Precept 5: Feedforward NNs, RNNs

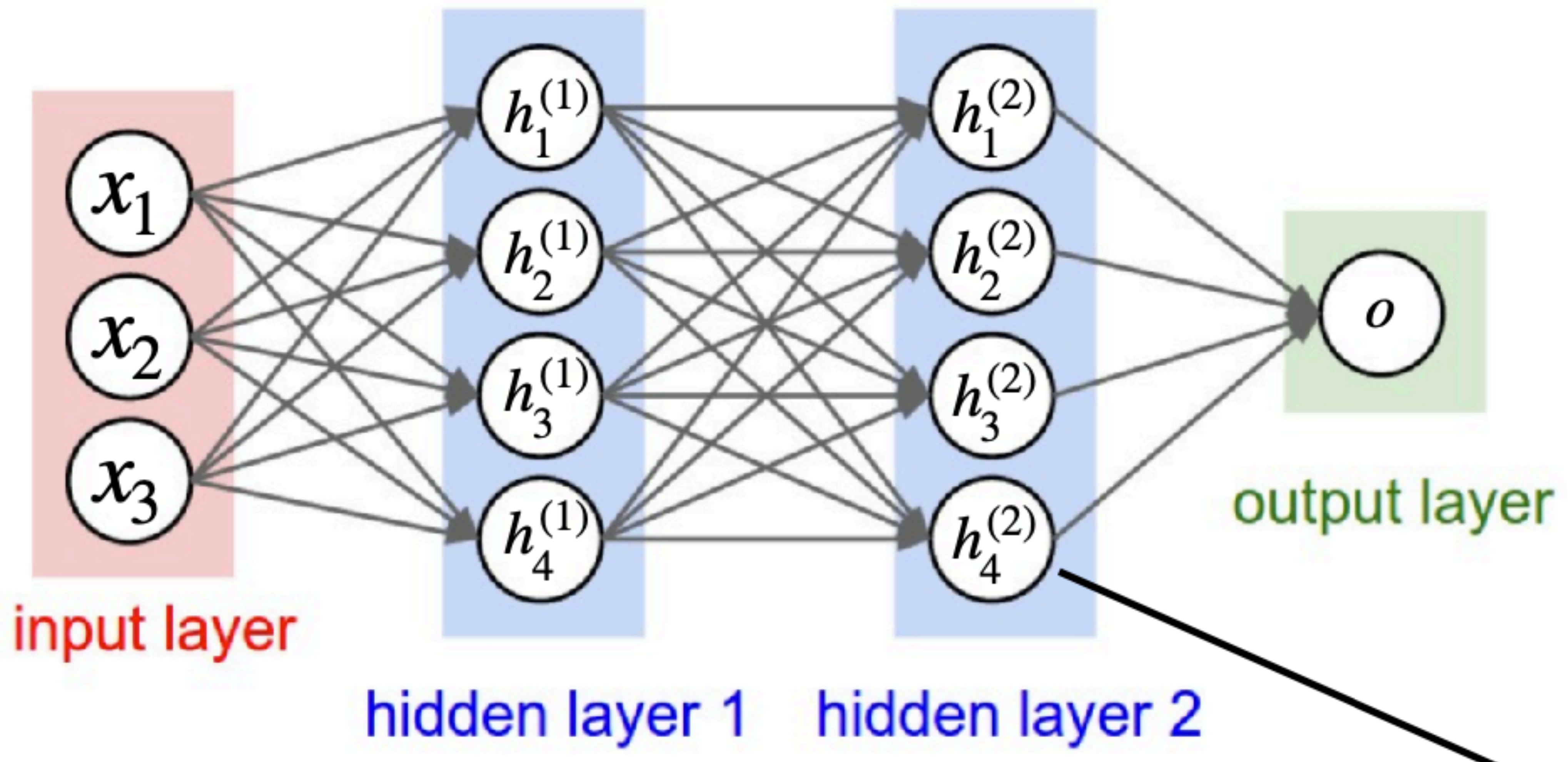
COS 484

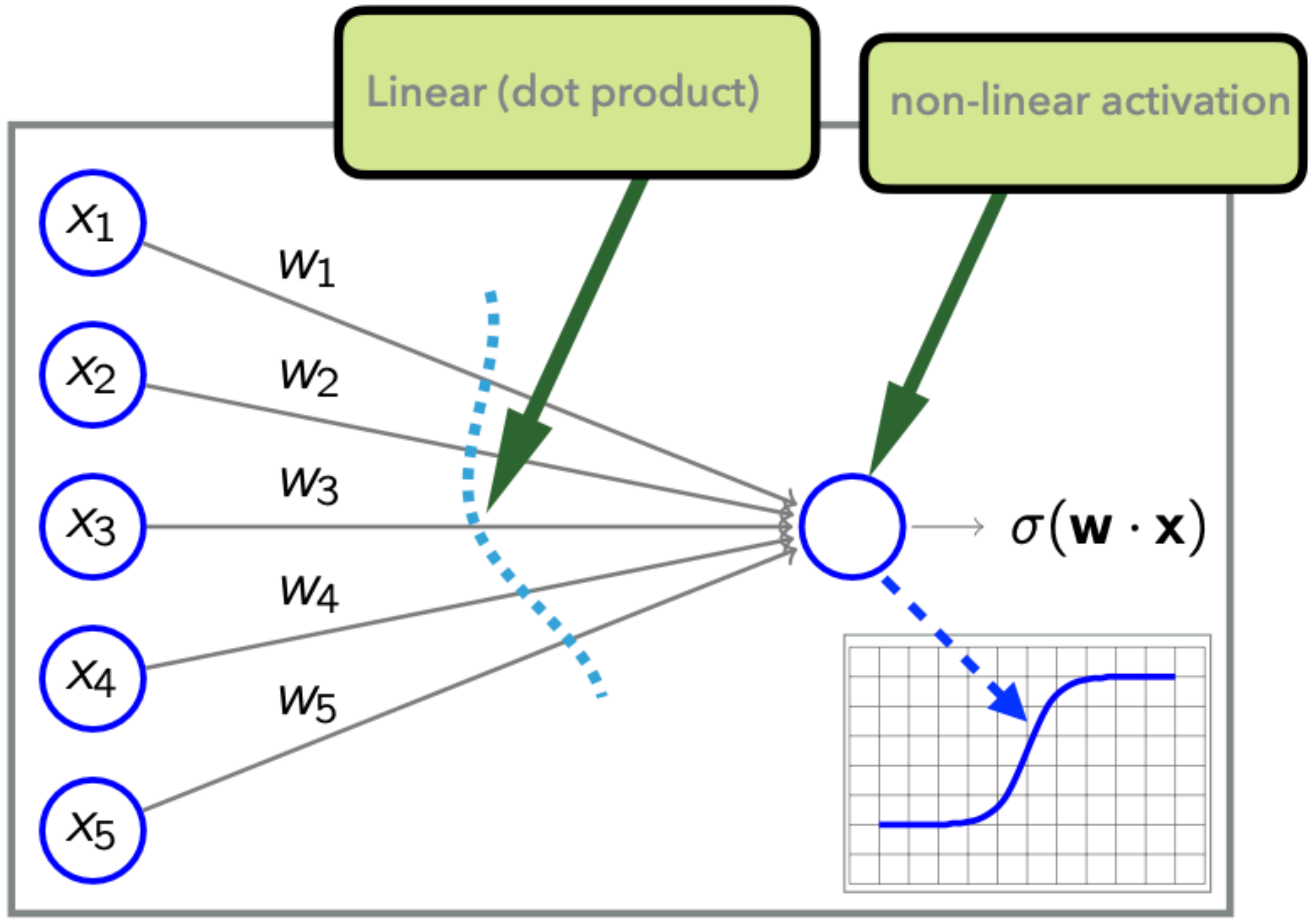
Tyler Zhu (figures adapted from Samyak Gupta)

Today's Plan

1. Feedforward Neural Networks for NLP
2. Recurrent Neural Networks
3. Q&A for Midterm?

Feedforward Networks



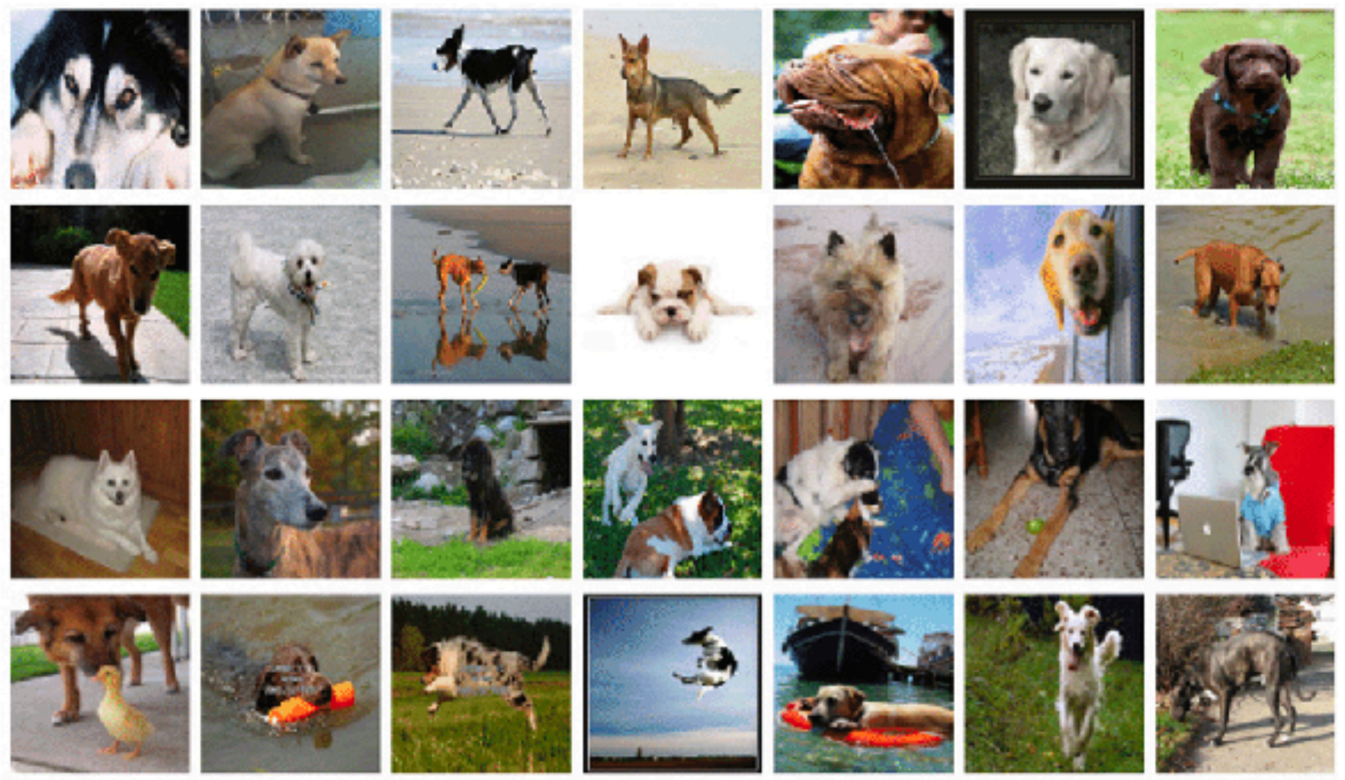


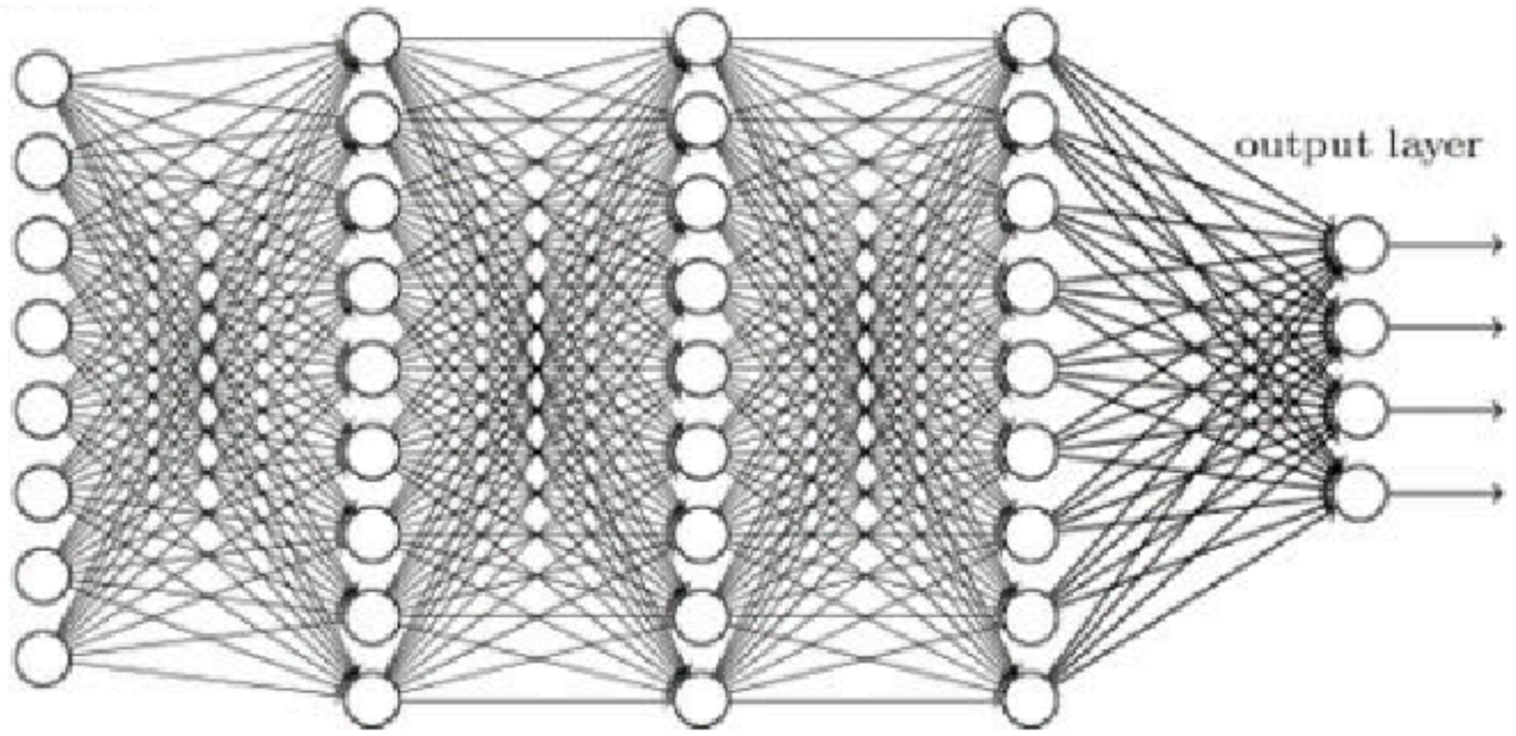
$$h_1^{(1)} = f(w_{1,1}^{(1)}x_1 + w_{1,2}^{(1)}x_2 + w_{1,3}^{(1)}x_3)$$

$$h_3^{(2)} = f(w_{3,1}^{(2)}h_1^{(1)} + w_{3,2}^{(2)}h_2^{(1)} + w_{3,3}^{(2)}h_3^{(1)} + w_{3,4}^{(2)}h_4^{(1)})$$

non-linearity f: σ, $\tanh$ or ReLU.

Using NNs for images vs. text





label = “dog”

a sometimes tedious film

i had to look away - this was god awful .

a gorgeous , witty , seductive movie .

label = positive

- Images: fixed-size input, continuous values
- Text: **variable-length** input, discrete words
  - need to convert into vectors - word embeddings!

Feedforward Neural Language Models

- Key idea: Instead of estimating raw probabilities, let's use a **neural network** to fit the **probabilistic distribution** of language!

$$P(w \mid \text{I am a good})$$

$$P(w \mid \text{I am a great})$$

- Allows us to move past naive Markov assumptions with more **powerful** models!
- Helps to have good word embeddings so that e(good) ~ e(great) (similar contexts)
  - Otherwise the distribution to learn becomes very noisy and sparse!

Feedforward Neural Language Models

- Feedforward neural language models approximate the probability based on the previous *m* (e.g., 5) words - *m* is a hyper-parameter!

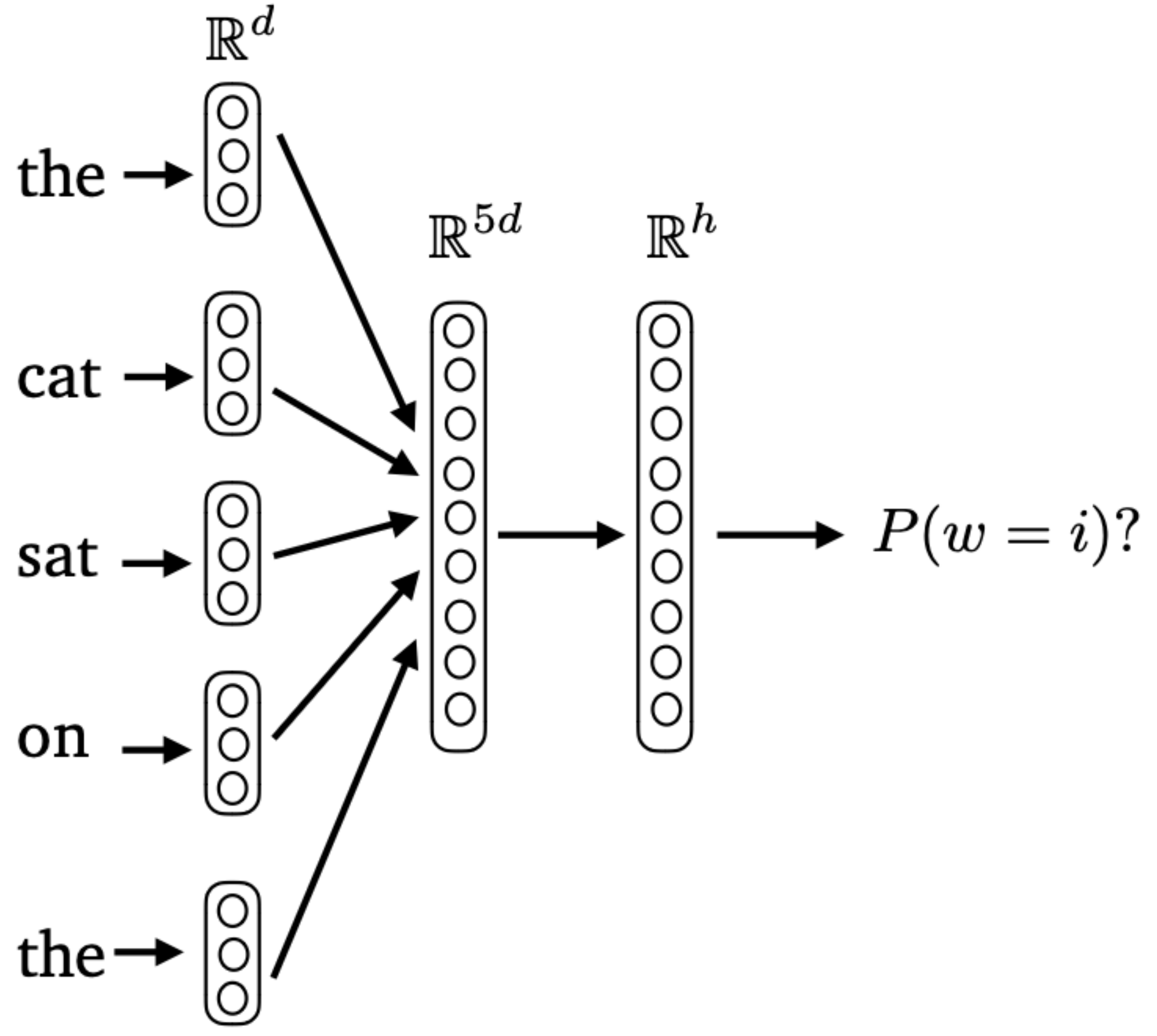
$$P(x_1, x_2, \ldots, x_n) \approx \prod_{i=1}^{n} P(x_i \mid x_{i-m+1}, \ldots, x_{i-1})$$

P(mat | the cat sat on the) = ?

d: word embedding size

h: hidden size

It is a |V|-way classification problem!



Feedforward Neural Language Models

P(mat | the cat sat on the) = ? d: word embedding size h: hidden size

- Input layer (m= 5): Q: why concat instead of taking the average?

  $\mathbf{x} = [e(\text{the}); e(\text{cat}); e(\text{sat}); e(\text{on}); e(\text{the})] \in \mathbb{R}^{md}$

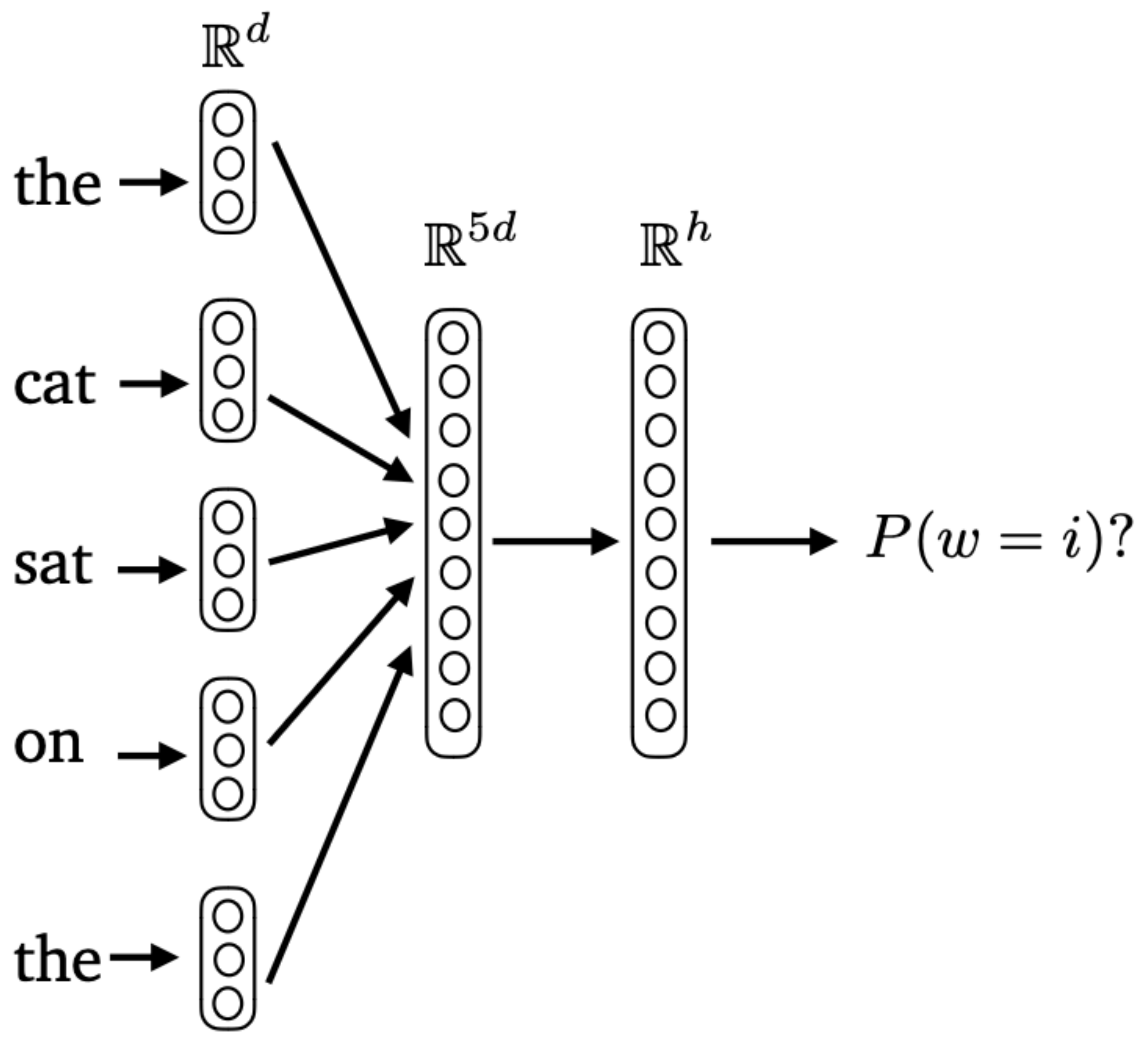
- Hidden layer:

  $$\mathbf{h} = \tanh(\mathbf{W}\mathbf{x} + \mathbf{b}) \in \mathbb{R}^h$$

- Output layer

  $$\mathbf{z} = \mathbf{U}\mathbf{h} \in \mathbb{R}^{|V|}$$

  $$P(w = i \mid \text{the cat sat on the}) = \text{softmax}_i(\mathbf{z}) = \frac{e^{z_i}}{\sum_k e^{z_k}}$$



What are the dimensions of **W** and **U**?

Feedforward Neural Language Models

P(mat | the cat sat on the) = ?   d: word embedding size   h: hidden size

- Input layer (m= 5):

Q: why concat instead of taking the average?

$$\mathbf{x} = [e(\text{the}); e(\text{cat}); e(\text{sat}); e(\text{on}); e(\text{the})] \in \mathbb{R}^{md}$$

- Hidden layer:

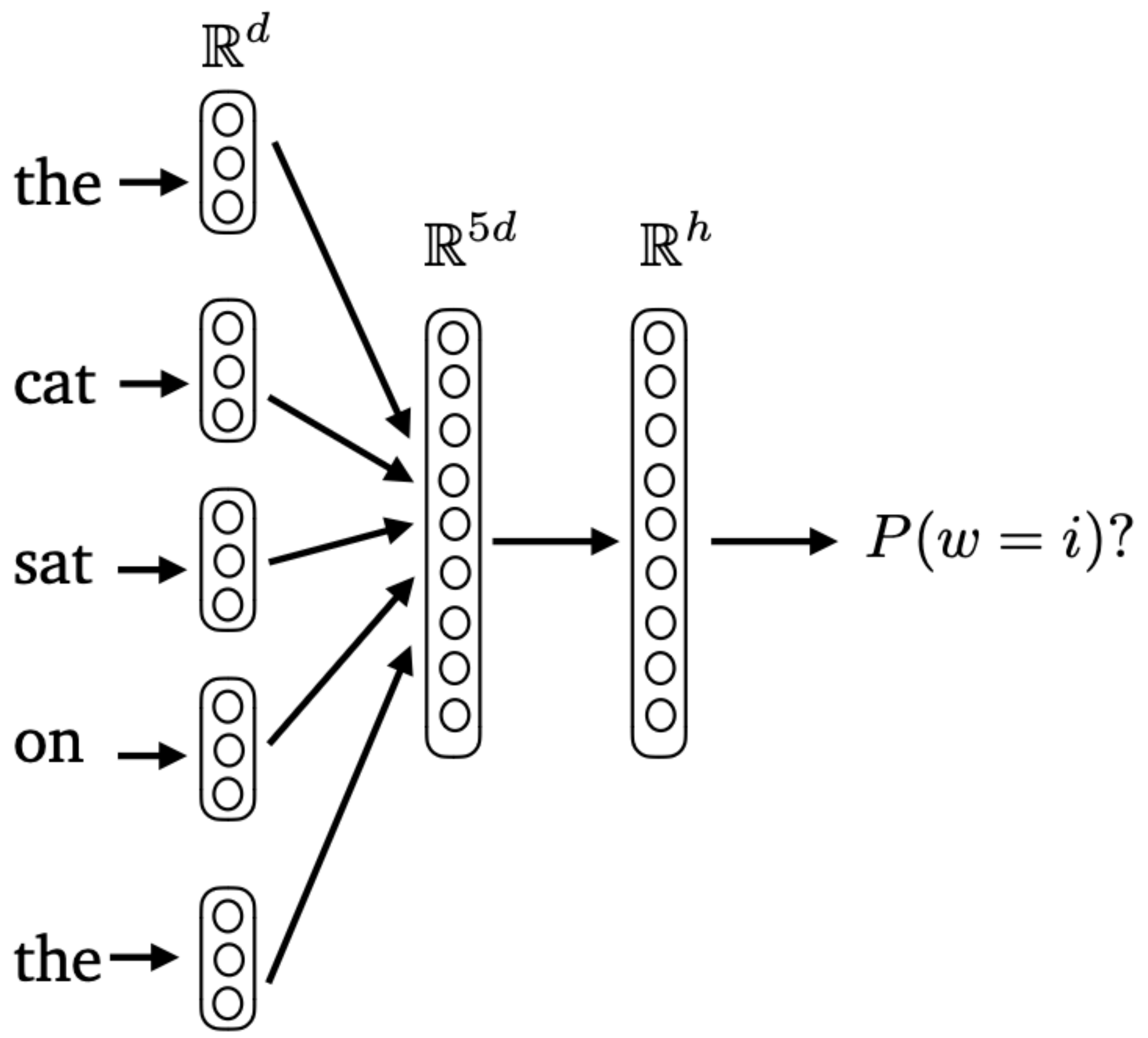
$$\mathbf{h} = \tanh(\mathbf{W}\mathbf{x} + \mathbf{b}) \in \mathbb{R}^h$$

- Output layer

$$\mathbf{z} = \mathbf{U}\mathbf{h} \in \mathbb{R}^{|V|}$$

$$P(w = i \mid \text{the cat sat on the})$$

$$= \text{softmax}_i(\mathbf{z}) = \frac{e^{z_i}}{\sum_k e^{z_k}}$$



What are the dimensions of **W** and **U**? $\mathbf{W} \in \mathbb{R}^{h \times 5d}$, $\mathbf{U} \in \mathbf{R}^{|V| \times h}$

Training FF Neural Language Models

- How to train this model? A: Use a lot of raw text to create training examples and run gradient-descent optimization!

The Fat Cat Sat on the Mat is a 1996 children's book by Nurit Karlin. Published by Harper Collins as part of the reading readiness program, the book stresses the ability to read words of specific structure, such as -at.

→

the fat cat sat on → the
fat cat sat on the → mat
cat sat on the mat → is
sat on the mat is → a
…

- Limitations?
  - $\mathbf{W}$ **linearly** scales with the context size m
  - The models learns separate patterns for different positions!
- Better solutions: recurrent NNs, Transformers..

the fat cat sat on → the
fat cat sat on the → mat
cat sat on the mat → is

“sat on” corresponds to different parameters in $\mathbf{W}$

Example through Code

- Walk through a programmatic example with window size 3 (m=3)
- See how the data is created as well

- Input layer (m= 5):

$$\mathbf{x} = [e(\text{the}); e(\text{cat}); e(\text{sat}); e(\text{on}); e(\text{the})] \in \mathbb{R}^{md}$$

- Hidden layer:

$$\mathbf{h} = \tanh(\mathbf{W}\mathbf{x} + \mathbf{b}) \in \mathbb{R}^{h}$$

- Output layer

$$\mathbf{z} = \mathbf{U}\mathbf{h} \in \mathbb{R}^{|V|}$$

$$P(w = i \mid \text{the cat sat on the})$$

$$= \text{softmax}_i(\mathbf{z}) = \frac{e^{z_i}}{\sum_k e^{z_k}}$$

First step: The data

- Take the same input sentence as before

“The fat cat sat on the mat.”

- We use simple 1-hot vectors to embed

- To create training data, we make

(word 1, word 2, word 3) —> (next word)

examples which make our labeled pairs

```
text = ["the", "fat", "cat", "sat", "on", "the", "mat"]

embed = {
    "the": [1, 0, 0, 0, 0, 0],
    "fat": [0, 1, 0, 0, 0, 0],
    "cat": [0, 0, 1, 0, 0, 0],
    "sat": [0, 0, 0, 1, 0, 0],
    "on": [0, 0, 0, 0, 1, 0],
    "mat": [0, 0, 0, 0, 0, 1]
}
```

```
# Simple Feedforward Neural Language Model
# Window size m = 3
import numpy as np

training_examples = [
    (text[i:i+3], text[i+3]) for i in range(0, len(text) - 3)
]
print("(x, y) labeled pairs")
for x, y in training_examples:
    print(f"({x}, {y})")
```

✓ 0.9s

```
(x, y) labeled pairs
(['the', 'fat', 'cat'], sat)
(['fat', 'cat', 'sat'], on)
(['cat', 'sat', 'on'], the)
(['sat', 'on', 'the'], mat)
```

Part 2: Initializing Weights

- We need to initialize our weights
- Dimensions are $\mathbf{W} \in \mathbb{R}^{h \times 3d}$, $\mathbf{U} \in \mathbf{R}^{|V| \times h}$

```python
# Initialize W to [h x 3d], b to [h], U to [|V| x h]
W = np.random.randn(4, 3*6)
b = np.random.randn(4, 1)
U = np.random.randn(6, 4)
```

- Input layer (m= 5):

$$\mathbf{x} = [e(\text{the}); e(\text{cat}); e(\text{sat}); e(\text{on}); e(\text{the})] \in \mathbb{R}^{md}$$

- Hidden layer:

$$\mathbf{h} = \tanh(\mathbf{W}\mathbf{x} + \mathbf{b}) \in \mathbb{R}^{h}$$

- Output layer

$$\mathbf{z} = \mathbf{U}\mathbf{h} \in \mathbb{R}^{|V|}$$

$$P(w = i \mid \text{the cat sat on the}) = \text{softmax}_i(\mathbf{z}) = \frac{e^{z_i}}{\sum_k e^{z_k}}$$

Part 3: Forward Pass

- Forward pass is what you expect:
  - Embed each word into a vector first with our embedding lookup
  - Concatenate all of the embeddings
  - "Neural Network": $h = \tanh(\mathbf{W}\mathbf{x} + \mathbf{b})$
  - Final logistic regression for a prediction

```
for i, (x_words, y) in enumerate(training_examples):
    print(f"Training example {i}: {x_words} -> {y}")
    x = np.array([embed[word] for word in x_words])
    x = x.reshape(-1, 1)

    h = np.tanh(W @ x + b)
    z = U @ h
    y_hat = np.argmax(softmax(z))
```

```
Training example 0: ['the', 'fat', 'cat'] -> sat
x: [1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0]
h = tanh(Wx + b):[-0.91  1.   -0.6  -0.79]
z = U @ h:[-2.04  0.73  0.79 -1.41 -2.1   0.77]
y_hat = argmax softmax(z): 2 (cat)
y = sat
```

Part 3: Forward Pass

- Forward pass is what you expect:
  - Embed each word into a vector first with our embedding lookup
  - Concatenate all of the embeddings
  - “Neural Network”: $h = \tanh(\mathbf{Wx} + \mathbf{b})$
  - Final logistic regression for a prediction

```python
for i, (x_words, y) in enumerate(training_examples):
    print(f"Training example {i}: {x_words} -> {y}")
    x = np.array([embed[word] for word in x_words])
    x = x.reshape(-1, 1)

    h = np.tanh(W @ x + b)
    z = U @ h
    y_hat = np.argmax(softmax(z))
```

```
Training example 0: ['the', 'fat', 'cat'] -> sat
x: [1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0]
h = tanh(Wx + b):[-0.91  1.   -0.6  -0.79]
z = U @ h:[-2.04  0.73  0.79 -1.41 -2.1   0.77]
y_hat = argmax softmax(z): 2 (cat)
y = sat
--------------------------------------------------
Training example 1: ['fat', 'cat', 'sat'] -> on
x: [0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0]
h = tanh(Wx + b):[ 1.    0.53 -0.72 -0.91]
z = U @ h:[ 1.88 -1.43 -2.15 -4.35  1.95 -1.38]
y_hat = argmax softmax(z): 4 (on)
y = on
```

Part 3: Forward Pass

- Forward pass is what you expect:
  - Embed each word into a vector first with our embedding lookup
  - Concatenate all of the embeddings
  - “Neural Network”: $h = \tanh(\mathbf{Wx} + \mathbf{b})$
  - Final logistic regression for a prediction

```
for i, (x_words, y) in enumerate(training_examples):
    print(f"Training example {i}: {x_words} -> {y}")
    x = np.array([embed[word] for word in x_words])
    x = x.reshape(-1, 1)

    h = np.tanh(W @ x + b)
    z = U @ h
    y_hat = np.argmax(softmax(z))
```

```
Training example 0: ['the', 'fat', 'cat'] -> sat
x: [1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0]
h = tanh(Wx + b):[-0.91  1.   -0.6  -0.79]
z = U @ h:[-2.04  0.73  0.79 -1.41 -2.1   0.77]
y_hat = argmax softmax(z): 2 (cat)
y = sat
--------------------------------------------------
Training example 1: ['fat', 'cat', 'sat'] -> on
x: [0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0]
h = tanh(Wx + b):[ 1.    0.53 -0.72 -0.91]
z = U @ h:[ 1.88 -1.43 -2.15 -4.35  1.95 -1.38]
y_hat = argmax softmax(z): 4 (on)
y = on
--------------------------------------------------
Training example 2: ['cat', 'sat', 'on'] -> the
x: [0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0]
h = tanh(Wx + b):[ 0.95  1.   -0.94 -0.99]
z = U @ h:[ 1.68 -1.14 -2.04 -4.82  1.42 -1.24]
y_hat = argmax softmax(z): 0 (the)
y = the
```

Part 3: Forward Pass

- Forward pass is what you expect:
  - Embed each word into a vector first with our embedding lookup
  - Concatenate all of the embeddings
  - “Neural Network”: $h = \tanh(\mathbf{Wx} + \mathbf{b})$
  - Final logistic regression for a prediction

```python
for i, (x_words, y) in enumerate(training_examples):
    print(f"Training example {i}: {x_words} -> {y}")
    x = np.array([embed[word] for word in x_words])
    x = x.reshape(-1, 1)

    h = np.tanh(W @ x + b)
    z = U @ h
    y_hat = np.argmax(softmax(z))
```

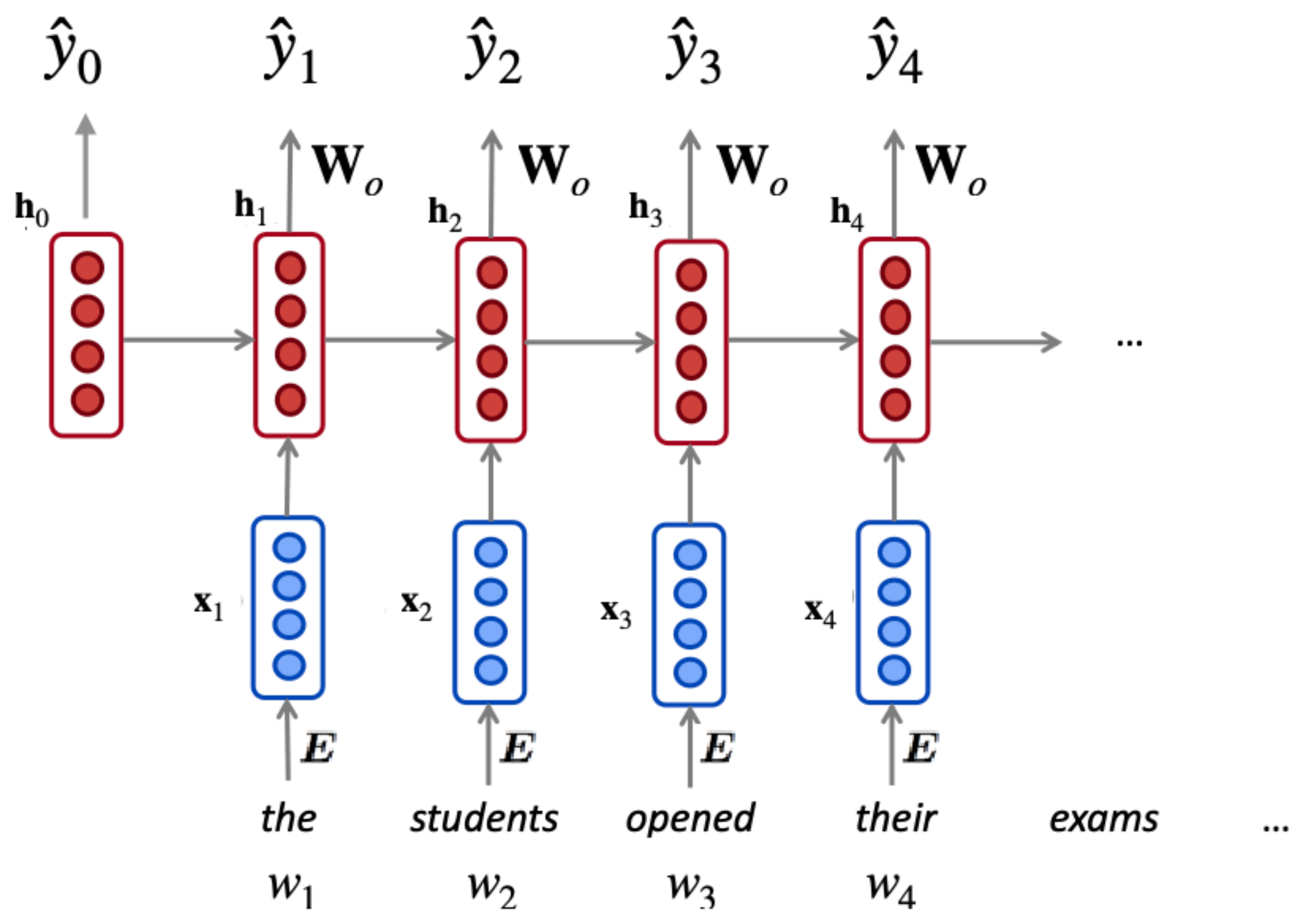
```
Training example 0: ['the', 'fat', 'cat'] -> sat
x: [1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0]
h = tanh(Wx + b):[-0.91  1.   -0.6  -0.79]
z = U @ h:[-2.04  0.73  0.79 -1.41 -2.1   0.77]
y_hat = argmax softmax(z): 2 (cat)
y = sat
------------------------------------------------
Training example 1: ['fat', 'cat', 'sat'] -> on
x: [0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0]
h = tanh(Wx + b):[ 1.    0.53 -0.72 -0.91]
z = U @ h:[ 1.88 -1.43 -2.15 -4.35  1.95 -1.38]
y_hat = argmax softmax(z): 4 (on)
y = on
------------------------------------------------
Training example 2: ['cat', 'sat', 'on'] -> the
x: [0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0]
h = tanh(Wx + b):[ 0.95  1.   -0.94 -0.99]
z = U @ h:[ 1.68 -1.14 -2.04 -4.82  1.42 -1.24]
y_hat = argmax softmax(z): 0 (the)
y = the
------------------------------------------------
Training example 3: ['sat', 'on', 'the'] -> mat
x: [0 0 0 1 0 0 0 0 0 0 1 0 1 0 0 0 0 0]
h = tanh(Wx + b):[ 0.47  0.95  0.96 -1.  ]
z = U @ h:[ 2.78 -0.25  0.88 -4.52 -1.24 -0.39]
y_hat = argmax softmax(z): 0 (the)
y = mat
```

Part 4: Training the Model

- Actually… this is your homework! :-)

Recurrent Neural Networks

- A family of neural networks that can handle **variable length inputs**
- Crucially, can learn the same pattern for different positions (efficient!)




```
Training example 2: ['cat', 'sat', 'on'] -> the
x: [0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0]
h = tanh(Wx + b):[ 0.95  1.   -0.94 -0.99]
z = U @ h:[ 1.68 -1.14 -2.04 -4.82  1.42 -1.24]
y_hat = argmax softmax(z): 0 (the)
y = the
------------------------------------------------
Training example 3: ['sat', 'on', 'the'] -> mat
x: [0 0 0 1 0 0 0 0 0 0 1 0 1 0 0 0 0 0]
h = tanh(Wx + b):[ 0.47  0.95  0.96 -1.  ]
z = U @ h:[ 2.78 -0.25  0.88 -4.52 -1.24 -0.39]
y_hat = argmax softmax(z): 0 (the)
y = mat
```

A simple RNN

A function: $\mathbf{y} = \text{RNN}(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n) \in \mathbb{R}^h$ where $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$

$\mathbf{h}_0 \in \mathbb{R}^h$ is an initial state

$$\boxed{\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t) \in \mathbb{R}^h}$$

$\mathbf{h}_t$: hidden states which store information from $\mathbf{x}_1$ to $\mathbf{x}_t$

Simple RNNs:

$$\boxed{\mathbf{h}_t = g(\mathbf{W}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t + \mathbf{b}) \in \mathbb{R}^h}$$

g: nonlinearity (e.g. tanh, ReLU),

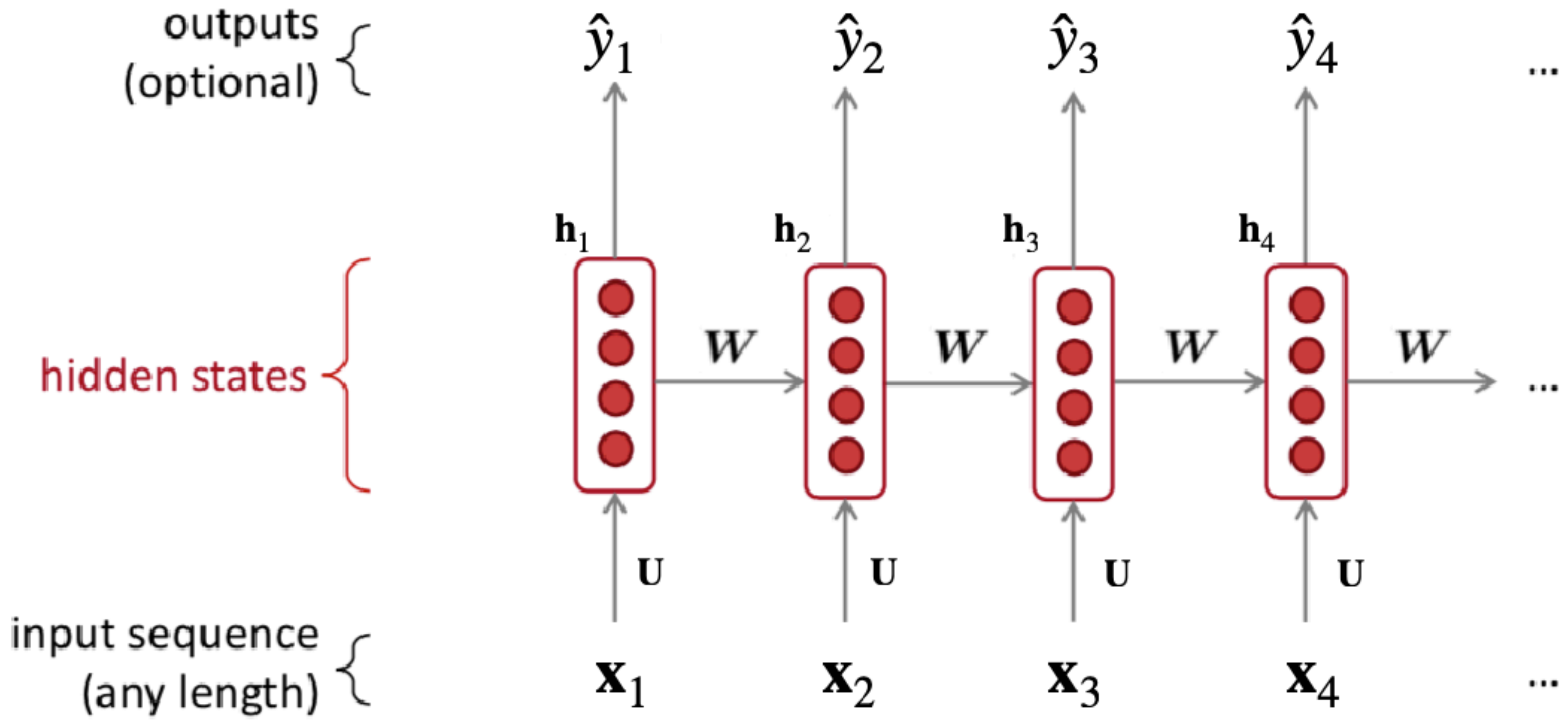
$$\mathbf{W} \in \mathbb{R}^{h \times h}, \mathbf{U} \in \mathbb{R}^{h \times d}, \mathbf{b} \in \mathbb{R}^h$$

This model contains $h \times (h + d + 1)$ parameters, and optionally *h for* $\mathbf{h}_0$ (a common way is just to set $\mathbf{h}_0$ as $\mathbf{0}$)

A simple RNN

$$\mathbf{h}_t = g(\mathbf{W}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t + \mathbf{b}) \in \mathbb{R}^h$$

Key idea: apply the same weights $\mathbf{W}, \mathbf{U}, \mathbf{b}$ repeatedly



Backpropagation through Time

Since $\mathbf{W}$ is used to compute at each timestep, we must "unroll" the loss through time

$$\mathbf{h}_1 = g(\mathbf{W}\mathbf{h}_0 + \mathbf{U}\mathbf{x}_1 + \mathbf{b})$$
$$\mathbf{h}_2 = g(\mathbf{W}\mathbf{h}_1 + \mathbf{U}\mathbf{x}_2 + \mathbf{b})$$
$$\mathbf{h}_3 = g(\mathbf{W}\mathbf{h}_2 + \mathbf{U}\mathbf{x}_3 + \mathbf{b}) \qquad \hat{\mathbf{y}}_3 = \mathrm{softmax}(\mathbf{W}_o\mathbf{h}_3)$$

$$L_3 = -\log \hat{\mathbf{y}}_3(w_4)$$

First, compute gradient with respect to hidden vector of last time step: $\frac{\partial L_3}{\partial \mathbf{h}_3}$

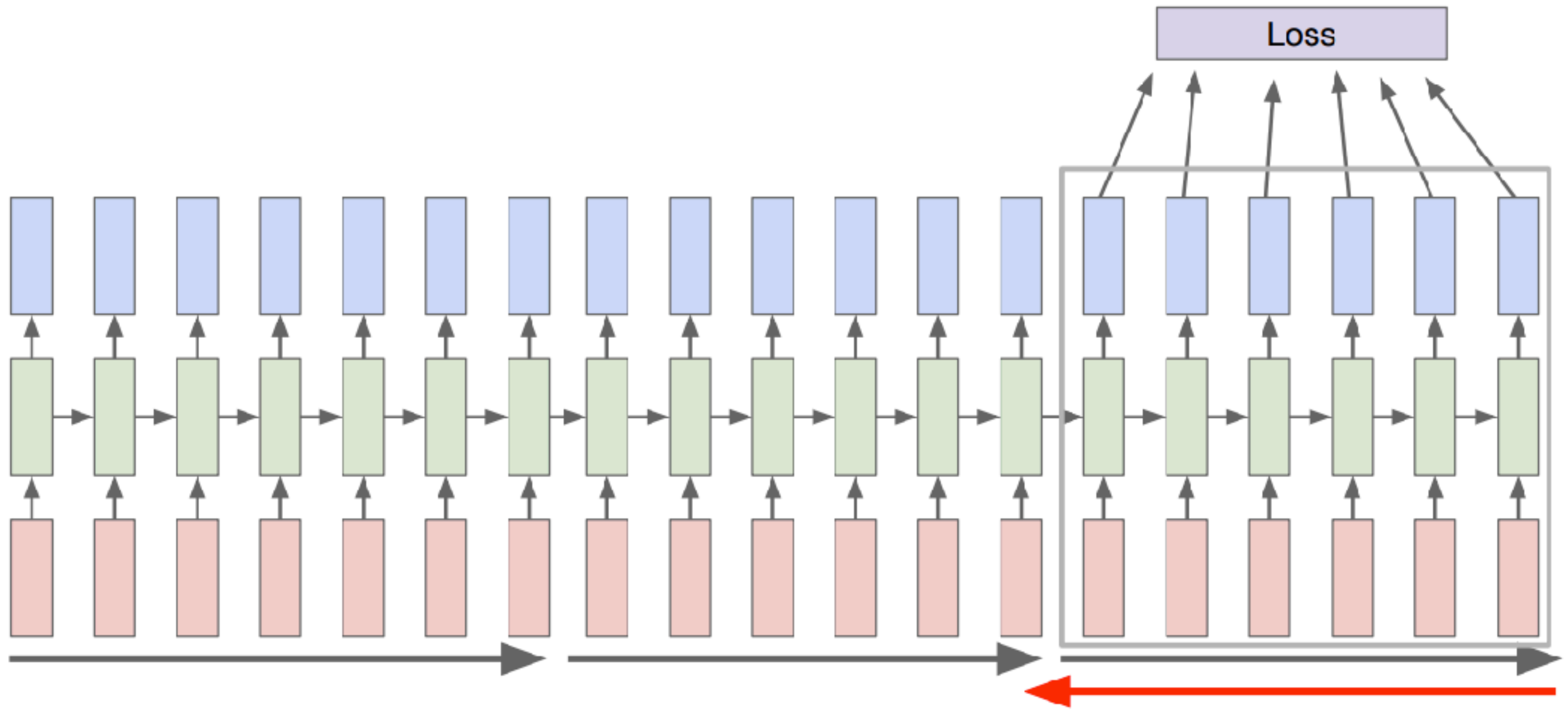
$$\frac{\partial L_3}{\partial \mathbf{W}} = \frac{\partial L_3}{\partial \mathbf{h}_3}\frac{\partial \mathbf{h}_3}{\partial \mathbf{W}} + \frac{\partial L_3}{\partial \mathbf{h}_3}\frac{\partial \mathbf{h}_3}{\partial \mathbf{h}_2}\frac{\partial \mathbf{h}_2}{\partial \mathbf{W}} + \frac{\partial L_3}{\partial \mathbf{h}_3}\frac{\partial \mathbf{h}_3}{\partial \mathbf{h}_2}\frac{\partial \mathbf{h}_2}{\partial \mathbf{h}_1}\frac{\partial \mathbf{h}_1}{\partial \mathbf{W}}$$

More generally, $\boxed{\frac{\partial L}{\partial \mathbf{W}} = -\frac{1}{n}\sum_{t=1}^{n}\sum_{k=1}^{t}\frac{\partial L_t}{\partial \mathbf{h}_t}\left(\prod_{j=k+1}^{t}\frac{\partial \mathbf{h}_j}{\partial \mathbf{h}_{j-1}}\right)\frac{\partial \mathbf{h}_k}{\partial \mathbf{W}}}$

If k and t are far away, the gradients can grow/shrink exponentially
(called the gradient exploding or gradient vanishing problem)

Truncated Backpropagation through Time

- Backpropagation is very expensive if you handle long sequences



- Run forward and backward through chunks of the sequence instead of whole sequence
- Carry hidden states forward in time forever, but only back-propagate for some smaller number of steps

RNN Tradeoffs

- Can handle arbitrary length inputs
- Reuse weights to reduce total model parameters (esp. w/ weight tying)
- Suffers from vanishing/exploding gradients
- Not very hardware friendly to train/deploy

Vanishing/Exploding Gradients

Two common reasons for why gradient issues arise:

1. The choice of activation function
2. Weight initialization

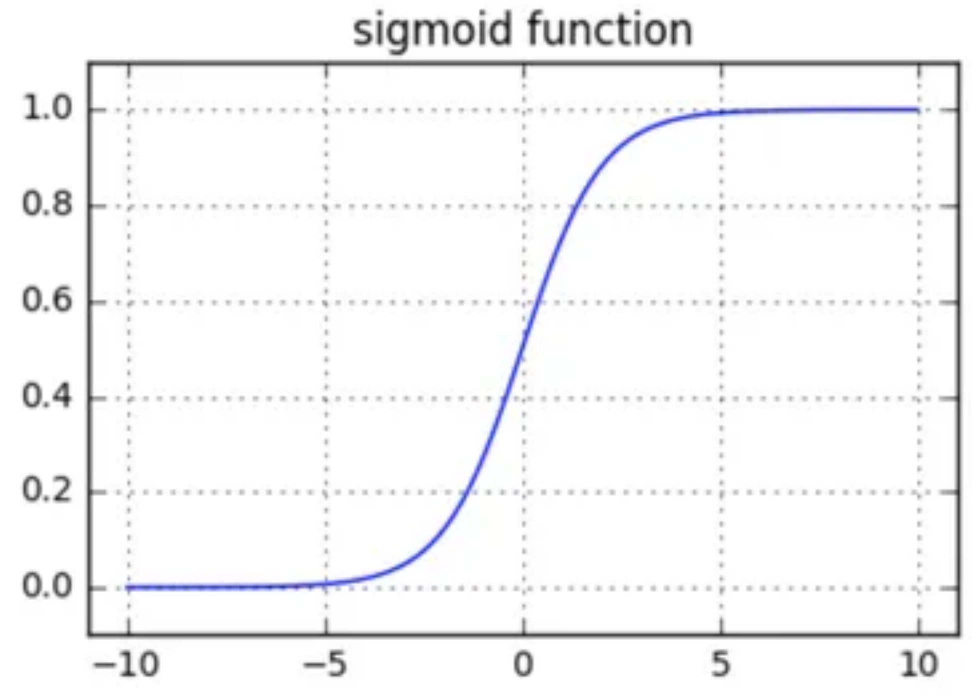
https://karpathy.medium.com/yes-you-should-understand-backprop-e2f06eab496b

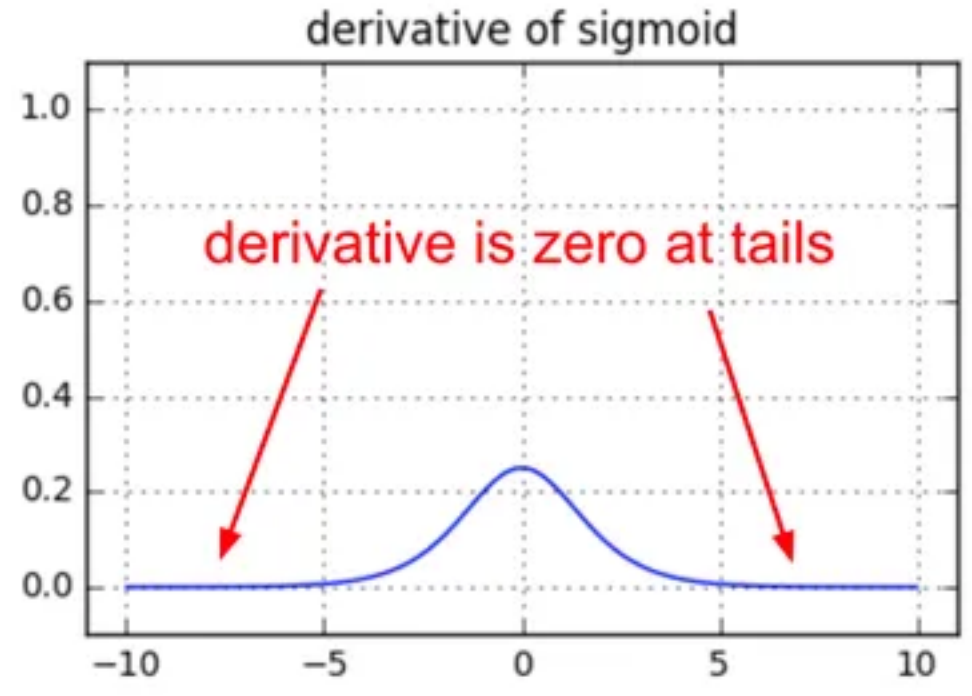
Vanishing/Exploding Gradients

Two common reasons for why gradient issues arise:

1. The choice of activation function
2. Weight initialization

What happens to the overall gradient signal for sigmoid?





https://karpathy.medium.com/yes-you-should-understand-backprop-e2f06eab496b

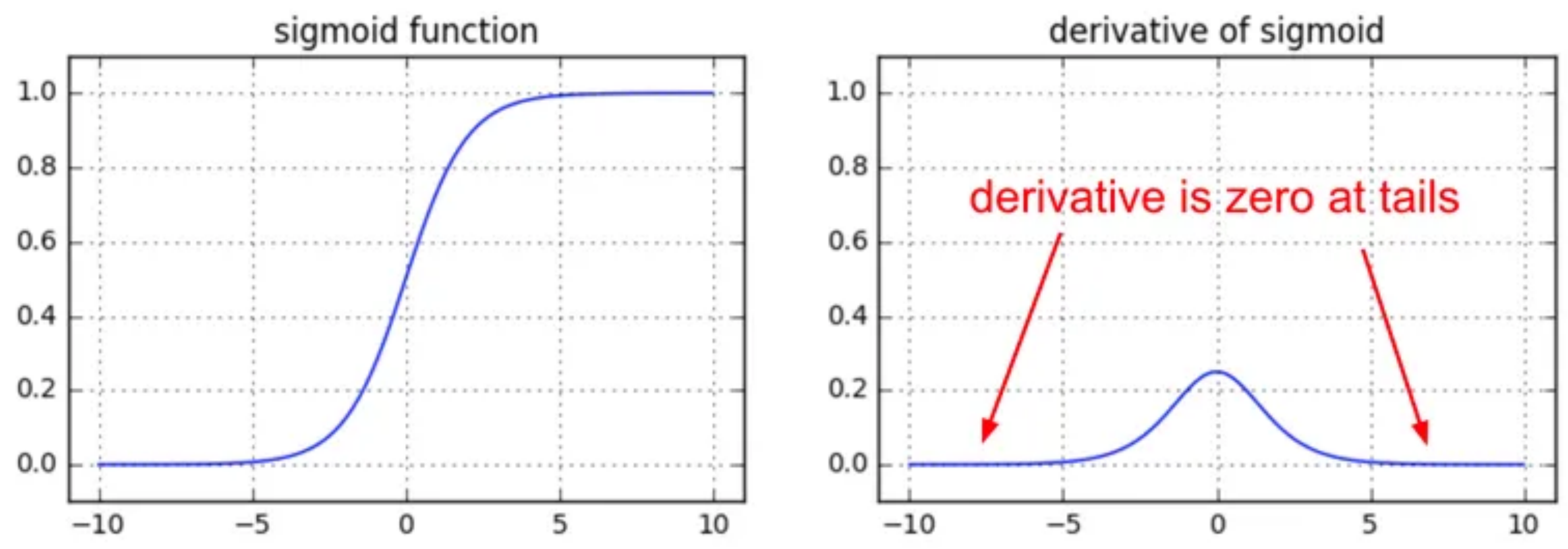
Vanishing/Exploding Gradients

Two common reasons for why gradient issues arise:

1. The choice of activation function
2. Weight initialization

What happens to the overall gradient signal for sigmoid?

1. Easily dies outside of small support ([-2,2])
2. Norm always decreases! $\sigma(0) = 0.25$, so gradient magnitude decreases vastly for early layers. Hence they learn slower…



https://karpathy.medium.com/yes-you-should-understand-backprop-e2f06eab496b

Vanishing/Exploding Gradients

Two common reasons for why gradient issues arise:

1. The choice of activation function

$$\mathbf{h}_t = g(\mathbf{W}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t + \mathbf{b}) \in \mathbb{R}^h$$

2. Weight initialization

For simplicity, say $\mathbf{x}, \mathbf{b} = 0$ and g is identity, so $h_t = \mathbf{W}h_{t-1}$.

1. What is h_t in terms of h_0?
2. What is $\dfrac{\partial h_t}{\partial h_0}$? When is this an issue?

https://karpathy.medium.com/yes-you-should-understand-backprop-e2f06eab496b

Vanishing/Exploding Gradients

Two common reasons for why gradient issues arise:

1. The choice of activation function

$$\mathbf{h}_t = g(\mathbf{W}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t + \mathbf{b}) \in \mathbb{R}^h$$

2. Weight initialization

For simplicity, say $\mathbf{x}, \mathbf{b} = 0$ and g is identity, so $h_t = \mathbf{W}h_{t-1}$.

1. What is h_t in terms of h_0? $h_t = \mathbf{W}^t h_0$
2. What is $\frac{\partial h_t}{\partial h_0}$? When is this an issue?

https://karpathy.medium.com/yes-you-should-understand-backprop-e2f06eab496b

Vanishing/Exploding Gradients

Two common reasons for why gradient issues arise:

1. The choice of activation function

$$\mathbf{h}_t = g(\mathbf{W}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t + \mathbf{b}) \in \mathbb{R}^h$$

2. Weight initialization

For simplicity, say $\mathbf{x}, \mathbf{b} = 0$ and g is identity, so $h_t = \mathbf{W}h_{t-1}$.

1. What is h_t in terms of h_0? $h_t = \mathbf{W}^t h_0$

2. What is $\frac{\partial h_t}{\partial h_0}$? When is this an issue? $\mathbf{W}^t$, which grows unbounded when $\lambda_{\max} > 1$ and to 0 when $\lambda_{\max} < 1$

https://karpathy.medium.com/yes-you-should-understand-backprop-e2f06eab496b

RNN Code Example: Data

$$\mathbf{h}_t = g(\mathbf{W}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t + \mathbf{b}) \in \mathbb{R}^h$$

- Same training text as before
- Unlike before now, our data pairs form **one example** together, not multiple ones.
  - We predict on all output possibilities at once (“teacher forcing”, more later)

```python
text = ["the", "fat", "cat", "sat", "on", "the", "mat"]

embed = {
    "the": [1, 0, 0, 0, 0, 0],
    "fat": [0, 1, 0, 0, 0, 0],
    "cat": [0, 0, 1, 0, 0, 0],
    "sat": [0, 0, 0, 1, 0, 0],
    "on": [0, 0, 0, 0, 1, 0],
    "mat": [0, 0, 0, 0, 0, 1]
}
```

```python
training_examples = [
    (text[0:i+1], text[i+1]) for i in range(0, len(text) - 1)
]
print("(x, y) labeled pairs")
for x, y in training_examples:
    print(f"({x}, {y})")
```

✓ 0.0s

```
(x, y) labeled pairs
(['the'], fat)
(['the', 'fat'], cat)
(['the', 'fat', 'cat'], sat)
(['the', 'fat', 'cat', 'sat'], on)
(['the', 'fat', 'cat', 'sat', 'on'], the)
(['the', 'fat', 'cat', 'sat', 'on', 'the'], mat)
```

RNN Code Example: Weights

$$\mathbf{h}_t = g(\mathbf{W}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t + \mathbf{b}) \in \mathbb{R}^h$$

- Main matrices: $\mathbf{W} \in \mathbb{R}^{h \times h}, \mathbf{U} \in \mathbb{R}^{h \times d}$
- $\mathbf{W}_o$ as an "un-embedding" to project back to vocabulary space.

```
# Initialize W to [h x h], U to [h x d], b to [h], and h0 to [h]
W = np.random.randn(4, 4)
U = np.random.randn(4, 6)
b = np.random.randn(4, 1)
Wo = np.random.randn(6, 4)
h0 = np.zeros((4, 1))
```

RNN Code Example: Forward Pass

$$\mathbf{h}_t = g(\mathbf{W}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t + \mathbf{b}) \in \mathbb{R}^h$$

```
Training example 1.0: ['the'] -> fat
x: [1 0 0 0 0 0]
h = relu(Wh + Ux + b):[0.09 0.26 0.   0.  ]
z = Wo @ h:[-0.36  0.1   0.48  0.32 -0.59  0.42]
y_hat = argmax softmax(z): 2 (cat)
y = fat
```

- Forward pass is similar to FFNN:
  - Embed each **new single** word into a vector first with our embedding lookup
  - Encode with **U** and add to hidden state
  - Get an output prediction w/ ReLU
  - Final logistic regression for a prediction

```
h = h0
for i, current_word in enumerate(text[:-1]):
    print(f"Training example 1.{i}: {text[0:i+1]} -> {text[i+1]}")
    x = np.array([embed[current_word]])
    x = x.reshape(-1, 1)

    h = np.maximum(W @ h + U @ x + b, 0)
    z = Wo @ h
    y_hat = np.argmax(softmax(z))
```

RNN Code Example: Forward Pass

$$\mathbf{h}_t = g(\mathbf{W}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t + \mathbf{b}) \in \mathbb{R}^h$$

```
Training example 1.0: ['the'] -> fat
x: [1 0 0 0 0 0]
h = relu(Wh + Ux + b):[0.09 0.26 0.   0.  ]
z = Wo @ h:[-0.36  0.1   0.48  0.32 -0.59  0.42]
y_hat = argmax softmax(z): 2 (cat)
y = fat
-----------------------------------------------------------------
Training example 1.1: ['the', 'fat'] -> cat
x: [0 1 0 0 0 0]
h = relu(Wh + Ux + b):[0.81 1.6  1.85 0.02]
z = Wo @ h:[-2.73  1.03  2.68  1.05 -1.2   3.28]
y_hat = argmax softmax(z): 5 (mat)
y = cat
```

- Forward pass is similar to FFNN:
  - Embed each **new single** word into a vector first with our embedding lookup
  - Encode with **U** and add to hidden state
  - Get an output prediction w/ ReLU
  - Final logistic regression for a prediction

```python
h = h0
for i, current_word in enumerate(text[:-1]):
    print(f"Training example 1.{i}: {text[0:i+1]} -> {text[i+1]}")
    x = np.array([embed[current_word]])
    x = x.reshape(-1, 1)

    h = np.maximum(W @ h + U @ x + b, 0)
    z = Wo @ h
    y_hat = np.argmax(softmax(z))
```

RNN Code Example: Forward Pass

$$\mathbf{h}_t = g(\mathbf{W}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t + \mathbf{b}) \in \mathbb{R}^h$$

- Forward pass is similar to FFNN:
  - Embed each **new single** word into a vector first with our embedding lookup
  - Encode with **U** and add to hidden state
  - Get an output prediction w/ ReLU
  - Final logistic regression for a prediction

```
h = h0
for i, current_word in enumerate(text[:-1]):
    print(f"Training example 1.{i}: {text[0:i+1]} -> {text[i+1]}")
    x = np.array([embed[current_word]])
    x = x.reshape(-1, 1)

    h = np.maximum(W @ h + U @ x + b, 0)
    z = Wo @ h
    y_hat = np.argmax(softmax(z))
```

```
Training example 1.0: ['the'] -> fat
x: [1 0 0 0 0 0]
h = relu(Wh + Ux + b):[0.09 0.26 0.   0.  ]
z = Wo @ h:[-0.36  0.1   0.48  0.32 -0.59  0.42]
y_hat = argmax softmax(z): 2 (cat)
y = fat
---------------------------------------------------------------
Training example 1.1: ['the', 'fat'] -> cat
x: [0 1 0 0 0 0]
h = relu(Wh + Ux + b):[0.81 1.6  1.85 0.02]
z = Wo @ h:[-2.73  1.03  2.68  1.05 -1.2   3.28]
y_hat = argmax softmax(z): 5 (mat)
y = cat
---------------------------------------------------------------
Training example 1.2: ['the', 'fat', 'cat'] -> sat
x: [0 0 1 0 0 0]
h = relu(Wh + Ux + b):[5.47 3.62 0.94 0.  ]
z = Wo @ h:[-4.83  0.17  6.38 10.45 -6.84  5.24]
y_hat = argmax softmax(z): 3 (sat)
y = sat
---------------------------------------------------------------
Training example 1.3: ['the', 'fat', 'cat', 'sat'] -> on
x: [0 0 0 1 0 0]
h = relu(Wh + Ux + b):[6.19 0.   1.82 2.13]
z = Wo @ h:[-1.34 -0.35 -2.73  9.59  4.39 -2.99]
y_hat = argmax softmax(z): 3 (sat)
y = on
---------------------------------------------------------------
Training example 1.4: ['the', 'fat', 'cat', 'sat', 'on'] -> the
x: [0 0 0 0 1 0]
h = relu(Wh + Ux + b):[0.46 0.   0.22 2.04]
z = Wo @ h:[-1.41  1.13 -2.23  1.67  2.05 -2.22]
y_hat = argmax softmax(z): 4 (on)
y = the
```