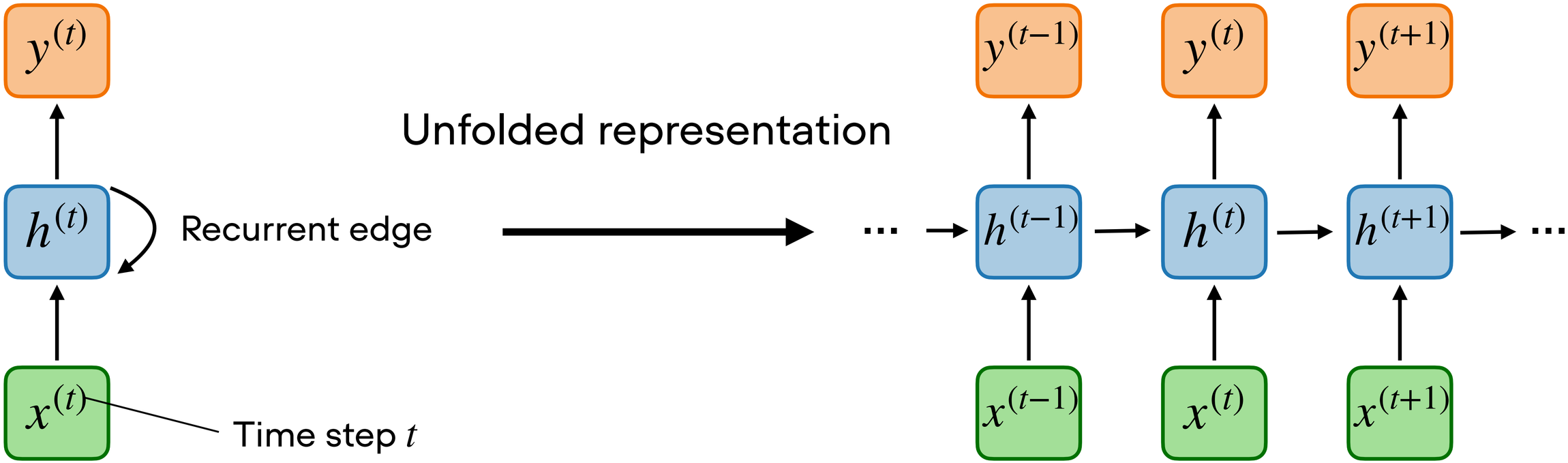
# 8.3

# Introduction to Recurrent Neural Networks

## Part 3: Encoding Inputs Using Embedding Layers

Sebastian Raschka and the Lightning AI Team

# Recurrent neural networks (RNNS) for modeling sequences




Recurrent neural network (RNN)

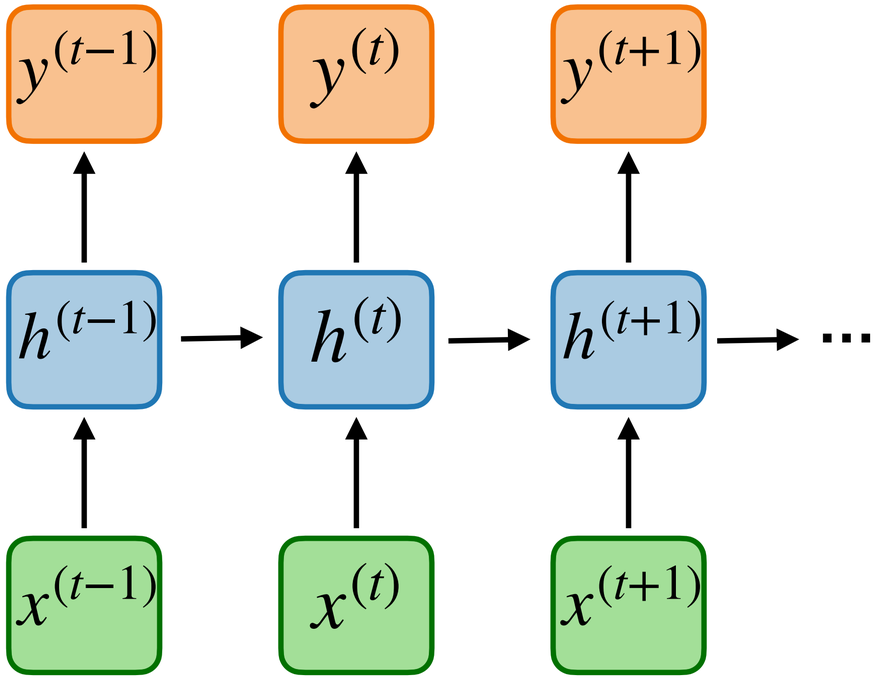
The same RNN

Suppose we implement a simple RNN that predicts the next character in a sentence:

*"Sunny days are the best days to go for a walk or have a picnic."*

# A simple RNN that predicts the next character

Desired outputs: U N N




Sentence:

*"Sunny days are the best days to go for a walk or have a picnic."*

Character inputs: S U N

# Remember one-hot encoding?

| | 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 |

# 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 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.]
```

## Weight matrix

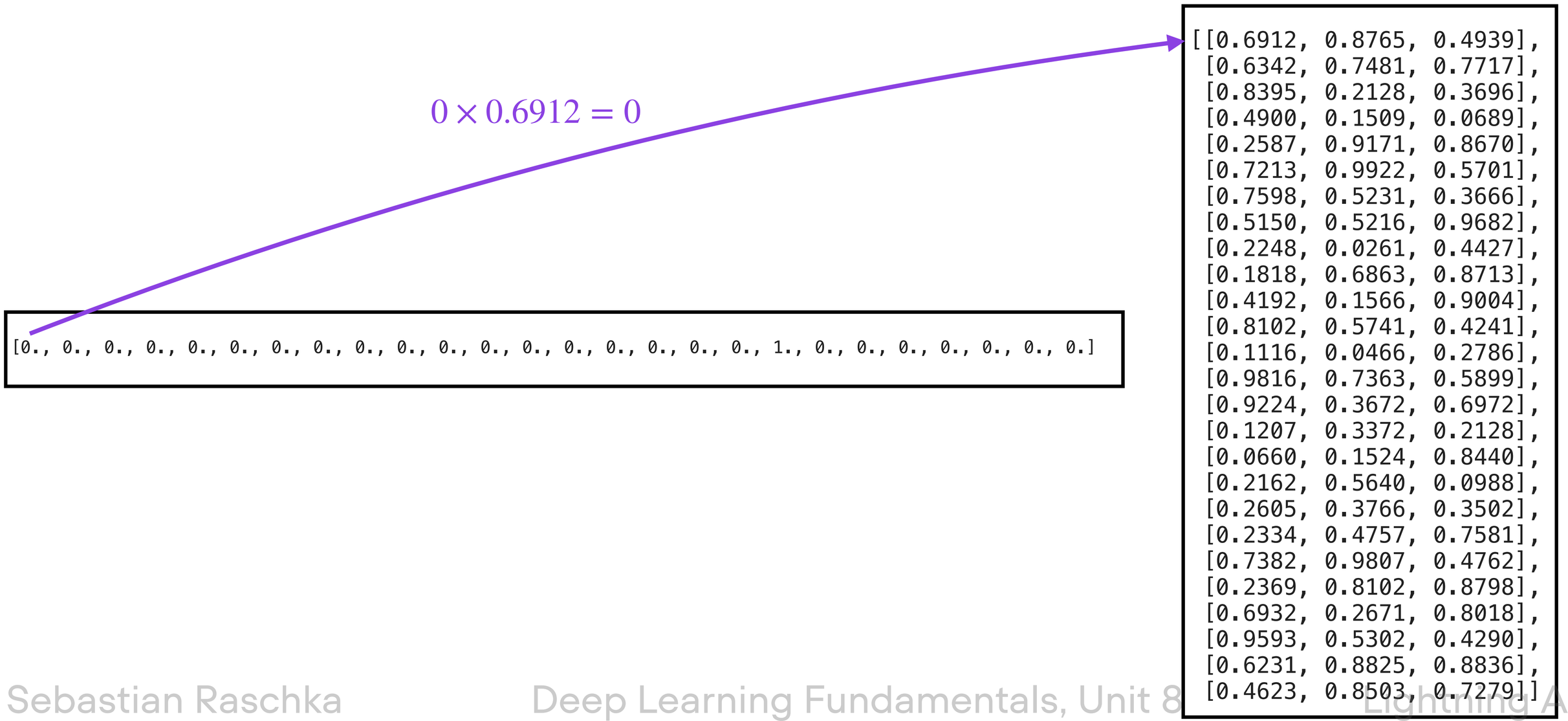
```
[[0.6912, 0.8765, 0.4939],
 [0.6342, 0.7481, 0.7717],
 [0.8395, 0.2128, 0.3696],
 [0.4900, 0.1509, 0.0689],
 [0.2587, 0.9171, 0.8670],
 [0.7213, 0.9922, 0.5701],
 [0.7598, 0.5231, 0.3666],
 [0.5150, 0.5216, 0.9682],
 [0.2248, 0.0261, 0.4427],
 [0.1818, 0.6863, 0.8713],
 [0.4192, 0.1566, 0.9004],
 [0.8102, 0.5741, 0.4241],
 [0.1116, 0.0466, 0.2786],
 [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],
 [0.2605, 0.3766, 0.3502],
 [0.2334, 0.4757, 0.7581],
 [0.7382, 0.9807, 0.4762],
 [0.2369, 0.8102, 0.8798],
 [0.6932, 0.2671, 0.8018],
 [0.9593, 0.5302, 0.4290],
 [0.6231, 0.8825, 0.8836],
 [0.4623, 0.8503, 0.7279]]
```

# Multiply input vector with weight matrix

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

```
[[0.6912, 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],
 [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],
 [0.2605, 0.3766, 0.3502],
 [0.2334, 0.4757, 0.7581],
 [0.7382, 0.9807, 0.4762],
 [0.2369, 0.8102, 0.8798],
 [0.6932, 0.2671, 0.8018],
 [0.9593, 0.5302, 0.4290],
 [0.6231, 0.8825, 0.8836],
 [0.4623, 0.8503, 0.7279]]
```

# Multiply input vector with weight matrix



# Multiply input vector with weight matrix

$0 \times 0.6342 = 0$

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

```
[[0.6912, 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],
 [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],
 [0.2605, 0.3766, 0.3502],
 [0.2334, 0.4757, 0.7581],
 [0.7382, 0.9807, 0.4762],
 [0.2369, 0.8102, 0.8798],
 [0.6932, 0.2671, 0.8018],
 [0.9593, 0.5302, 0.4290],
 [0.6231, 0.8825, 0.8836],
 [0.4623, 0.8503, 0.7279]]
```

# Multiply input vector with weight matrix

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

$1 \times 0.2605 = 0.2605$

```
[[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],
 [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],
 [0.2605, 0.3766, 0.3502],
 [0.2334, 0.4757, 0.7581],
 [0.7382, 0.9807, 0.4762],
 [0.2369, 0.8102, 0.8798],
 [0.6932, 0.2671, 0.8018],
 [0.9593, 0.5302, 0.4290],
 [0.6231, 0.8825, 0.8836],
 [0.4623, 0.8503, 0.7279]]
```

# Multiply input vector with weight matrix

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

$1 \times 0.3766 = 0.3766$

```
[[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],
 [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],
 [0.2605, 0.3766, 0.3502],
 [0.2334, 0.4757, 0.7581],
 [0.7382, 0.9807, 0.4762],
 [0.2369, 0.8102, 0.8798],
 [0.6932, 0.2671, 0.8018],
 [0.9593, 0.5302, 0.4290],
 [0.6231, 0.8825, 0.8836],
 [0.4623, 0.8503, 0.7279]]
```

# Multiply input vector with weight matrix

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

$1 \times 0.3502 = 0.3502$

```
[[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],
 [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],
 [0.2605, 0.3766, 0.3502],
 [0.2334, 0.4757, 0.7581],
 [0.7382, 0.9807, 0.4762],
 [0.2369, 0.8102, 0.8798],
 [0.6932, 0.2671, 0.8018],
 [0.9593, 0.5302, 0.4290],
 [0.6231, 0.8825, 0.8836],
 [0.4623, 0.8503, 0.7279]]
```

# Multiply input vector with weight matrix

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

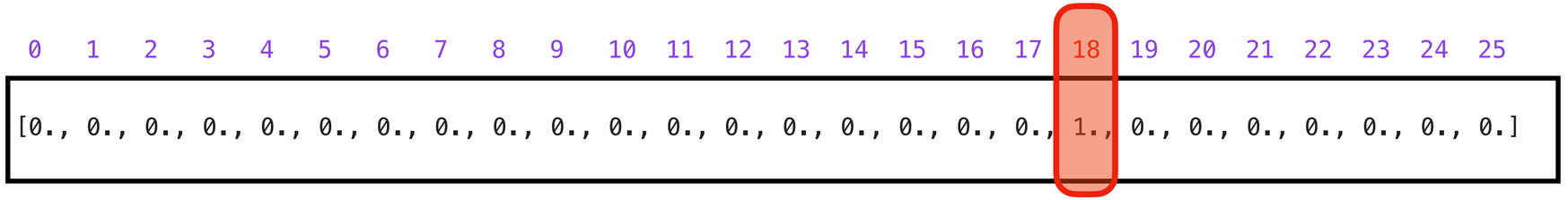
```
[[0.6912, 0.8765, 0.4939],
 [0.6342, 0.7481, 0.7717],
 [0.8395, 0.2128, 0.3696],
 [0.4900, 0.1509, 0.0689],
 [0.2587, 0.9171, 0.8670],
 [0.7213, 0.9922, 0.5701],
 [0.7598, 0.5231, 0.3666],
 [0.5150, 0.5216, 0.9682],
 [0.2248, 0.0261, 0.4427],
 [0.1818, 0.6863, 0.8713],
 [0.4192, 0.1566, 0.9004],
 [0.8102, 0.5741, 0.4241],
 [0.1116, 0.0466, 0.2786],
 [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],
 [0.2605, 0.3766, 0.3502],
 [0.2334, 0.4757, 0.7581],
 [0.7382, 0.9807, 0.4762],
 [0.2369, 0.8102, 0.8798],
 [0.6932, 0.2671, 0.8018],
 [0.9593, 0.5302, 0.4290],
 [0.6231, 0.8825, 0.8836],
 [0.4623, 0.8503, 0.7279]]
```

Result of the vector matrix multiplication:

```
[0.2605, 0.3766, 0.3502]
```

# This matrix multiplication is very inefficient compared to an index-based lookup

# Replace multiplication between one-hot encoded vector and matrix with index look-up




```
0  [[0.6912, 0.8765, 0.4939],
1   [0.6342, 0.7481, 0.7717],
2   [0.8395, 0.2128, 0.3696],
3   [0.4900, 0.1509, 0.0689],
4   [0.2587, 0.9171, 0.8670],
5   [0.7213, 0.9922, 0.5701],
6   [0.7598, 0.5231, 0.3666],
7   [0.5150, 0.5216, 0.9682],
8   [0.2248, 0.0261, 0.4427],
9   [0.1818, 0.6863, 0.8713],
10  [0.4192, 0.1566, 0.9004],
11  [0.8102, 0.5741, 0.4241],
12  [0.1116, 0.0466, 0.2786],
13  [0.9816, 0.7363, 0.5899],
14  [0.9224, 0.3672, 0.6972],
15  [0.1207, 0.3372, 0.2128],
16  [0.0660, 0.1524, 0.8440],
17  [0.2162, 0.5640, 0.0988],
18  [0.2605, 0.3766, 0.3502],
19  [0.2334, 0.4757, 0.7581],
20  [0.7382, 0.9807, 0.4762],
21  [0.2369, 0.8102, 0.8798],
22  [0.6932, 0.2671, 0.8018],
23  [0.9593, 0.5302, 0.4290],
24  [0.6231, 0.8825, 0.8836],
25  [0.4623, 0.8503, 0.7279]]
```

Result of the vector matrix multiplication:

```
[0.2605, 0.3766, 0.3502]
```

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]]
```

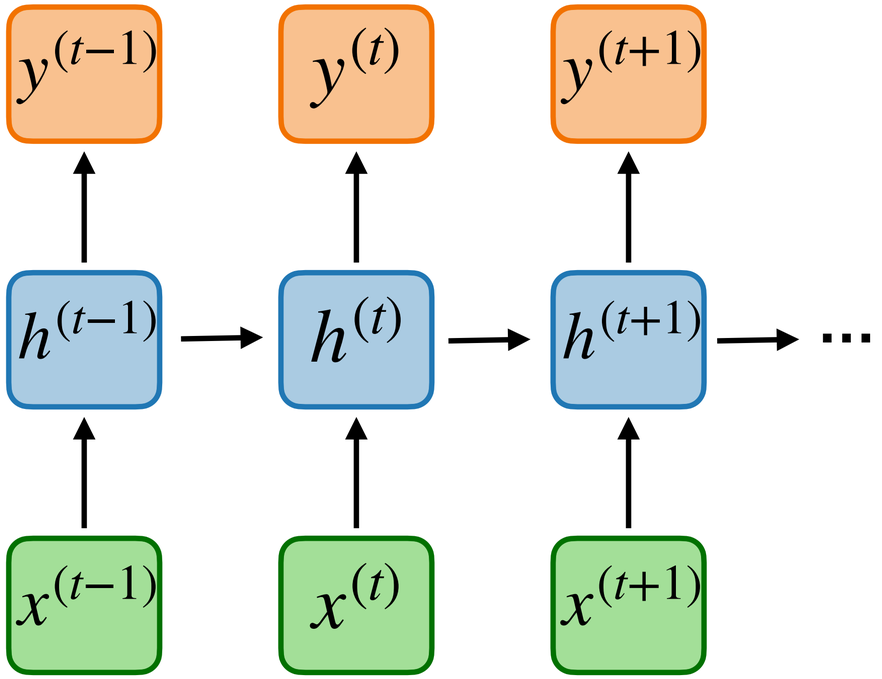
Embedding layer

```
[[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]]
```

# Coming back to the character-level RNN ...

# A simple RNN that predicts the next character

Desired outputs: U N N

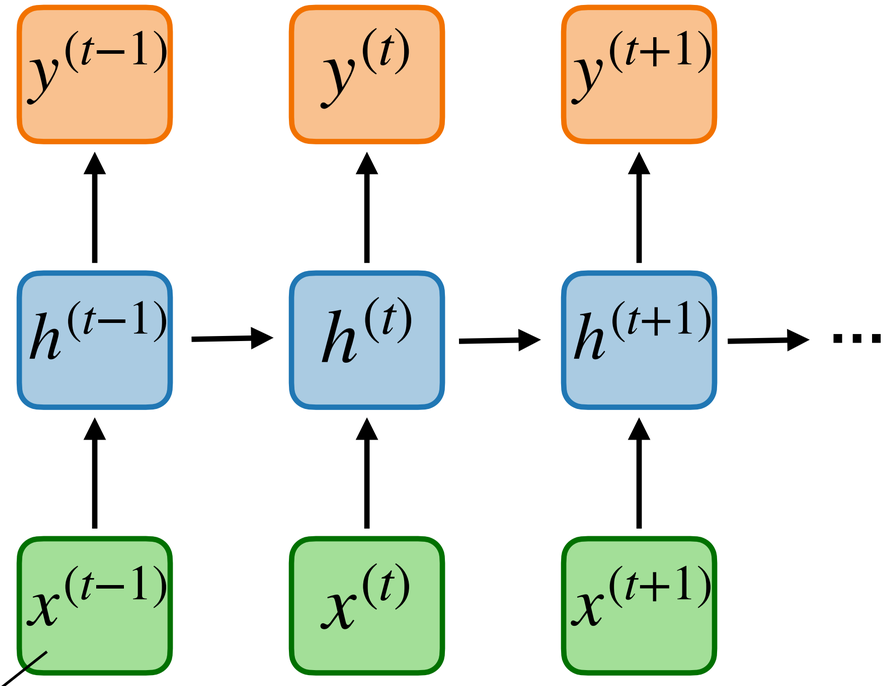



Sentence:

*"Sunny days are the best days to go for a walk or have a picnic."*

Character inputs: S U N

Desired outputs: U N N




a) feed embedded inputs

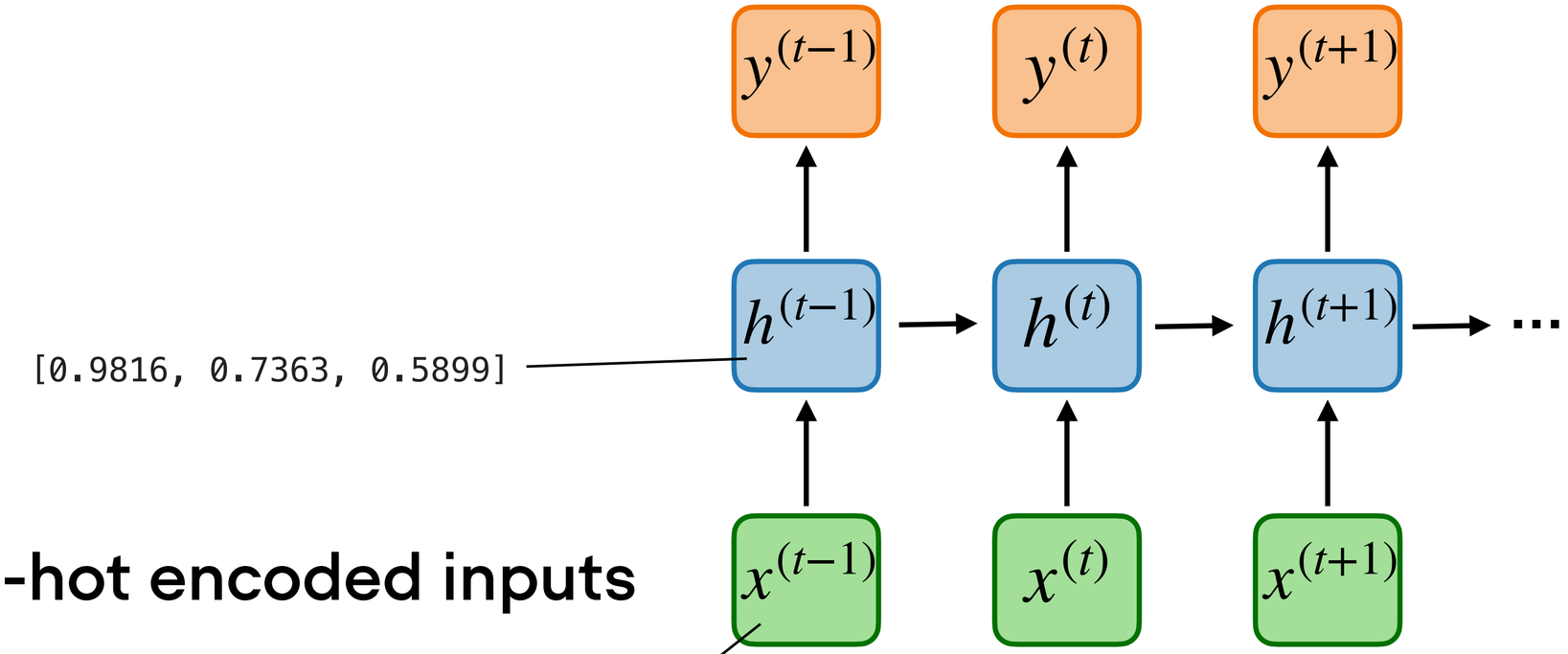
```
[0.9816, 0.7363, 0.5899]
```

Character inputs: S U N

Feed one-hot encoded inputs and let the hidden layer do the embedding.

Optimal embedding is “learned” during training.

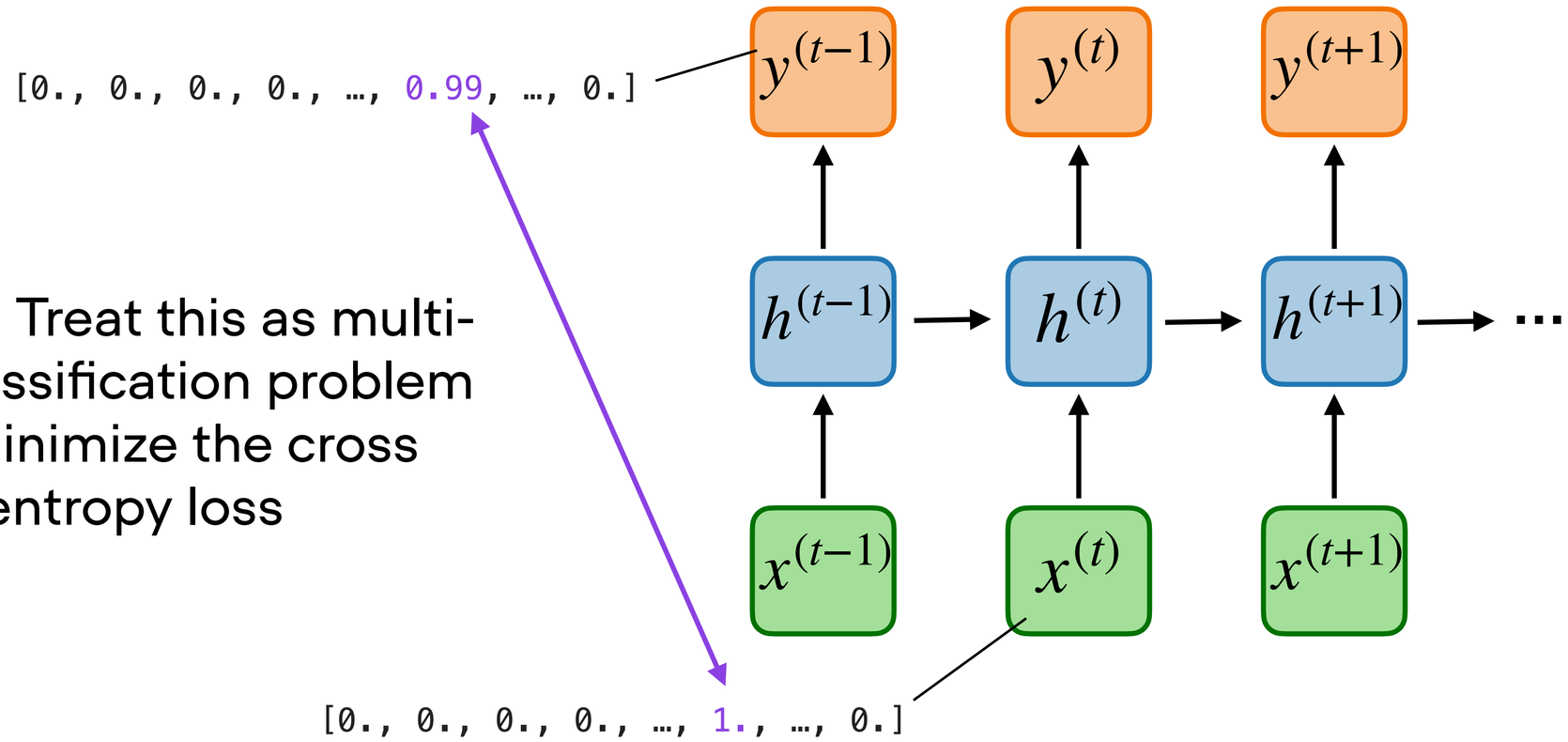
Desired outputs: U N N




b) feed one-hot encoded inputs

Character inputs: S U N

Desired outputs: U N N




Training: Treat this as multi-class classification problem and minimize the cross entropy loss

Character inputs: S U N

# Word vs character embeddings

# Advantages / disadvantages of character embeddings vs. word embeddings

+ require less memory (English: 26 letters + punctuation)

+ Smaller output layers

- Can create nonsense words

- Worse at capturing long-distance dependencies

# Next: PyTorch Embedding Layer Example