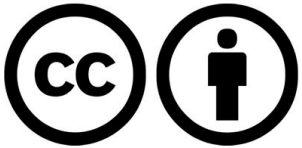
With PyTorch 0.2.0_2

Lab 12
RNN

Sung Kim <hunkim+ml@gmail.com>
Code: https://github.com/hunkim/DeepLearningZeroToAll/



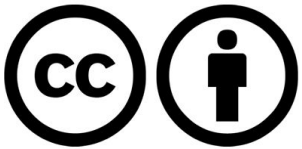
With PyTorch 0.2.0_2

Lab 12-1

Hi Hello RNN

Sung Kim <hunkim+ml@gmail.com>

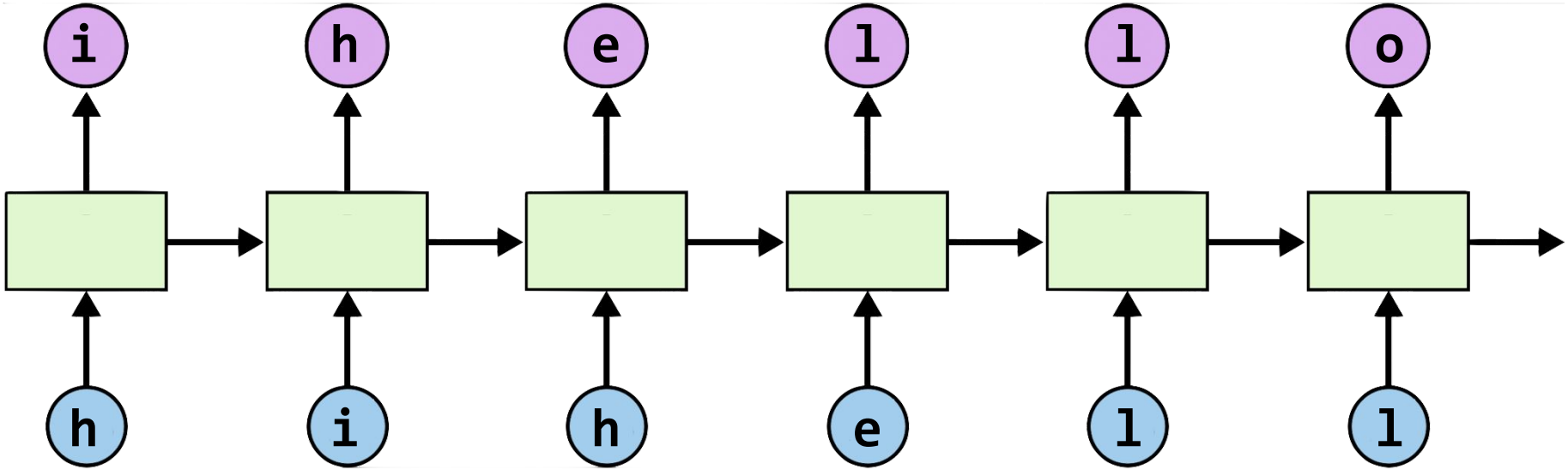
Code: https://github.com/hunkim/DeepLearningZeroToAll/



https://github.com/hunkim/DeepLearningZeroToAll/tree/master/pytorch

https://github.com/hunkim/DeepLearningZeroToAll/blob/master/pytorch/lab-12-1-hello-rnn.py

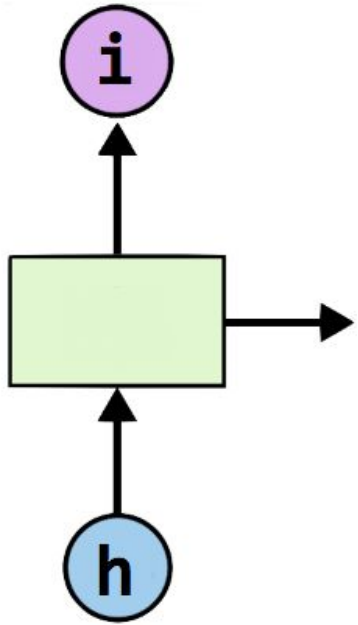
Teach RNN ‘hihello’



- text: ‘hihello’
- unique chars (vocabulary, voc):
  h, i, e, l, o
- voc index:
  h:0, i:1, e:2, l:3, o:4

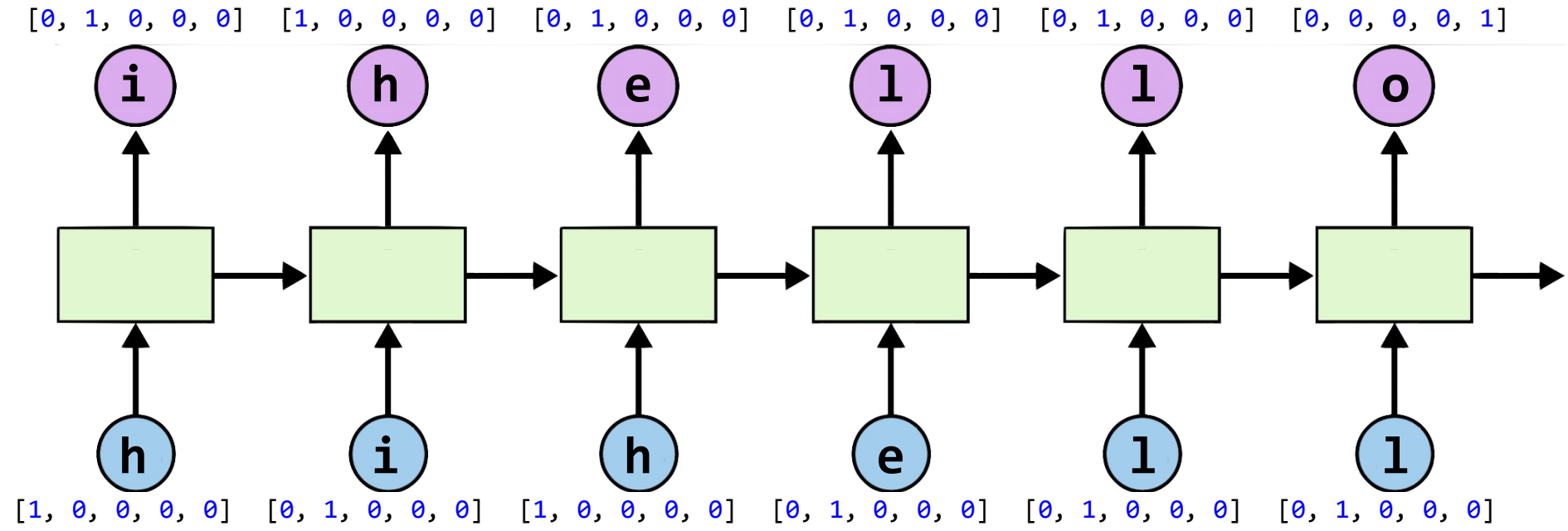
One-hot encoding

```
[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
```



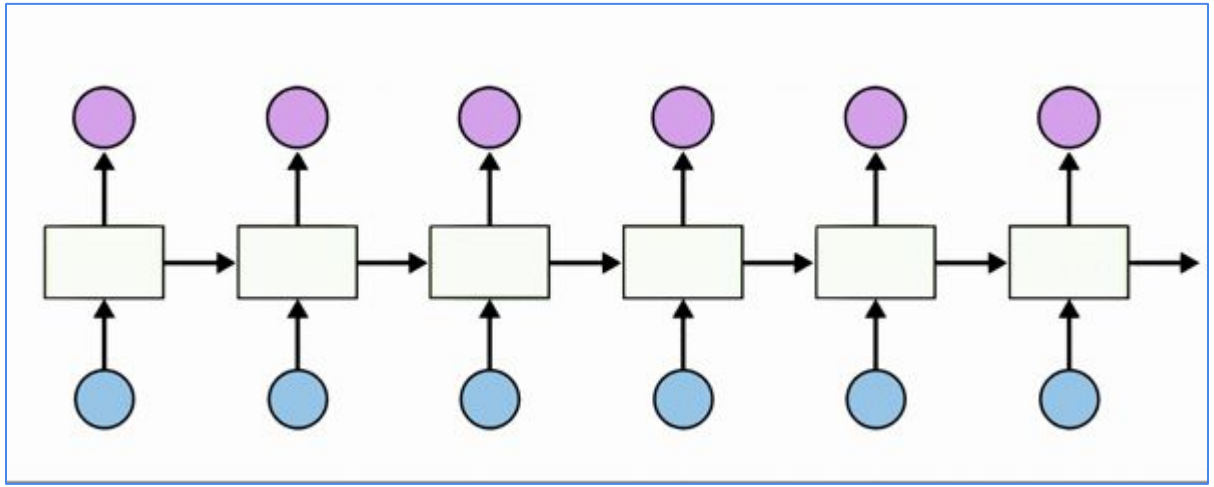
Teach RNN ‘hihello’

```
[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
```



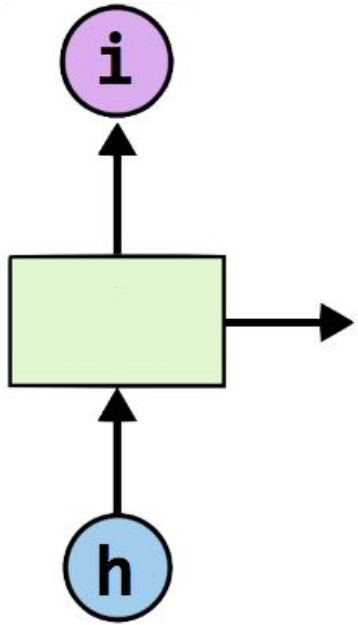
Teach RNN ‘hihello’

```
[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 ‘hihello’

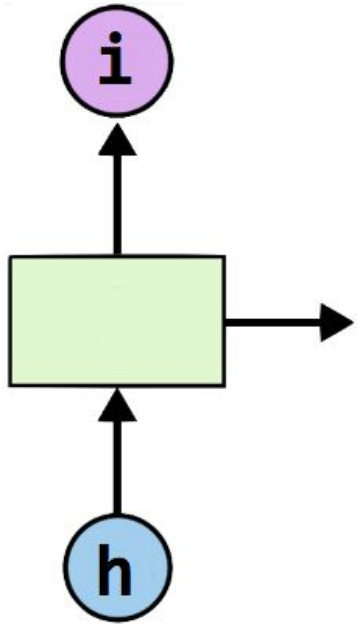
```
idx2char = ['h', 'i', 'e', 'l', 'o']

# Teach hello: 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
```



https://github.com/hunkim/DeepLearningZeroToAll/blob/master/pytorch/lab-12-1-hello-rnn.py

RNN parameters

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

inputs = Variable(inputs)
labels = Variable(labels)

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
```



https://github.com/hunkim/DeepLearningZeroToAll/blob/master/pytorch/lab-12-1-hello-rnn.py

Class RNN

```python
class RNN(nn.Module):

    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
        # Set parameters for RNN block
        # Note: batch_first=False by default.
        # When true, inputs are (batch_size, sequence_length, input_dimension)
        # instead of (sequence_length, batch_size, input_dimension)
        self.rnn = nn.RNN(input_size=input_size, hidden_size=hidden_size,
                          num_layers=num_layers, batch_first=True)
        # Fully connected layer to obtain outputs corresponding to the number
        # of classes
        self.fc = nn.Linear(hidden_size, num_classes)
```

```python
    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)

        # Reshape output from (batch, seq_len, hidden_size) to (batch *
        # seq_len, hidden_size)
        out = out.view(-1, self.hidden_size)
        # Return outputs applied to fully connected layer
        out = self.fc(out)
        return out
```

https://github.com/hunkim/DeepLearningZeroToAll/blob/master/pytorch/lab-12-1-hello-rnn.py

Loss, Optimizer

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

# Set loss and optimizer function
criterion = torch.nn.CrossEntropyLoss()    # Softmax is internally computed.
optimizer = torch.optim.Adam(rnn.parameters(), lr=learning_rate)
```

https://github.com/hunkim/DeepLearningZeroToAll/blob/master/pytorch/lab-12-1-hello-rnn.py

Training

```
# Train the model
for epoch in range(num_epochs):
    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))
```

https://github.com/hunkim/DeepLearningZeroToAll/blob/master/pytorch/lab-12-1-hello-rnn.py

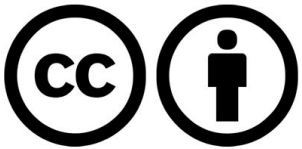
With PyTorch 0.2.0_2

Lab 12-2

RNN with long sequences

Sung Kim <hunkim+ml@gmail.com>

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



https://github.com/hunkim/DeepLearningZeroToAll/tree/master/pytorch

https://github.com/hunkim/DeepLearningZeroToAll/blob/master/pytorch/lab-12-2-char-seq-rnn.py

Manual data creation

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

https://github.com/hunkim/DeepLearningZeroToAll/blob/master/pytorch/lab-12-1-hello-rnn.py

Better data creation

```
sample = " if you want you"
idx2char = list(set(sample))  # index -> char
char2idx = {c: i for i, c in enumerate(idx2char)}  # char -> idx
```

https://github.com/hunkim/DeepLearningZeroToAll/blob/master/pytorch/lab-12-2-char-seq-rnn.py

Hyper parameters

```
sample = " if you want you"
idx2char = list(set(sample))  # index -> char
char2idx = {c: i for i, c in enumerate(idx2char)}  # char -> idx

# hyper parameters
dic_size = len(char2idx)  # RNN input size (one hot size)
rnn_hidden_size = len(char2idx)  # RNN output size
num_classes = len(char2idx)  # final output size (RNN or softmax, etc.)
batch_size = 1  # one sample data, one batch
sequence_length = len(sample) - 1  # number of lstm unfolding (unit #)
```

https://github.com/hunkim/DeepLearningZeroToAll/blob/master/pytorch/lab-12-2-char-seq-rnn.py

One hot encoding

```
sample_idx = [char2idx[c] for c in sample]  # char to index
x_data = [sample_idx[:-1]]  # X data sample (0 ~ n-1) hello: hell
y_data = [sample_idx[1:]]   # Y label sample (1 ~ n)  hello: ello
```

```
x_data = torch.Tensor(x_data)
y_data = torch.LongTensor(y_data)

# one hot encoding


def one_hot(x, num_classes):
    idx = x.long()
    idx = idx.view(-1, 1)
    x_one_hot = torch.zeros(x.size()[0] * x.size()[1], num_classes)
    x_one_hot.scatter_(1, idx, 1)
    x_one_hot = x_one_hot.view(x.size()[0], x.size()[1], num_classes)
    return x_one_hot


x_one_hot = one_hot(x_data, num_classes)

inputs = Variable(x_one_hot)
labels = Variable(y_data)
```

https://github.com/hunkim/DeepLearningZeroToAll/blob/master/pytorch/lab-12-2-char-seq-rnn.py

LSTM

```
class LSTM(nn.Module):

    def __init__(self, num_classes, input_size, hidden_size, num_layers):
        super(LSTM, 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
        # Set parameters for RNN block
        # Note: batch_first=False by default.
        # When true, inputs are (batch_size, sequence_length, input_dimensio
        # instead of (sequence_length, batch_size, input_dimension)
        self.lstm = nn.LSTM(input_size=input_size, hidden_size=hidden_size,
                            num_layers=num_layers, batch_first=True)
        # Fully connected layer to obtain outputs corresponding to the numbe
        # of classes
        self.fc = nn.Linear(hidden_size, num_classes)
```

```
def forward(self, x):
    # Initialize hidden and cell states
    h_0 = Variable(torch.zeros(
        self.num_layers, x.size(0), self.hidden_size))
    c_0 = Variable(torch.zeros(
        self.num_layers, x.size(0), 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: (num_layers * num_directions, batch, hidden_size)
    out, _ = self.lstm(x, (h_0, c_0))

    # Reshape output from (batch, seq_len, hidden_size) to (batch *
    # seq_len, hidden_size)
    out = out.view(-1, self.hidden_size)
    # Return outputs applied to fully connected layer
    out = self.fc(out)
    return out
```

https://github.com/hunkim/DeepLearningZeroToAll/blob/master/pytorch/lab-12-2-char-seq-rnn.py

Loss, Optimizer

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

# Set loss and optimizer function
criterion = torch.nn.CrossEntropyLoss()    # Softmax is internally computed.
optimizer = torch.optim.Adam(rnn.parameters(), lr=learning_rate)
```

https://github.com/hunkim/DeepLearningZeroToAll/blob/master/pytorch/lab-12-2-char-seq-rnn.py

Training

```
# Train the model
for epoch in range(num_epochs):
    outputs = lstm(inputs)
    optimizer.zero_grad()
    loss = criterion(outputs, labels.view(-1))
    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!")
```

https://github.com/hunkim/DeepLearningZeroToAll/blob/master/pytorch/lab-12-2-char-seq-rnn.py

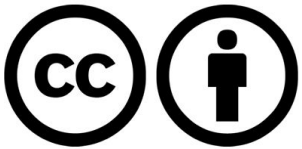
With PyTorch 0.2.0_2

Lab 12-4

RNN with long sentences

Sung Kim <hunkim+ml@gmail.com>

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



https://github.com/hunkim/DeepLearningZeroToAll/tree/master/pytorch

https://github.com/hunkim/DeepLearningZeroToAll/blob/master/pytorch/lab-12-4-rnn-long_char.py

Really long sentence?

```
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.")
```

https://github.com/hunkim/DeepLearningZeroToAll/blob/master/pytorch/lab-12-4-rnn-long_char.py

Really long sentence?

```
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.")
```

```
# training dataset
0 if you wan -> f you want
1 f you want ->  you want
2  you want  -> you want t
3 you want t -> ou want to
…
168  of the se -> of the sea
169 of the sea -> f the sea.
```

https://github.com/hunkim/DeepLearningZeroToAll/blob/master/pytorch/lab-12-4-rnn-long_char.py

RNN parameters

```
char_set = list(set(sentence))
char_dic = {w: i for i, w in enumerate(char_set)}

# hyperparameters
learning_rate = 0.1
num_epochs = 500
input_size = len(char_set) # RNN input size (one hot
size)
hidden_size = len(char_set) # RNN output size
num_classes = len(char_set) # final output size (RNN
or softmax, etc.)
sequence_length = 10 # any arbitrary number
num_layers = 2 # number of layers in RNN
```

 # training dataset
0 if you wan -> f you want
1 f you want -> you want
2 you want -> you want t
3 you want t -> ou want to
…
168 of the se -> of the sea
169 of the sea -> f the sea.

https://github.com/hunkim/DeepLearningZeroToAll/blob/master/pytorch/lab-12-4-rnn-long_char.py

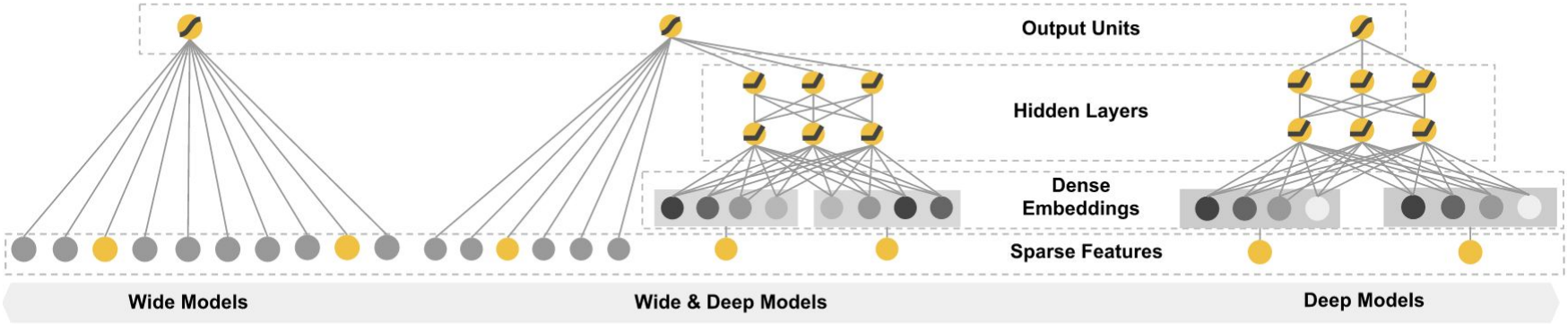
Making dataset

```
char_set = list(set(sentence))
char_dic = {w: i for i, w in enumerate(char_set)}

dataX = []
dataY = []
for i in range(0, len(sentence) - seq_length):
    x_str = sentence[i:i + seq_length]
    y_str = sentence[i + 1: i + seq_length + 1]
    print(i, x_str, '->', y_str)

    x = [char_dic[c] for c in x_str]  # x str to index
    y = [char_dic[c] for c in y_str]  # y str to index

    dataX.append(x)
    dataY.append(y)
```

 # training dataset
0 if you wan -> f you want
1 f you want -> you want
2 you want -> you want t
3 you want t -> ou want to
…
168 of the se -> of the sea
169 of the sea -> f the sea.

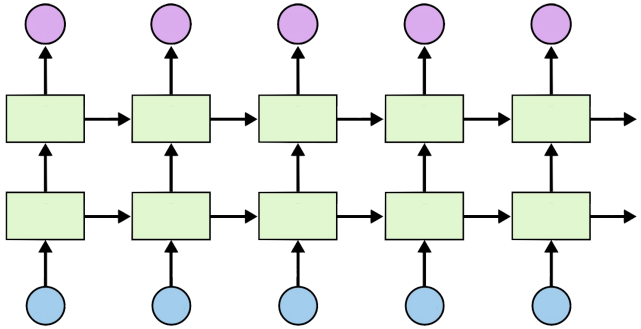
https://github.com/hunkim/DeepLearningZeroToAll/blob/master/pytorch/lab-12-4-rnn-long_char.py

Wide & Deep



https://www.tensorflow.org/versions/r0.11/tutorials/wide_and_deep/index.html

Stacked RNN



```
# one hot encoding


def one_hot(x, num_classes):
    idx = x.long()
    idx = idx.view(-1, 1)
    x_one_hot = torch.zeros(x.size()[0] * x.size()[1], num_classes)
    x_one_hot.scatter_(1, idx, 1)
    x_one_hot = x_one_hot.view(x.size()[0], x.size()[1], num_classes)
    return x_one_hot


x_one_hot = one_hot(x_data, num_classes)

inputs = Variable(x_one_hot)
labels = Variable(y_data)
```

https://github.com/hunkim/DeepLearningZeroToAll/blob/master/pytorch/lab-12-4-rnn-long_char.py

LSTM

```python
class LSTM(nn.Module):

    def __init__(self, num_classes, input_size, hidden_size, num_layers):
        super(LSTM, 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
        # Set parameters for RNN block
        # Note: batch_first=False by default.
        # When true, inputs are (batch_size, sequence_length, input_dimen
        # instead of (sequence_length, batch_size, input_dimension)
        self.lstm = nn.LSTM(input_size=input_size, hidden_size=hidden_siz
                            num_layers=num_layers, batch_first=True)
        # Fully connected layer
        self.fc = nn.Linear(hidden_size, num_classes)
```

```python
def forward(self, x):
    # Initialize hidden and cell states
    h_0 = Variable(torch.zeros(
        self.num_layers, x.size(0), self.hidden_size))
    c_0 = Variable(torch.zeros(
        self.num_layers, x.size(0), self.hidden_size))
    # h_0 = Variable(torch.zeros(
    # self.num_layers, x.size(0), self.hidden_size))
    # c_0 = Variable(torch.zeros(
    # self.num_layers, x.size(0), self.hidden_size))

    # Propagate input through LSTM
    # Input: (batch, seq_len, input_size)
    out, _ = self.lstm(x, (h_0, c_0))
    # Note: the output tensor of LSTM in this case is a block with holes
    # > add .contiguous() to apply view()
    out = out.contiguous().view(-1, self.hidden_size)
    # Return outputs applied to fully connected layer
    out = self.fc(out)
    return out
```

https://github.com/hunkim/DeepLearningZeroToAll/blob/master/pytorch/lab-12-4-rnn-long_char.py

Softmax (FC) in Deep CNN

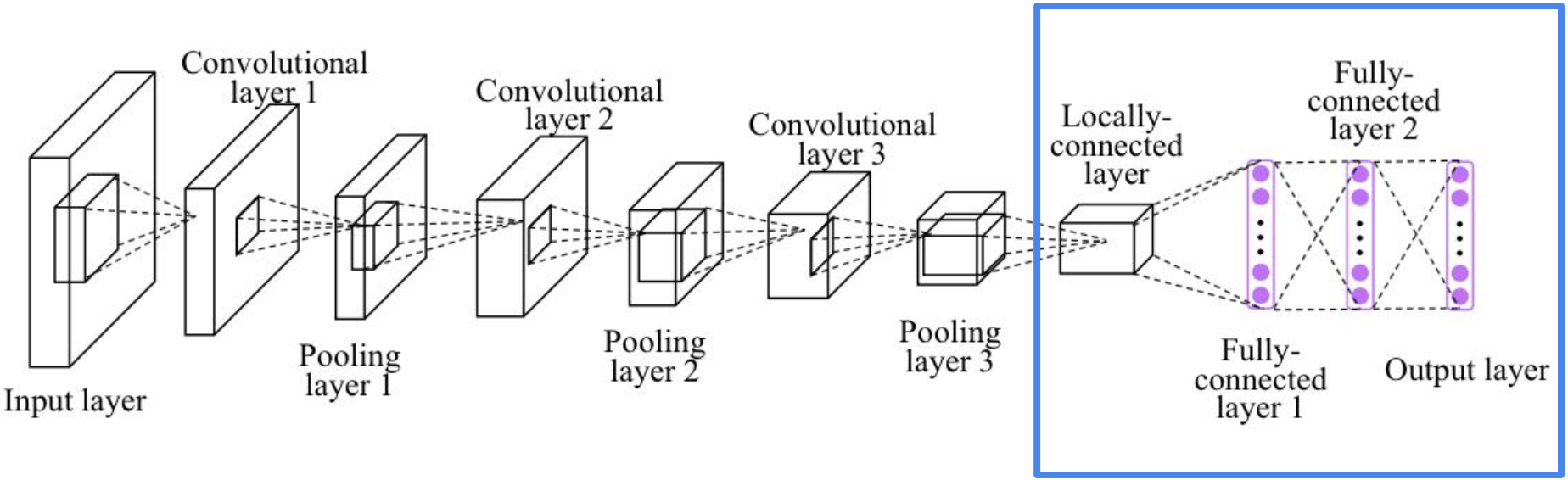
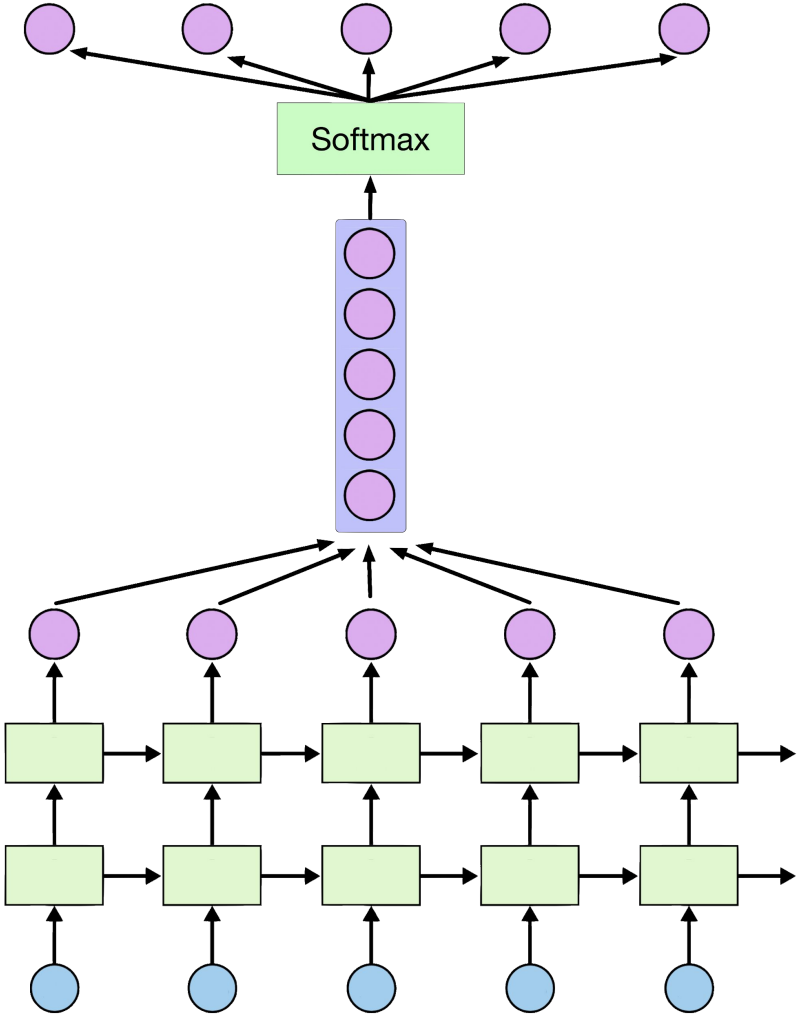


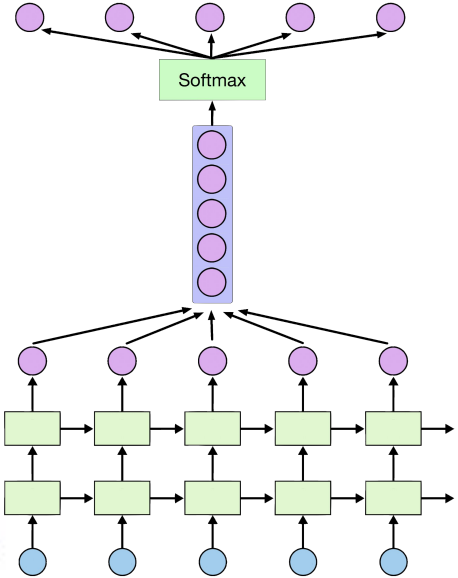
Image credit: http://personal.ie.cuhk.edu.hk/~ccloy/project_target_code/index.html

Softmax



Loss, Optimizer

```
# Instantiate RNN model
lstm = LSTM(num_classes, input_size, hidden_size, num_layers)

# Set loss and optimizer function
criterion = torch.nn.CrossEntropyLoss()    # Softmax is internally computed.
optimizer = torch.optim.Adam(lstm.parameters(), lr=learning_rate)
```



https://github.com/hunkim/DeepLearningZeroToAll/blob/master/pytorch/lab-12-4-rnn-long_char.py

Training

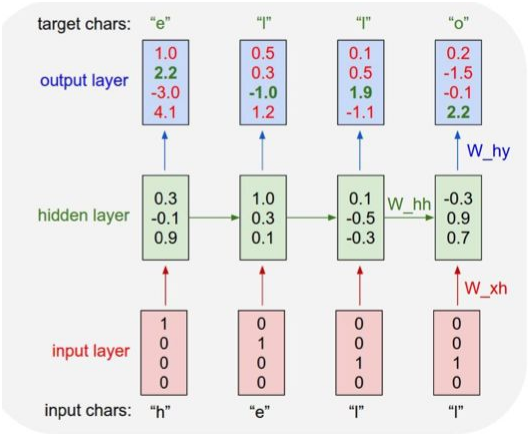
```
# Train the model
for epoch in range(num_epochs):
    outputs = lstm(inputs)
    optimizer.zero_grad()
    # obtain the loss function
    # flatten target labels to match output
    loss = criterion(outputs, labels.view(-1))
    loss.backward()
    optimizer.step()
    # obtain the predicted indices of the next character
    _, idx = outputs.max(1)
    idx = idx.data.numpy()
    idx = idx.reshape(-1, sequence_length)  # (170,10)
    # display the prediction of the last sequence
    result_str = [char_set[c] for c in idx[-1]]
    print("epoch: %d, loss: %1.3f" % (epoch + 1, loss.data[0]))
    print("Predicted string: ", ''.join(result_str))

print("Learning finished!")
```

https://github.com/hunkim/DeepLearningZeroToAll/blob/master/pytorch/lab-12-4-rnn-long_char.py

Print results

g 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.

char-rnn

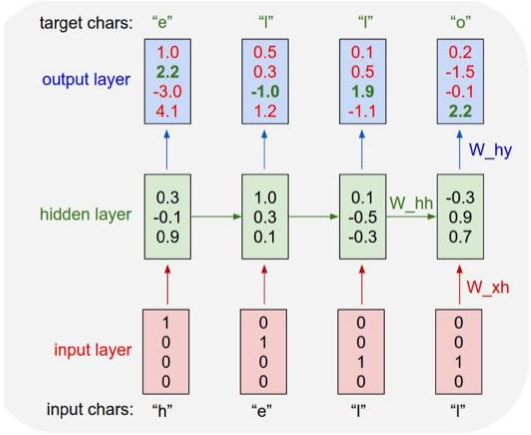


Shakespeare

It looks like we can learn to spell English words. But how about if there is more structure and style in the data? To examine this I downloaded all the works of Shakespeare and concatenated them into a single (4.4MB) file. We can now afford to train a larger network, in this case lets try a 3-layer RNN with 512 hidden nodes on each layer. After we train the network for a few hours we obtain samples such as:

```
PANDARUS:
Alas, I think he shall be come approached and the day
When little srain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

Second Senator:
They are away this miseries, produced upon my soul,
Breaking and strongly should be buried, when I perish
The earth and thoughts of many states.

DUKE VINCENTIO:
Well, your wit is in the care of side and that.

Second Lord:
They would be ruled after this chamber, and
my fair nues begun out of the fact, to be conveyed,
Whose noble souls I'll have the heart of the wars.

Clown:
Come, sir, I will make did behold your worship.

VIOLA:
I'll drink it.
```

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



Linux Source Code

I wanted to push structured data to its limit, so for the final challenge I decided to use code. In particular, I took all the source and header files found in the Linux repo on Github, concatenated all of them in a single giant file (474MB of C code) (I was originally going to train only on the kernel but that by itself is only ~16MB). Then I trained several as-large-as-fits-on-my-GPU 3-layer LSTMs over a period of a few days. These models have about 10 million parameters, which is still on the lower end for RNN models. The results are superfun:

```
/*
 * Increment the size file of the new incorrect UI_FILTER group information
 * of the size generatively.
 */
static int indicate_policy(void)
{
  int error;
  if (fd == MARN_EPT) {
    /*
     * The kernel blank will coeld it to userspace.
     */
    if (ss->segment < mem_total)
      unblock_graph_and_set_blocked();
    else
      ret = 1;
    goto bail;
  }
  segaddr = in_SB(in.addr);
  selector = seg / 16;
  setup_works = true;
  for (i = 0; i < blocks; i++) {
    seq = buf[i++];
    bpf = bd->bd.next + i * search;
    if (fd) {
      current = blocked;
    }
  }
  rw->name = "Getjbbregs";
  bprm_self_clearl(&iv->version);
  regs->new = blocks[(BPF_STATS << info->historidac)] | PFMR_CLOBATHINC_SECONDS << 12;
  return segtable;
}
```

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

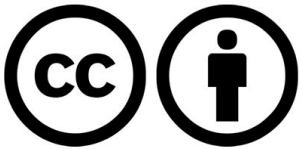
With TF 1.0!

Lab 12-5

RNN with time series data (stock)

Sung Kim <hunkim+ml@gmail.com>

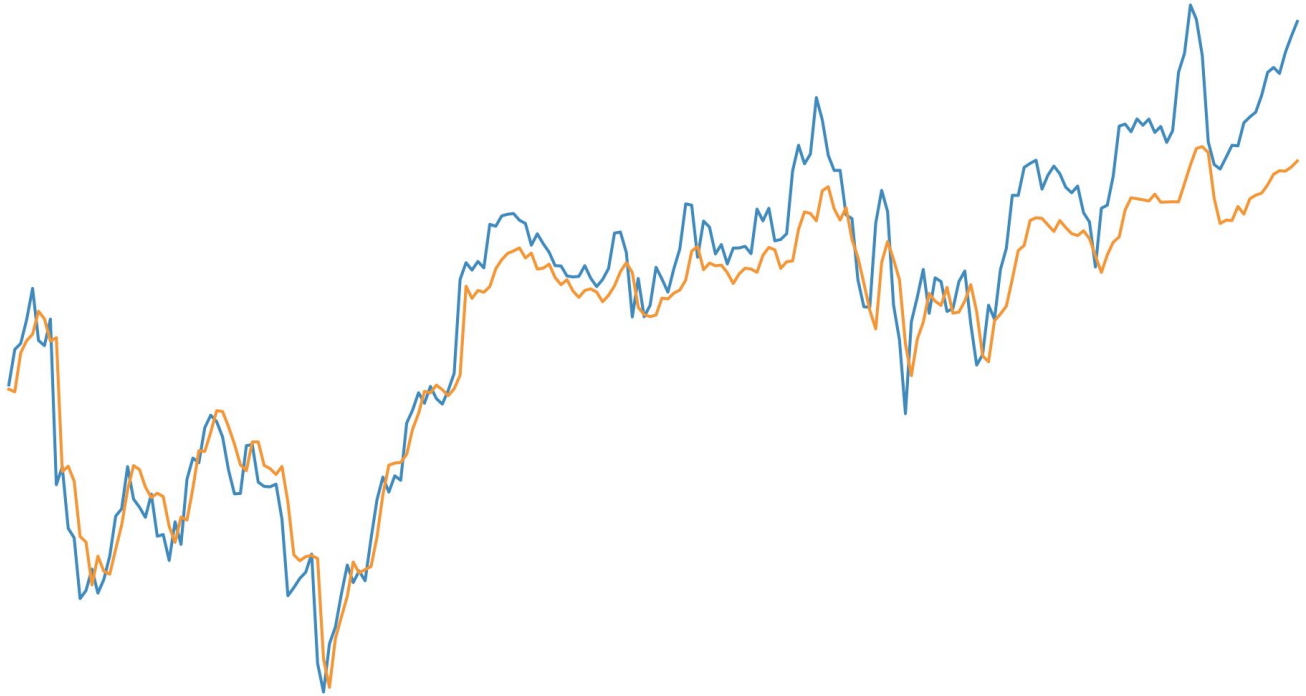
Code: https://github.com/hunkim/DeepLearningZeroToAll/



https://github.com/hunkim/DeepLearningZeroToAll/tree/master/pytorch

https://github.com/hunkim/DeepLearningZeroToAll/blob/master/pytorch/lab-12-5-stock_prediction.py

Time series data

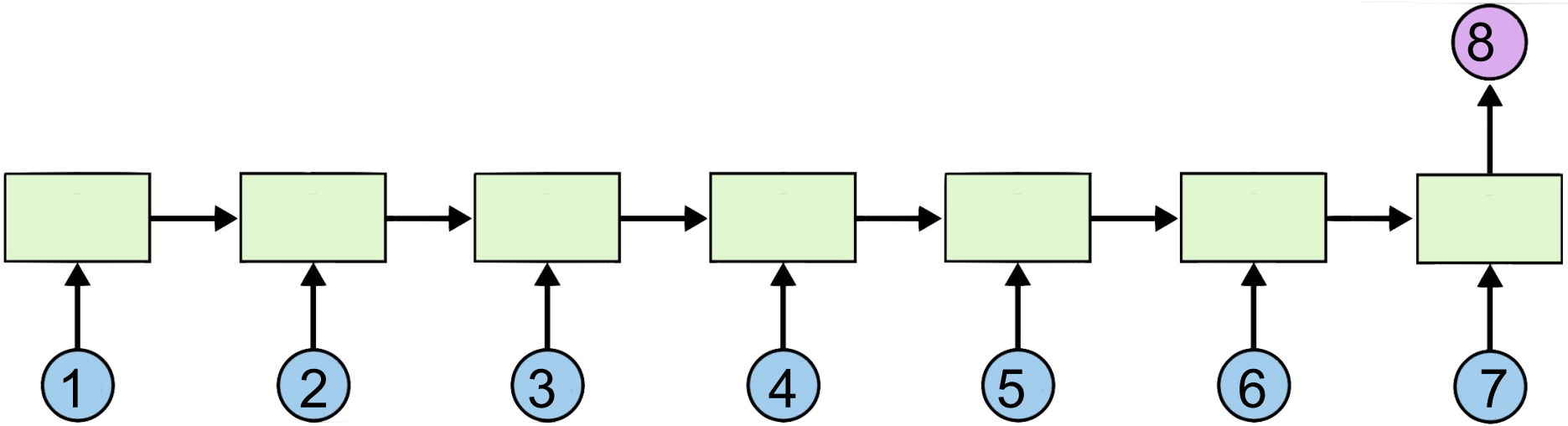


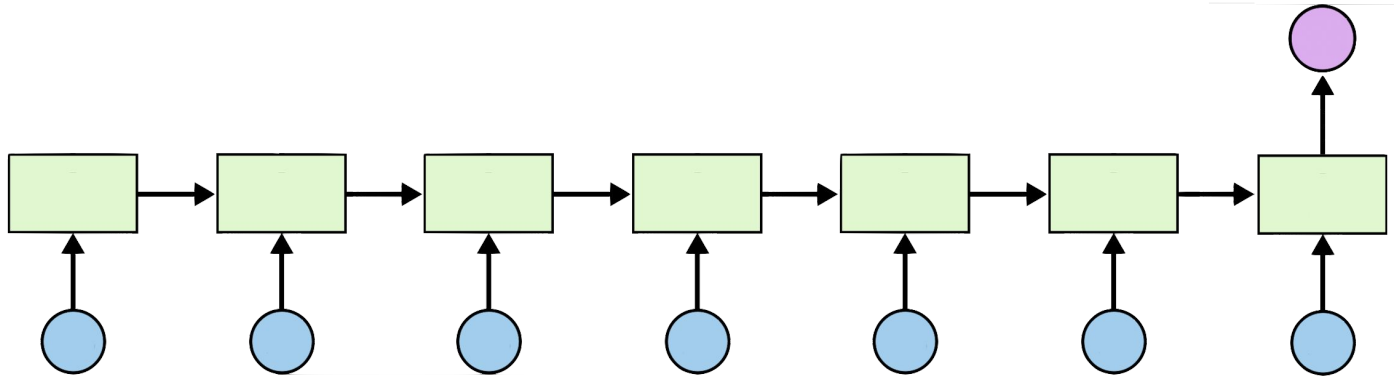
Time series data

Open	High	Low	Volume	Close
828.659973	833.450012	828.349976	1247700	**831.659973**
823.02002	828.070007	821.655029	1597800	**828.070007**
819.929993	824.400024	818.97998	1281700	**824.159973**
819.359985	823	818.469971	1304000	**818.97998**
819	823	816	1053600	**820.450012**
816	820.958984	815.48999	1198100	**819.23999**
811.700012	815.25	809.780029	1129100	**813.669983**
809.51001	810.659973	804.539978	989700	**809.559998**
807	811.840027	803.190002	1155300	**808.380005**

`'data-02-stock_daily.csv'`

Many to one





Open	High	Low	Volume	Close
828.659973	833.450012	828.349976	1247700	**831.659973**
823.02002	828.070007	821.655029	1597800	**828.070007**
819.929993	824.400024	818.97998	1281700	**824.159973**
819.359985	823	818.469971	1304000	**818.97998**
819	823	816	1053600	**820.450012**
816	820.958984	815.48999	1198100	**819.23999**
811.700012	815.25	809.780029	1129100	**813.669983**
809.51001	810.659973	804.539978	989700	**?**
807	811.840027	803.190002	1155300	**?**

Reading data

```
timesteps = seq_length = 7
input_size = 5
num_layers = 1 # number of layers in RNN
# Open,High,Low,Close,Volume
xy = np.loadtxt('data-02-stock_daily.csv', delimiter=',')
xy = xy[::-1]  # reverse order (chronically ordered)
xy = MinMaxScaler(xy)
x = xy
y = xy[:, [-1]]  # Close as label

dataX = []
dataY = []
for i in range(0, len(y) - seq_length):
    _x = x[i:i + seq_length]
    _y = y[i + seq_length]  # Next close price
    print(_x, "->", _y)
    dataX.append(_x)
    dataY.append(_y)
```

```
[ 0.18667876  0.20948057  0.20878184  0.
0.21744815]

[ 0.30697388  0.31463414  0.21899367
0.01247647  0.21698189]

[ 0.21914211  0.26390721  0.2246864
0.45632338  0.22496747]

[ 0.23312993  0.23641916  0.16268272
0.57017119  0.14744274]

[ 0.13431201  0.15175877  0.11617252
0.39380658  0.13289962]

[ 0.13973232  0.17060429  0.15860382
0.28173344  0.18171679]

[ 0.18933069  0.20057799  0.19187983
0.29783096  0.2086465 ]]

-> [ 0.14106001]
```

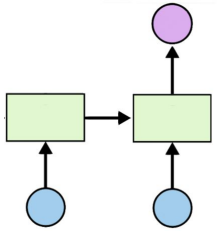
https://github.com/hunkim/DeepLearningZeroToAll/blob/master/pytorch/lab-12-5-stock_prediction.py

Train/Test split

```
# split to train and testing
train_size = int(len(dataY) * 0.7)
test_size = len(dataY) - train_size
trainX = torch.Tensor(np.array(dataX[0:train_size]))
trainX = Variable(trainX)
testX = torch.Tensor(np.array(dataX[train_size:len(dataX)]))
testX = Variable(testX)
trainY = torch.Tensor(np.array(dataY[0:train_size]))
trainY = Variable(trainY)
testY = torch.Tensor(np.array(dataY[train_size:len(dataY)]))
testY = Variable(testY)
```

https://github.com/hunkim/DeepLearningZeroToAll/blob/master/pytorch/lab-12-5-stock_prediction.py

LSTM

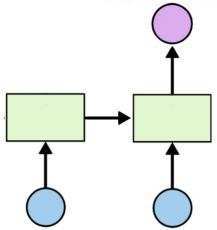


```python
class LSTM(nn.Module):

    def __init__(self, num_classes, input_size, hidden_size, num_layers):
        super(LSTM, self).__init__()
        self.num_classes = num_classes
        self.num_layers = num_layers
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.seq_length = seq_length
        # Set parameters for RNN block
        # Note: batch_first=False by default.
        # When true, inputs are (batch_size, sequence_length, input_dimension)
        # instead of (sequence_length, batch_size, input_dimension)
        self.lstm = nn.LSTM(input_size=input_size, hidden_size=hidden_size,
                            num_layers=num_layers, batch_first=True)
        # Fully connected layer
        self.fc = nn.Linear(hidden_size, num_classes)
```

```python
def forward(self, x):
    # Initialize hidden and cell states
    h_0 = Variable(torch.zeros(
        self.num_layers, x.size(0), self.hidden_size))
    c_0 = Variable(torch.zeros(
        self.num_layers, x.size(0), self.hidden_size))

    # Propagate input through LSTM
    _, (h_out, _) = self.lstm(x, (h_0, c_0))
    h_out = h_out.view(-1, self.hidden_size)
    out = self.fc(h_out)
    return out
```

https://github.com/hunkim/DeepLearningZeroToAll/blob/master/pytorch/lab-12-5-stock_prediction.py

Loss, Optimizer

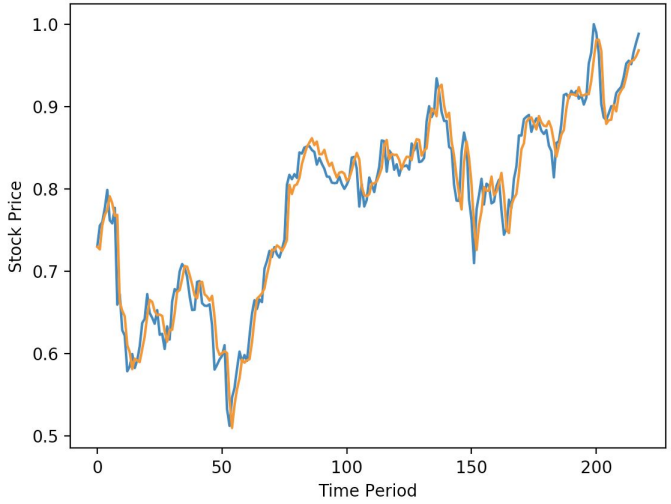


```
# Instantiate RNN model
lstm = LSTM(num_classes, input_size, hidden_size, num_layers)

# Set loss and optimizer function
criterion = torch.nn.MSELoss()    # mean-squared error for regression
optimizer = torch.optim.Adam(lstm.parameters(), lr=learning_rate)
```

https://github.com/hunkim/DeepLearningZeroToAll/blob/master/pytorch/lab-12-5-stock_prediction.py

Training and Results

```
# Train the model
for epoch in range(num_epochs):
    outputs = lstm(trainX)
    optimizer.zero_grad()
    # obtain the loss function
    loss = criterion(outputs, trainY)
    loss.backward()
    optimizer.step()
    print("Epoch: %d, loss: %1.5f" % (epoch, loss.data[0]))

print("Learning finished!")

# Test the model
lstm.eval()
test_predict = lstm(testX)

# Plot predictions
test_predict = test_predict.data.numpy()
testY = testY.data.numpy()
plt.plot(testY)
plt.plot(test_predict)
plt.xlabel("Time Period")
plt.ylabel("Stock Price")
plt.show()
```



https://github.com/hunkim/DeepLearningZeroToAll/blob/master/pytorch/lab-12-5-stock_prediction.py

Exercise

- Implement stock prediction using linear regression only
- Improve results using more features such as keywords and/or sentiments in top news

Other RNN applications

- Language Modeling
- Speech Recognition
- Machine Translation
- Conversation Modeling/Question Answering
- Image/Video Captioning
- Image/Music/Dance Generation

http://jiwonkim.org/awesome-rnn/