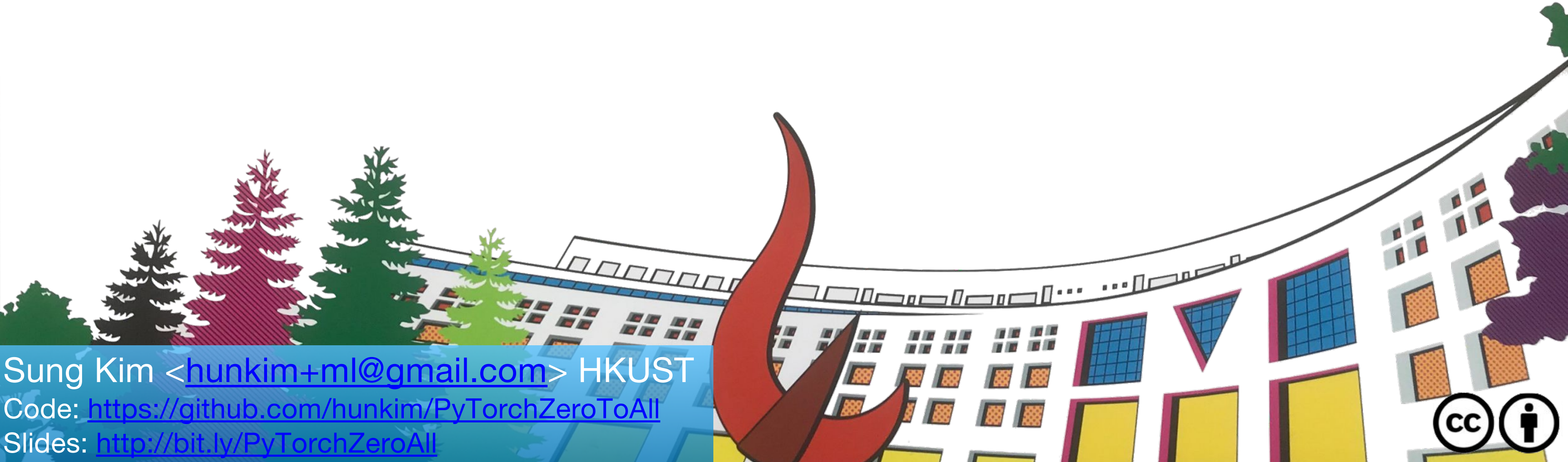
ML/DL for Everyone with PYTORCH

Lecture 11: RNN



Sung Kim <hunkim+ml@gmail.com> HKUST
Code: https://github.com/hunkim/PyTorchZeroToAll
Slides: http://bit.ly/PyTorchZeroAll

Call for Comments

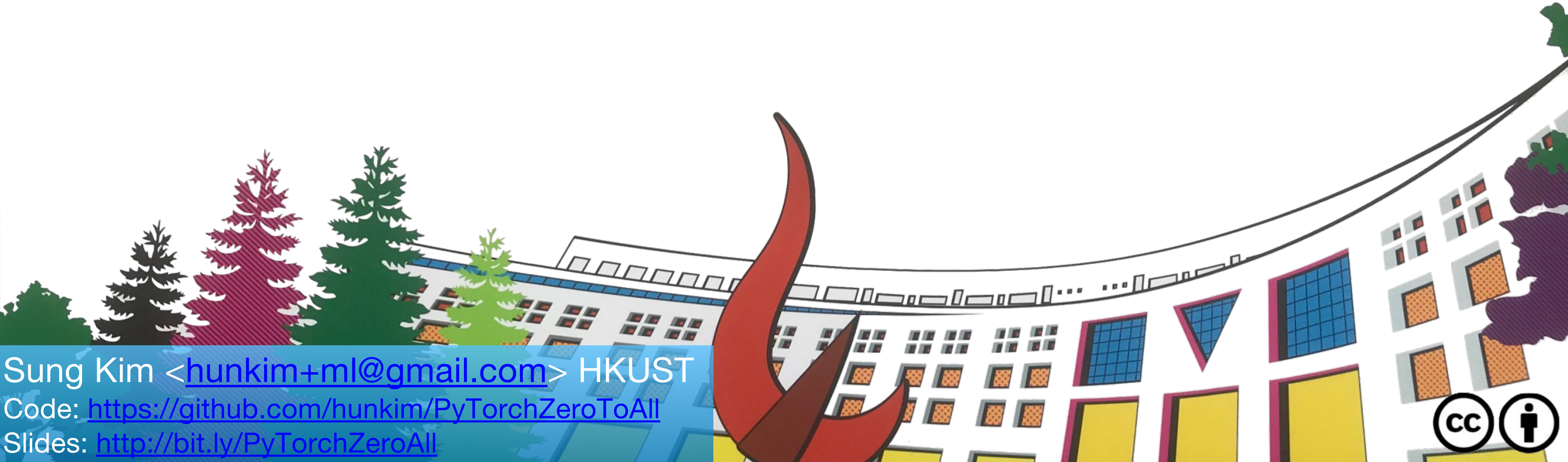
Please feel free to add comments directly on these slides.

Other slides: http://bit.ly/PyTorchZeroAll

Picture from http://www.tssablog.org/archives/3280

ML/DL for Everyone with PYTORCH

Lecture 11: RNN

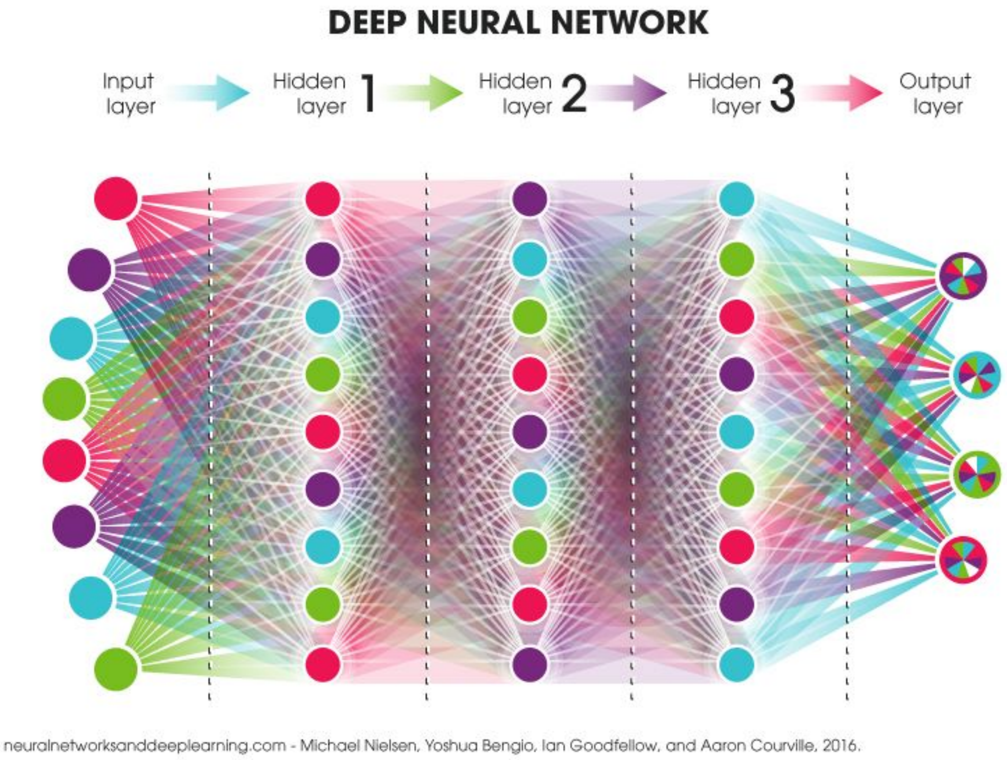


Sung Kim <hunkim+ml@gmail.com> HKUST

Code: https://github.com/hunkim/PyTorchZeroToAll

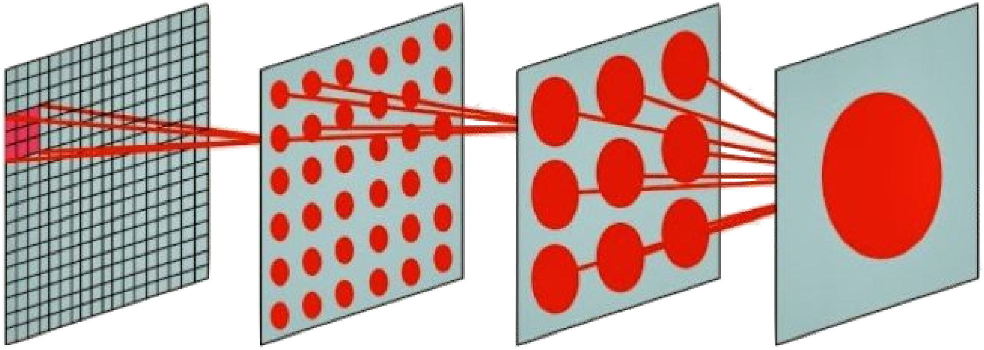
Slides: http://bit.ly/PyTorchZeroAll

DNN, CNN, RNN



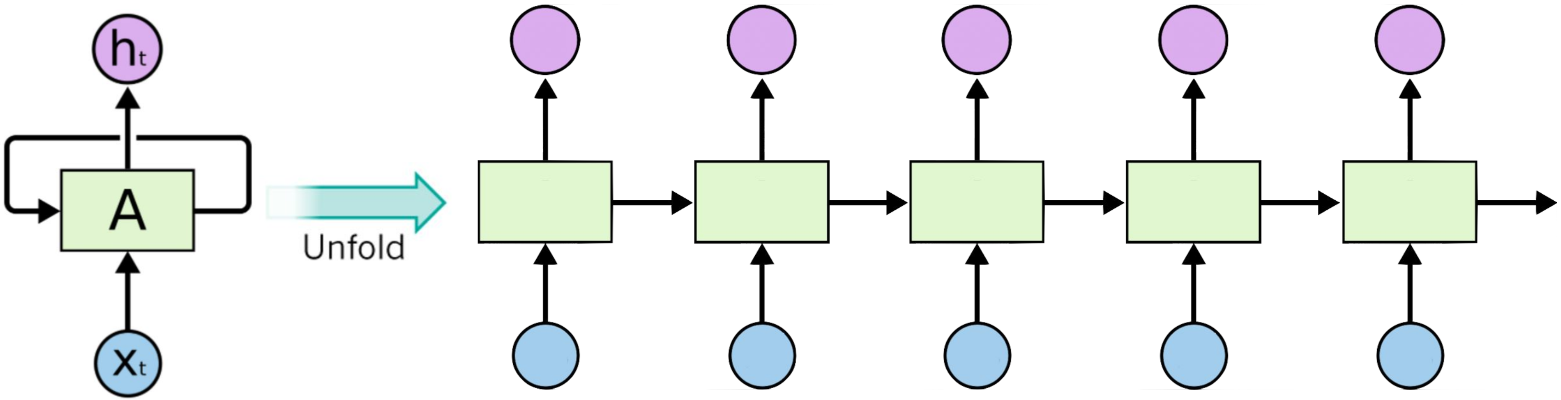
https://blog.ttro.com/artificial-intelligence-will-shape-e-learning-for-good/

DNN, CNN, RNN



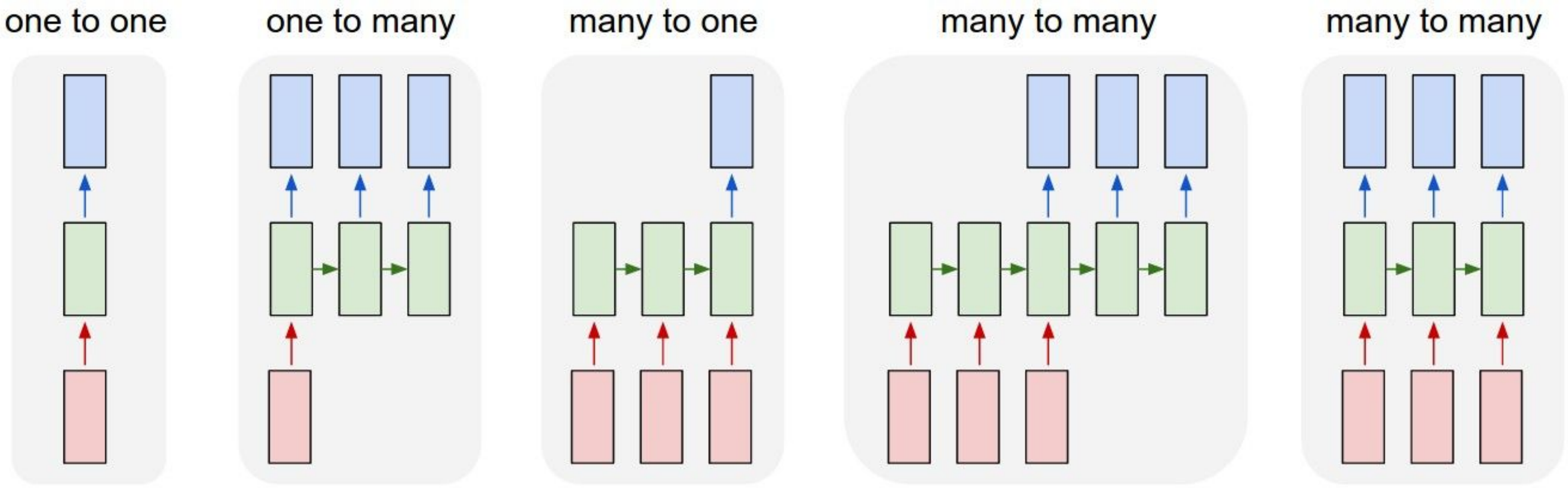
http://practicalquant.blogspot.hk/2013/10/deep-learning-oral-traditions.html

DNN, CNN, RNN



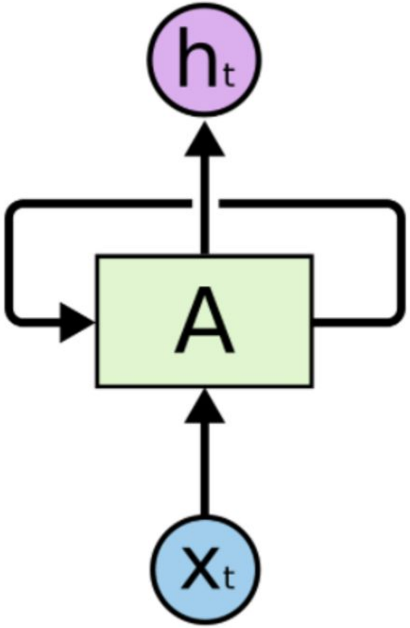
http://practicalquant.blogspot.hk/2013/10/deep-learning-oral-traditions.html

RNN Applications



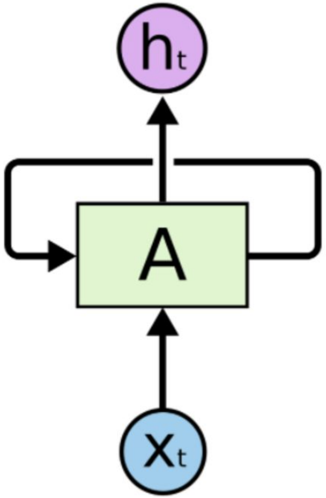
http://karpathy.github.io/2015/05/21/rnn-effectiveness/

RNN in PyTorch




```
cell = nn.LSTM(input_size=4, hidden_size=2, batch_first=True)

...

inputs = ... # (batch, seq_len, input_size) with batch_first=True
hidden = (... , ...) # (num_layers * num_directions, batch, hidden_size)

...

out, hidden = cell(inputs, hidden)
```

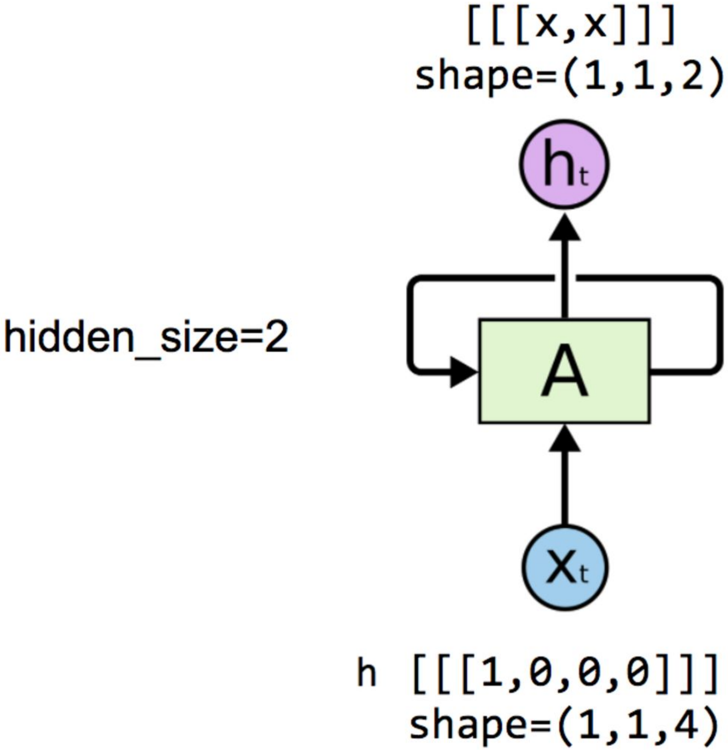
One node: 4 (*input-dim*) in 2 (*hidden_size*)




```
h [[[1,0,0,0]]]
  shape=(1,1,4)
```

```
# One hot encoding
h = [1, 0, 0, 0]
e = [0, 1, 0, 0]
l = [0, 0, 1, 0]
o = [0, 0, 0, 1]
```

One node: 4 (*input-dim*) in 2 (*hidden_size*)

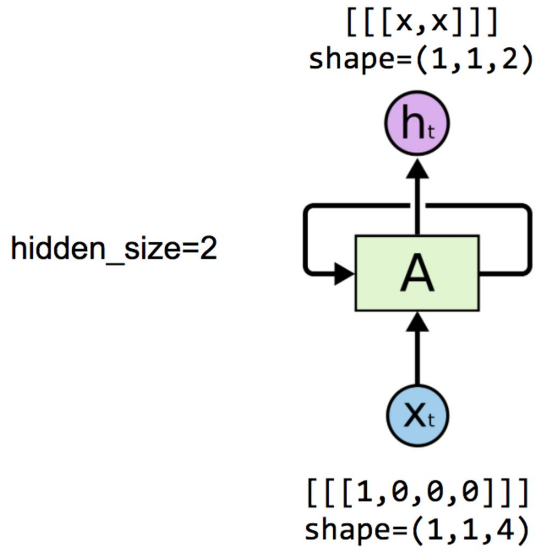



```
# One hot encoding
h = [1, 0, 0, 0]
e = [0, 1, 0, 0]
l = [0, 0, 1, 0]
o = [0, 0, 0, 1]
```

One node: 4 (*input_dim*) in 2 (*hidden_size*)

```
# One cell RNN input_dim (4) -> output_dim (2)
cell = nn.LSTM(input_size=4, hidden_size=2, batch_first=True)

# One letter input
inputs = autograd.Variable(torch.Tensor([[h]])) # rank = (1, 1, 4)

# initialize the hidden state.
# (num_layers * num_directions, batch, hidden_size)
hidden = (autograd.Variable(torch.randn(1, 1, 2)),
          autograd.Variable(torch.randn((1, 1, 2))))

# Feed to one element at a time.
# after each step, hidden contains the hidden state.
out, hidden = cell(inputs, hidden)
print("out", out.data)
```

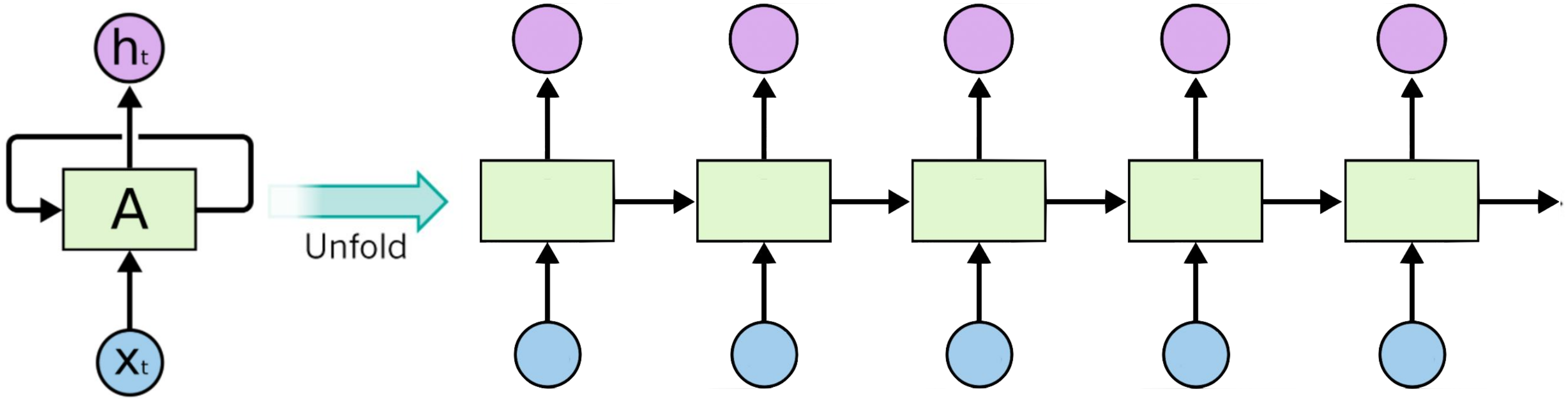
```
-0.1243  0.0738
[torch.FloatTensor of size 1x1x2]
```



Unfolding to n sequences

```
hidden_size=2
seq_len=5
```

shape=(1,5,2): [[[x,x], [x,x], [x,x], [x,x], [x,x]]]



shape=(1,5,4): [[[1,0,0,0], [0,1,0,0], [0,0,1,0], [0,0,1,0], [0,0,0,1]]]
h e l l o

https://github.com/hunkim/DeepLearningZeroToAll/blob/master/lab-12-0-rnn_basics.ipynb

Unfolding to n sequences

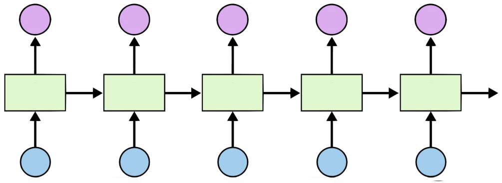
```
# One cell RNN input_dim (4) -> output_dim (2). sequence: 5
cell = nn.LSTM(input_size=4, hidden_size=2, batch_first=True)

inputs = autograd.Variable(torch.Tensor([[h, e, l, l, o]]))
print("input size", inputs.size())

hidden = (autograd.Variable(torch.randn(1, 1, 2)), autograd.Variable(
    torch.randn((1, 1, 2))))  # clean out hidden state

out, hidden = cell(inputs, hidden)
print(out.data)
```

```
# One hot encoding
h = [1, 0, 0, 0]
e = [0, 1, 0, 0]
l = [0, 0, 1, 0]
o = [0, 0, 0, 1]
```

```
input size torch.Size([1, 5, 4])

(0 ,.,.) =
 -0.1825  0.0737
 -0.1981  0.1164
 -0.3367  0.2095
 -0.3625  0.2503
 -0.2038  0.3626
[torch.FloatTensor of size 1x5x2]
```

hidden_size=2
seq_len=5

shape=(1,5,2): [[[x,x], [x,x], [x,x], [x,x], [x,x]]]

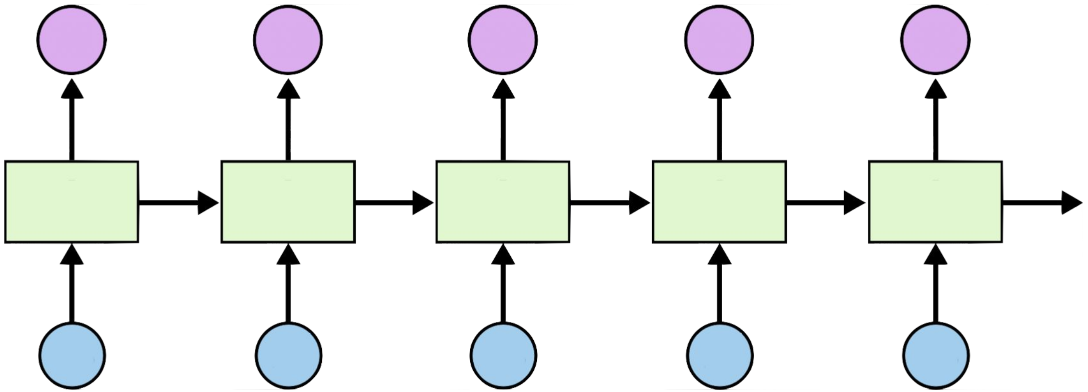


shape=(1,5,4): [[[1,0,0,0], [0,1,0,0], [0,0,1,0], [0,0,1,0], [0,0,0,1]]]
h e l l o

```
Hidden_size=2
sequence_length=5
batch_size=3
```

Batching input

```
shape=(3,5,2): [[[x,x], [x,x], [x,x], [x,x], [x,x]],
                [[x,x], [x,x], [x,x], [x,x], [x,x]],
                [[x,x], [x,x], [x,x], [x,x], [x,x]]]
```

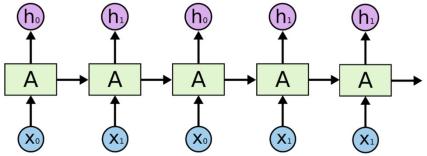


```
shape=(3,5,4): [[[1,0,0,0], [0,1,0,0], [0,0,1,0], [0,0,1,0], [0,0,0,1]], # hello
                [[0,1,0,0], [0,0,0,1], [0,0,1,0], [0,0,1,0], [0,0,1,0]]  # eolll
                [[0,0,1,0], [0,0,1,0], [0,1,0,0], [0,1,0,0], [0,0,1,0]]] # lleel
```

Batching input

```python
# One cell RNN input_dim (4) -> output_dim (2). sequence: 5, batch 3
# 3 batches 'hello', 'eolll', 'lleel'
# rank = (3, 5, 4)
inputs = autograd.Variable(torch.Tensor([[h, e, l, l, o],
                                         [e, o, l, l, l],
                                         [l, l, e, e, l]]))
print("input size", inputs.size()) # input size torch.Size([3, 5, 4])

# (num_layers * num_directions, batch, hidden_size)
hidden = (autograd.Variable(torch.randn(1, 3, 2)), autograd.Variable(
   torch.randn((1, 3, 2))))

out, hidden = cell(inputs, hidden)
print("out size", out.size()) # out size torch.Size([3, 5, 2])
```

```
shape=(3,5,2): [[[x,x], [x,x], [x,x], [x,x], [x,x]],
                [[x,x], [x,x], [x,x], [x,x], [x,x]],
                [[x,x], [x,x], [x,x], [x,x], [x,x]]]
```

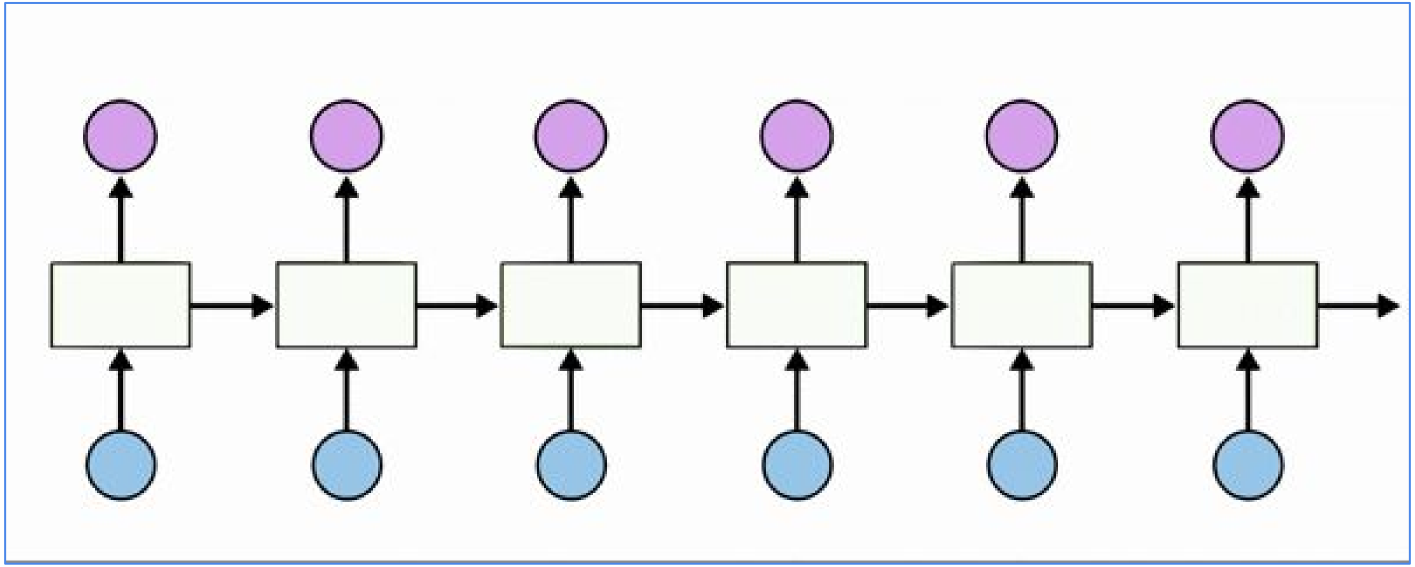



```
shape=(3,5,4): [[[1,0,0,0], [0,1,0,0], [0,0,1,0], [0,0,1,0], [0,0,0,1]], # hello
                [[0,1,0,0], [0,0,0,1], [0,0,1,0], [0,0,1,0], [0,0,1,0]]  # eolll
                [[0,0,1,0], [0,0,1,0], [0,1,0,0], [0,1,0,0], [0,0,1,0]]] # lleel
```

Hidden_size=2
sequence_length=5
batch_size=3

Teach RNN ‘hihell’ to ‘ihello’

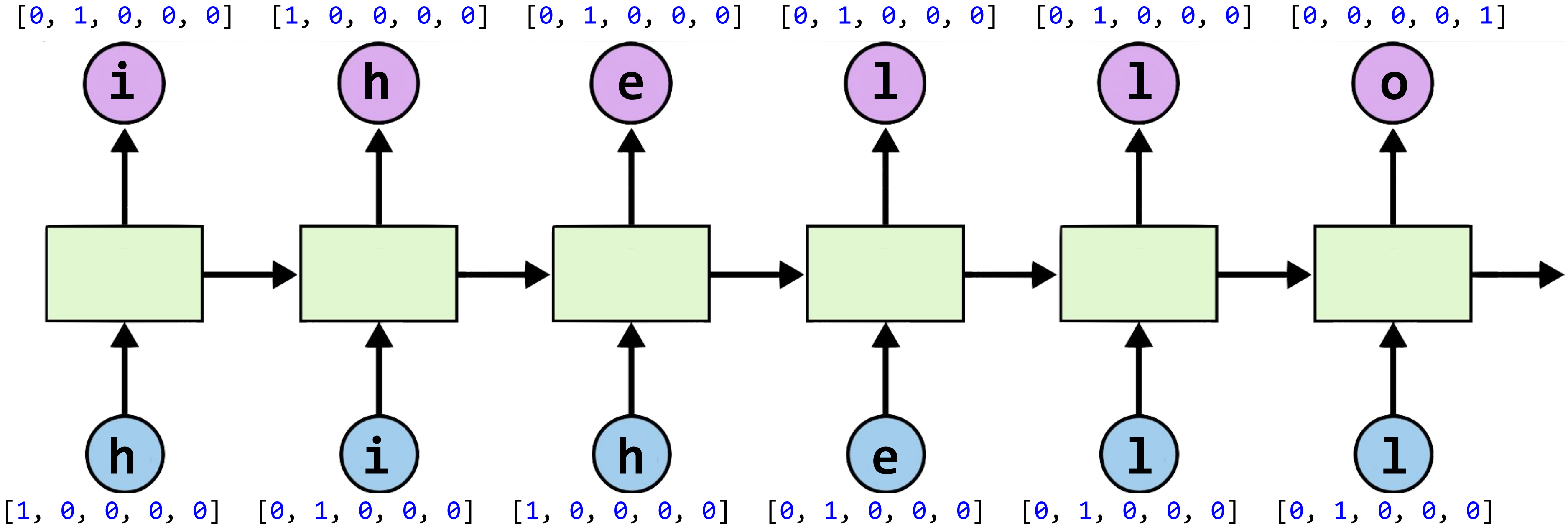
```
[1, 0, 0, 0, 0],   # h 0
[0, 1, 0, 0, 0],   # i 1
[0, 0, 1, 0, 0],   # e 2
[0, 0, 0, 1, 0],   # l 3
[0, 0, 0, 0, 1],   # o 4
```



Teach RNN ‘hihell’ to ‘ihello’

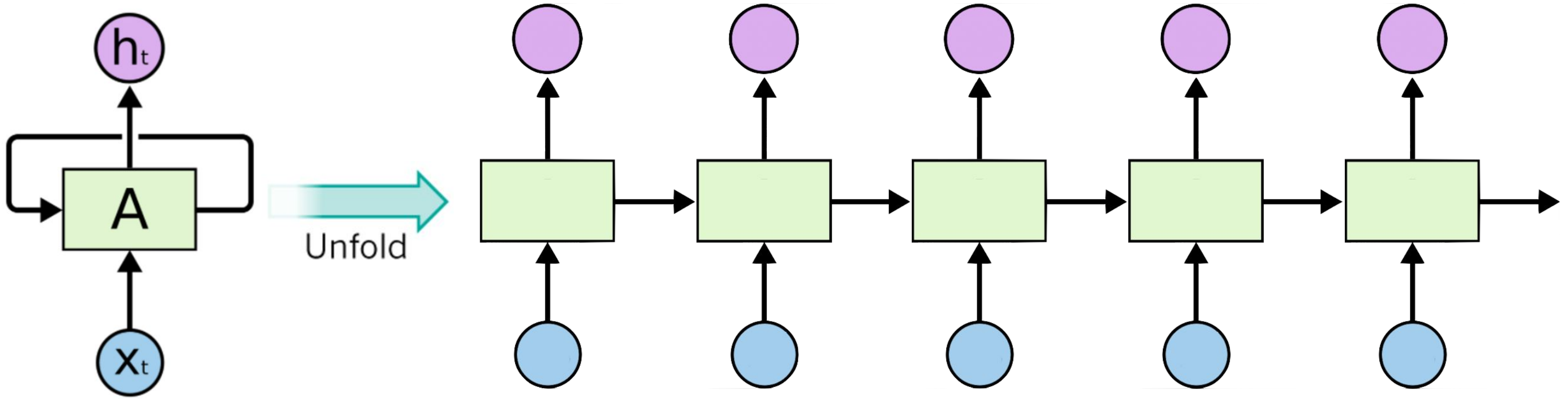
```
[1, 0, 0, 0, 0],   # h 0
[0, 1, 0, 0, 0],   # i 1
[0, 0, 1, 0, 0],   # e 2
[0, 0, 0, 1, 0],   # l 3
[0, 0, 0, 0, 1],   # o 4
```

`output_dim = 5`



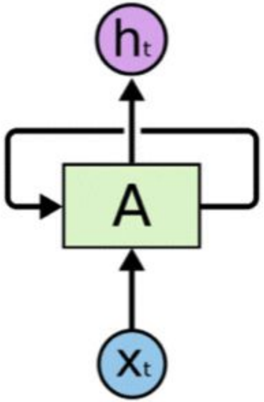
`Input_dim = 5`

Unfolding one to n sequences



Teach RNN ‘hihell’ to ‘ihello’

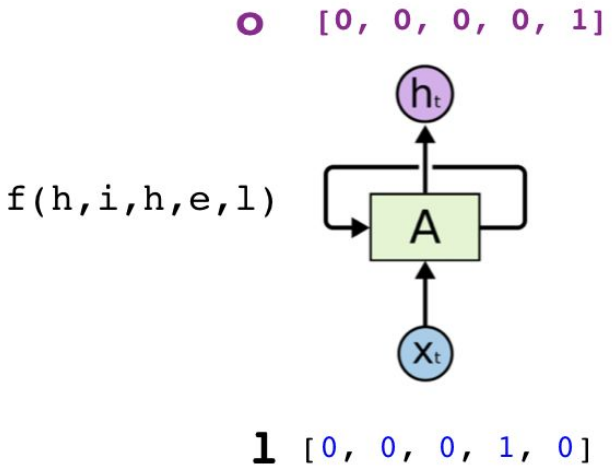
```
[1, 0, 0, 0, 0],   # h 0
[0, 1, 0, 0, 0],   # i 1
[0, 0, 1, 0, 0],   # e 2
[0, 0, 0, 1, 0],   # l 3
[0, 0, 0, 0, 1],   # o 4
```



RNN inout and output

```
[1, 0, 0, 0, 0],    # h 0
[0, 1, 0, 0, 0],    # i 1
[0, 0, 1, 0, 0],    # e 2
[0, 0, 0, 1, 0],    # l 3
[0, 0, 0, 0, 1],    # o 4
```

```
self.rnn = nn.RNN(input_size=5, hidden_size=5, batch_first=True)
```

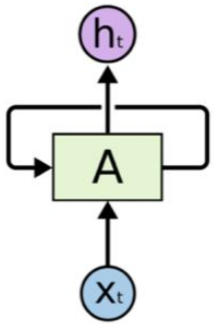


Designing Loss

Predict Y, one of five {h, i, e, l, o}

o [0, 0, 0, 0, 1]

f(h,i,h,e,l)



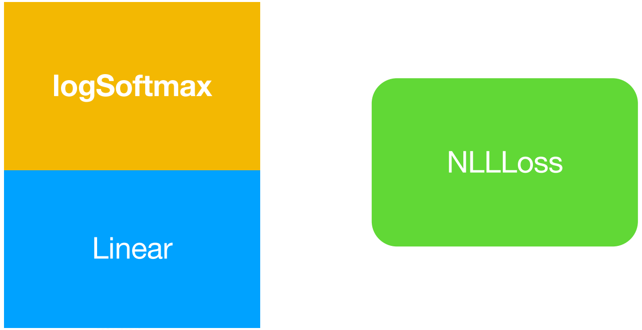
Some input: X l [0, 0, 0, 1, 0]

Designing Loss

Predict Y, one of five {h, i, e, l, o} $Y \in R^{N \times ?}$

`o [0, 0, 0, 0, 1]`

`f(h,i,h,e,l)`



Some input: X `l [0, 0, 0, 1, 0]`

Designing Loss

```
out = rnn_out.view(-1, 5)
```

Predict Y, one of five {h, i, e, l, o} $Y \in R^{N \times 5}$

o [0, 0, 0, 0, 1]

f(h,i,h,e,l)

logSoftmax

Linear

With NLLLoss

Some input: X l [0, 0, 0, 1, 0]

(I) Data preperation

```python
idx2char = ['h', 'i', 'e', 'l', 'o']

# Teach hihell -> ihello
x_data = [[0, 1, 0, 2, 3, 3]]   # hihell
x_one_hot = [[[1, 0, 0, 0, 0],   # h 0
              [0, 1, 0, 0, 0],   # i 1
              [1, 0, 0, 0, 0],   # h 0
              [0, 0, 1, 0, 0],   # e 2
              [0, 0, 0, 1, 0],   # l 3
              [0, 0, 0, 1, 0]]]  # l 3

y_data = [1, 0, 2, 3, 3, 4]    # ihello

# As we have one batch of samples, we will change them to variables only once
inputs = Variable(torch.Tensor(x_one_hot))
labels = Variable(torch.LongTensor(y_data))
```

(2) Parameters

```
num_classes = 5
input_size = 5  # one-hot size
hidden_size = 5  # output from the LSTM. 5 to directly predict one-hot
batch_size = 1   # one sentence
sequence_length = 6  # |ihello| == 6
num_layers = 1  # one-layer rnn
```

(3) Our model

```python
def __init__(self, num_classes, input_size, hidden_size, num_layers):
  super(RNN, self).__init__()

   self.num_classes = num_classes
   self.num_layers = num_layers
   self.input_size = input_size
   self.hidden_size = hidden_size
   self.sequence_length = sequence_length

   self.rnn = nn.RNN(input_size=5, hidden_size=5, batch_first=True)

def forward(self, x):
   # Initialize hidden and cell states
   h_0 = Variable(torch.zeros(
       x.size(0), self.num_layers, self.hidden_size))

   # Reshape input
   x.view(x.size(0), self.sequence_length, self.input_size)

   # Propagate input through RNN
   # Input: (batch, seq_len, input_size)
   # h_0: (batch, num_layers * num_directions, hidden_size)

   out, _ = self.rnn(x, h_0)
   print(out.size())
   return out.view(-1, num_classes)
```

$$Y \in R^{N \times 5}$$

```python
num_classes = 5
input_size = 5  # one-hot size
hidden_size = 5  # output from the LSTM.
batch_size = 1   # one sentence
sequence_length = 6  # |ihello| == 6
num_layers = 1  # one-layer rnn
```

(4) Loss & Training

```python
# Instantiate RNN model
rnn = RNN(num_classes, input_size, hidden_size, num_layers)
print(rnn)

# Set loss and optimizer function
# CrossEntropyLoss = LogSoftmax + NLLLoss
criterion = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(rnn.parameters(), lr=0.1)

# Train the model
for epoch in range(100):
   outputs = rnn(inputs)
   optimizer.zero_grad()
   loss = criterion(outputs, labels)
   loss.backward()
   optimizer.step()
   _, idx = outputs.max(1)
   idx = idx.data.numpy()
   result_str = [idx2char[c] for c in idx.squeeze()]
   print("epoch: %d, loss: %1.3f" % (epoch + 1, loss.data[0]))
   print("Predicted string: ", ''.join(result_str))

print("Learning finished!")
```

```python
num_classes = 5
input_size = 5  # one-hot size
hidden_size = 5  # output from the LSTM.
batch_size = 1   # one sentence
sequence_length = 6  # |ihello| == 6
num_layers = 1  # one-layer rnn
```

(5) ENJOY the results

```python
# Instantiate RNN model
rnn = RNN(num_classes, input_size, hidden_size, num_layers)
print(rnn)

# Set loss and optimizer function
# CrossEntropyLoss = LogSoftmax + NLLLoss
criterion = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(rnn.parameters(), lr=0.1)

# Train the model
for epoch in range(100):
    outputs = rnn(inputs)
    optimizer.zero_grad()
    loss = criterion(outputs, labels)
    loss.backward()
    optimizer.step()
    _, idx = outputs.max(1)
    idx = idx.data.numpy()
    result_str = [idx2char[c] for c in idx.squeeze()]
    print("epoch: %d, loss: %1.3f" % (epoch + 1, loss.data[0]))
    print("Predicted string: ", ''.join(result_str))

print("Learning finished!")
```

```
epoch: 1, loss: 1.673
Predicted string:  ehehee
epoch: 2, loss: 1.403
Predicted string:  ehehel
epoch: 3, loss: 1.240
Predicted string:  ehelll
...
epoch: 95, loss: 0.458
Predicted string:  ihello
epoch: 96, loss: 0.458
Predicted string:  ihello
epoch: 97, loss: 0.458
Predicted string:  ihello
epoch: 98, loss: 0.458
Predicted string:  ihello
epoch: 99, loss: 0.458
Predicted string:  ihello
epoch: 100, loss: 0.458
Predicted string:  ihello
```

Exercise 11-1:
Implement softmax classifier for 'hihell' to 'ihello'

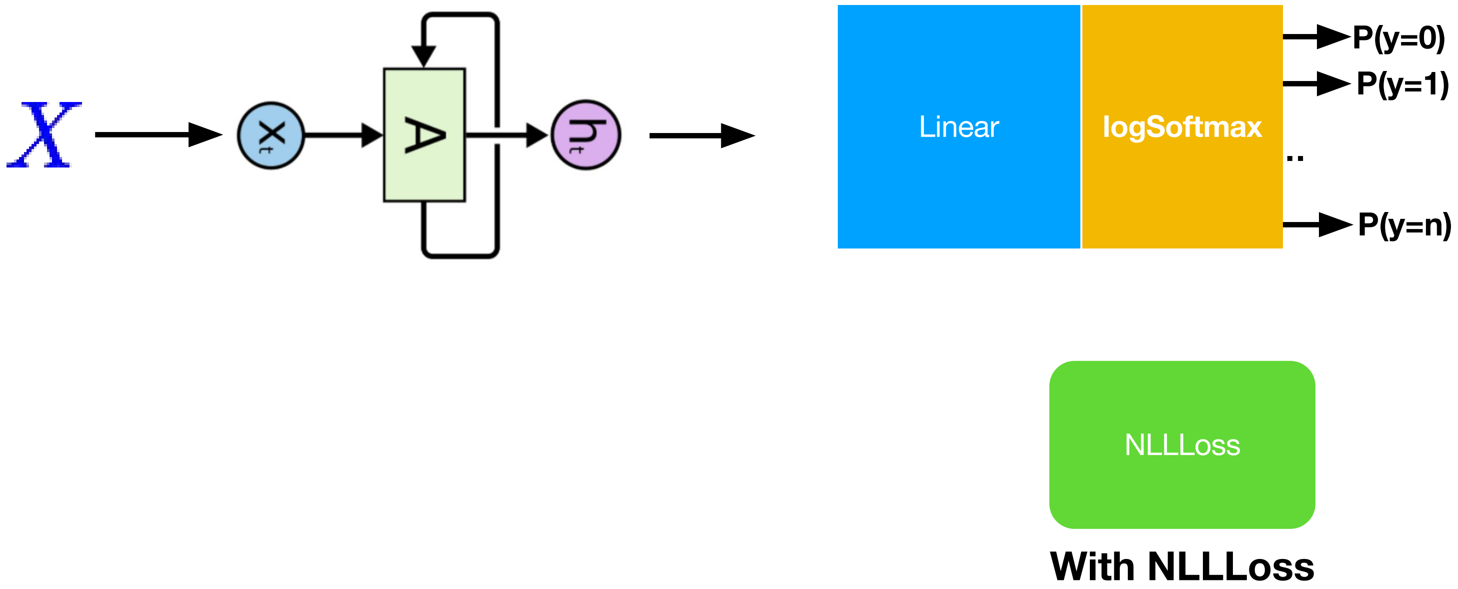
Predict Y, one of five {h, i, e, l, o} $Y \in R^{N \times 5}$

o [0, 0, 0, 0, 1]

Some input: X l [0, 0, 0, 1, 0]

Why does it not work?

Exercise 11-2: Combine RNN+Linear



Why does it train faster (more stable)?

Exercise 11-3: Teach RNN a long sequence

```
sentence = ("if you want to build a ship, don't drum up people together to "
            "collect wood and don't assign them tasks and work, but rather "
            "teach them to long for the endless immensity of the sea.")
```

Under the hood: RNN

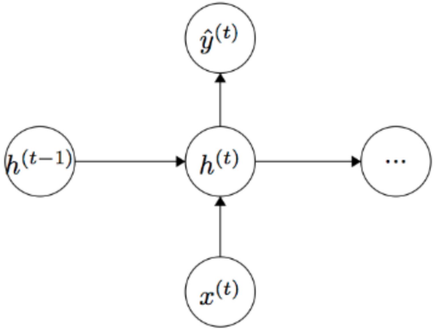


Figure 3: The inputs and outputs to a neuron of a RNN

https://cs224d.stanford.edu/lecture_notes/LectureNotes4.pdf

Under the hood: RNN

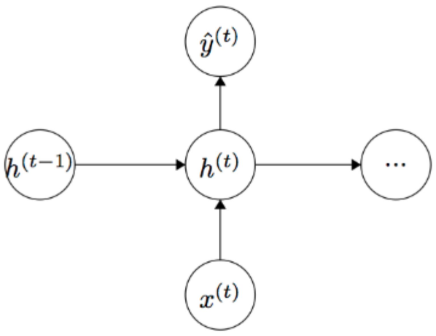


Figure 3: The inputs and outputs to a neuron of a RNN

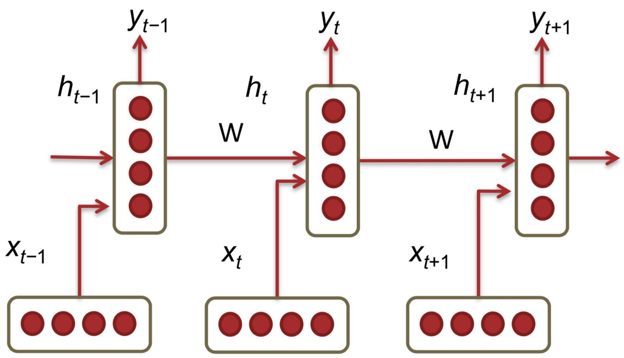


Figure 2: A Recurrent Neural Network (RNN). Three time-steps are shown.

https://cs224d.stanford.edu/lecture_notes/LectureNotes4.pdf

Under the hood: RNN

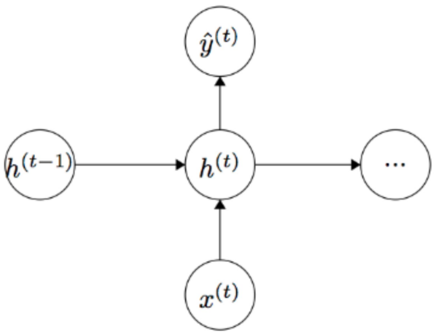


Figure 3: The inputs and outputs to a neuron of a RNN

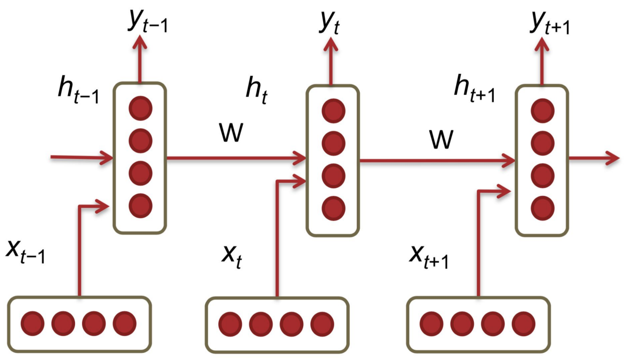


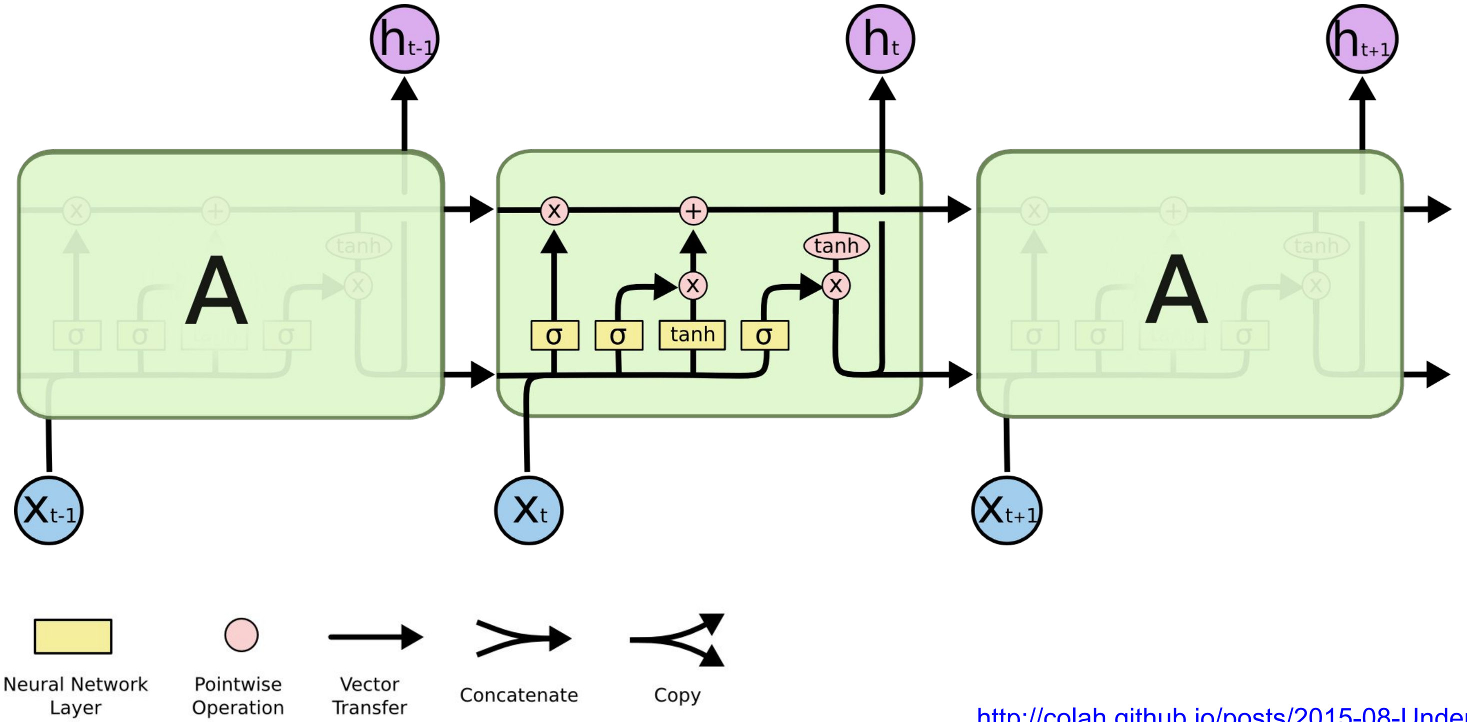
Figure 2: A Recurrent Neural Network (RNN). Three time-steps are shown.

$$h_t = W f(h_{t-1}) + W^{(hx)} x_t$$

$$\hat{y} = W^{(S)} f(h_t)$$

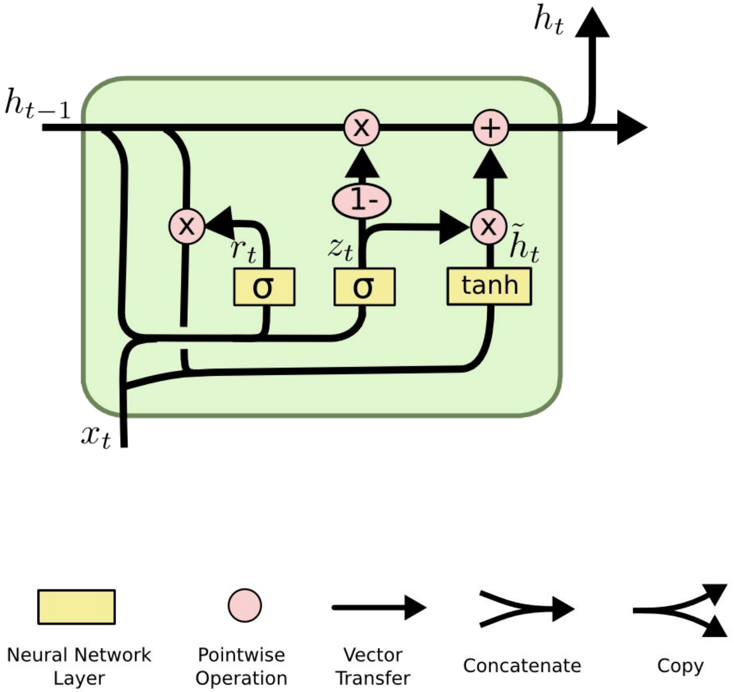
https://cs224d.stanford.edu/lecture_notes/LectureNotes4.pdf

Under the hood: LSTM



http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Under the hood: LSTM



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

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

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

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

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Exercise 11-4: Implement RNN using Numpy

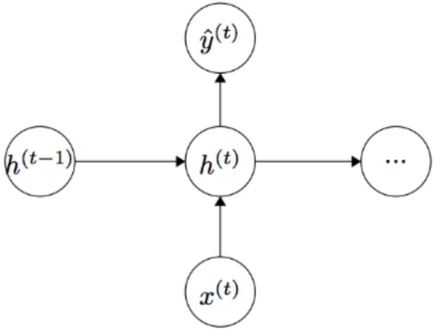


Figure 3: The inputs and outputs to a neuron of a RNN

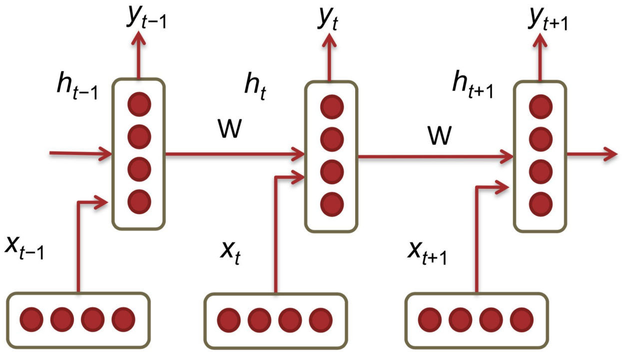


Figure 2: A Recurrent Neural Network (RNN). Three time-steps are shown.

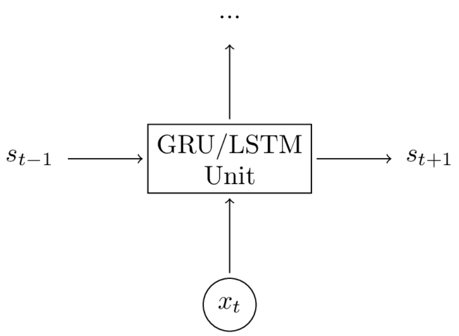
$$h_t = Wf(h_{t-1}) + W^{(hx)}x_t$$

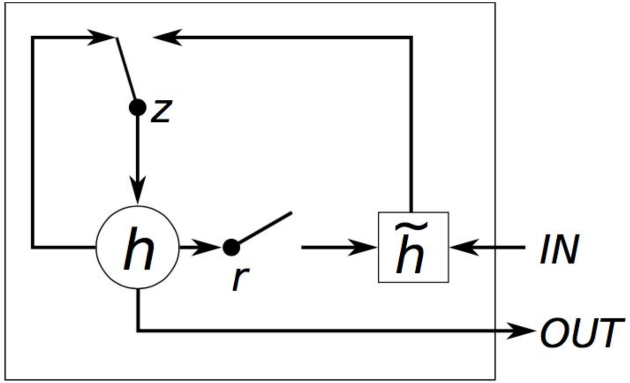
$$\hat{y} = W^{(S)}f(h_t)$$

Hint: http://blog.varunajayasiri.com/numpy_lstm.html

https://cs224d.stanford.edu/lecture_notes/LectureNotes4.pdf

Exercise 11-5: Implement GRU using Numpy





$$z = \sigma(x_t U^z + s_{t-1} W^z)$$
$$r = \sigma(x_t U^r + s_{t-1} W^r)$$
$$h = tanh(x_t U^h + (s_{t-1} \circ r) W^h)$$
$$s_t = (1 - z) \circ h + z \circ s_{t-1}$$

- http://blog.varunajayasiri.com/numpy_lstm.html
- http://www.wildml.com/2015/10/recurrent-neural-network-tutorial-part-4-implementing-a-grulstm-rnn-with-python-and-theano/

Advanced Topics

- Sequence to Sequence
  - Sequence to Sequence models: https://github.com/MaximumEntropy/Seq2Seq-PyTorch
  - Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation: https://arxiv.org/abs/1406.1078
- Attention Models
  - Attention and Augmented Recurrent Neural Networks https://distill.pub/2016/augmented-rnns/
  - Neural Machine Translation by Jointly Learning to Align and Translate: https://arxiv.org/abs/1409.0473
  - Effective Approaches to Attention-based Neural Machine Translation: https://arxiv.org/abs/1508.04025

Exercise 11-6:

Implement Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation: https://arxiv.org/abs/1406.1078

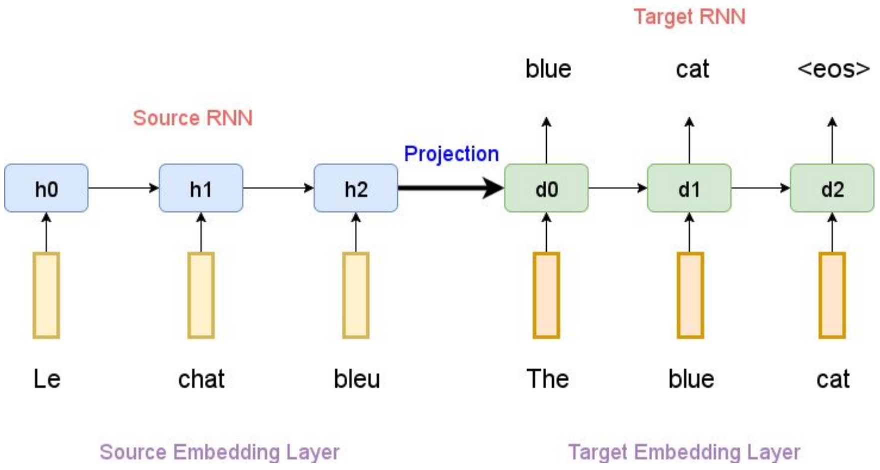


Image source: https://github.com/MaximumEntropy/Seq2Seq-PyTorch

Exercise 11-7:

Implement Neural Machine Translation by Jointly Learning to Align and Translate: https://arxiv.org/abs/1409.0473

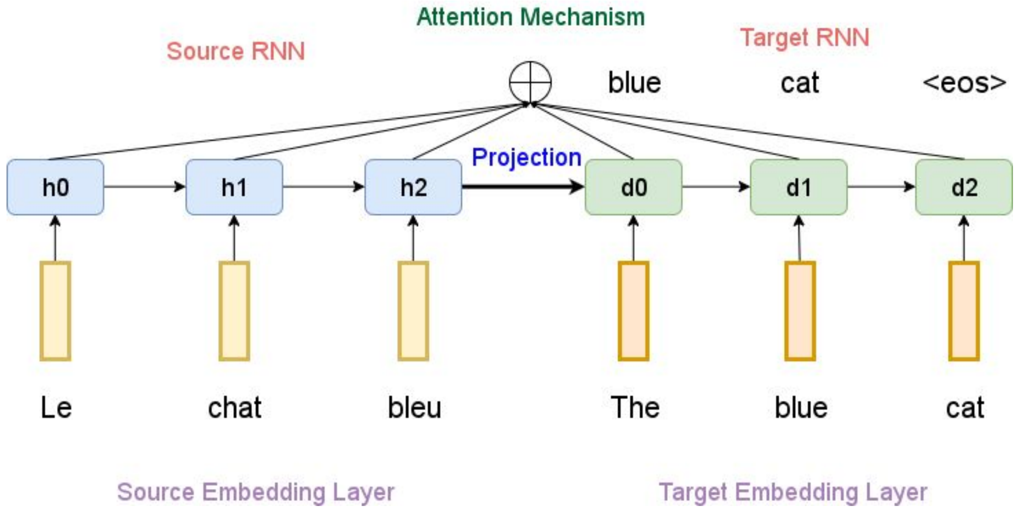


Image source: https://github.com/MaximumEntropy/Seq2Seq-PyTorch

WHAT NEXT

Lecture 12:
NSML,
Smartest ML Platform