ML/DL for Everyone with PYTORCH

Lecture 13: RNN II (classification)

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

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

Slides: http://bit.ly/PyTorchZeroAll

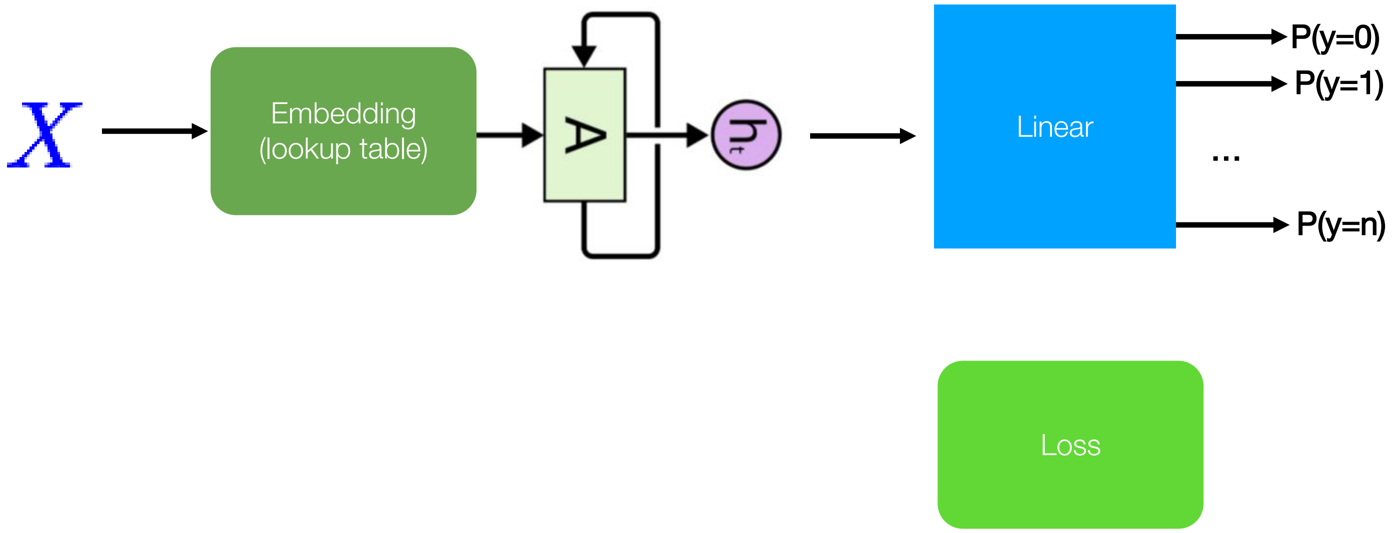
Call for Comments

Please feel free to add comments directly on these slides.

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

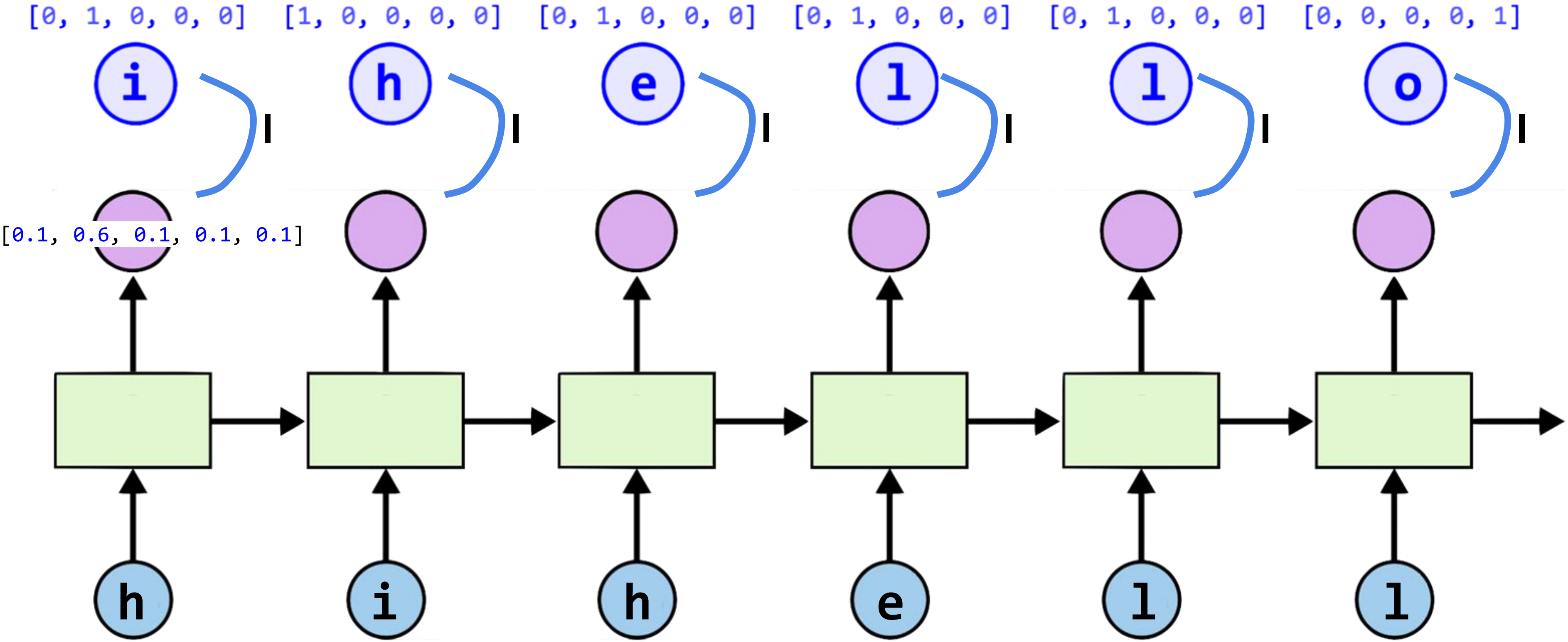
Picture from http://www.tssablog.org/archives/3280

Typical RNN Models

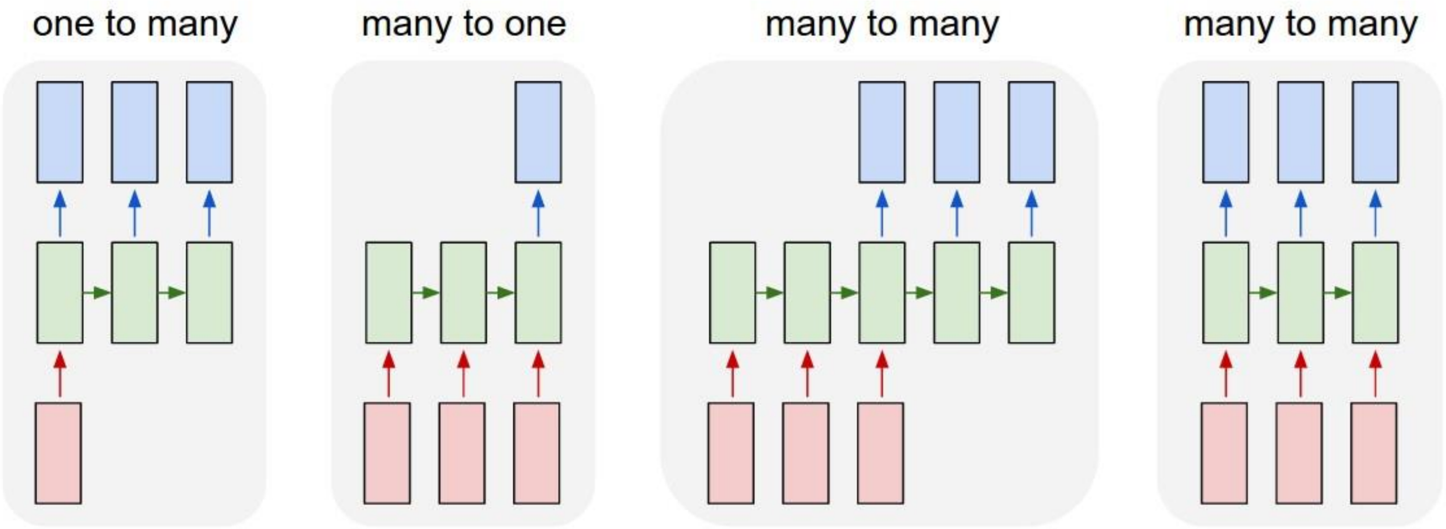


With CrossEntropy

RNN Loss and training

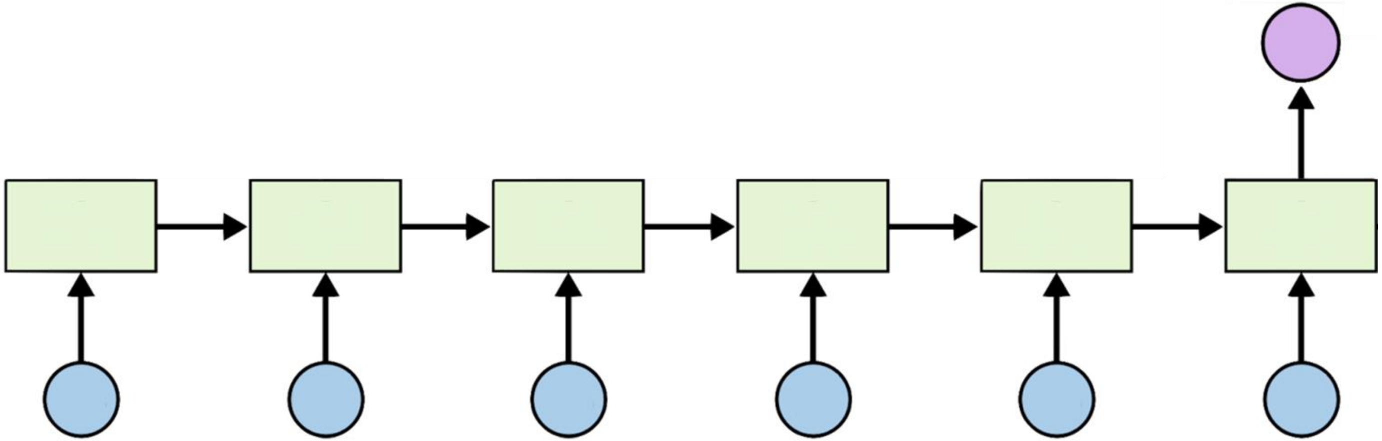


RNN Applications



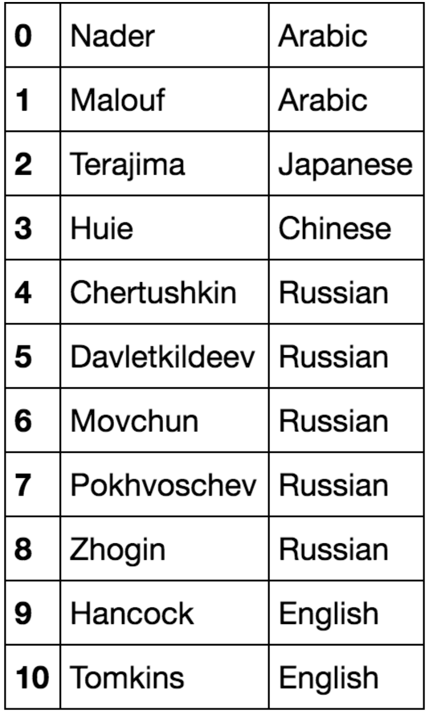
http://karpathy.github.io/2015/05/21/rnn-effectiveness/

RNN Classification

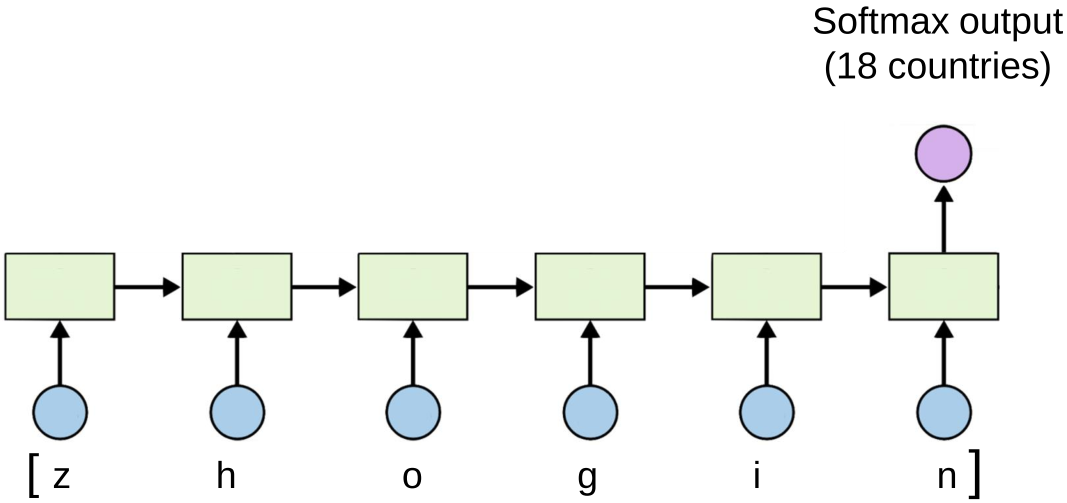


http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Name Classification: Dataset

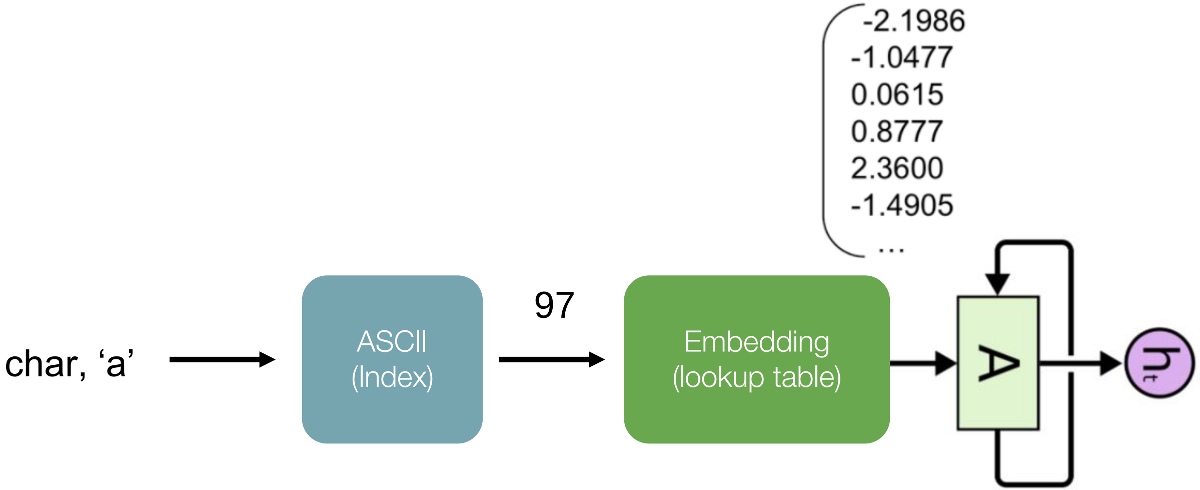


0	Nader	Arabic
1	Malouf	Arabic
2	Terajima	Japanese
3	Huie	Chinese
4	Chertushkin	Russian
5	Davletkildeev	Russian
6	Movchun	Russian
7	Pokhvoschev	Russian
8	Zhogin	Russian
9	Hancock	English
10	Tomkins	English



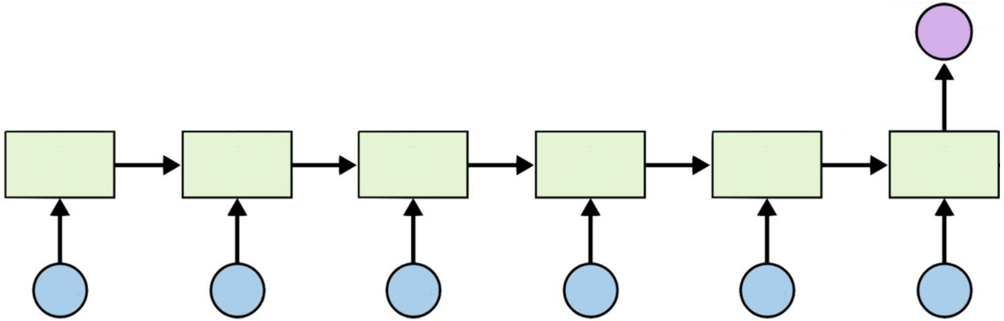
https://github.com/hunkim/PyTorchZeroToAll

Char embedding



Input representation

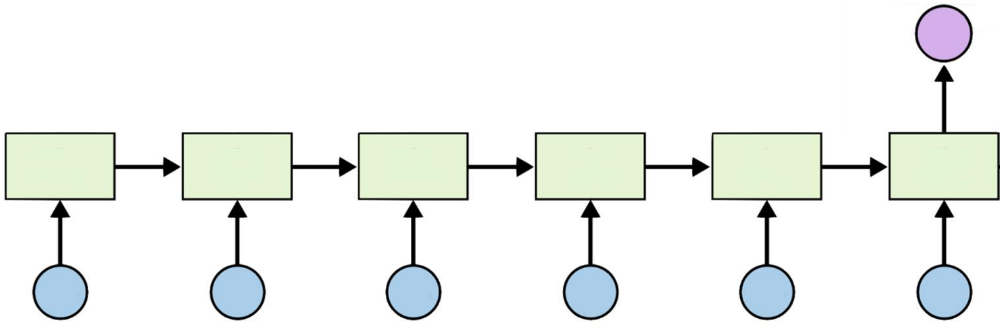
Softmax output
(18 countries)



$$\begin{pmatrix} a & d & y & l & o & v \end{pmatrix} \quad \text{char}$$

Matrix visualization from Nicolas, https://github.com/ngarneau

Input representation



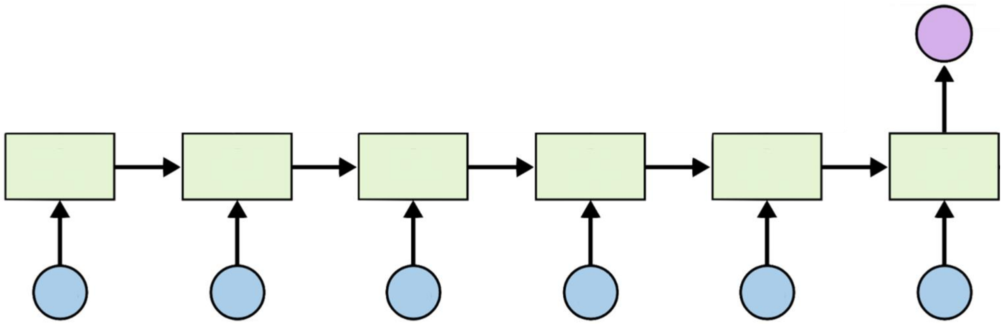
Softmax output
(18 countries)

97	100	121	108	111	118	ascii/index
a	d	y	l	o	v	char

Matrix visualization from Nicolas, https://github.com/ngarneau

Input representation

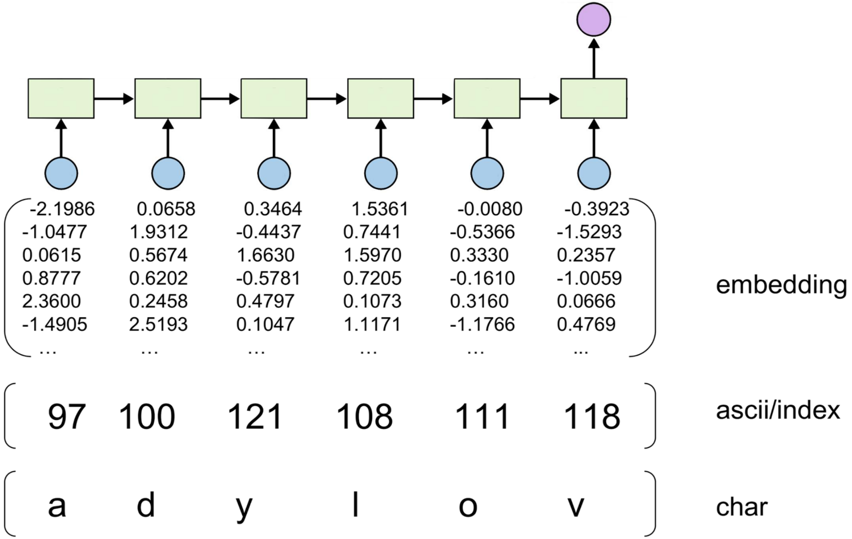
Softmax output
(18 countries)



-2.1986	0.0658	0.3464	1.5361	-0.0080	-0.3923	
-1.0477	1.9312	-0.4437	0.7441	-0.5366	-1.5293	
0.0615	0.5674	1.6630	1.5970	0.3330	0.2357	
0.8777	0.6202	-0.5781	0.7205	-0.1610	-1.0059	embedding
2.3600	0.2458	0.4797	0.1073	0.3160	0.0666	
-1.4905	2.5193	0.1047	1.1171	-1.1766	0.4769	
...	...	...	...	...	...	
97	100	121	108	111	118	ascii/index
a	d	y	l	o	v	char

Matrix visualization from Nicolas, https://github.com/ngarneau

Data preparation



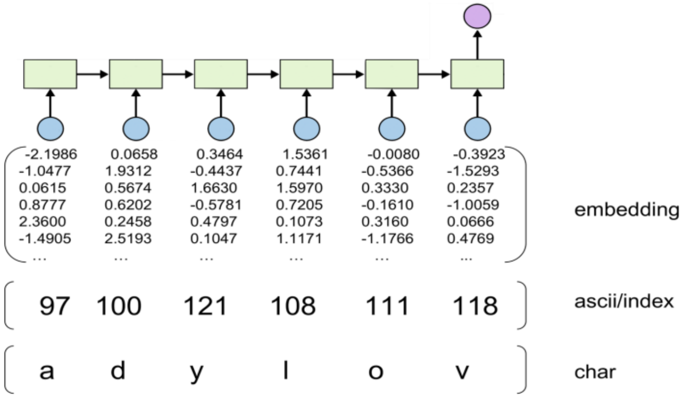
-2.1986	0.0658	0.3464	1.5361	-0.0080	-0.3923	
-1.0477	1.9312	-0.4437	0.7441	-0.5366	-1.5293	
0.0615	0.5674	1.6630	1.5970	0.3330	0.2357	
0.8777	0.6202	-0.5781	0.7205	-0.1610	-1.0059	embedding
2.3600	0.2458	0.4797	0.1073	0.3160	0.0666	
-1.4905	2.5193	0.1047	1.1171	-1.1766	0.4769	
…	…	…	…	…	…	
97	100	121	108	111	118	ascii/index
a	d	y	l	o	v	char

```
self.embedding =
 nn.Embedding(input_voc_size,
rnn_input_size)

...
embedded = self.embedding(input)
```

```
def str2ascii_arr(name):
    arr = [ord(c) for c in name]
    return arr, len(arr)
```

Matrix visualization from Nicolas, https://github.com/ngarneau

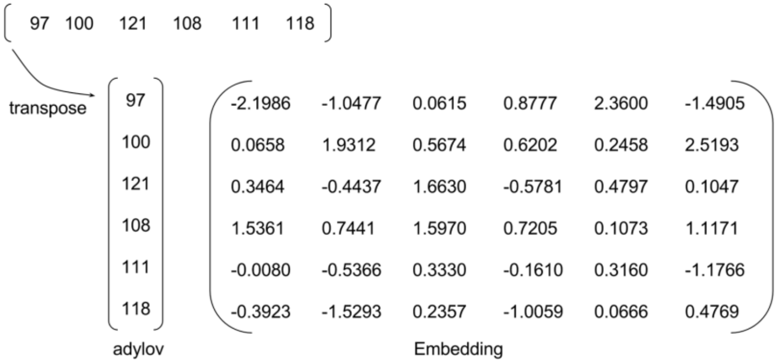
Implementation



```python
if __name__ == '__main__':
    classifier = RNNClassifier(N_CHARS,
                  HIDDEN_SIZE, N_CLASSES)

    arr, _ = str2ascii_arr('adylov')
    inp = Variable(torch.LongTensor([arr]))
    out = classifier(inp)
    print("in", inp.size(), "out", out.size())
    # in torch.Size([1, 6])
    #out torch.Size([1, 1, 18])
```

```python
class RNNClassifier(nn.Module):
    def __init__(self, input_size, hidden_size, output_size, n_layers=1):
        super(RNNClassifier, self).__init__()
        self.hidden_size = hidden_size
        self.n_layers = n_layers
        self.embedding = nn.Embedding(input_size, hidden_size)
        self.gru = nn.GRU(hidden_size, hidden_size, n_layers)
        self.fc = nn.Linear(hidden_size, output_size)

    def forward(self, input):
        # Note: we run this all at once (over the whole input sequence)
        # input = B x S . size(0) = B
        batch_size = input.size(0)

        # input:  B x S  -- (transpose) --> S x B
        input = input.t()

        # Embedding S x B -> S x B x I (embedding size)
        print("  input", input.size())
        embedded = self.embedding(input)
        print("  embedding", embedded.size())

        # Make a hidden
        hidden = self._init_hidden(batch_size)
        output, hidden = self.gru(embedded, hidden)
        print("  gru hidden output", hidden.size())
        # Use the last layer output as FC's input
        # No need to unpack, since we are going to use hidden
        fc_output = self.fc(hidden)
        print("  fc output", fc_output.size())
        return fc_output

    def _init_hidden(self, batch_size):
        hidden = torch.zeros(self.n_layers, batch_size, self.hidden_size)
        return Variable(hidden)
```

Implementation



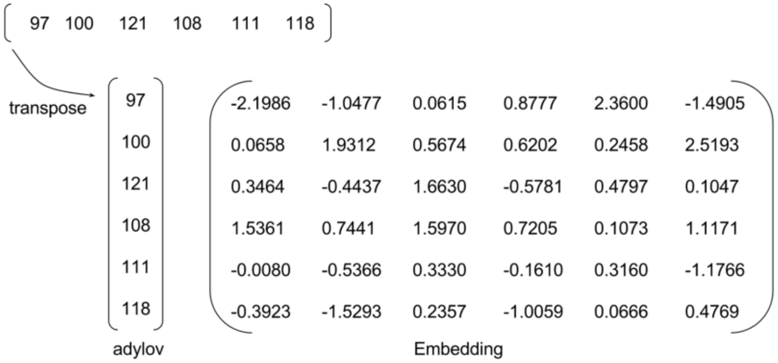
$$\begin{bmatrix} 97 & 100 & 121 & 108 & 111 & 118 \end{bmatrix}$$

$$\begin{bmatrix} 97 \\ 100 \\ 121 \\ 108 \\ 111 \\ 118 \end{bmatrix} \quad \begin{pmatrix} -2.1986 & -1.0477 & 0.0615 & 0.8777 & 2.3600 & -1.4905 \\ 0.0658 & 1.9312 & 0.5674 & 0.6202 & 0.2458 & 2.5193 \\ 0.3464 & -0.4437 & 1.6630 & -0.5781 & 0.4797 & 0.1047 \\ 1.5361 & 0.7441 & 1.5970 & 0.7205 & 0.1073 & 1.1171 \\ -0.0080 & -0.5366 & 0.3330 & -0.1610 & 0.3160 & -1.1766 \\ -0.3923 & -1.5293 & 0.2357 & -1.0059 & 0.0666 & 0.4769 \end{pmatrix}$$

```python
if __name__ == '__main__':
    classifier = RNNClassifier(N_CHARS,
                         HIDDEN_SIZE, N_CLASSES)

    arr, _ = str2ascii_arr('adylov')
    inp = Variable(torch.LongTensor([arr]))
    out = classifier(inp)
    print("in", inp.size(), "out", out.size())
    # in torch.Size([1, 6])
    #out torch.Size([1, 1, 18])
```

```python
class RNNClassifier(nn.Module):
    def __init__(self, input_size, hidden_size, output_size, n_layers=1):
        super(RNNClassifier, self).__init__()
        self.hidden_size = hidden_size
        self.n_layers = n_layers
        self.embedding = nn.Embedding(input_size, hidden_size)
        self.gru = nn.GRU(hidden_size, hidden_size, n_layers)
        self.fc = nn.Linear(hidden_size, output_size)

    def forward(self, input):
        # Note: we run this all at once (over the whole input sequence)
        # input = B x S . size(0) = B
        batch_size = input.size(0)

        # input:  B x S  -- (transpose) --> S x B
        input = input.t()

        # Embedding S x B -> S x B x I (embedding size)
        print("  input", input.size())
        embedded = self.embedding(input)
        print("  embedding", embedded.size())

        # Make a hidden
        hidden = self._init_hidden(batch_size)
        output, hidden = self.gru(embedded, hidden)
        print("  gru hidden output", hidden.size())
        # Use the last layer output as FC's input
        # No need to unpack, since we are going to use hidden
        fc_output = self.fc(hidden)
        print("  fc output", fc_output.size())
        return fc_output

    def _init_hidden(self, batch_size):
        hidden = torch.zeros(self.n_layers, batch_size, self.hidden_size)
        return Variable(hidden)
```

Batch?



$$\begin{bmatrix} 97 & 100 & 121 & 108 & 111 & 118 \end{bmatrix}$$

$$\begin{bmatrix} 97 \\ 100 \\ 121 \\ 108 \\ 111 \\ 118 \end{bmatrix} \quad \begin{pmatrix} -2.1986 & -1.0477 & 0.0615 & 0.8777 & 2.3600 & -1.4905 \\ 0.0658 & 1.9312 & 0.5674 & 0.6202 & 0.2458 & 2.5193 \\ 0.3464 & -0.4437 & 1.6630 & -0.5781 & 0.4797 & 0.1047 \\ 1.5361 & 0.7441 & 1.5970 & 0.7205 & 0.1073 & 1.1171 \\ -0.0080 & -0.5366 & 0.3330 & -0.1610 & 0.3160 & -1.1766 \\ -0.3923 & -1.5293 & 0.2357 & -1.0059 & 0.0666 & 0.4769 \end{pmatrix}$$

```
if __name__ == '__main__':
    classifier = RNNClassifier(N_CHARS,
                               HIDDEN_SIZE, N_CLASSES)

    arr, _ = str2ascii_arr('adylov')
    inp = Variable(torch.LongTensor([arr]))
    out = classifier(inp)
    print("in", inp.size(), "out", out.size())
    # in torch.Size([1, 6])
    #out torch.Size([1, 1, 18])
```

```
if __name__ == '__main__':
    names = ['adylov', 'solan', 'hard', 'san']
    classifier = RNNClassifier(N_CHARS, HIDDEN_SIZE, N_CLASSES)

    for name in names:
        arr, _ = str2ascii_arr(name)
        inp = Variable(torch.LongTensor([arr]))
        out = classifier(inp)
        print("in", inp.size(), "out", out.size())

    # in torch.Size([1, 6]) out torch.Size([1, 1, 18])
    # in torch.Size([1, 5]) out torch.Size([1, 1, 18])
    # ...
```

Batch?

$$\begin{pmatrix} 97 & 100 & 121 & 108 & 111 & 118 \end{pmatrix} \xrightarrow{\text{transpose}} \underbrace{\begin{pmatrix} 97 \\ 100 \\ 121 \\ 108 \\ 111 \\ 118 \end{pmatrix}}_{\text{adylov}} \quad \underbrace{\begin{pmatrix} -2.1986 & -1.0477 & 0.0615 & 0.8777 & 2.3600 & -1.4905 \\ 0.0658 & 1.9312 & 0.5674 & 0.6202 & 0.2458 & 2.5193 \\ 0.3464 & -0.4437 & 1.6630 & -0.5781 & 0.4797 & 0.1047 \\ 1.5361 & 0.7441 & 1.5970 & 0.7205 & 0.1073 & 1.1171 \\ -0.0080 & -0.5366 & 0.3330 & -0.1610 & 0.3160 & -1.1766 \\ -0.3923 & -1.5293 & 0.2357 & -1.0059 & 0.0666 & 0.4769 \end{pmatrix}}_{\text{Embedding}}$$

```python
if __name__ == '__main__':
    classifier = RNNClassifier(N_CHARS,
                               HIDDEN_SIZE, N_CLASSES)

    arr, _ = str2ascii_arr('adylov')
    inp = Variable(torch.LongTensor([arr]))
    out = classifier(inp)
    print("in", inp.size(), "out", out.size())
    # in torch.Size([1, 6])
    #out torch.Size([1, 1, 18])
```

$$\begin{array}{cccc} \text{adylov} & \text{sloan} & \text{harb} & \text{san} \\ \begin{pmatrix} 97 \\ 100 \\ 121 \\ 108 \\ 111 \\ 118 \end{pmatrix} & \begin{pmatrix} 115 \\ 111 \\ 108 \\ 97 \\ 110 \\ \end{pmatrix} & \begin{pmatrix} 104 \\ 97 \\ 114 \\ 100 \\ \\ \end{pmatrix} & \begin{pmatrix} 115 \\ 97 \\ 110 \\ \\ \\ \end{pmatrix} \end{array}$$

```python
if __name__ == '__main__':
    names = ['adylov', 'solan', 'hard', 'san']
    classifier = RNNClassifier(N_CHARS, HIDDEN_SIZE, N_CLASSES)

    for name in names:
        arr, _ = str2ascii_arr(name)
        inp = Variable(torch.LongTensor([arr]))
        out = classifier(inp)
        print("in", inp.size(), "out", out.size())

    # in torch.Size([1, 6]) out torch.Size([1, 1, 18])
    # in torch.Size([1, 5]) out torch.Size([1, 1, 18])
    # ...
```

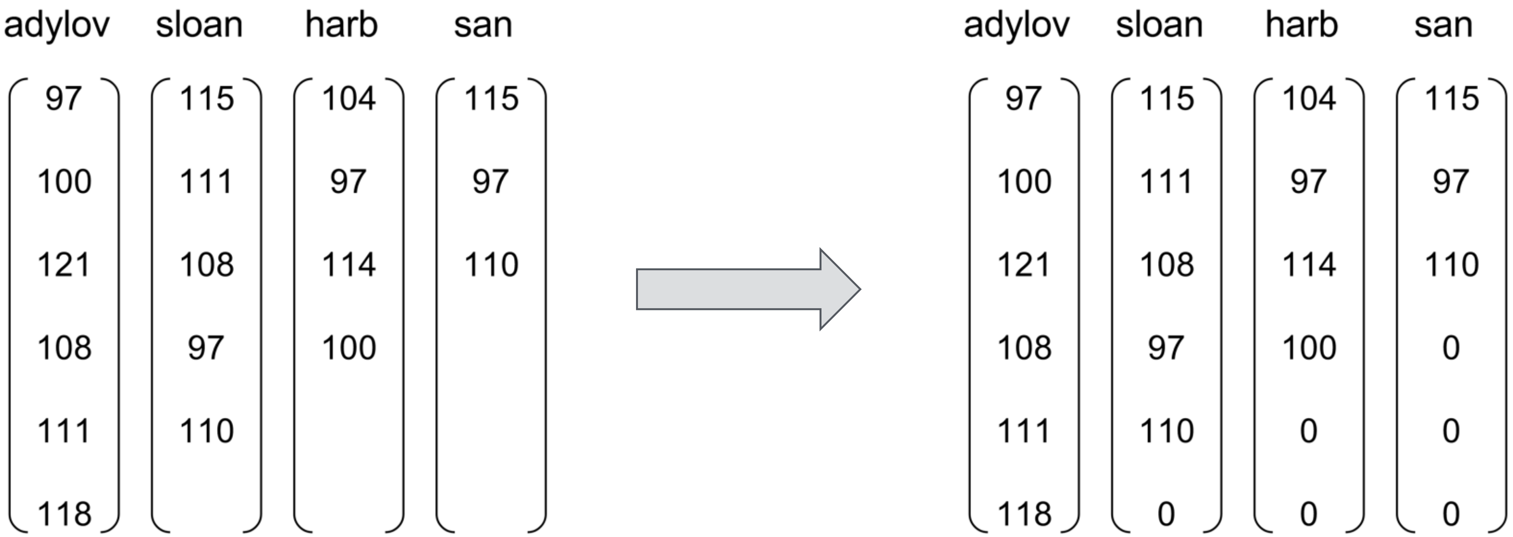
Zero padding

$$\begin{array}{cccc} \text{adylov} & \text{sloan} & \text{harb} & \text{san} \\ \begin{pmatrix} 97 \\ 100 \\ 121 \\ 108 \\ 111 \\ 118 \end{pmatrix} & \begin{pmatrix} 115 \\ 111 \\ 108 \\ 97 \\ 110 \\ \end{pmatrix} & \begin{pmatrix} 104 \\ 97 \\ 114 \\ 100 \\ \\ \end{pmatrix} & \begin{pmatrix} 115 \\ 97 \\ 110 \\ \\ \\ \end{pmatrix} \end{array}$$

Zero padding

$$\begin{array}{cccc} \text{adylov} & \text{sloan} & \text{harb} & \text{san} \\ \begin{pmatrix}97\\100\\121\\108\\111\\118\end{pmatrix} & \begin{pmatrix}115\\111\\108\\97\\110\\ \end{pmatrix} & \begin{pmatrix}104\\97\\114\\100\\ \\ \end{pmatrix} & \begin{pmatrix}115\\97\\110\\ \\ \\ \end{pmatrix} \end{array} \Longrightarrow \begin{array}{cccc} \text{adylov} & \text{sloan} & \text{harb} & \text{san} \\ \begin{pmatrix}97\\100\\121\\108\\111\\118\end{pmatrix} & \begin{pmatrix}115\\111\\108\\97\\110\\0\end{pmatrix} & \begin{pmatrix}104\\97\\114\\100\\0\\0\end{pmatrix} & \begin{pmatrix}115\\97\\110\\0\\0\\0\end{pmatrix} \end{array}$$

Zero padding

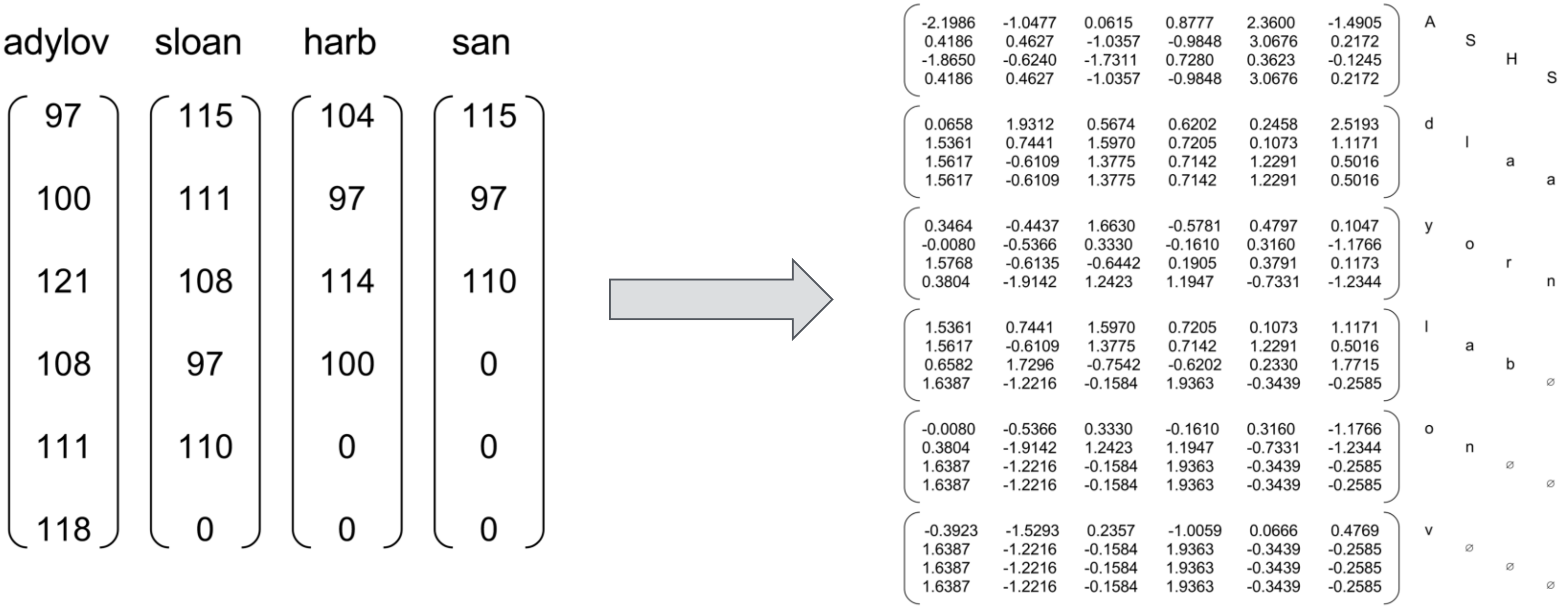


adylov	sloan	harb	san
97	115	104	115
100	111	97	97
121	108	114	110
108	97	100	
111	110		
118			

adylov	sloan	harb	san
97	115	104	115
100	111	97	97
121	108	114	110
108	97	100	0
111	110	0	0
118	0	0	0

```
def pad_sequences(vectorized_seqs, seq_lengths):
   seq_tensor = torch.zeros((len(vectorized_seqs), seq_lengths.max())).long()
   for idx, (seq, seq_len) in enumerate(zip(vectorized_seqs, seq_lengths)):
       seq_tensor[idx, :seq_len] = torch.LongTensor(seq)
   return  seq_tensor
```

Embedding



adylov	sloan	harb	san
97	115	104	115
100	111	97	97
121	108	114	110
108	97	100	0
111	110	0	0
118	0	0	0

-2.1986	-1.0477	0.0615	0.8777	2.3600	-1.4905	A			
0.4186	0.4627	-1.0357	-0.9848	3.0676	0.2172		S		
-1.8650	-0.6240	-1.7311	0.7280	0.3623	-0.1245			H	
0.4186	0.4627	-1.0357	-0.9848	3.0676	0.2172				S

0.0658	1.9312	0.5674	0.6202	0.2458	2.5193	d			
1.5361	0.7441	1.5970	0.7205	0.1073	1.1171		l		
1.5617	-0.6109	1.3775	0.7142	1.2291	0.5016			a	
1.5617	-0.6109	1.3775	0.7142	1.2291	0.5016				a

0.3464	-0.4437	1.6630	-0.5781	0.4797	0.1047	y			
-0.0080	-0.5366	0.3330	-0.1610	0.3160	-1.1766		o		
1.5768	-0.6135	-0.6442	0.1905	0.3791	0.1173			r	
0.3804	-1.9142	1.2423	1.1947	-0.7331	-1.2344				n

1.5361	0.7441	1.5970	0.7205	0.1073	1.1171	l			
1.5617	-0.6109	1.3775	0.7142	1.2291	0.5016		a		
0.6582	1.7296	-0.7542	-0.6202	0.2330	1.7715			b	
1.6387	-1.2216	-0.1584	1.9363	-0.3439	-0.2585				∅

-0.0080	-0.5366	0.3330	-0.1610	0.3160	-1.1766	o			
0.3804	-1.9142	1.2423	1.1947	-0.7331	-1.2344		n		
1.6387	-1.2216	-0.1584	1.9363	-0.3439	-0.2585			∅	
1.6387	-1.2216	-0.1584	1.9363	-0.3439	-0.2585				∅

-0.3923	-1.5293	0.2357	-1.0059	0.0666	0.4769	v			
1.6387	-1.2216	-0.1584	1.9363	-0.3439	-0.2585		∅		
1.6387	-1.2216	-0.1584	1.9363	-0.3439	-0.2585			∅	
1.6387	-1.2216	-0.1584	1.9363	-0.3439	-0.2585				∅

```
# Embedding S x B -> S x B x I (embedding size)
embedded = self.embedding(input)
```

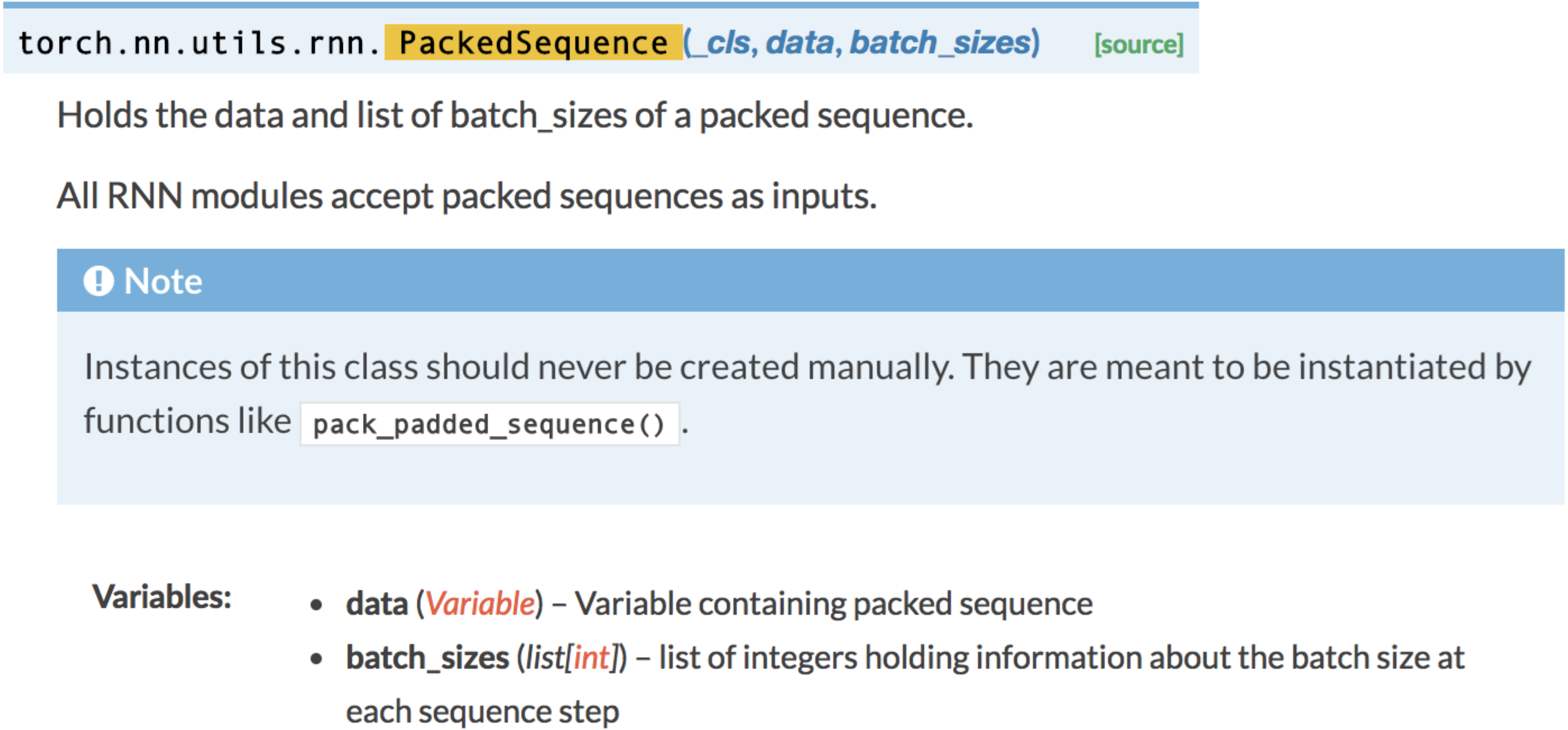
Full implementation

```python
def str2ascii_arr(msg):
   arr = [ord(c) for c in msg]
   return arr, len(arr)

# pad sequences and sort the tensor
def pad_sequences(vectorized_seqs, seq_lengths):
   seq_tensor = torch.zeros((len(vectorized_seqs), seq_lengths.max())).long()
   for idx, (seq, seq_len) in enumerate(zip(vectorized_seqs, seq_lengths)):
       seq_tensor[idx, :seq_len] = torch.LongTensor(seq)
   return seq_tensor

# Create necessary variables, lengths, and target
def make_variables(names):
   sequence_and_length = [str2ascii_arr(name) for name in names]
   vectorized_seqs = [sl[0] for sl in sequence_and_length]
   seq_lengths = torch.LongTensor([sl[1] for sl in sequence_and_length])
   return pad_sequences(vectorized_seqs, seq_lengths)


if __name__ == '__main__':
   names = ['adylov', 'solan', 'hard', 'san']
   classifier = RNNClassifier(N_CHARS, HIDDEN_SIZE, N_CLASSES)

   inputs = make_variables(names)
   out = classifier(inp)
   print("batch in", inp.size(), "batch out", out.size())

   # batch in torch.Size([4, 6]) batch out torch.Size([1, 4, 18])
```

Full implementation

```python
def str2ascii_arr(msg):
   arr = [ord(c) for c in msg]
   return arr, len(arr)

# pad sequences and sort the tensor
def pad_sequences(vectorized_seqs, seq_lengths):
   seq_tensor = torch.zeros((len(vectorized_seqs), seq_lengths.max())).long()
   for idx, (seq, seq_len) in enumerate(zip(vectorized_seqs, seq_lengths)):
       seq_tensor[idx, :seq_len] = torch.LongTensor(seq)
   return seq_tensor

# Create necessary variables, lengths, and target
def make_variables(names):
   sequence_and_length = [str2ascii_arr(name) for name in names]
   vectorized_seqs = [sl[0] for sl in sequence_and_length]
   seq_lengths = torch.LongTensor([sl[1] for sl in sequence_and_length])
   return pad_sequences(vectorized_seqs, seq_lengths)


if __name__ == '__main__':
   names = ['adylov', 'solan', 'hard', 'san']
   classifier = RNNClassifier(N_CHARS, HIDDEN_SIZE, N_CLASSES)

   inputs = make_variables(names)
   out = classifier(inputs)
   print("batch in", inputs.size(), "batch out", out.size())

   # batch in torch.Size([4, 6]) batch out torch.Size([1, 4, 18])
```

```python
optimizer =
torch.optim.Adam(classifier.parameters(),
                                    lr=0.001)
criterion = nn.CrossEntropyLoss()

…

loss = criterion(output, target)

classifier.zero_grad()
loss.backward()
optimizer.step()
```



`torch.nn.utils.rnn.PackedSequence`(*_cls*, *data*, *batch_sizes*) [source]

Holds the data and list of batch_sizes of a packed sequence.

All RNN modules accept packed sequences as inputs.

> **Note**
>
> Instances of this class should never be created manually. They are meant to be instantiated by functions like `pack_padded_sequence()`.

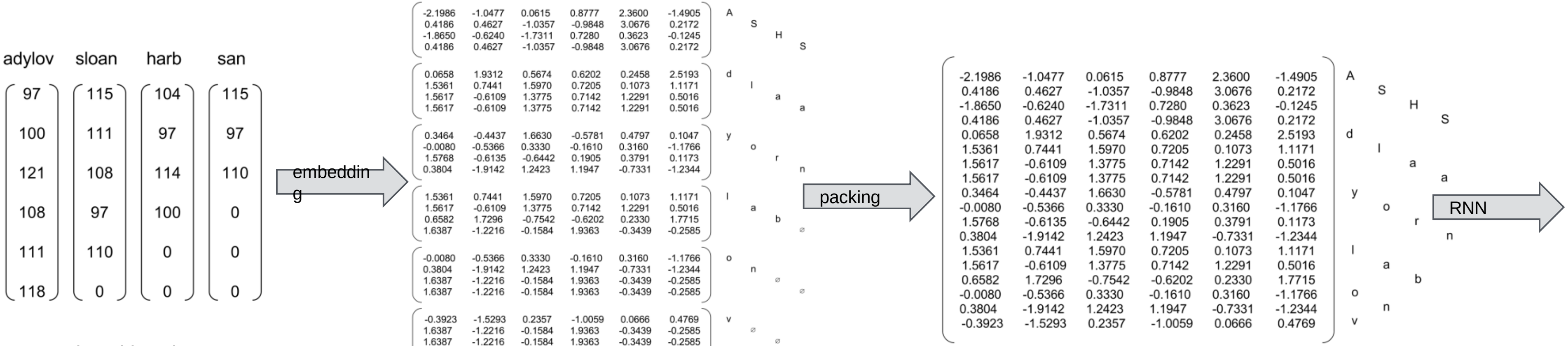
Variables:

- **data** (*Variable*) – Variable containing packed sequence
- **batch_sizes** (*list[int]*) – list of integers holding information about the batch size at each sequence step

http://pytorch.org/docs/0.3.0/nn.html?highlight=packedsequence#torch.nn.utils.rnn.PackedSequence

Efficiently handling batched sequences with variable lengths:

pack_padded_sequence



adylov	sloan	harb	san
97	115	104	115
100	111	97	97
121	108	114	110
108	97	100	0
111	110	0	0
118	0	0	0

$$\left(\begin{array}{cccccc} -2.1986 & -1.0477 & 0.0615 & 0.8777 & 2.3600 & -1.4905 \\ 0.4186 & 0.4627 & -1.0357 & -0.9848 & 3.0676 & 0.2172 \\ -1.8650 & -0.6240 & -1.7311 & 0.7280 & 0.3623 & -0.1245 \\ 0.4186 & 0.4627 & -1.0357 & -0.9848 & 3.0676 & 0.2172 \end{array}\right) \begin{array}{l} A \\ S \\ H \\ S \end{array}$$

$$\left(\begin{array}{cccccc} 0.0658 & 1.9312 & 0.5674 & 0.6202 & 0.2458 & 2.5193 \\ 1.5361 & 0.7441 & 1.5970 & 0.7205 & 0.1073 & 1.1171 \\ 1.5617 & -0.6109 & 1.3775 & 0.7142 & 1.2291 & 0.5016 \\ 1.5617 & -0.6109 & 1.3775 & 0.7142 & 1.2291 & 0.5016 \end{array}\right) \begin{array}{l} d \\ l \\ a \\ a \end{array}$$

$$\left(\begin{array}{cccccc} 0.3464 & -0.4437 & 1.6630 & -0.5781 & 0.4797 & 0.1047 \\ -0.0080 & -0.5366 & 0.3330 & -0.1610 & 0.3160 & -1.1766 \\ 1.5768 & -0.6135 & -0.6442 & 0.1905 & 0.3791 & 0.1173 \\ 0.3804 & -1.9142 & 1.2423 & 1.1947 & -0.7331 & -1.2344 \end{array}\right) \begin{array}{l} y \\ o \\ r \\ n \end{array}$$

$$\left(\begin{array}{cccccc} 1.5361 & 0.7441 & 1.5970 & 0.7205 & 0.1073 & 1.1171 \\ 1.5617 & -0.6109 & 1.3775 & 0.7142 & 1.2291 & 0.5016 \\ 0.6582 & 1.7296 & -0.7542 & -0.6202 & 0.2330 & 1.7715 \\ 1.6387 & -1.2216 & -0.1584 & 1.9363 & -0.3439 & -0.2585 \end{array}\right) \begin{array}{l} l \\ a \\ b \\ \varnothing \end{array}$$

$$\left(\begin{array}{cccccc} -0.0080 & -0.5366 & 0.3330 & -0.1610 & 0.3160 & -1.1766 \\ 0.3804 & -1.9142 & 1.2423 & 1.1947 & -0.7331 & -1.2344 \\ 1.6387 & -1.2216 & -0.1584 & 1.9363 & -0.3439 & -0.2585 \\ 1.6387 & -1.2216 & -0.1584 & 1.9363 & -0.3439 & -0.2585 \end{array}\right) \begin{array}{l} o \\ n \\ \varnothing \\ \varnothing \end{array}$$

$$\left(\begin{array}{cccccc} -0.3923 & -1.5293 & 0.2357 & -1.0059 & 0.0666 & 0.4769 \\ 1.6387 & -1.2216 & -0.1584 & 1.9363 & -0.3439 & -0.2585 \\ 1.6387 & -1.2216 & -0.1584 & 1.9363 & -0.3439 & -0.2585 \\ 1.6387 & -1.2216 & -0.1584 & 1.9363 & -0.3439 & -0.2585 \end{array}\right) \begin{array}{l} v \\ \varnothing \\ \varnothing \\ \varnothing \end{array}$$

$$\left(\begin{array}{cccccc} -2.1986 & -1.0477 & 0.0615 & 0.8777 & 2.3600 & -1.4905 \\ 0.4186 & 0.4627 & -1.0357 & -0.9848 & 3.0676 & 0.2172 \\ -1.8650 & -0.6240 & -1.7311 & 0.7280 & 0.3623 & -0.1245 \\ 0.4186 & 0.4627 & -1.0357 & -0.9848 & 3.0676 & 0.2172 \\ 0.0658 & 1.9312 & 0.5674 & 0.6202 & 0.2458 & 2.5193 \\ 1.5361 & 0.7441 & 1.5970 & 0.7205 & 0.1073 & 1.1171 \\ 1.5617 & -0.6109 & 1.3775 & 0.7142 & 1.2291 & 0.5016 \\ 1.5617 & -0.6109 & 1.3775 & 0.7142 & 1.2291 & 0.5016 \\ 0.3464 & -0.4437 & 1.6630 & -0.5781 & 0.4797 & 0.1047 \\ -0.0080 & -0.5366 & 0.3330 & -0.1610 & 0.3160 & -1.1766 \\ 1.5768 & -0.6135 & -0.6442 & 0.1905 & 0.3791 & 0.1173 \\ 0.3804 & -1.9142 & 1.2423 & 1.1947 & -0.7331 & -1.2344 \\ 1.5361 & 0.7441 & 1.5970 & 0.7205 & 0.1073 & 1.1171 \\ 1.5617 & -0.6109 & 1.3775 & 0.7142 & 1.2291 & 0.5016 \\ 0.6582 & 1.7296 & -0.7542 & -0.6202 & 0.2330 & 1.7715 \\ -0.0080 & -0.5366 & 0.3330 & -0.1610 & 0.3160 & -1.1766 \\ 0.3804 & -1.9142 & 1.2423 & 1.1947 & -0.7331 & -1.2344 \\ -0.3923 & -1.5293 & 0.2357 & -1.0059 & 0.0666 & 0.4769 \end{array}\right) \begin{array}{l} A \\ S \\ H \\ S \\ d \\ l \\ a \\ a \\ y \\ o \\ r \\ n \\ l \\ a \\ b \\ o \\ n \\ v \end{array}$$

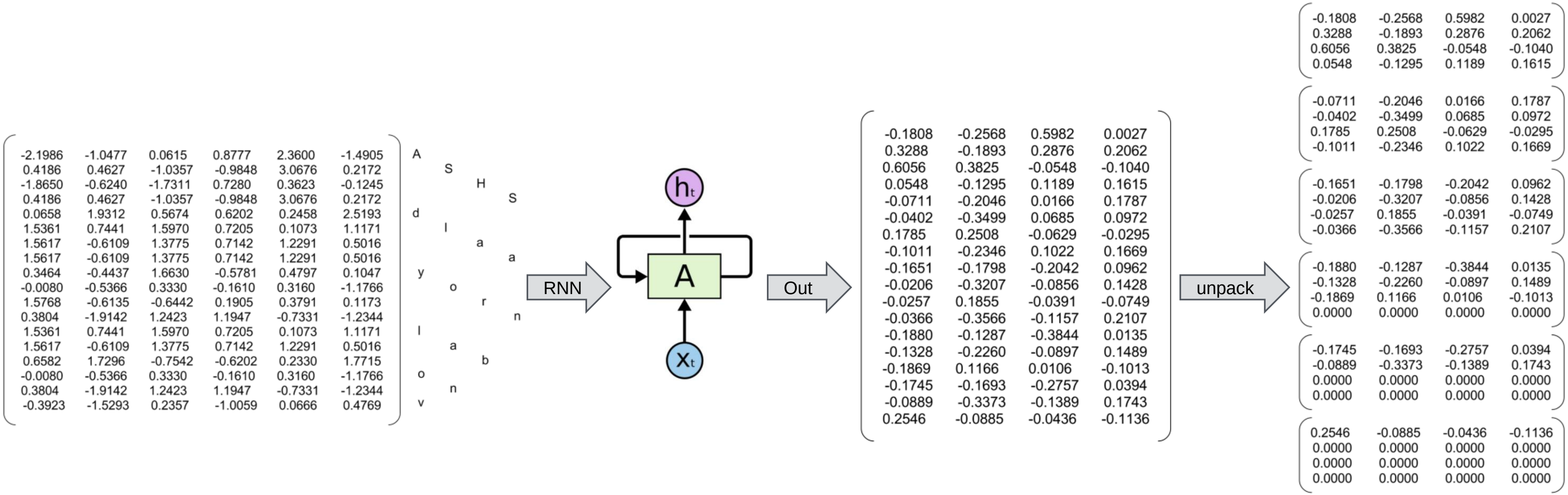
Ordered batch
(by length)

https://github.com/hunkim/PyTorchZeroToAll/blob/master/13_4_pack_pad.py

Matrix visualization from Nicolas, https://github.com/ngarneau

Efficiently handling batched sequences with variable lengths:

pack_padded_sequence



$$\left(\begin{array}{cccccc}
-2.1986 & -1.0477 & 0.0615 & 0.8777 & 2.3600 & -1.4905 \\
0.4186 & 0.4627 & -1.0357 & -0.9848 & 3.0676 & 0.2172 \\
-1.8650 & -0.6240 & -1.7311 & 0.7280 & 0.3623 & -0.1245 \\
0.4186 & 0.4627 & -1.0357 & -0.9848 & 3.0676 & 0.2172 \\
0.0658 & 1.9312 & 0.5674 & 0.6202 & 0.2458 & 2.5193 \\
1.5361 & 0.7441 & 1.5970 & 0.7205 & 0.1073 & 1.1171 \\
1.5617 & -0.6109 & 1.3775 & 0.7142 & 1.2291 & 0.5016 \\
1.5617 & -0.6109 & 1.3775 & 0.7142 & 1.2291 & 0.5016 \\
0.3464 & -0.4437 & 1.6630 & -0.5781 & 0.4797 & 0.1047 \\
-0.0080 & -0.5366 & 0.3330 & -0.1610 & 0.3160 & -1.1766 \\
1.5768 & -0.6135 & -0.6442 & 0.1905 & 0.3791 & 0.1173 \\
0.3804 & -1.9142 & 1.2423 & 1.1947 & -0.7331 & -1.2344 \\
1.5361 & 0.7441 & 1.5970 & 0.7205 & 0.1073 & 1.1171 \\
1.5617 & -0.6109 & 1.3775 & 0.7142 & 1.2291 & 0.5016 \\
0.6582 & 1.7296 & -0.7542 & -0.6202 & 0.2330 & 1.7715 \\
-0.0080 & -0.5366 & 0.3330 & -0.1610 & 0.3160 & -1.1766 \\
0.3804 & -1.9142 & 1.2423 & 1.1947 & -0.7331 & -1.2344 \\
-0.3923 & -1.5293 & 0.2357 & -1.0059 & 0.0666 & 0.4769
\end{array}\right)
\begin{array}{llll}
\text{A} & & & \\
 & \text{S} & & \\
 & & \text{H} & \\
 & & & \text{S} \\
\text{d} & & & \\
 & \text{l} & & \\
 & & \text{a} & \\
 & & & \text{a} \\
\text{y} & & & \\
 & \text{o} & & \\
 & & \text{r} & \\
 & & & \text{n} \\
\text{l} & & & \\
 & \text{a} & & \\
 & & \text{b} & \\
\text{o} & & & \\
 & \text{n} & & \\
\text{v} & & &
\end{array}$$

$$\begin{pmatrix}
-0.1808 & -0.2568 & 0.5982 & 0.0027 \\
0.3288 & -0.1893 & 0.2876 & 0.2062 \\
0.6056 & 0.3825 & -0.0548 & -0.1040 \\
0.0548 & -0.1295 & 0.1189 & 0.1615 \\
-0.0711 & -0.2046 & 0.0166 & 0.1787 \\
-0.0402 & -0.3499 & 0.0685 & 0.0972 \\
0.1785 & 0.2508 & -0.0629 & -0.0295 \\
-0.1011 & -0.2346 & 0.1022 & 0.1669 \\
-0.1651 & -0.1798 & -0.2042 & 0.0962 \\
-0.0206 & -0.3207 & -0.0856 & 0.1428 \\
-0.0257 & 0.1855 & -0.0391 & -0.0749 \\
-0.0366 & -0.3566 & -0.1157 & 0.2107 \\
-0.1880 & -0.1287 & -0.3844 & 0.0135 \\
-0.1328 & -0.2260 & -0.0897 & 0.1489 \\
-0.1869 & 0.1166 & 0.0106 & -0.1013 \\
-0.1745 & -0.1693 & -0.2757 & 0.0394 \\
-0.0889 & -0.3373 & -0.1389 & 0.1743 \\
0.2546 & -0.0885 & -0.0436 & -0.1136
\end{pmatrix}$$

$$\begin{pmatrix}
-0.1808 & -0.2568 & 0.5982 & 0.0027 \\
0.3288 & -0.1893 & 0.2876 & 0.2062 \\
0.6056 & 0.3825 & -0.0548 & -0.1040 \\
0.0548 & -0.1295 & 0.1189 & 0.1615
\end{pmatrix}$$

$$\begin{pmatrix}
-0.0711 & -0.2046 & 0.0166 & 0.1787 \\
-0.0402 & -0.3499 & 0.0685 & 0.0972 \\
0.1785 & 0.2508 & -0.0629 & -0.0295 \\
-0.1011 & -0.2346 & 0.1022 & 0.1669
\end{pmatrix}$$

$$\begin{pmatrix}
-0.1651 & -0.1798 & -0.2042 & 0.0962 \\
-0.0206 & -0.3207 & -0.0856 & 0.1428 \\
-0.0257 & 0.1855 & -0.0391 & -0.0749 \\
-0.0366 & -0.3566 & -0.1157 & 0.2107
\end{pmatrix}$$

$$\begin{pmatrix}
-0.1880 & -0.1287 & -0.3844 & 0.0135 \\
-0.1328 & -0.2260 & -0.0897 & 0.1489 \\
-0.1869 & 0.1166 & 0.0106 & -0.1013 \\
0.0000 & 0.0000 & 0.0000 & 0.0000
\end{pmatrix}$$

$$\begin{pmatrix}
-0.1745 & -0.1693 & -0.2757 & 0.0394 \\
-0.0889 & -0.3373 & -0.1389 & 0.1743 \\
0.0000 & 0.0000 & 0.0000 & 0.0000 \\
0.0000 & 0.0000 & 0.0000 & 0.0000
\end{pmatrix}$$

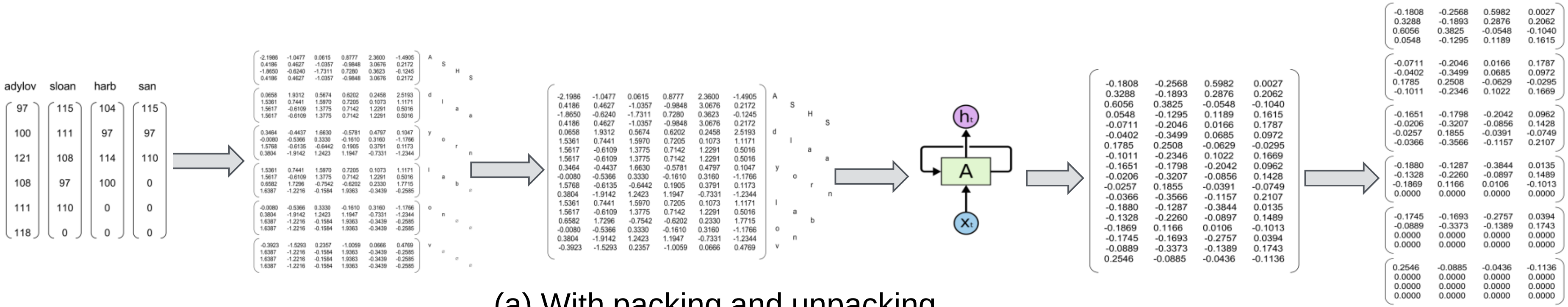
$$\begin{pmatrix}
0.2546 & -0.0885 & -0.0436 & -0.1136 \\
0.0000 & 0.0000 & 0.0000 & 0.0000 \\
0.0000 & 0.0000 & 0.0000 & 0.0000 \\
0.0000 & 0.0000 & 0.0000 & 0.0000
\end{pmatrix}$$

https://github.com/hunkim/PyTorchZeroToAll/blob/master/13_4_pack_pad.py

Matrix visualization from Nicolas, https://github.com/ngarneau

Efficiently handling batched sequences with variable lengths:

pack_padded_sequence



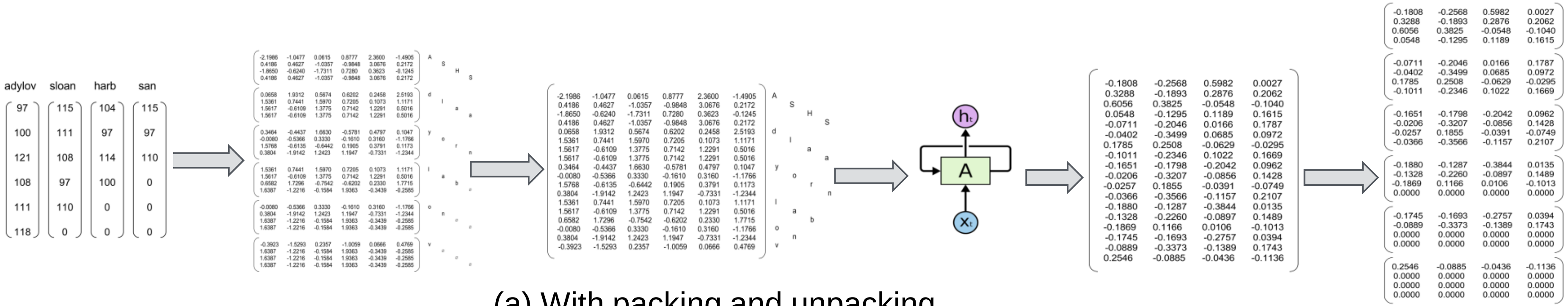
(a) With packing and unpacking

https://github.com/hunkim/PyTorchZeroToAll/blob/master/13_4_pack_pad.py

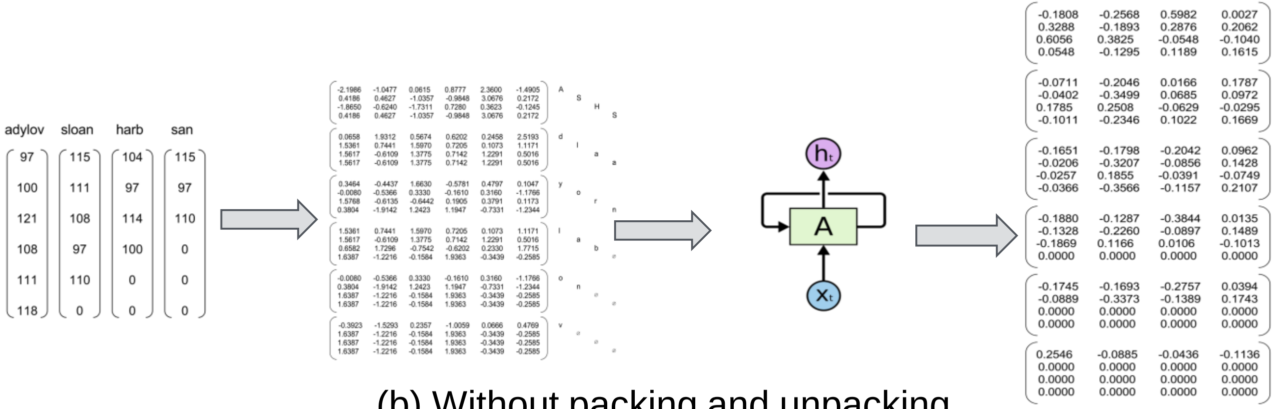
Matrix visualization from Nicolas, https://github.com/ngarneau

Efficiently handling batched sequences with variable lengths:

pack_padded_sequence



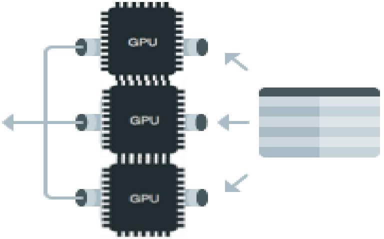
(a) With packing and unpacking



(b) Without packing and unpacking

Matrix visualization from Nicolas, https://github.com/ngarneau

GPU/Data Parallel



1 Copy all variables to gpu

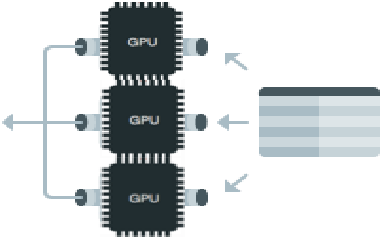
```
if torch.cuda.is_available():
    return Variable(tensor.cuda())
else:
    return Variable(tensor)
```

2 Put your models on gpu

```
classifier = RNNClassifier(N_CHARS, HIDDEN_SIZE, ...
if torch.cuda.is_available():
    classifier.cuda()
```

http://pytorch.org/tutorials/beginner/blitz/data_parallel_tutorial.html#sphx-glr-beginner-blitz-data-parallel-tutorial-py

GPU/Data Parallel



1 Copy all variables to gpu

```
if torch.cuda.is_available():
   return Variable(tensor.cuda())
else:
   return Variable(tensor)
```

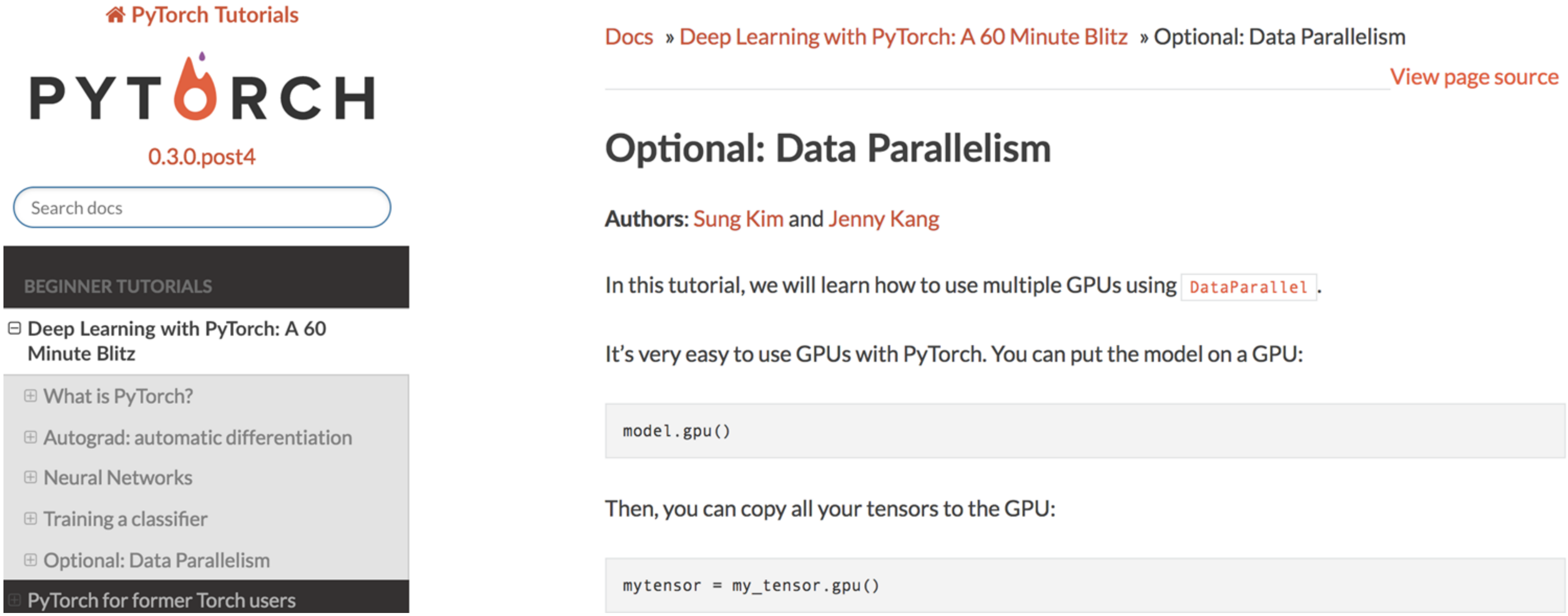
2 Put your models on gpu

```
classifier = RNNClassifier(N_CHARS, HIDDEN_SIZE, ...
if torch.cuda.is_available():
   classifier.cuda()
```

3 Wrap your model using data parallel

```
if torch.cuda.device_count() > 1:
   print("Let's use", torch.cuda.device_count(), "GPUs!")
   # dim = 0 [33, xxx] -> [11, ...], [11, ...], [11, ...] on 3 GPUs
   classifier = nn.DataParallel(classifier)
```

http://pytorch.org/tutorials/beginner/blitz/data_parallel_tutorial.html#sphx-glr-beginner-blitz-data-parallel-tutorial-py



PyTorch Tutorials

PYTORCH

0.3.0.post4

Search docs

BEGINNER TUTORIALS

⊟ Deep Learning with PyTorch: A 60 Minute Blitz

⊞ What is PyTorch?

⊞ Autograd: automatic differentiation

⊞ Neural Networks

⊞ Training a classifier

⊞ Optional: Data Parallelism

⊞ PyTorch for former Torch users

Docs » Deep Learning with PyTorch: A 60 Minute Blitz » Optional: Data Parallelism

View page source

Optional: Data Parallelism

Authors: Sung Kim and Jenny Kang

In this tutorial, we will learn how to use multiple GPUs using `DataParallel`.

It's very easy to use GPUs with PyTorch. You can put the model on a GPU:

```
model.gpu()
```

Then, you can copy all your tensors to the GPU:

```
mytensor = my_tensor.gpu()
```

http://pytorch.org/tutorials/beginner/blitz/data_parallel_tutorial.html#sphx-glr-beginner-blitz-data-parallel-tutorial-py

Exercise 13-1: Implement the full name classification

- With GPU
- With data parallel
- Use pad-pack

Exercise 13-2: Sentiment analysis on movie reviews

- The sentiment labels are:
  0 - negative
  1 - somewhat negative
  2 - neutral
  3 - somewhat positive
  4 - positive

- https://www.kaggle.com/c/sentiment-analysis-on-movie-reviews/data

Lecture 14: Language modeling

ML/DL for Everyone with PYTORCH

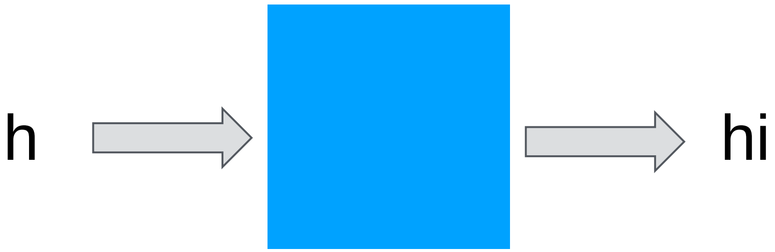
Lecture 14: RNN III (language modeling)

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

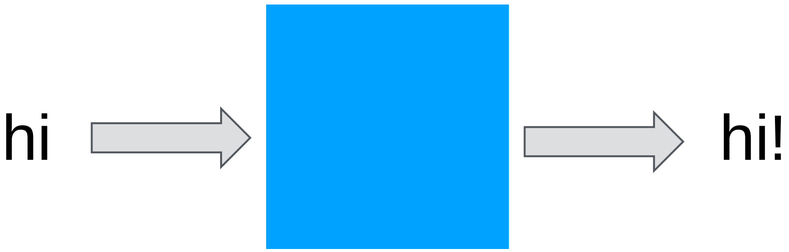
Code: https://github.com/hunkim/PyTorchZeroToAll

Slides: http://bit.ly/PyTorchZeroAll

(Character level) Language Modeling



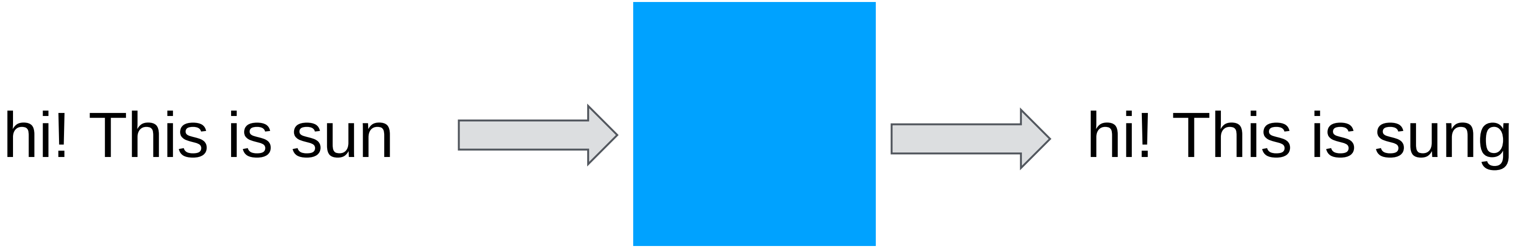
(Character level) Language Modeling



hi

hi!

(Character level) Language Modeling



hi! This is sun

hi! This is sung

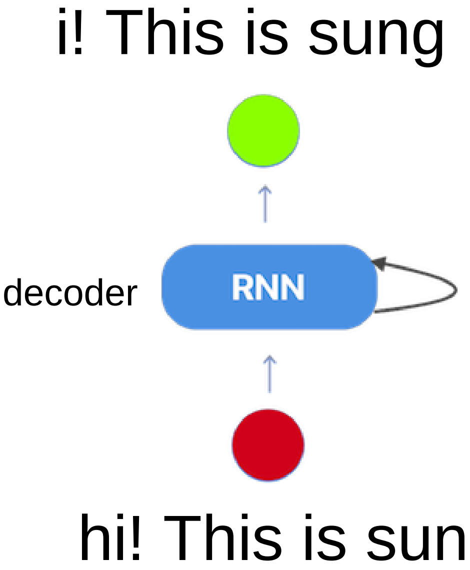
(Character level) Language Modeling

hi! This is sun

hi! This is sung

$$p(y_t | y_1, y_2, \dots, y_{t-1})$$

Language Modeling using RNN



i! This is sung

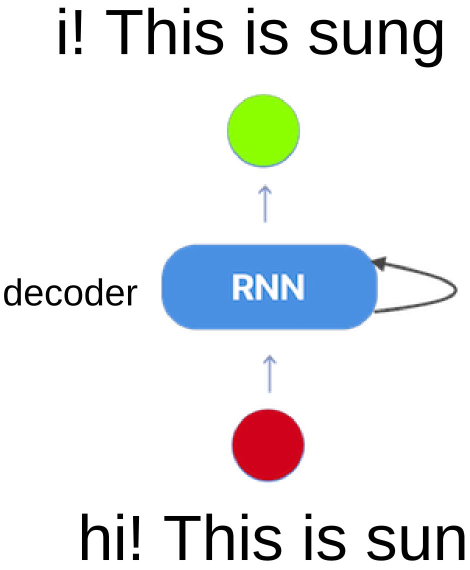
decoder

RNN

hi! This is sun

```python
def generate(decoder, prime_str='A', predict_len=100, temperature=0.8):
    hidden = decoder.init_hidden()
    prime_input = str2tensor(prime_str)
    predicted = prime_str

    # Use priming string to "build up" hidden state
    for p in range(len(prime_str) - 1):
        _, hidden = decoder(prime_input[p], hidden)

    inp = prime_input[-1]
    for p in range(predict_len):
        output, hidden = decoder(inp, hidden)

        # Sample from the network as a multinomial distribution
        output_dist = output.data.view(-1).div(temperature).exp()
        top_i = torch.multinomial(output_dist, 1)[0]

        # Add predicted character to string and use as next input
        predicted_char = chr(top_i)
        predicted += predicted_char
        inp = str2tensor(predicted_char)

    return predicted
```

Training



i! This is sung

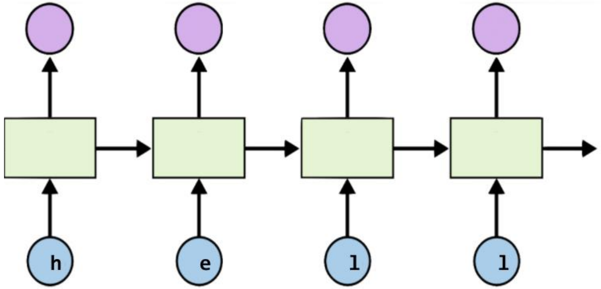
decoder

RNN

hi! This is sun

```
def train_teacher_forching(line):
   input = str2tensor(line[:-1])
   target = str2tensor(line[1:])

   hidden = RNNdecoder.init_hidden()
   loss = 0

   for c in range(len(input)):
       output, hidden = decoder(input[c], hidden)
       loss += criterion(output, target[c])

   decoder.zero_grad()
   loss.backward()
   decoder_optimizer.step()

   return loss.data[0] / len(input)
```

Teacher Forcing

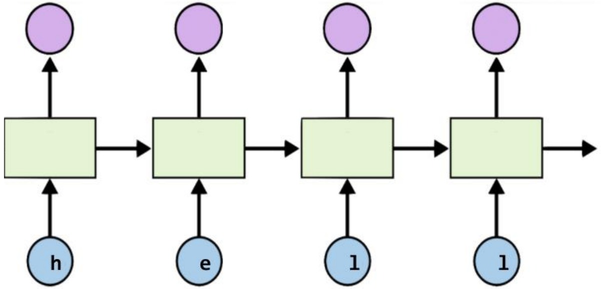


Teacher Forcing

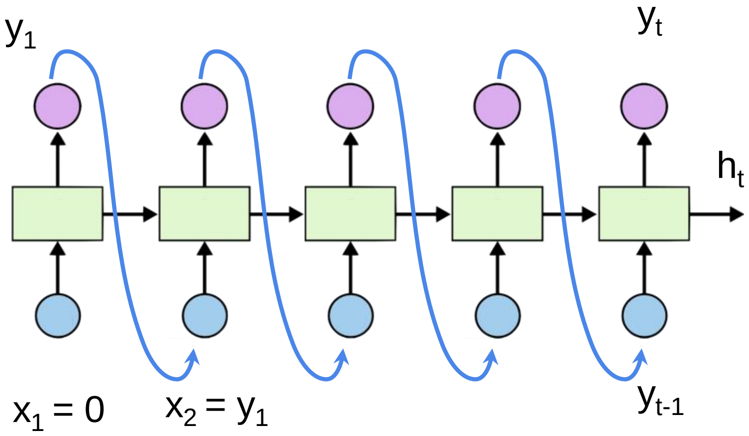
```python
def train_teacher_forching(line):
    input = str2tensor(line[:-1])
    target = str2tensor(line[1:])

    hidden = RNNdecoder.init_hidden()
    loss = 0

    for c in range(len(input)):
        output, hidden = decoder(input[c], hidden)
        loss += criterion(output, target[c])

    decoder.zero_grad()
    loss.backward()
    decoder_optimizer.step()

    return loss.data[0] / len(input)
```

https://www.quora.com/What-is-the-teacher-forcing-in-RNN

Teacher Forcing



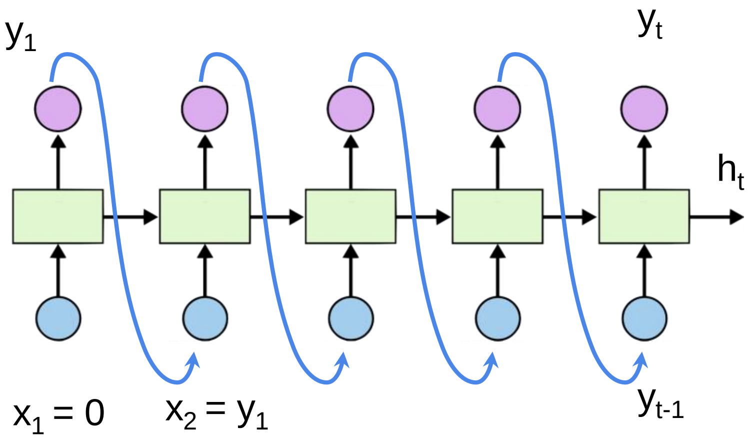
Teacher Forcing



No Teacher Forcing
(more natural)

https://www.quora.com/What-is-the-teacher-forcing-in-RNN

Training: no teacher forcing

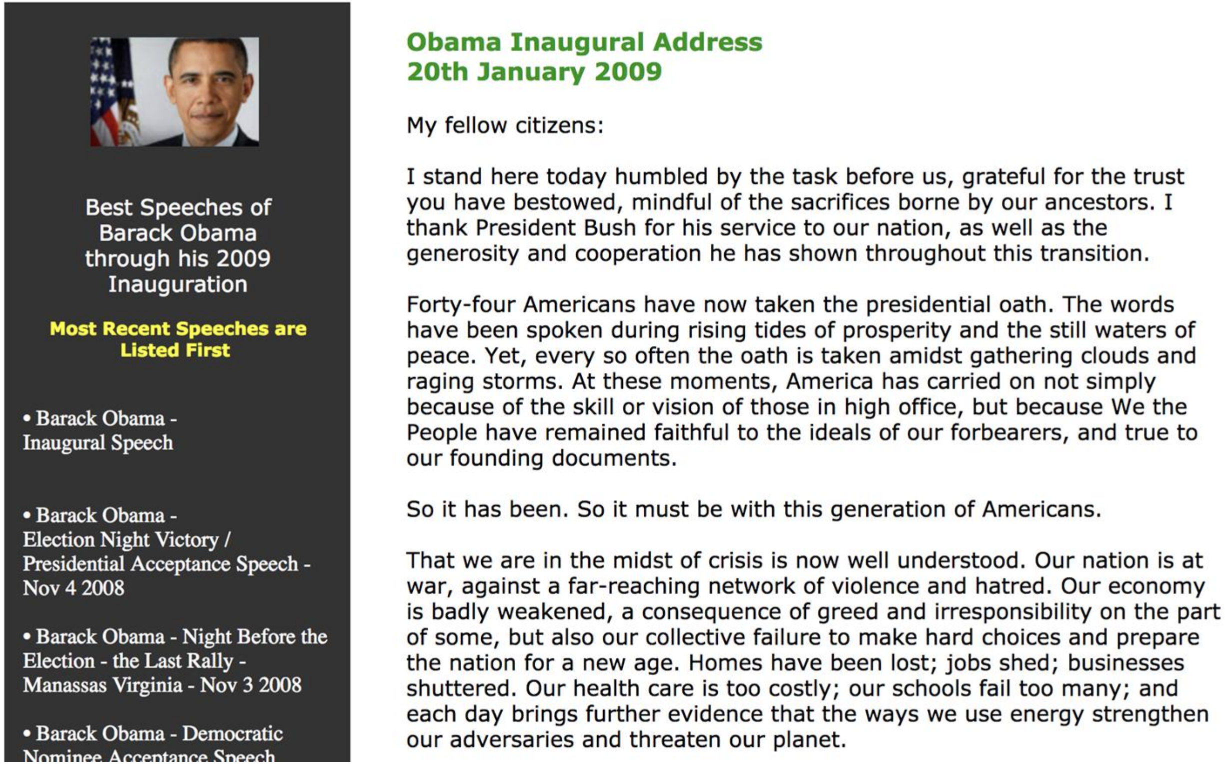


```
def train(line):
    input = str2tensor(line[:-1])
    target = str2tensor(line[1:])

    hidden = decoder.init_hidden()
    decoder_in = input[0]
    loss = 0

    for c in range(len(input)):
        output, hidden = decoder(decoder_in, hidden)
        loss += criterion(output, target[c])
        decoder_in = output.max(1)[1]

    decoder.zero_grad()
    loss.backward()
    decoder_optimizer.step()

    return loss.data[0] / len(input)
```

Data preparation and implementation details

https://github.com/hunkim/PyTorchZeroToAll

Exercise 14-1: Language model with Obama speech



Best Speeches of Barack Obama through his 2009 Inauguration

Most Recent Speeches are Listed First

- Barack Obama - Inaugural Speech
- Barack Obama - Election Night Victory / Presidential Acceptance Speech - Nov 4 2008
- Barack Obama - Night Before the Election - the Last Rally - Manassas Virginia - Nov 3 2008
- Barack Obama - Democratic Nominee Acceptance Speech

Obama Inaugural Address
20th January 2009

My fellow citizens:

I stand here today humbled by the task before us, grateful for the trust you have bestowed, mindful of the sacrifices borne by our ancestors. I thank President Bush for his service to our nation, as well as the generosity and cooperation he has shown throughout this transition.

Forty-four Americans have now taken the presidential oath. The words have been spoken during rising tides of prosperity and the still waters of peace. Yet, every so often the oath is taken amidst gathering clouds and raging storms. At these moments, America has carried on not simply because of the skill or vision of those in high office, but because We the People have remained faithful to the ideals of our forbearers, and true to our founding documents.

So it has been. So it must be with this generation of Americans.

That we are in the midst of crisis is now well understood. Our nation is at war, against a far-reaching network of violence and hatred. Our economy is badly weakened, a consequence of greed and irresponsibility on the part of some, but also our collective failure to make hard choices and prepare the nation for a new age. Homes have been lost; jobs shed; businesses shuttered. Our health care is too costly; our schools fail too many; and each day brings further evidence that the ways we use energy strengthen our adversaries and threaten our planet.

http://obamaspeeches.com/

WHAT NEXT?

Lecture 14: Sequence to Sequence