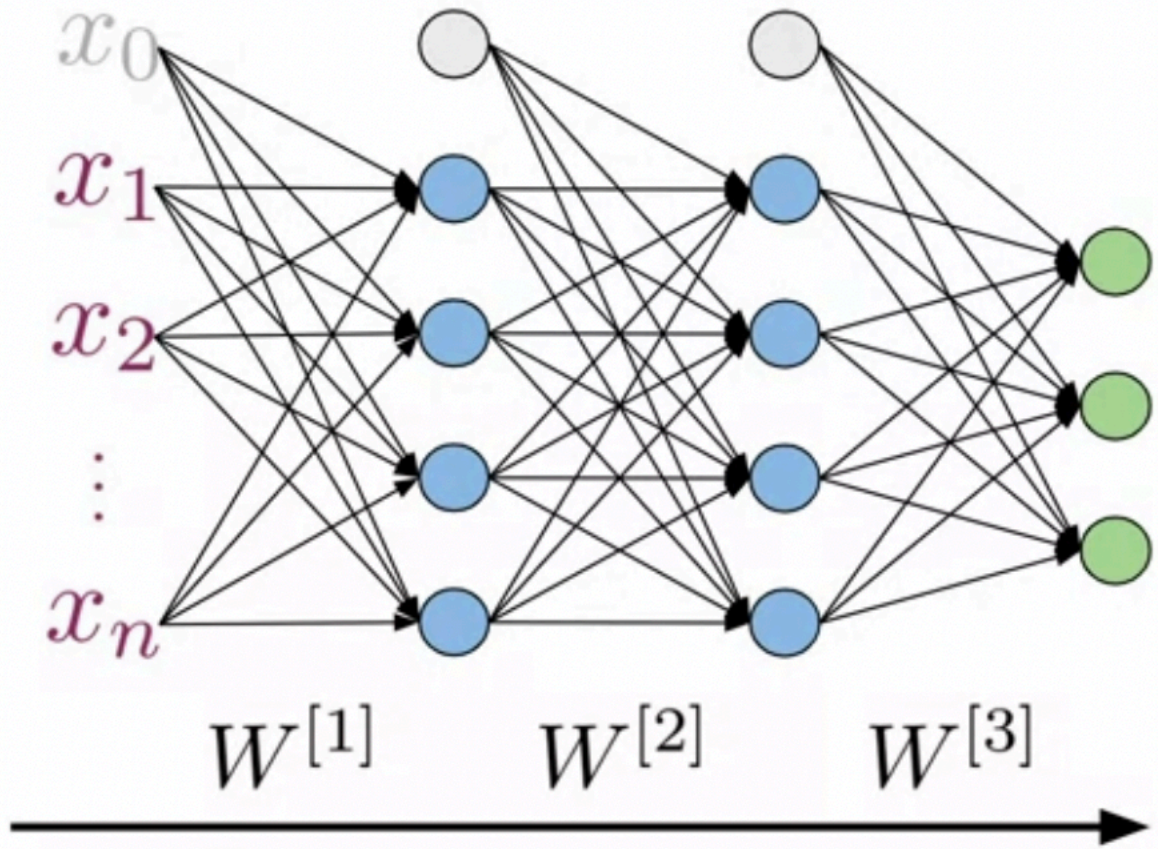
Outline

- Neural networks and forward propagation
- Structure for sentiment analysis

Forward propagation



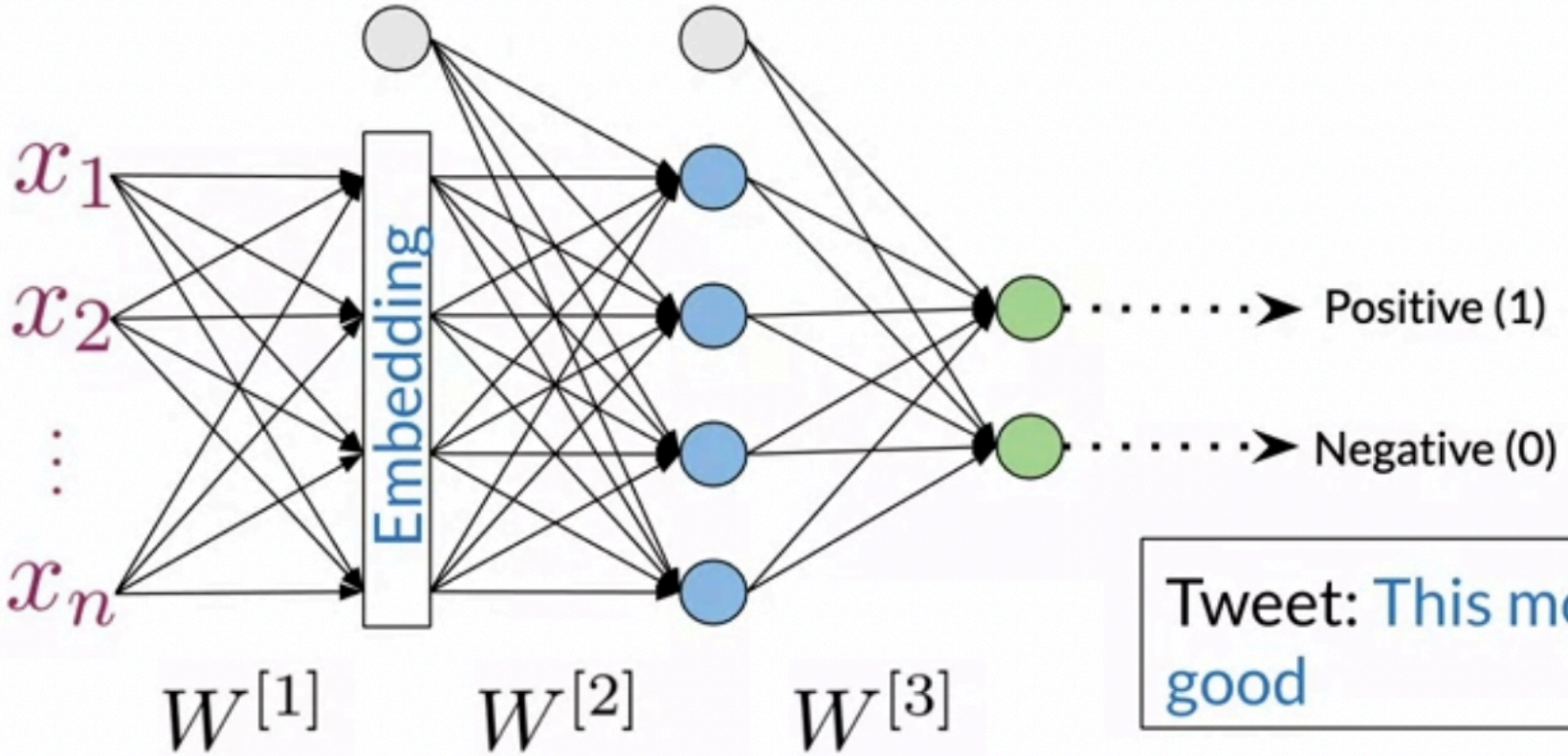
$a^{[i]}$ Activations ith layer

$$a^{[0]} = X$$

$$z^{[i]} = W^{[i]} a^{[i-1]}$$

$$a^{[i]} = g^{[i]}(z^{[i]})$$

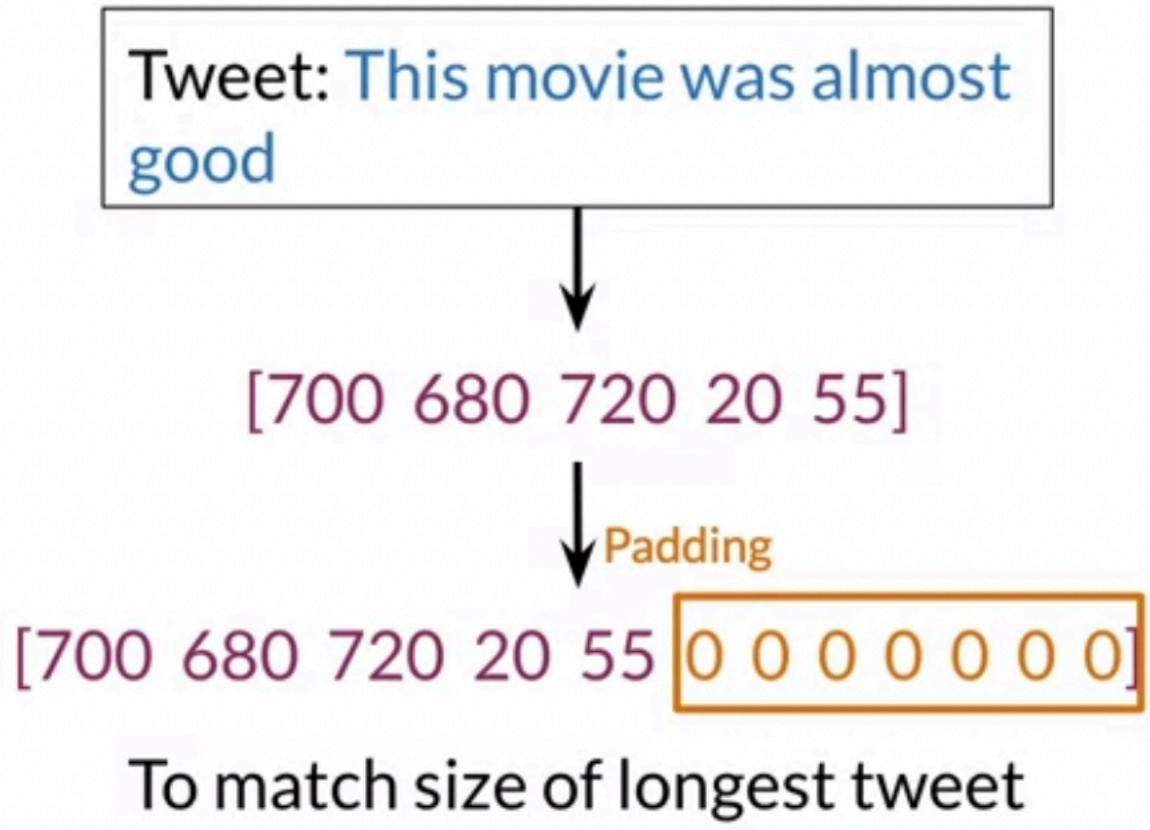
Neural Networks for sentiment analysis



Tweet: This movie was almost good

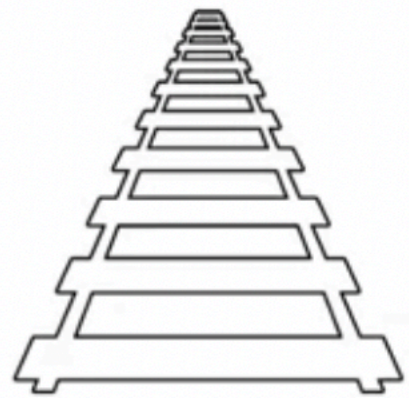
Initial Representation

Word	Number
a	1
able	2
about	3
...	...
hand	615
...	...
happy	621
...	...
zebra	1000

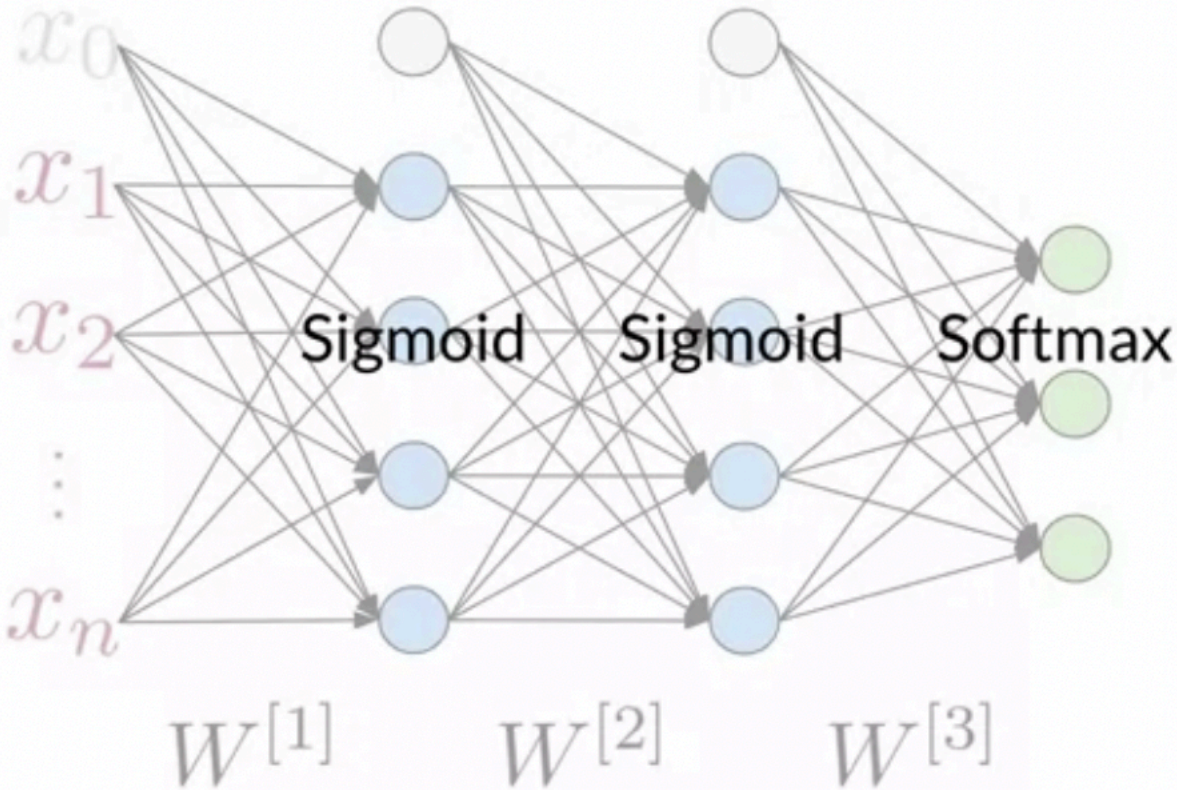


Outline

- Define a basic neural network using Trax
- Benefits of Trax



Neural Networks in Trax




```
from trax import layers as tl
Model = tl.Serial(
          tl.Dense(4),
          tl.Sigmoid(),
          tl.Dense(4),
          tl.Sigmoid(),
          tl.Dense(3),
          tl.Softmax())
```

Advantages of using frameworks

- Run fast on CPUs, GPUs and TPUs
- Parallel computing
- Record algebraic computations for gradient evaluation

Tensorflow Pytorch JAX

Summary

- Order of computation $\longrightarrow$ Model in Trax
- Benefits from using frameworks

deeplearning.ai

Trax: Layers

Classes in Python

```
class MyClass(Object):
    def __init__(self, y):
        self.y = y
    def my_method(self,x):
        return x + self.y

    def __call__(self, x):
        return self.my_method(x)
```

```
f = MyClass(7)
```

Classes in Python

```
class MyClass(Object):

    def __init__(self, y):
        self.y = y

    def my_method(self,x):
        return x + self.y

    def __call__(self, x):
        return self.my_method(x)
```

```
f = MyClass(7)

print(f(3))
```

```
10
```

Subclasses

```
class MyClass(Object):

    def __init__(self,y):
        self.y = y

    def my_method(self,x):
        return x + self.y

    def __call__(self,x):
        return self.my_method(x)
```

```
class SubClass(MyClass):
```

Subclasses

```
class MyClass(Object):

    def __init__(self,y):
        self.y = y

    def my_method(self,x):
        return x + self.y

    def __call__(self,x):
        return self.my_method(x)
```

```
class SubClass(MyClass):

    def my_method(self,x):
        return x + self.y**2
```

Subclasses

```
class MyClass(Object):

    def __init__(self,y):
        self.y = y

    def my_method(self,x):
        return x + self.y

    def __call__(self,x):
        return self.my_method(x)
```

```
class SubClass(MyClass):

    def my_method(self,x):
        return x + self.y**2
```

```
f = SubClass(7)

print(f(3))
```

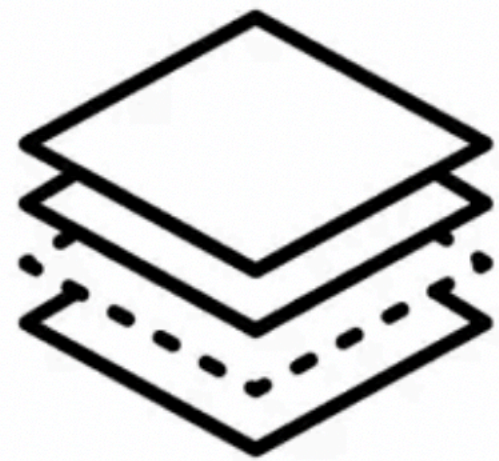
```
52
```

deeplearning.ai

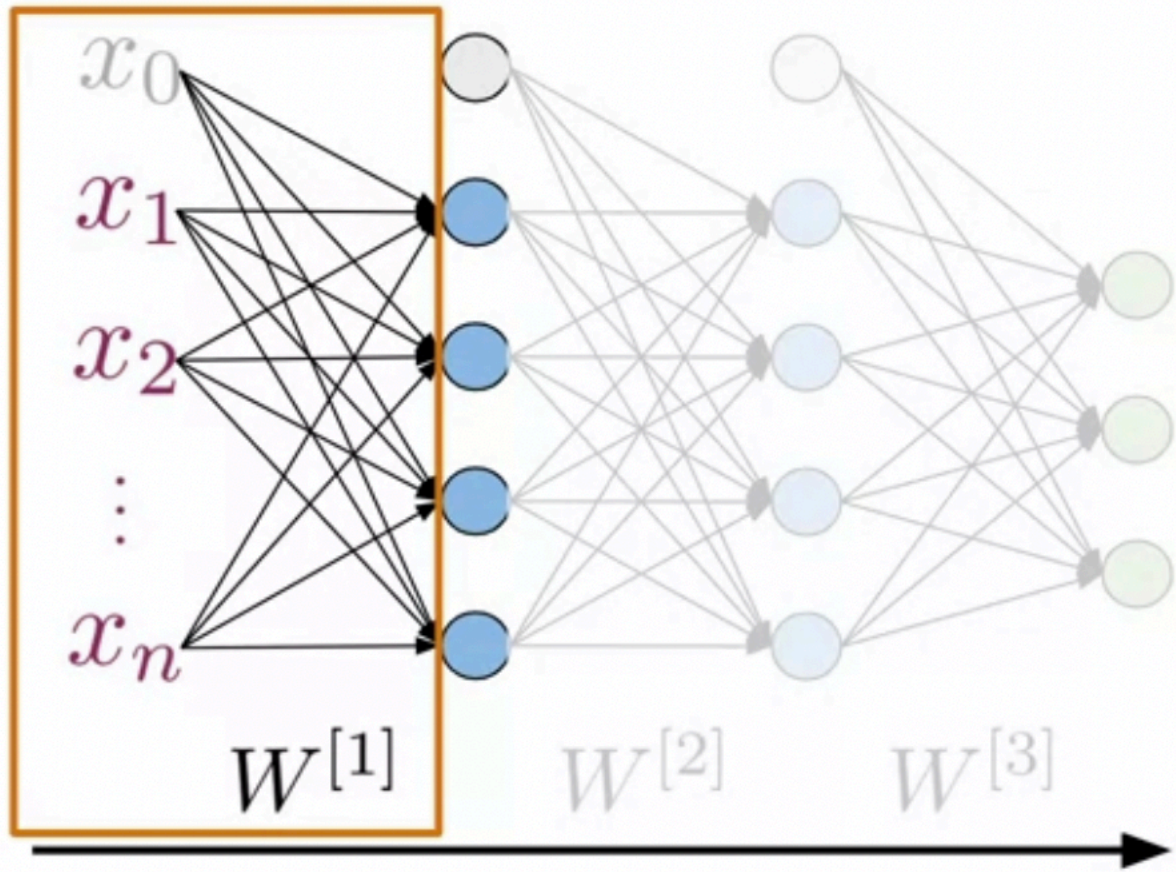
Dense and ReLU Layers

Outline

- Dense and ReLU layers



Dense Layer

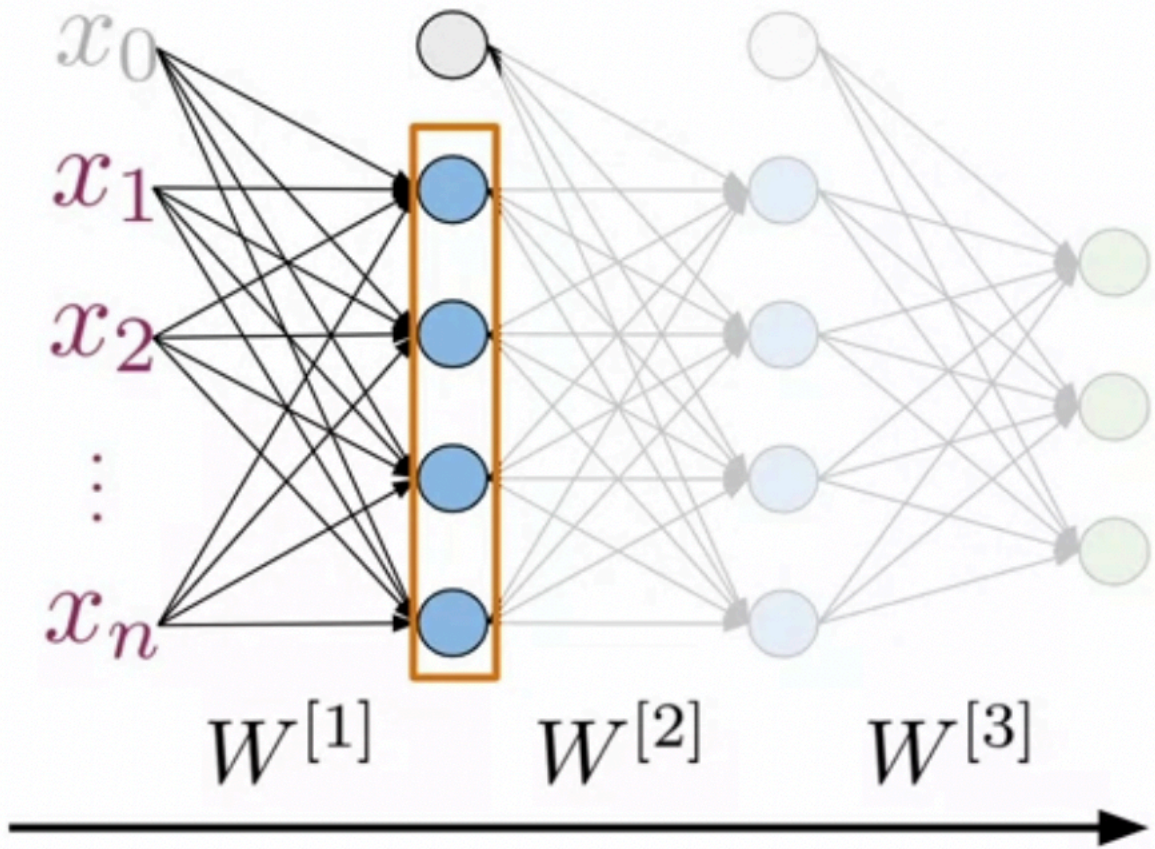


Fully connected layer

$$z^{[i]} = W^{[i]} a^{[i-1]}$$

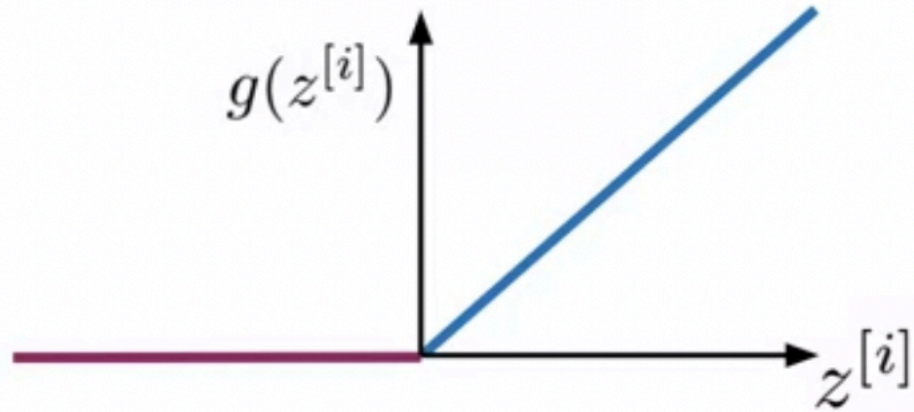
Trainable parameters

ReLU Layer



ReLU = Rectified linear unit

$$a^{[i]} = g(z^{[i]})$$
$$g(z^{[i]}) = \max(0, z^{[i]})$$



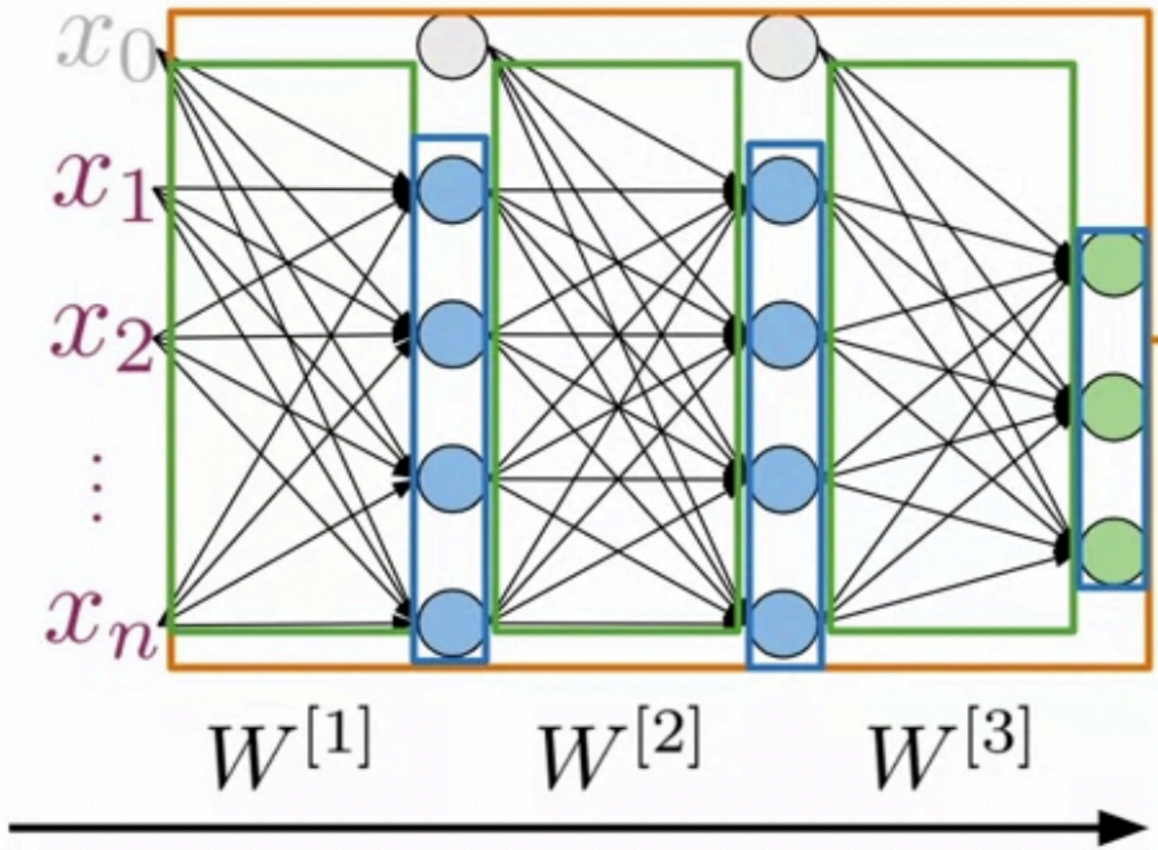
Summary

- Dense Layer $\longrightarrow z^{[i]} = W^{[i]} a^{[i-1]}$
- ReLU Layer $\longrightarrow g(z^{[i]}) = \max(0, z^{[i]})$

deeplearning.ai

Serial Layer

Serial Layer



Composition of layers in *serial* arrangement

- Dense Layers
- Activation Layers

Summary

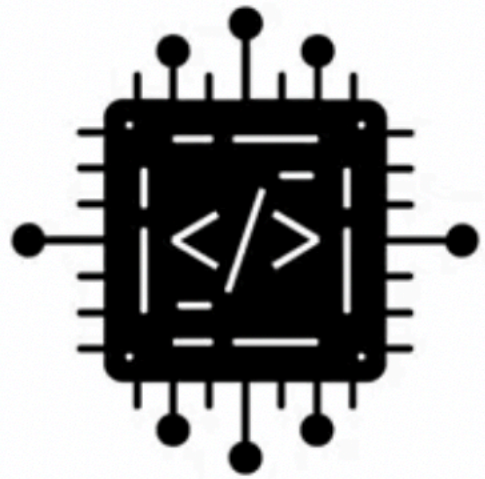
- Serial layer is a composition of sublayers

deeplearning.ai

Trax: Other Layers

Outline

- Embedding layer
- Mean layer



Embedding Layer

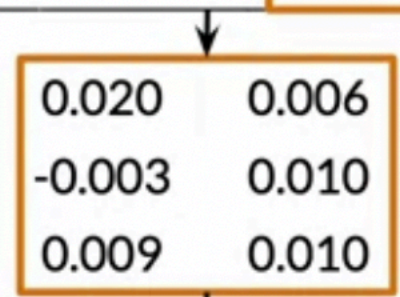
Vocabulary	Index		
I	1	0.020	0.006
am	2	-0.003	0.010
happy	3	0.009	0.010
because	4	-0.011	-0.018
learning	5	-0.040	-0.047
NLP	6	-0.009	0.050
sad	7	-0.044	0.001
not	8	0.011	-0.022

Vocabulary x Embedding

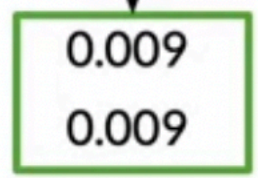
Mean Layer

Tweet: I am happy

Vocabulary	Index		
I	1	0.020	0.006
am	2	-0.003	0.010
happy	3	0.009	0.010



Mean of the word embeddings



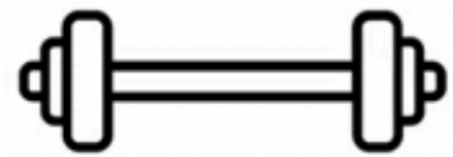
No trainable parameters

Summary

- Embedding is trainable using an embedding layer
- Mean layer gives a vector representation

Outline

- Computing gradients in trax
- Training using grad()



Computing gradients in Trax

$$f(x) = 3x^2 + x$$

$$\frac{\delta f(x)}{\delta x} = 6x + 1$$

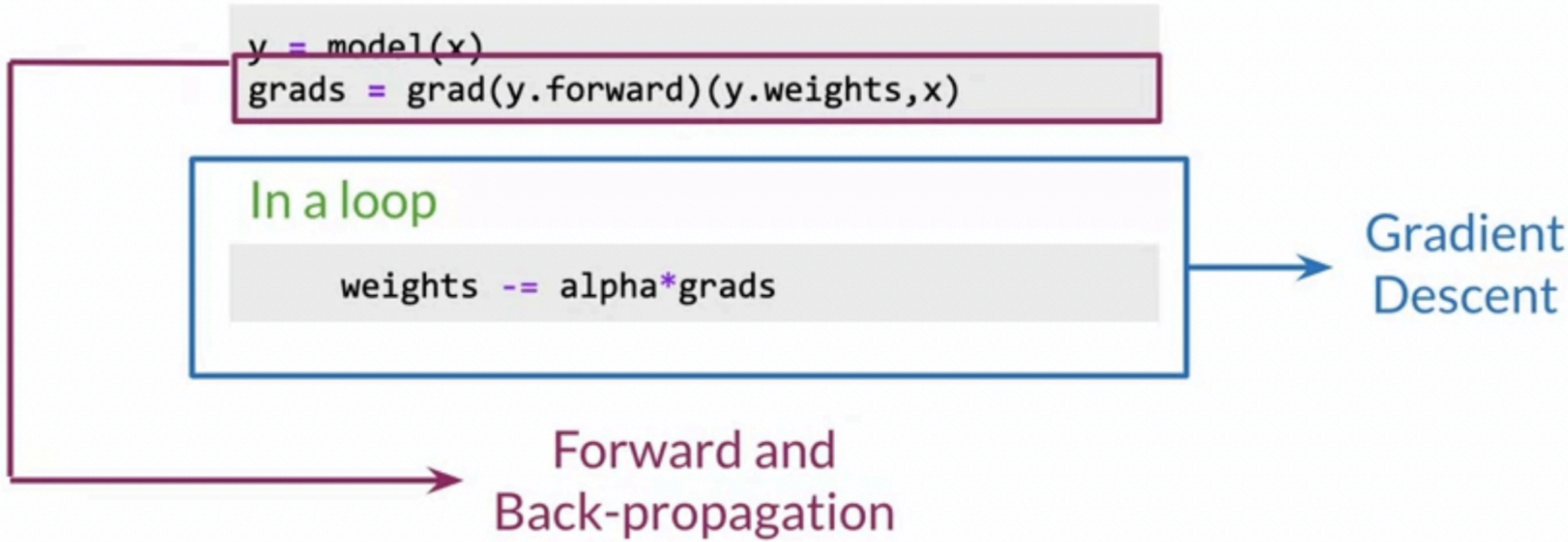
Gradient

```
def f(x):
    return 3*x**2 + x

grad_f = trax.math.grad(f)
```

Returns a function

Training with grad()



Summary

- grad() allows much easier training
- Forward and backpropagation in one line!