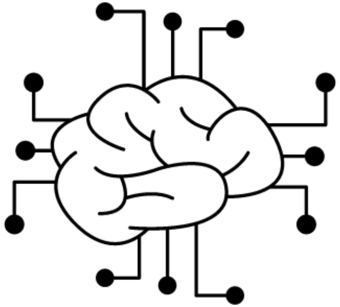
Copyright Notice

deeplearning.ai

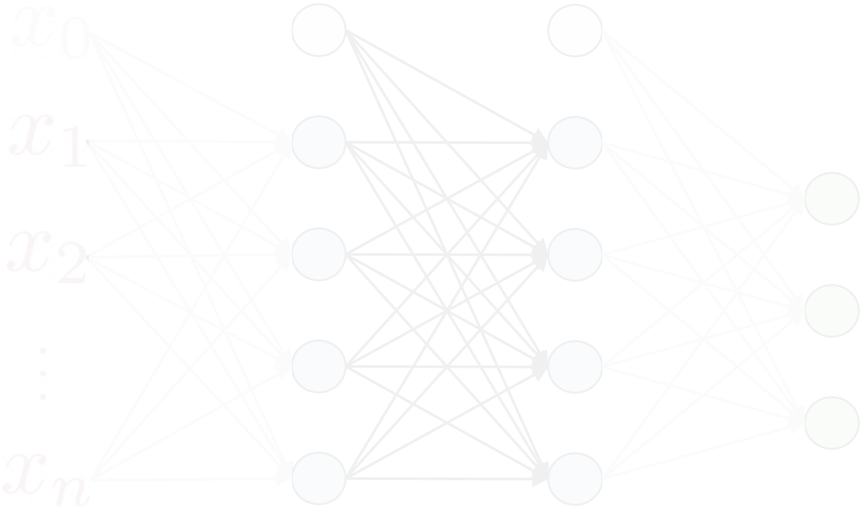
Neural Networks for Sentiment Analysis

Outline

- Neural networks and forward propagation
- Structure for sentiment analysis

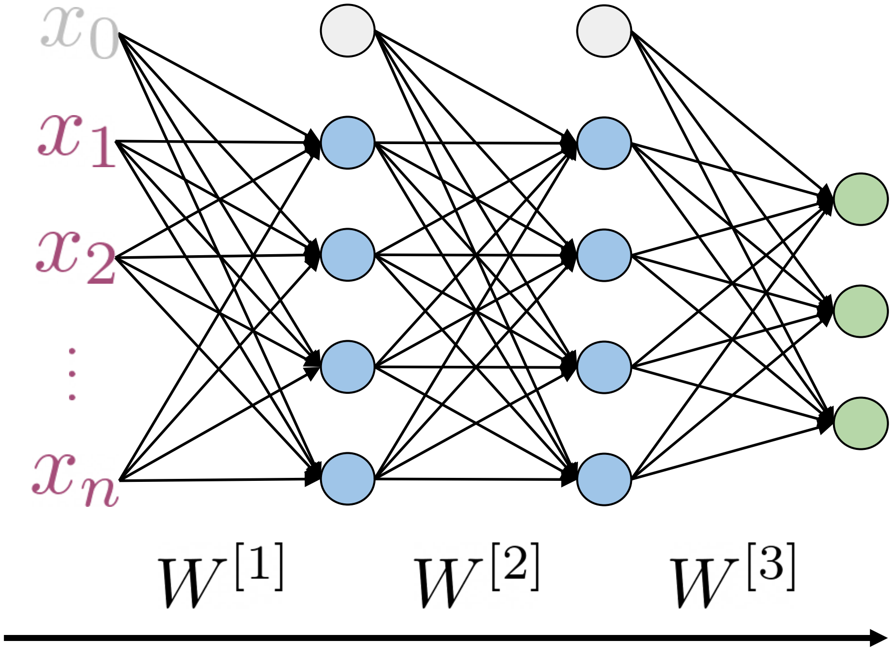


Neural Networks



$W^{[1]}$ $W^{[2]}$ $W^{[3]}$

Forward propagation



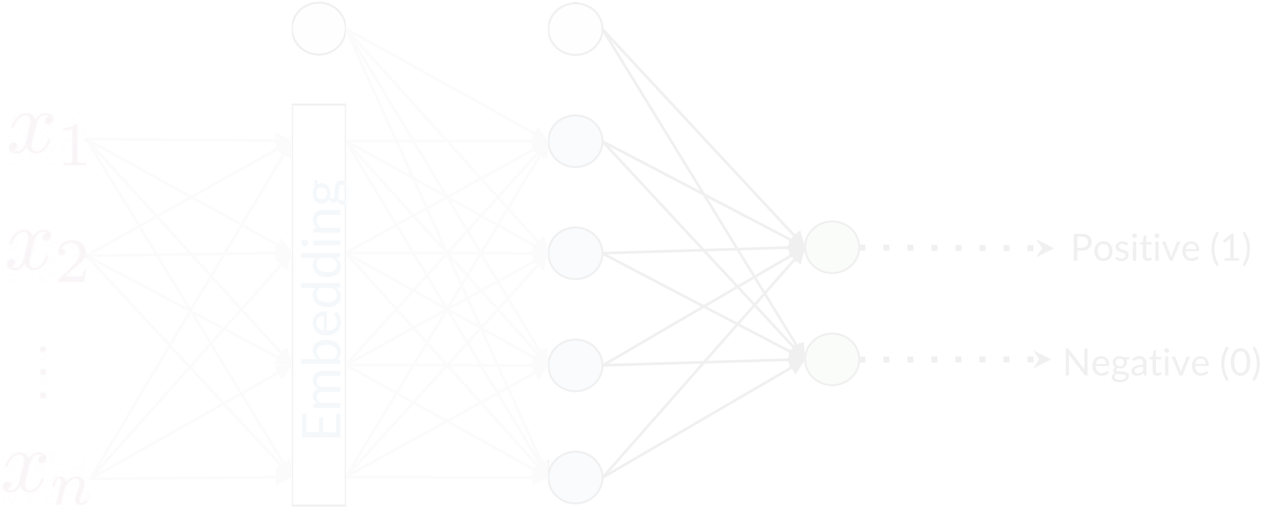
$a^{[i]}$ Activations ith layer

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

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

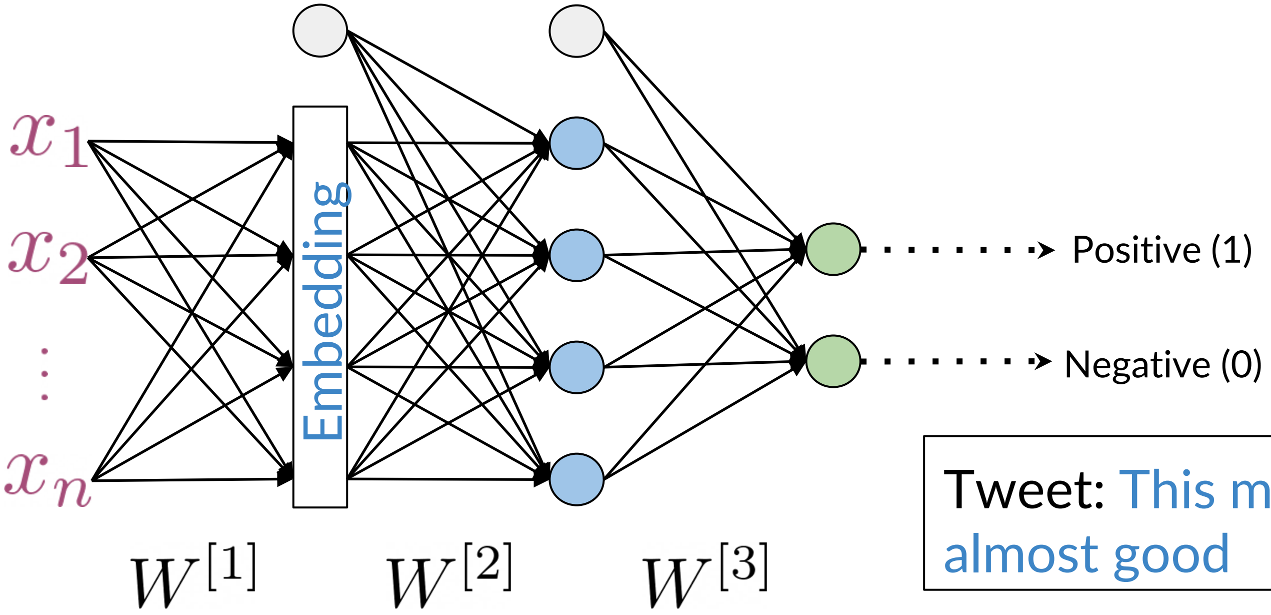
$$a^{[i]} = g^{[i]}(z^{[i]})$$

Neural Networks for sentiment analysis



$W^{[1]}$ $W^{[2]}$ $W^{[3]}$

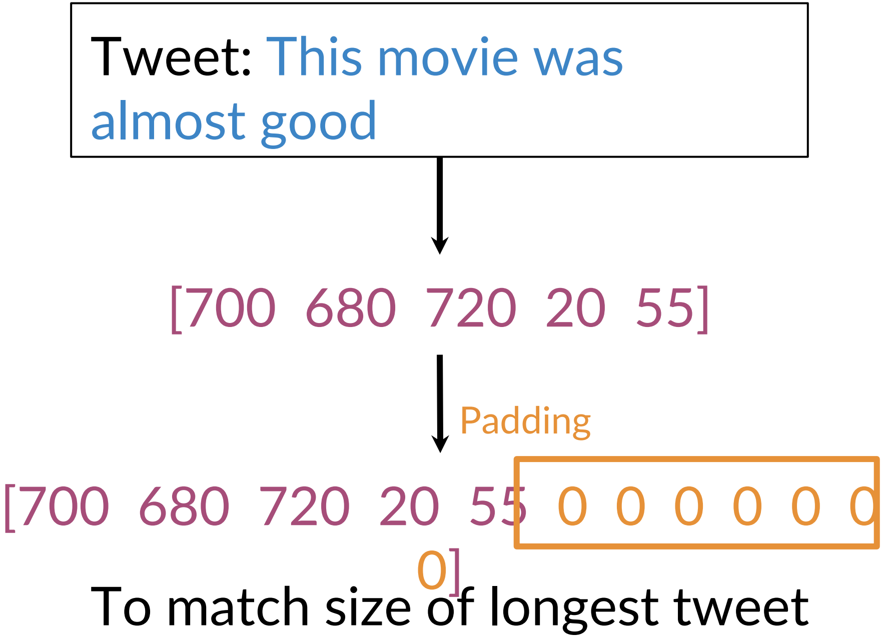
Neural Networks for sentiment analysis



Tweet: This movie was almost good

Initial Representation

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



Tweet: This movie was almost good

[700 680 720 20 55]

Padding

[700 680 720 20 55 0 0 0 0 0 0 0]

To match size of longest tweet

Summary

- Structure for sentiment analysis
- Classify complex tweets
- Initial representation

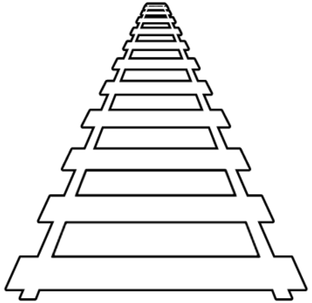
deeplearning.ai

Trax: Neural Networks

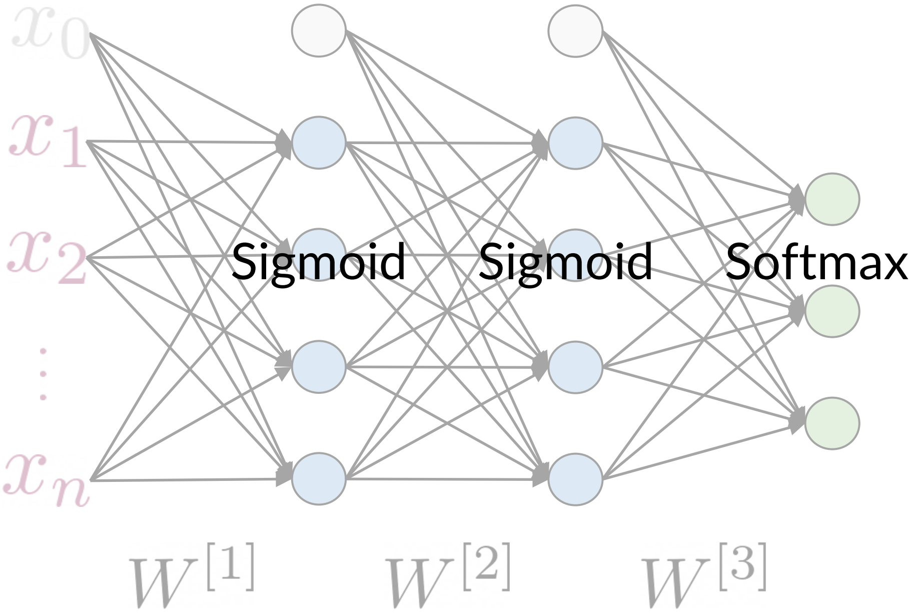
Outline

- Define a basic neural network using Trax

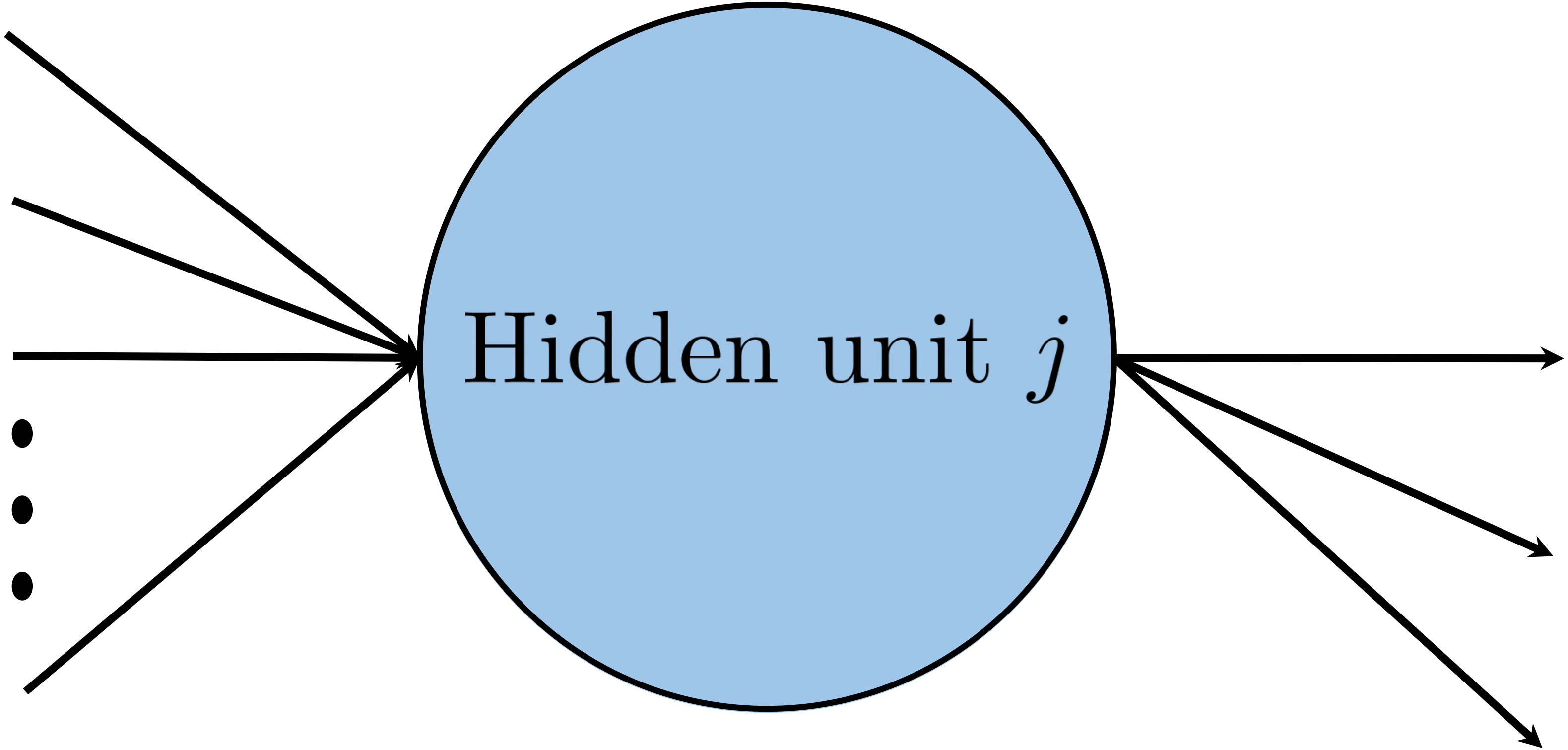
- Benefits of Trax



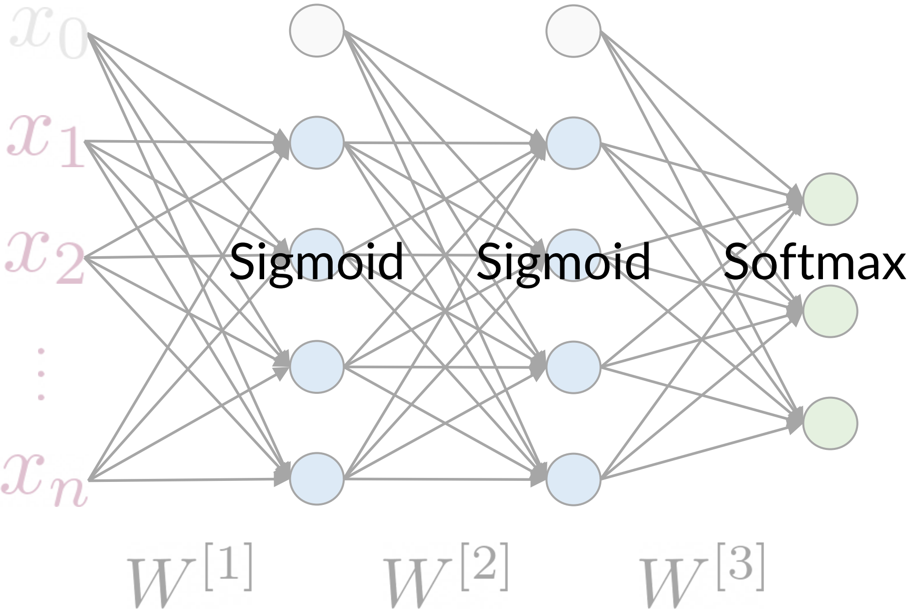
Neural Networks in Trax



Neural networks in 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

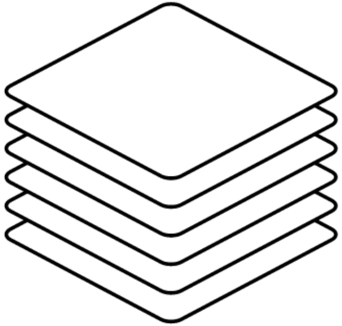
- Order of computation ⟶ Model in Trax
- Benefits from using frameworks

deeplearning.ai

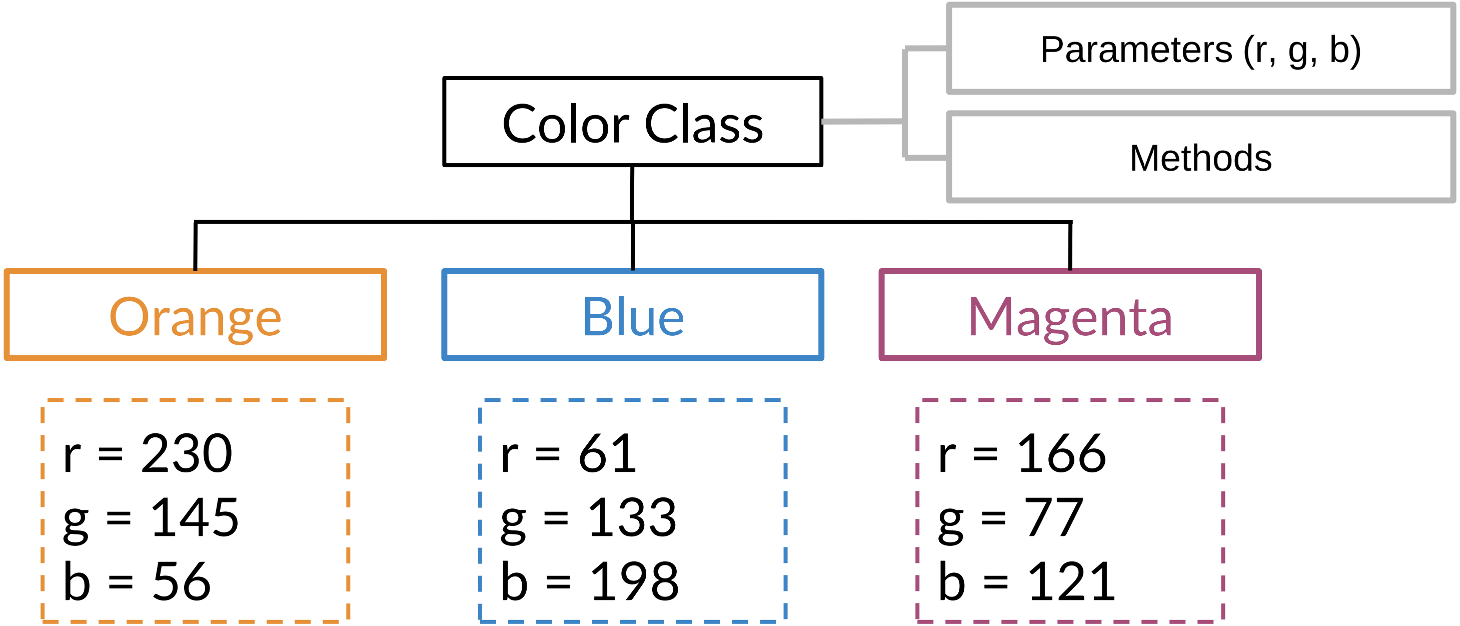
Classes, Subclasses and Inheritance

Outline

- How classes work and their implementation
- Subclasses and inheritance



Classes



Color Class

Parameters (r, g, b)

Methods

Orange

r = 230
g = 145
b = 56

Blue

r = 61
g = 133
b = 198

Magenta

r = 166
g = 77
b = 121

Classes in Python

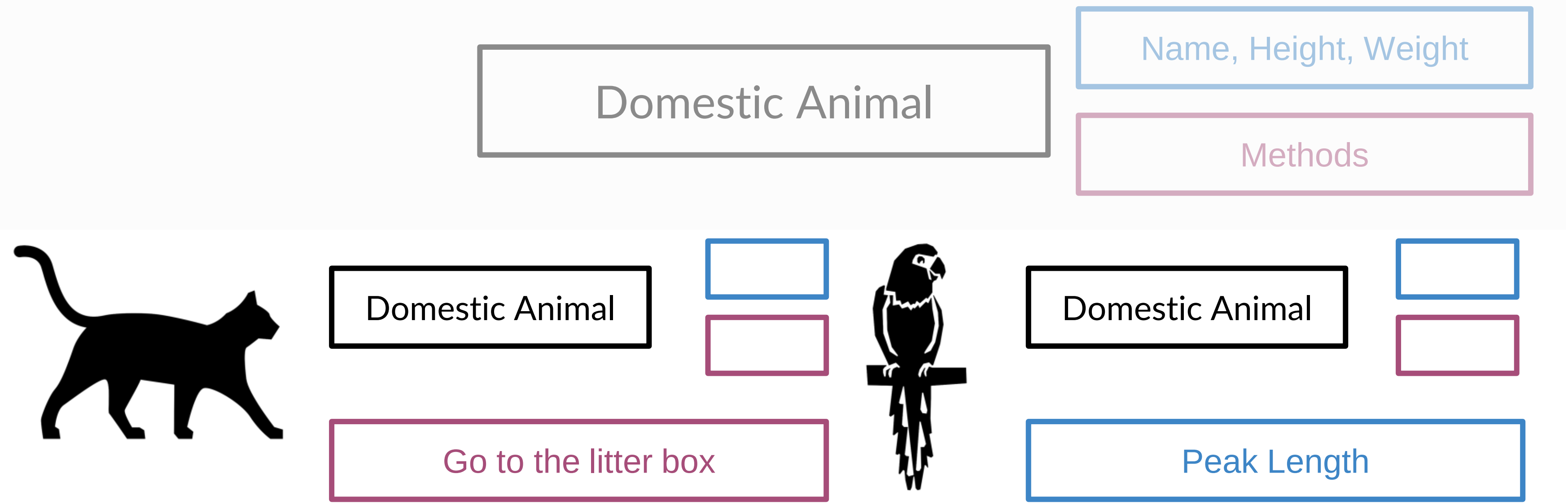
```
class MyClass:
    def __init__(self, y):
              self.y = y
    def my_method(self,x):
              return x + self.y
    def __call__(self, x):
              return 
self.my_method(x)
```

```
f = MyClass(7)

print(f(3))
```

```
10
```

Subclasses and Inheritance



Convenient when classes share common parameters and methods.

Subclasses

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

Summary

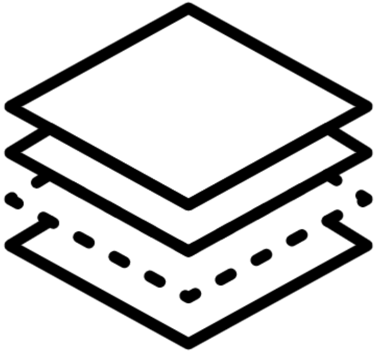
- Classes, subclasses, instances and inheritance.

deeplearning.ai

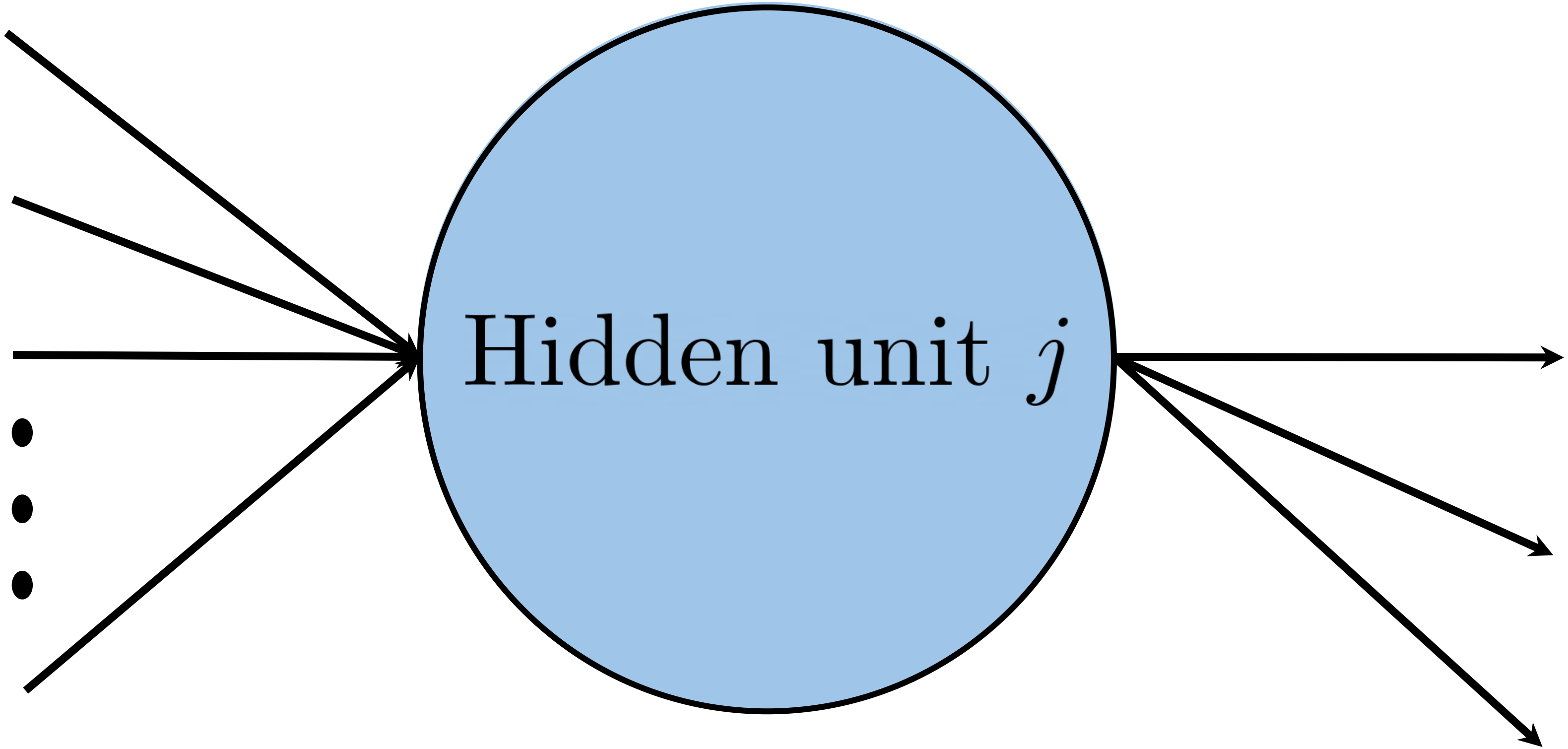
Dense and ReLU Layers

Outline

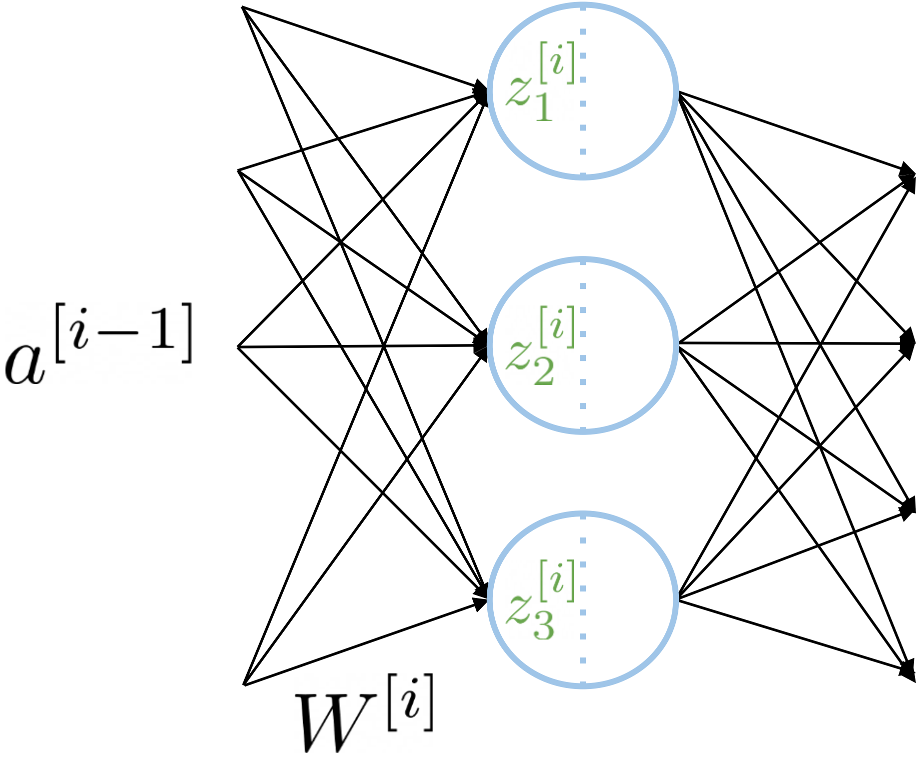
- Dense layer in detail
- ReLU function



Neural networks in Trax



Dense Layer



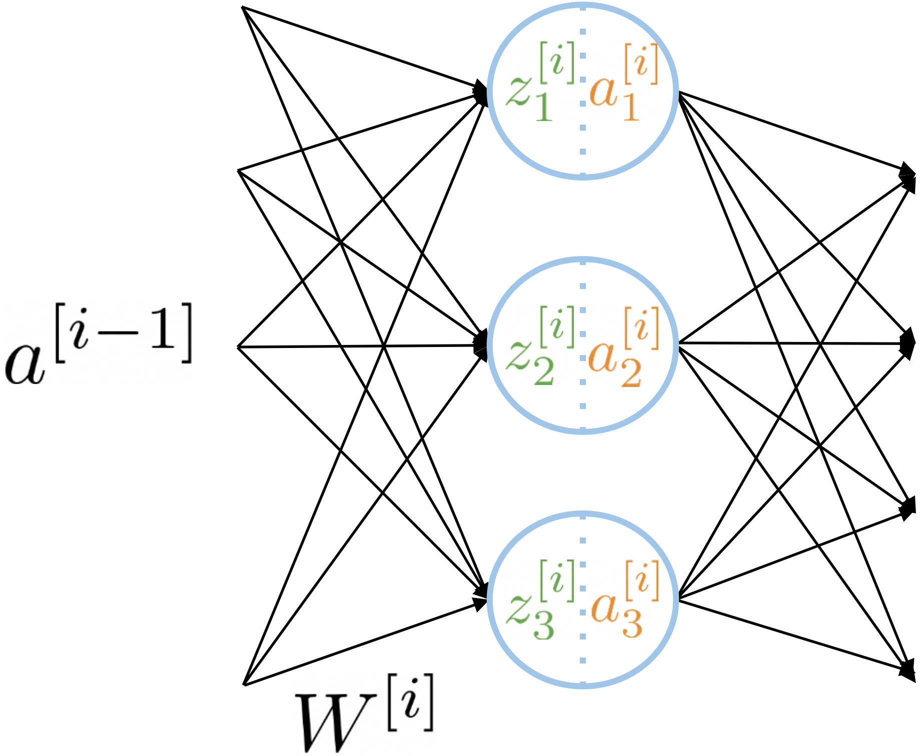
$$z_j^{[i]} = {w_j^{[i]}}^T a^{[i-1]}$$

Dense layer

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

Trainable parameters

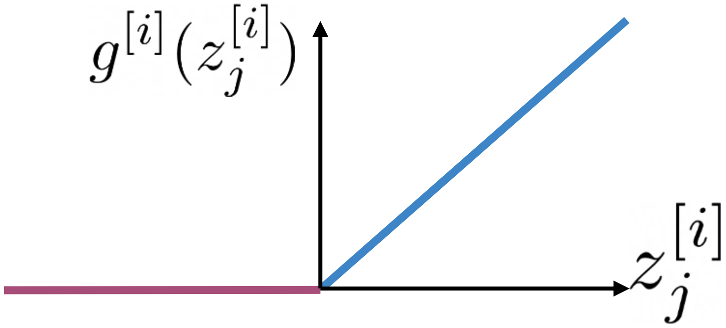
ReLU Layer



$$a_j^{[i]} = g^{[i]}(z_j^{[i]})$$

ReLU = Rectified linear unit

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



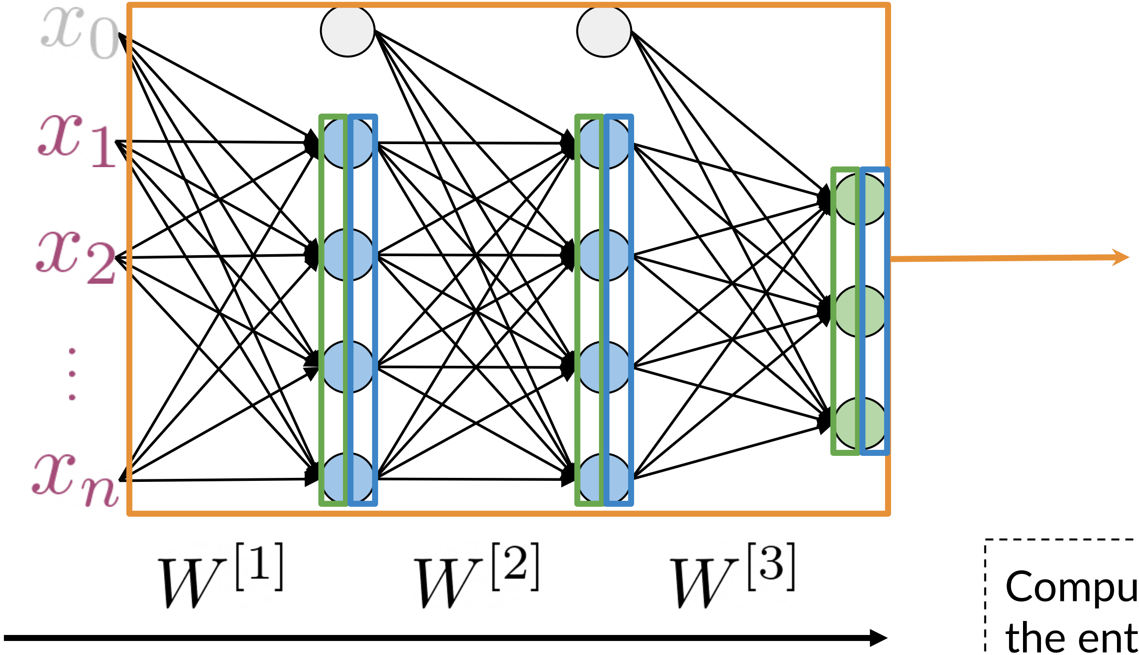
Summary

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

deeplearning.ai

Serial Layer

Serial Layer



Composition of layers in *serial* arrangement

- 🟩 Dense Layers
- 🟦 Activation Layers

Compute forward propagation of the entire model

Summary

- Serial layer is a composition of sublayers
- Forward propagation by calling the method from the serial layer

deeplearning.ai

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

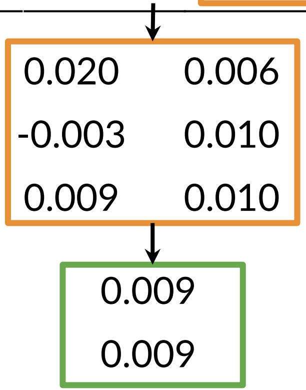
Trainable weights

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



$$\begin{bmatrix} 0.020 & 0.006 \\ -0.003 & 0.010 \\ 0.009 & 0.010 \end{bmatrix} \to \begin{bmatrix} 0.009 \\ 0.009 \end{bmatrix}$$

Mean of the word embeddings

No trainable parameters

Summary

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

deeplearning.ai

Training

Outline

- Computing gradients

- Training

Computing gradients in Trax

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

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

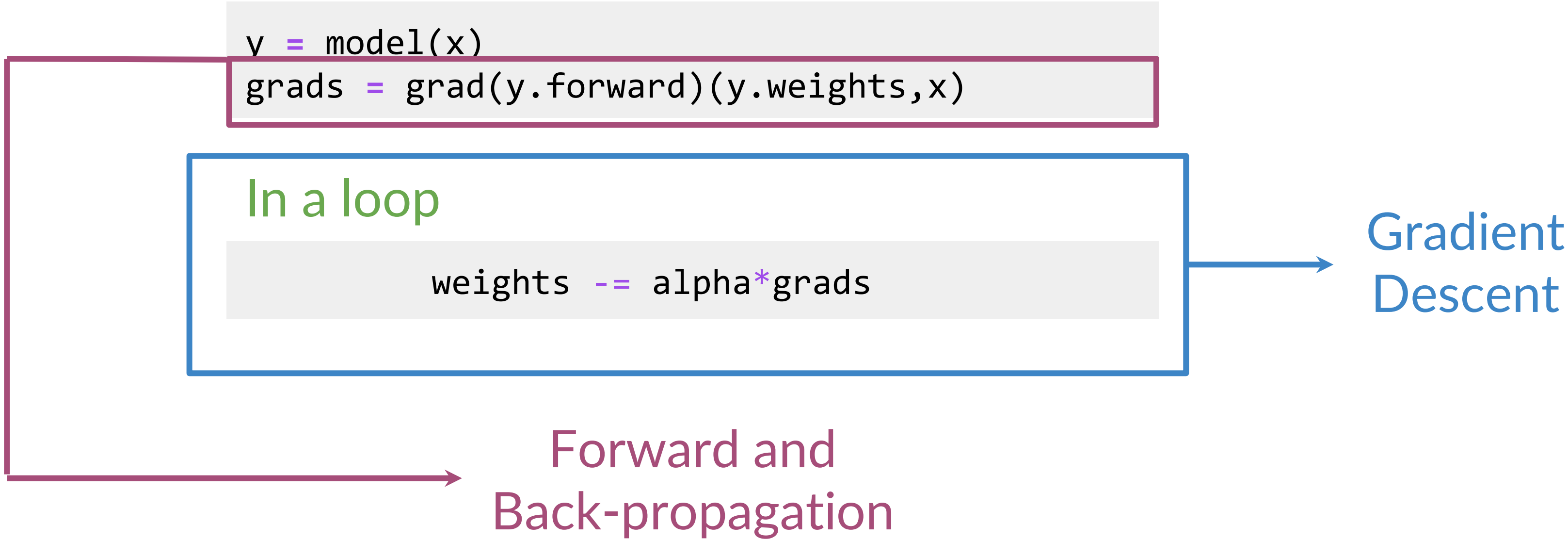
Gradient

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

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

Returns a function

Training with grad()



```
y = model(x)
grads = grad(y.forward)(y.weights,x)
```

In a loop

```
          weights -= alpha*grads
```

Gradient Descent

Forward and Back-propagation

Summary

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