8.3

Introduction to Recurrent Neural Networks

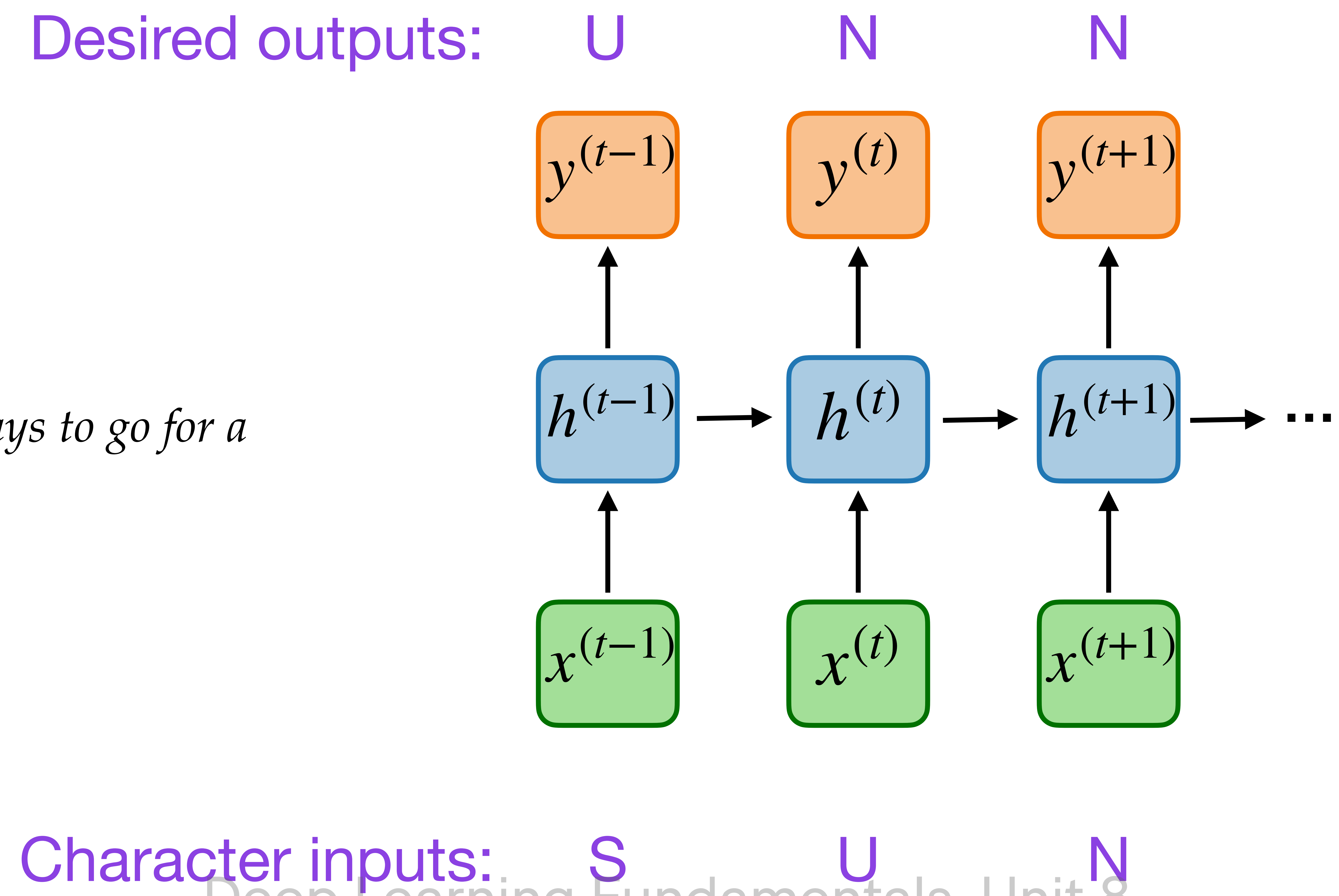
Part 4: Embedding Layers in PyTorch

Sebastian Raschka and the Lightning AI Team

A simple RNN that predicts the next character

Sentence:

“Sunny days are the best days to go for a walk or have a picnic.”



One-hot encoded input letter

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
S	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0

```
[0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0.]
```

One-hot encoded ("sparse") representation of "S U N N Y"

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
S	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
U	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
N	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
N	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
Y	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

Embedded ("dense") representation of "S U N N Y"

```
[[0.9816, 0.7363, 0.5899],
 [0.2605, 0.3766, 0.3502],
 [0.7382, 0.9807, 0.4762],
 [0.6231, 0.8825, 0.8836]]
```

Embedding layer

```
  [[0.6912, 0.8765, 0.4939],
   [0.6342, 0.7481, 0.7717],
   [0.8395, 0.2128, 0.3696],
   [0.4900, 0.1509, 0.0689],
   [0.2587, 0.9171, 0.8670],
   [0.7213, 0.9922, 0.5701],
   [0.7598, 0.5231, 0.3666],
   [0.5150, 0.5216, 0.9682],
   [0.2248, 0.0261, 0.4427],
   [0.1818, 0.6863, 0.8713],
   [0.4192, 0.1566, 0.9004],
   [0.8102, 0.5741, 0.4241],
   [0.1116, 0.0466, 0.2786],
S  [0.9816, 0.7363, 0.5899],
   [0.9224, 0.3672, 0.6972],
   [0.1207, 0.3372, 0.2128],
   [0.0660, 0.1524, 0.8440],
   [0.2162, 0.5640, 0.0988],
U  [0.2605, 0.3766, 0.3502],
   [0.2334, 0.4757, 0.7581],
N  [0.7382, 0.9807, 0.4762],
   [0.2369, 0.8102, 0.8798],
   [0.6932, 0.2671, 0.8018],
   [0.9593, 0.5302, 0.4290],
Y  [0.6231, 0.8825, 0.8836],
   [0.4623, 0.8503, 0.7279]]
```

We learned that embeddings layers are efficient forms of matrix multiplications when working with one-hot encoded vectors

1) Using torch.nn.Embedding

```python
import torch

torch.manual_seed(123);
```

```python
idx = torch.tensor([2, 3, 1]) # 3 training examples
```

```
import torch

torch.manual_seed(123);
```

```
idx = torch.tensor([2, 3, 1]) # 3 training examples

num_idx = max(idx)+1
out_dim = 5
```

Input dimension of a one-hot encoded vector is the number of indices (the highest index + 1)

Suppose we want embeddings of size 5

```python
import torch

torch.manual_seed(123);
```

```python
idx = torch.tensor([2, 3, 1]) # 3 training examples

num_idx = max(idx)+1
out_dim = 5

embedding = torch.nn.Embedding(num_idx, out_dim)
embedding(idx)
```

```
tensor([[ 0.6957, -1.8061, -1.1589,  0.3255, -0.6315],
        [-2.8400, -0.7849, -1.4096, -0.4076,  0.7953],
        [ 1.3010,  1.2753, -0.2010, -0.1606, -0.4015]],
       grad_fn=<EmbeddingBackward0>)
```

Each row is a training example

```python
import torch

torch.manual_seed(123);
```

```python
idx = torch.tensor([2, 3, 1]) # 3 training examples

num_idx = max(idx)+1
out_dim = 5

embedding = torch.nn.Embedding(num_idx, out_dim)
embedding(idx)
```

```
tensor([[ 0.6957, -1.8061, -1.1589,  0.3255, -0.6315],
        [-2.8400, -0.7849, -1.4096, -0.4076,  0.7953],
        [ 1.3010,  1.2753, -0.2010, -0.1606, -0.4015]],
       grad_fn=<EmbeddingBackward0>)
```

Each training example has 5 feature values

Let’s summarize this step by step

```python
import torch

torch.manual_seed(123);
```

```python
idx = torch.tensor([2, 3, 1]) # 3 training examples
```

1) Indices of 3 training examples $\begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix}$

```python
import torch

torch.manual_seed(123);
```

```python
idx = torch.tensor([2, 3, 1]) # 3 training examples

num_idx = max(idx)+1
out_dim = 5

embedding = torch.nn.Embedding(num_idx, out_dim)
```

1) Indices of 3 training examples $\begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix}$

2) Embedding matrix

$$\begin{bmatrix} 0.3374 & -0.1778 & -0.3035 & -0.5880 & 1.5810 \\ 1.3010 & 1.2753 & -0.2010 & -0.1606 & -0.4015 \\ 0.6957 & -1.8061 & -1.1589 & 0.3255 & -0.6315 \\ -2.8400 & -0.7849 & -1.4096 & -0.4076 & 0.7953 \end{bmatrix}$$

```python
import torch

torch.manual_seed(123);
```

```python
idx = torch.tensor([2, 3, 1]) # 3 training examples

num_idx = max(idx)+1
out_dim = 5

embedding = torch.nn.Embedding(num_idx, out_dim)
embedding(idx)
```

```
tensor([[ 0.6957, -1.8061, -1.1589,  0.3255, -0.6315],
        [-2.8400, -0.7849, -1.4096, -0.4076,  0.7953],
        [ 1.3010,  1.2753, -0.2010, -0.1606, -0.4015]],
       grad_fn=<EmbeddingBackward0>)
```

1) Indices of 3 training examples $\begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix}$

2) Embedding matrix

$$\begin{bmatrix} 0.3374 & -0.1778 & -0.3035 & -0.5880 & 1.5810 \\ 1.3010 & 1.2753 & -0.2010 & -0.1606 & -0.4015 \\ 0.6957 & -1.8061 & -1.1589 & 0.3255 & -0.6315 \\ -2.8400 & -0.7849 & -1.4096 & -0.4076 & 0.7953 \end{bmatrix}$$

3) Look up entries by index

$$\begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix} \quad \begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} 0.6957 & -1.8061 & -1.1589 & 0.3255 & -0.6315 \\ -2.8400 & -0.7849 & -1.4096 & -0.4076 & 0.7953 \\ 1.3010 & 1.2753 & -0.2010 & -0.1606 & -0.4015 \end{bmatrix}$$

2) Using torch.nn.Linear

```python
idx = torch.tensor([2, 3, 1]) # 3 training examples

num_idx = max(idx)+1
out_dim = 5

embedding = torch.nn.Embedding(num_idx, out_dim)
embedding(idx)
```

```
tensor([[ 0.6957, -1.8061, -1.1589,  0.3255, -0.6315],
        [-2.8400, -0.7849, -1.4096, -0.4076,  0.7953],
        [ 1.3010,  1.2753, -0.2010, -0.1606, -0.4015]],
       grad_fn=<EmbeddingBackward0>)
```

```python
onehot = torch.nn.functional.one_hot(idx)
linear = torch.nn.Linear(num_idx, 5, bias=False)
linear.weight = torch.nn.Parameter(embedding.weight.T.detach())
```

Transpose required, because nn.Linear multiplies $xW^\top$ instead of xW

```
idx = torch.tensor([2, 3, 1]) # 3 training examples

num_idx = max(idx)+1
out_dim = 5

embedding = torch.nn.Embedding(num_idx, out_dim)
embedding(idx)
```

```
tensor([[ 0.6957, -1.8061, -1.1589,  0.3255, -0.6315],
        [-2.8400, -0.7849, -1.4096, -0.4076,  0.7953],
        [ 1.3010,  1.2753, -0.2010, -0.1606, -0.4015]],
       grad_fn=<EmbeddingBackward0>)
```

```
onehot = torch.nn.functional.one_hot(idx)
linear = torch.nn.Linear(num_idx, 5, bias=False)
linear.weight = torch.nn.Parameter(embedding.weight.T.detach())
```

```
linear(onehot.float())
```

```
tensor([[ 0.6957, -1.8061, -1.1589,  0.3255, -0.6315],
        [-2.8400, -0.7849, -1.4096, -0.4076,  0.7953],
        [ 1.3010,  1.2753, -0.2010, -0.1606, -0.4015]], grad_fn=<MmBackward0>)
```

```python
idx = torch.tensor([2, 3, 1]) # 3 training examples

num_idx = max(idx)+1
out_dim = 5

embedding = torch.nn.Embedding(num_idx, out_dim)
embedding(idx)
```

```
tensor([[ 0.6957, -1.8061, -1.1589,  0.3255, -0.6315],
        [-2.8400, -0.7849, -1.4096, -0.4076,  0.7953],
        [ 1.3010,  1.2753, -0.2010, -0.1606, -0.4015]],
       grad_fn=<EmbeddingBackward0>)
```

```python
onehot = torch.nn.functional.one_hot(idx)
linear = torch.nn.Linear(num_idx, 5, bias=False)
linear.weight = torch.nn.Parameter(embedding.weight.T.detach())
```

```python
linear(onehot.float())
```

```
tensor([[ 0.6957, -1.8061, -1.1589,  0.3255, -0.6315],
        [-2.8400, -0.7849, -1.4096, -0.4076,  0.7953],
        [ 1.3010,  1.2753, -0.2010, -0.1606, -0.4015]], grad_fn=<MmBackward0>)
```

Let’s summarize this step by step

```python
idx = torch.tensor([2, 3, 1]) # 3 training examples

num_idx = max(idx)+1
out_dim = 5

embedding = torch.nn.Embedding(num_idx, out_dim)
embedding(idx)
```

```
tensor([[ 0.6957, -1.8061, -1.1589,  0.3255, -0.6315],
        [-2.8400, -0.7849, -1.4096, -0.4076,  0.7953],
        [ 1.3010,  1.2753, -0.2010, -0.1606, -0.4015]],
       grad_fn=<EmbeddingBackward0>)
```

```python
onehot = torch.nn.functional.one_hot(idx)
```

1) Convert indices of 3 training examples to one-hot encoding

$$\begin{bmatrix}2\\3\\1\end{bmatrix} \longrightarrow \begin{bmatrix}0 & 0 & 1 & 0\\0 & 0 & 0 & 1\\0 & 1 & 0 & 0\end{bmatrix}$$

```python
idx = torch.tensor([2, 3, 1]) # 3 training examples

num_idx = max(idx)+1
out_dim = 5

embedding = torch.nn.Embedding(num_idx, out_dim)
embedding(idx)
```

```
tensor([[ 0.6957, -1.8061, -1.1589,  0.3255, -0.6315],
        [-2.8400, -0.7849, -1.4096, -0.4076,  0.7953],
        [ 1.3010,  1.2753, -0.2010, -0.1606, -0.4015]],
       grad_fn=<EmbeddingBackward0>)
```

```python
onehot = torch.nn.functional.one_hot(idx)
linear = torch.nn.Linear(num_idx, 5, bias=False)
linear.weight = torch.nn.Parameter(embedding.weight.T.detach())
```

1) Convert indices of 3 training examples to one-hot encoding

$$\begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix} \longrightarrow \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

2) Weight (embedding) matrix

$$\begin{bmatrix} 0.3374 & -0.1778 & -0.3035 & -0.5880 & 1.5810 \\ 1.3010 & 1.2753 & -0.2010 & -0.1606 & -0.4015 \\ 0.6957 & -1.8061 & -1.1589 & 0.3255 & -0.6315 \\ -2.8400 & -0.7849 & -1.4096 & -0.4076 & 0.7953 \end{bmatrix}$$

```python
idx = torch.tensor([2, 3, 1]) # 3 training examples

num_idx = max(idx)+1
out_dim = 5

embedding = torch.nn.Embedding(num_idx, out_dim)
embedding(idx)
```

```
tensor([[ 0.6957, -1.8061, -1.1589,  0.3255, -0.6315],
        [-2.8400, -0.7849, -1.4096, -0.4076,  0.7953],
        [ 1.3010,  1.2753, -0.2010, -0.1606, -0.4015]],
       grad_fn=<EmbeddingBackward0>)
```

```python
onehot = torch.nn.functional.one_hot(idx)
linear = torch.nn.Linear(num_idx, 5, bias=False)
linear.weight = torch.nn.Parameter(embedding.weight.T.detach())
```

```python
linear(onehot.float())
```

```
tensor([[ 0.6957, -1.8061, -1.1589,  0.3255, -0.6315],
        [-2.8400, -0.7849, -1.4096, -0.4076,  0.7953],
```

1) Convert indices of 3 training examples to one-hot encoding

$$\begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix} \longrightarrow \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

2) Weight (embedding) matrix

$$\begin{bmatrix} 0.3374 & -0.1778 & -0.3035 & -0.5880 & 1.5810 \\ 1.3010 & 1.2753 & -0.2010 & -0.1606 & -0.4015 \\ 0.6957 & -1.8061 & -1.1589 & 0.3255 & -0.6315 \\ -2.8400 & -0.7849 & -1.4096 & -0.4076 & 0.7953 \end{bmatrix}$$

3) Multiply one-hot encoded inputs with weight matrix

$$\begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0.3374 & -0.1778 & -0.3035 & -0.5880 & 1.5810 \\ 1.3010 & 1.2753 & -0.2010 & -0.1606 & -0.4015 \\ 0.6957 & -1.8061 & -1.1589 & 0.3255 & -0.6315 \\ -2.8400 & -0.7849 & -1.4096 & -0.4076 & 0.7953 \end{bmatrix}$$

$$= \begin{bmatrix} 0.6957 & -1.8061 & -1.1589 & 0.3255 & -0.6315 \\ -2.8400 & -0.7849 & -1.4096 & -0.4076 & 0.7953 \\ 1.3010 & 1.2753 & -0.2010 & -0.1606 & -0.4015 \end{bmatrix}$$

$$\begin{aligned} & 0 \times 0.3374 \\ & + 0 \times 1.3010 \\ & + 1 \times 0.6957 \\ & + 0 \times -2.8400 \\ & = 0.6957 \end{aligned}$$

Next: RNNs with Attentions and Transformers