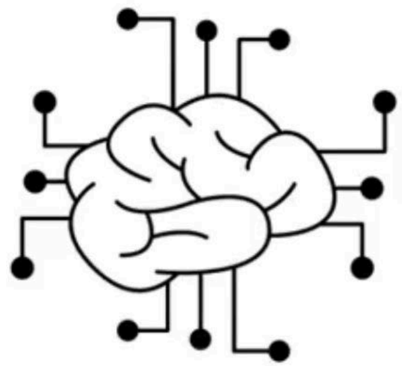
Outline

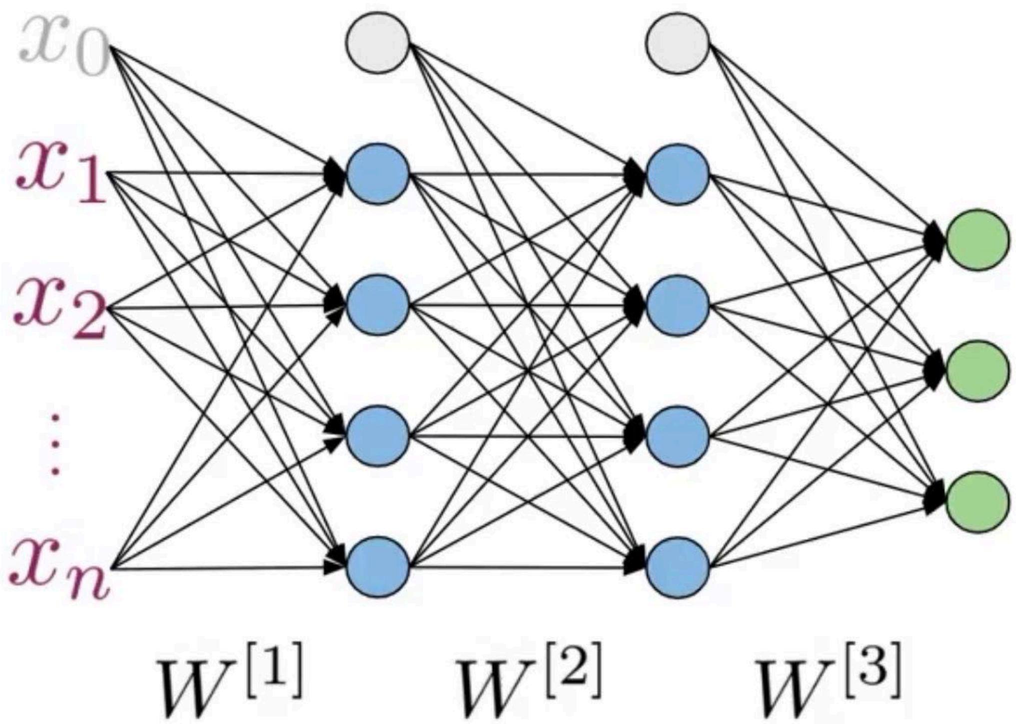
- Neural networks and forward propagation
- Structure for sentiment analysis



5:29 PM Tue Jul 28 71%

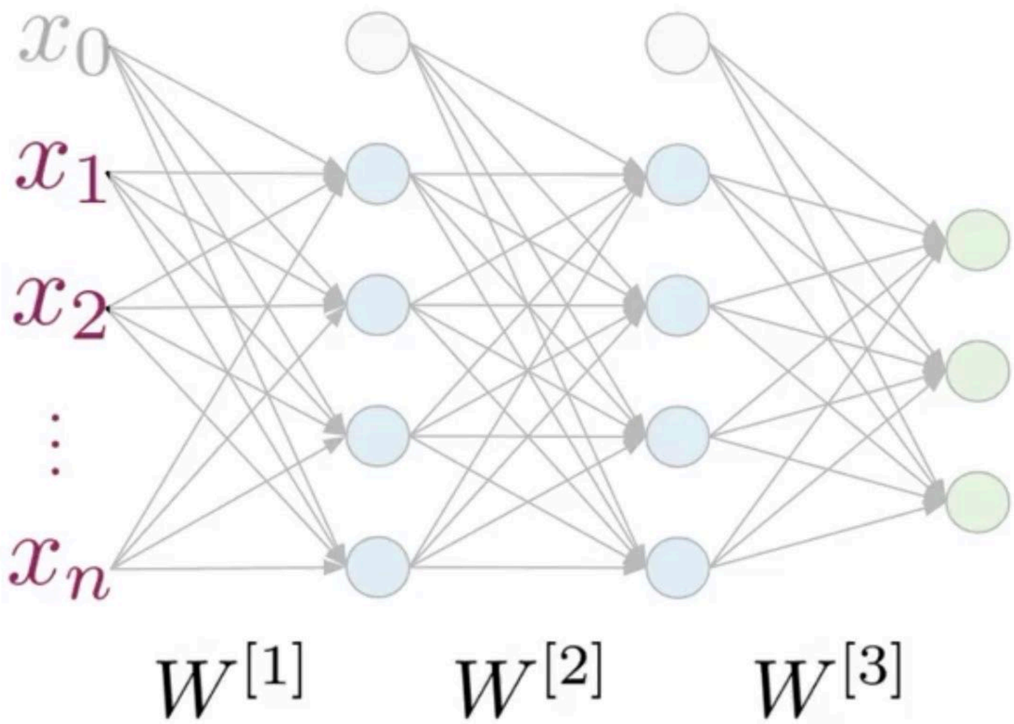
Neural Networks for Sentiment Analysis

Neural Networks

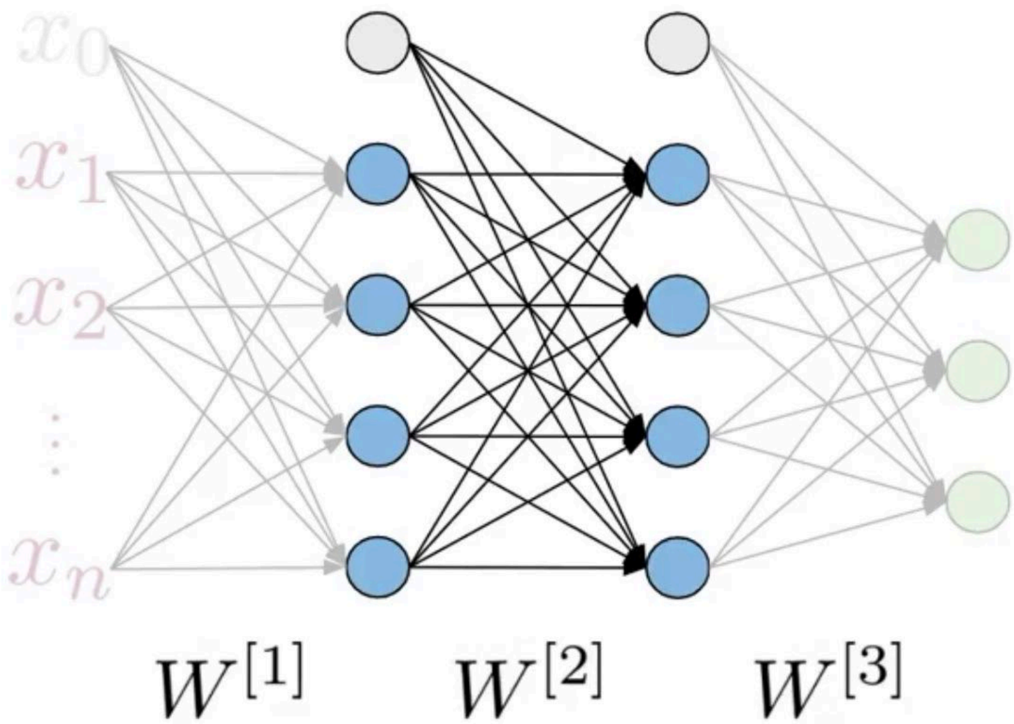


deeplearning.ai

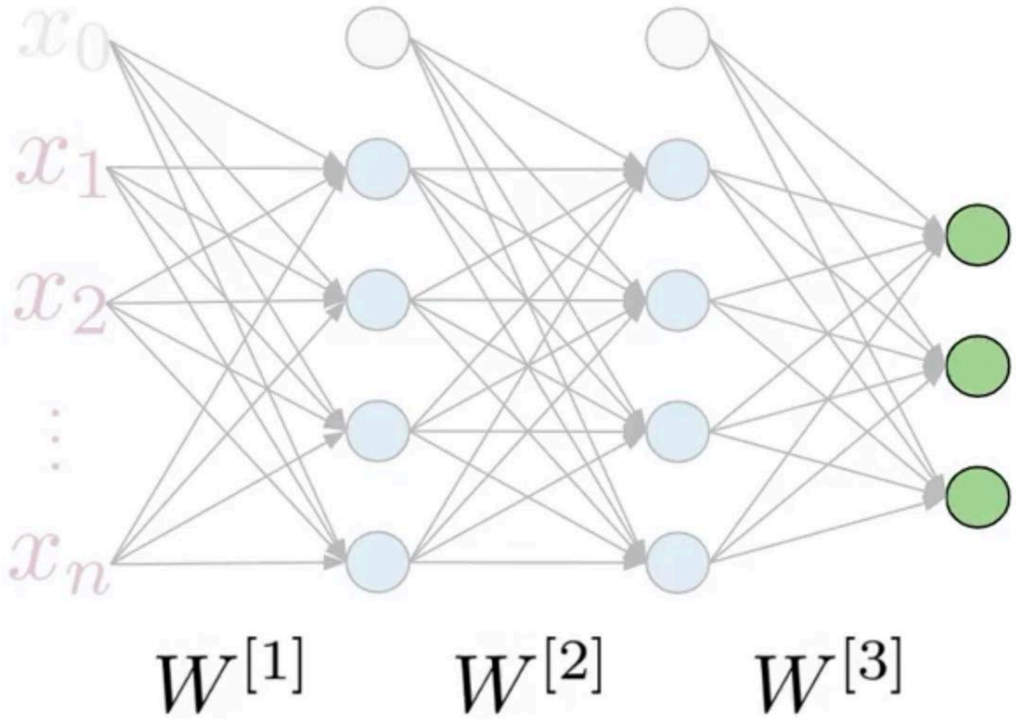
Neural Networks



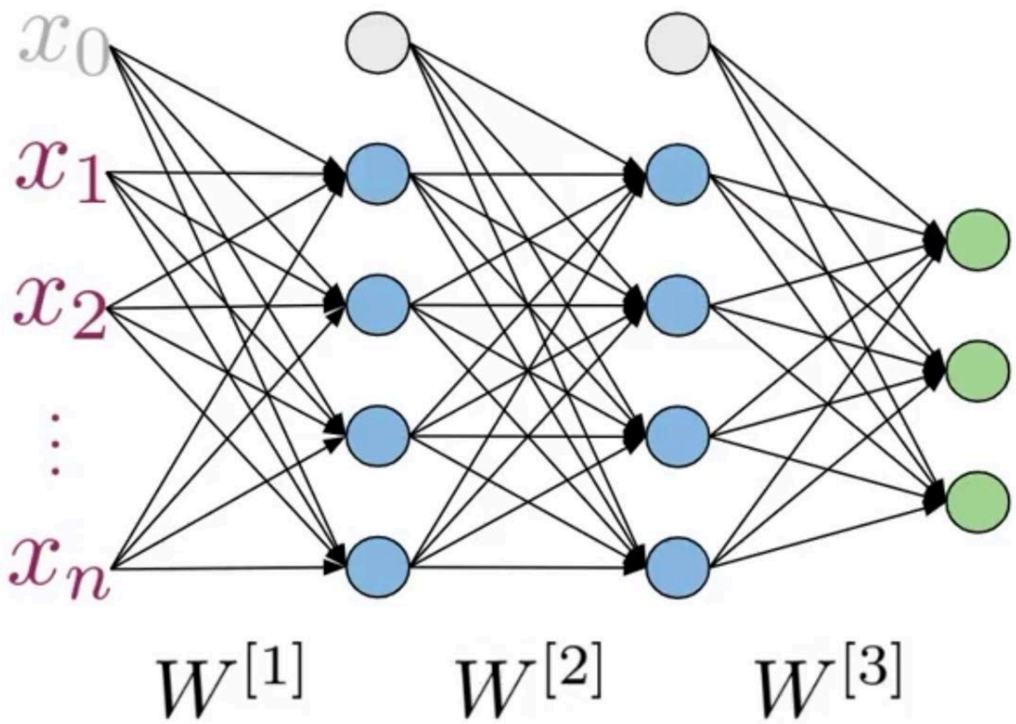
Neural Networks



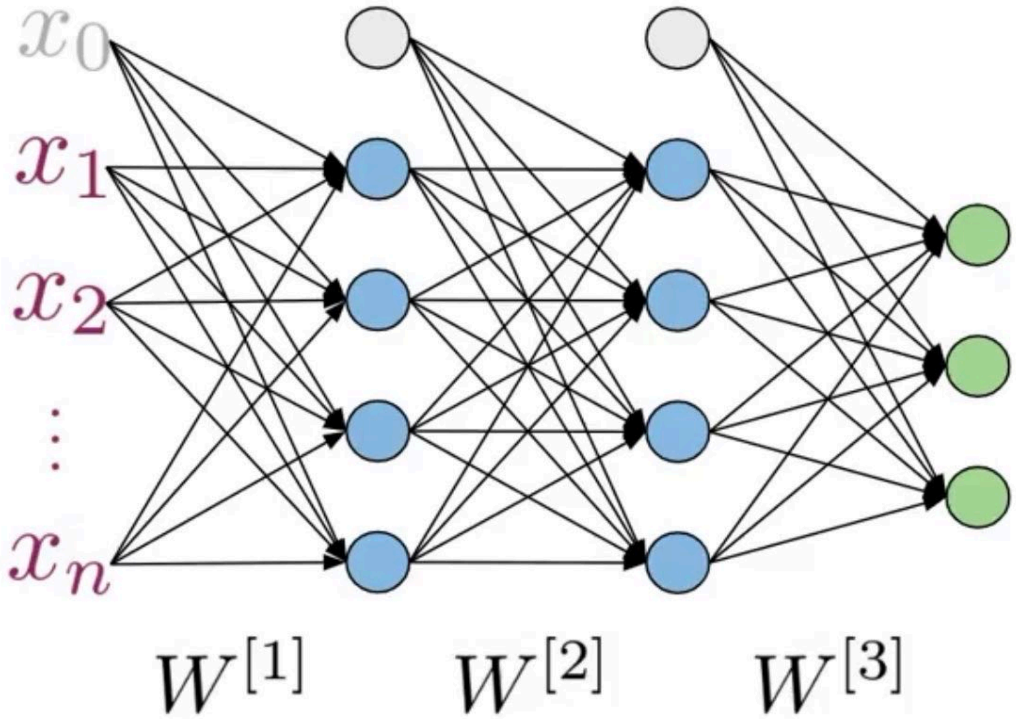
Neural Networks



Forward propagation

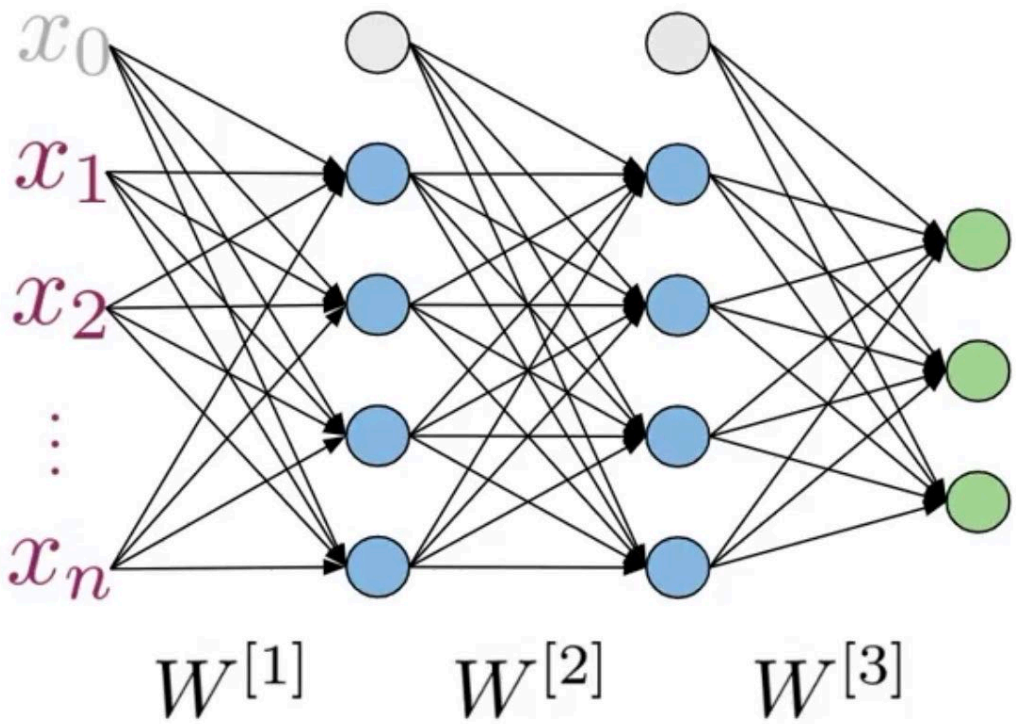


Forward propagation



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

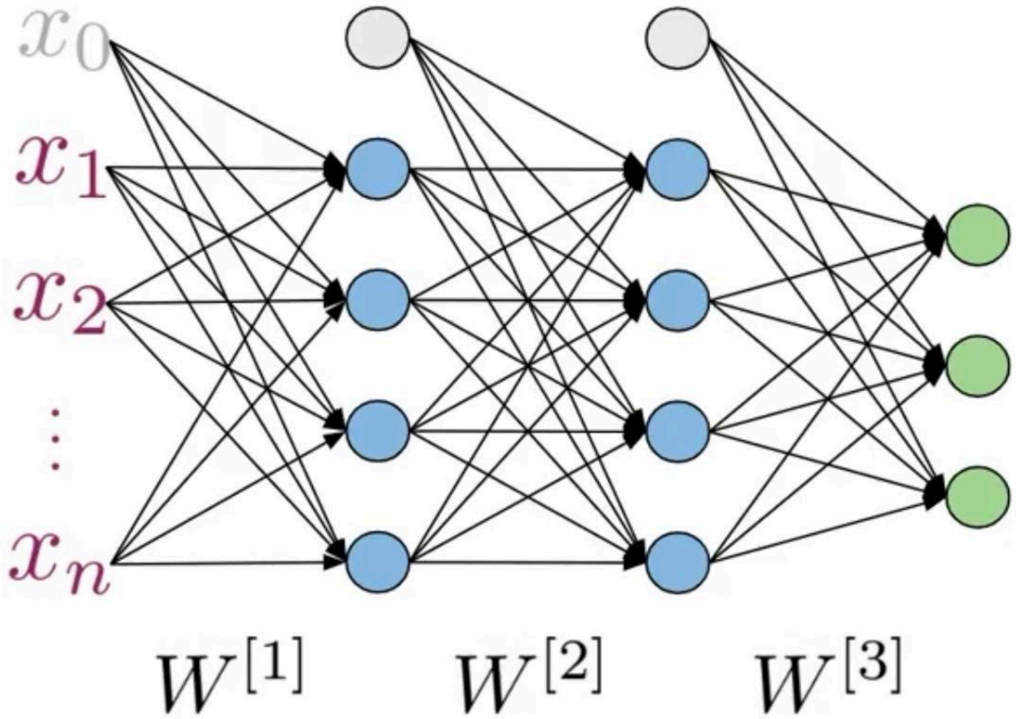
Forward propagation



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

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

Forward propagation

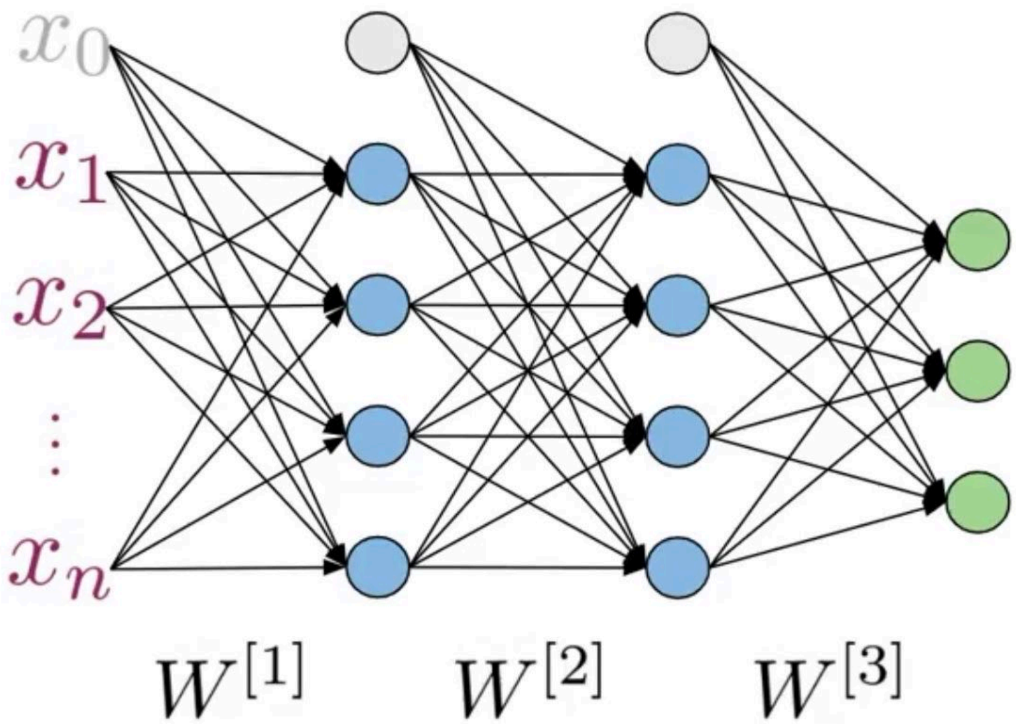


$a^{[i]}$ Activations ith layer

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

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

Forward propagation



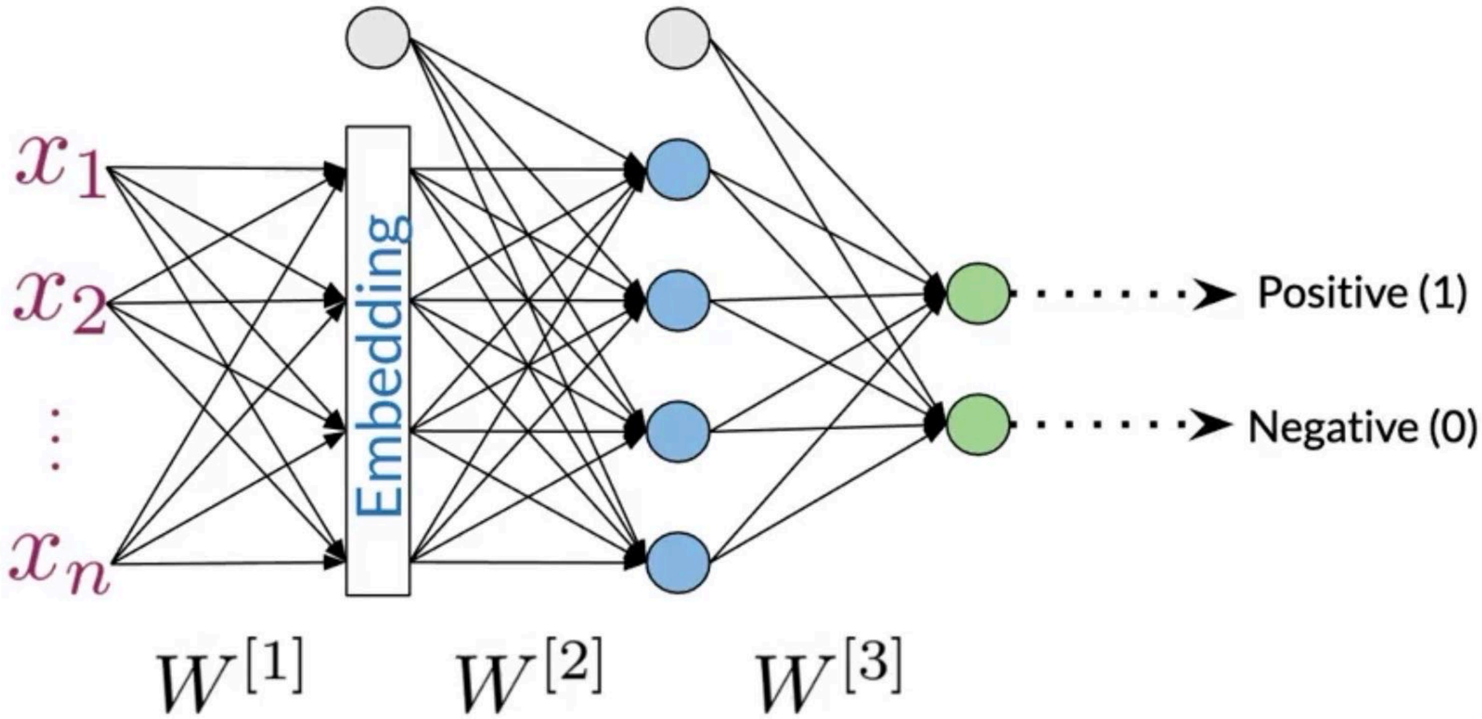
$a^{[i]}$ Activations ith layer

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

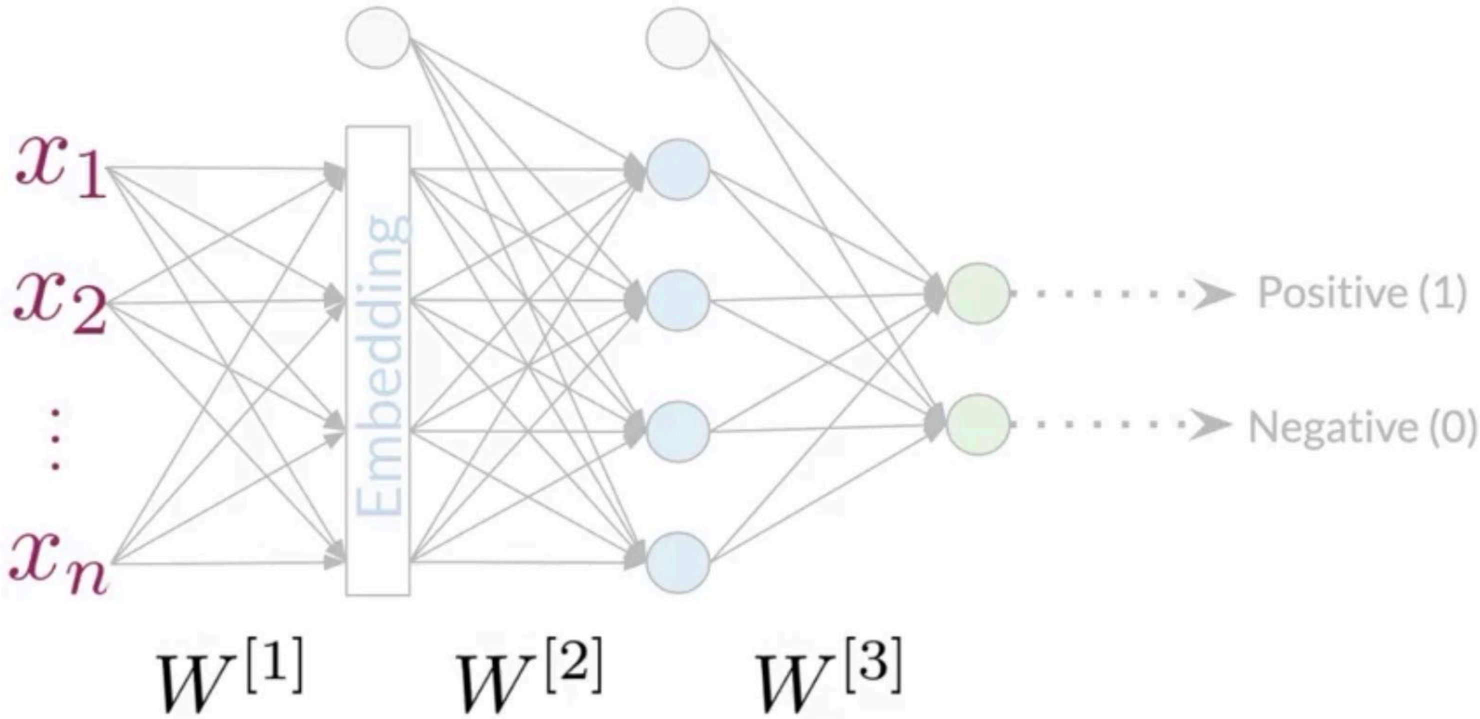
$$z^{[i]} = W^{[i]} a^{[i-1]}$$

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

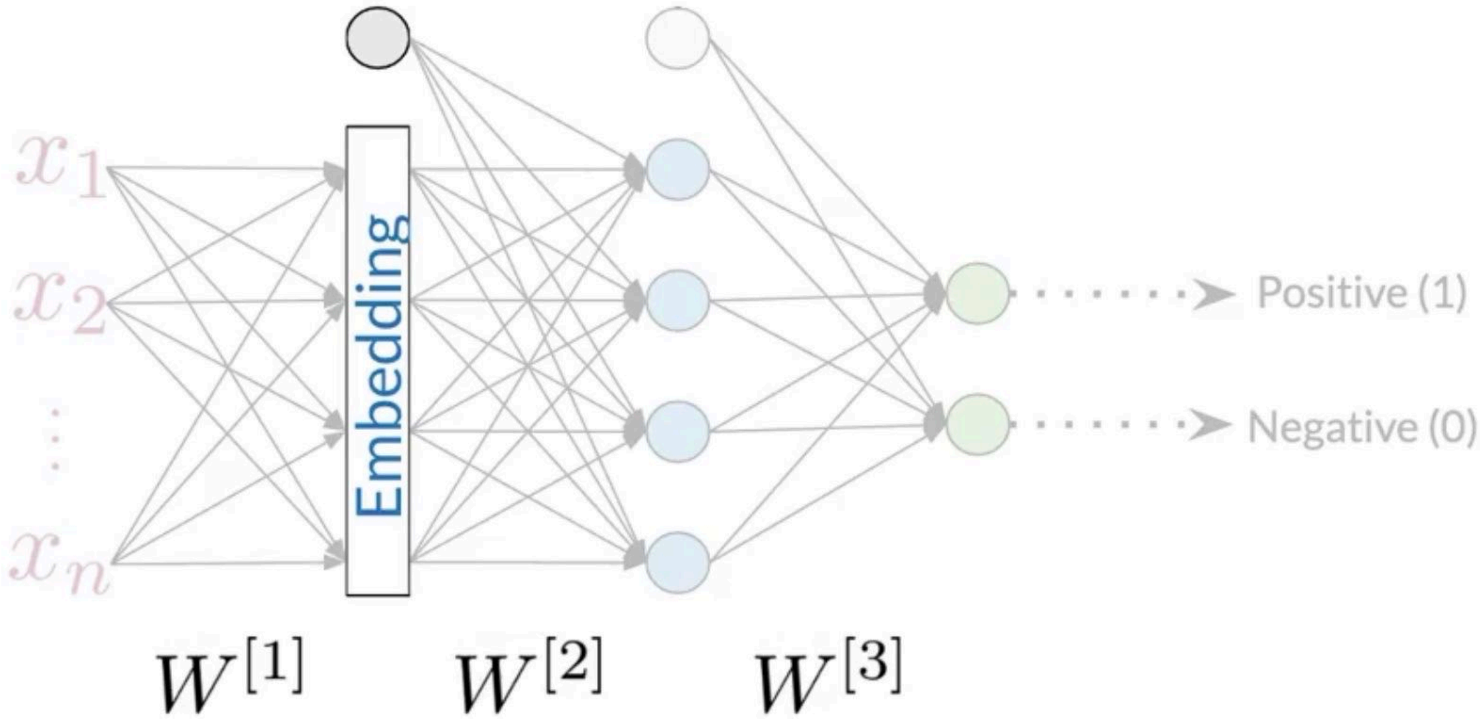
Neural Networks for sentiment analysis



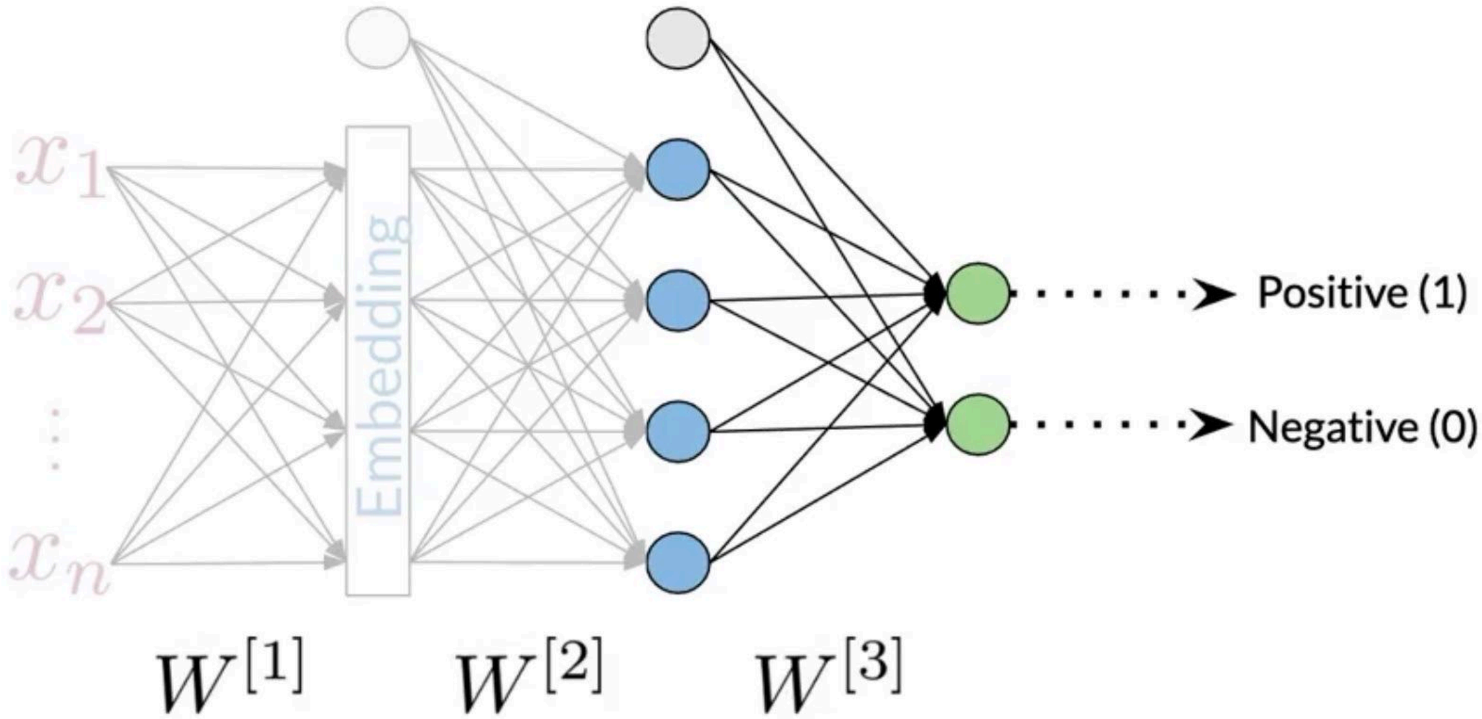
Neural Networks for sentiment analysis



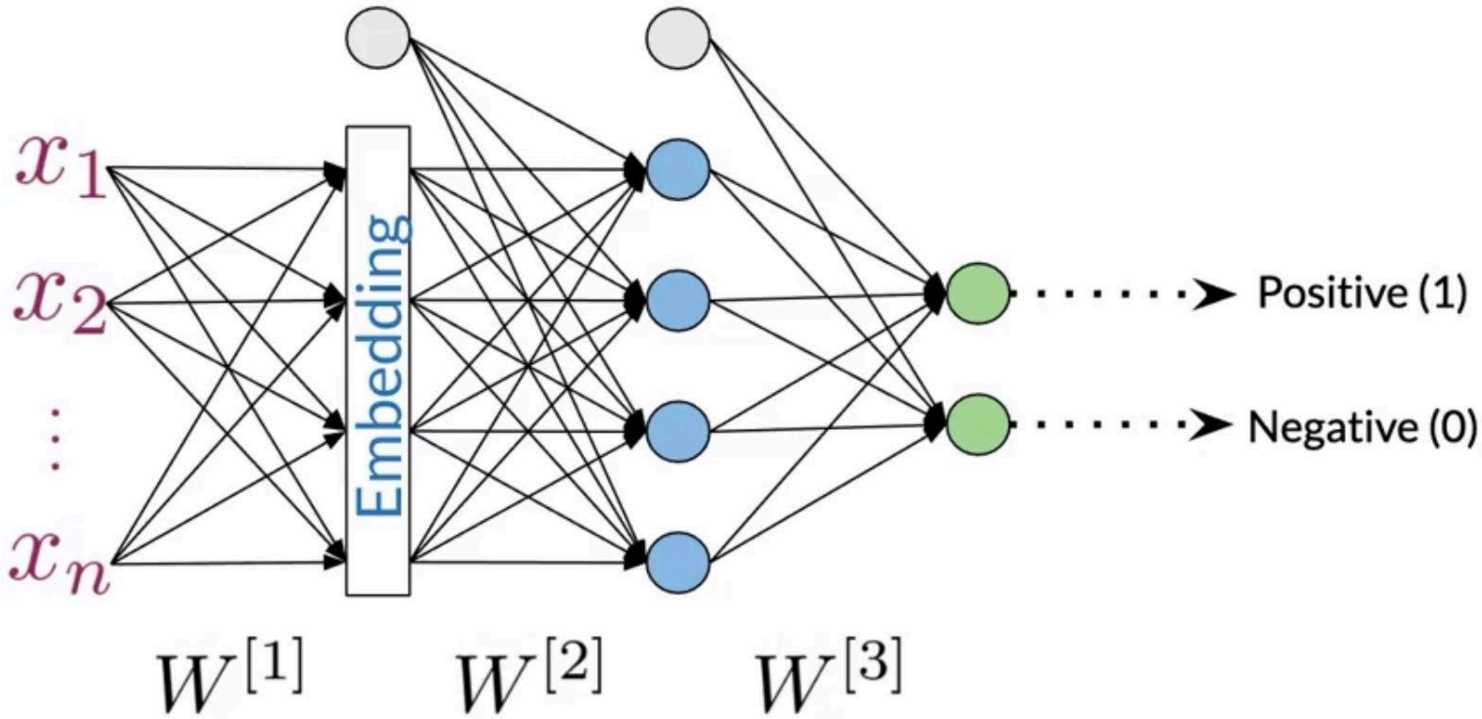
Neural Networks for sentiment analysis



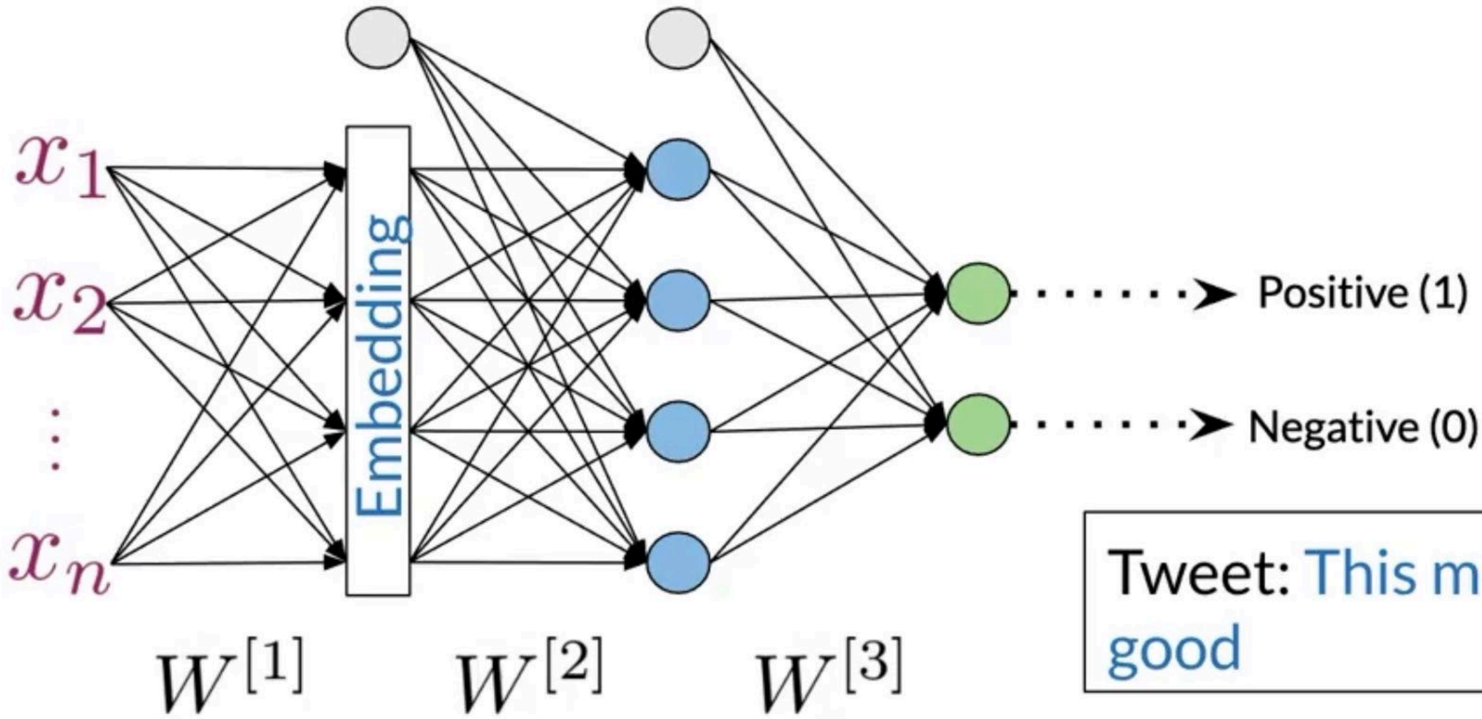
Neural Networks for sentiment analysis



Neural Networks for sentiment analysis



Neural Networks for sentiment analysis



Tweet: This movie was almost good

Initial Representation

Word
a
able
about
...
hand
...
happy
...
zebra

Initial Representation

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

Initial Representation

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

Tweet: This movie was almost good

$\downarrow$

[700 680 720 20 55]

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]

↓

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

To match size of longest tweet

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]

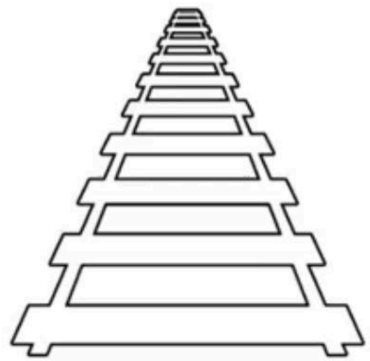
To match size of longest tweet

Summary

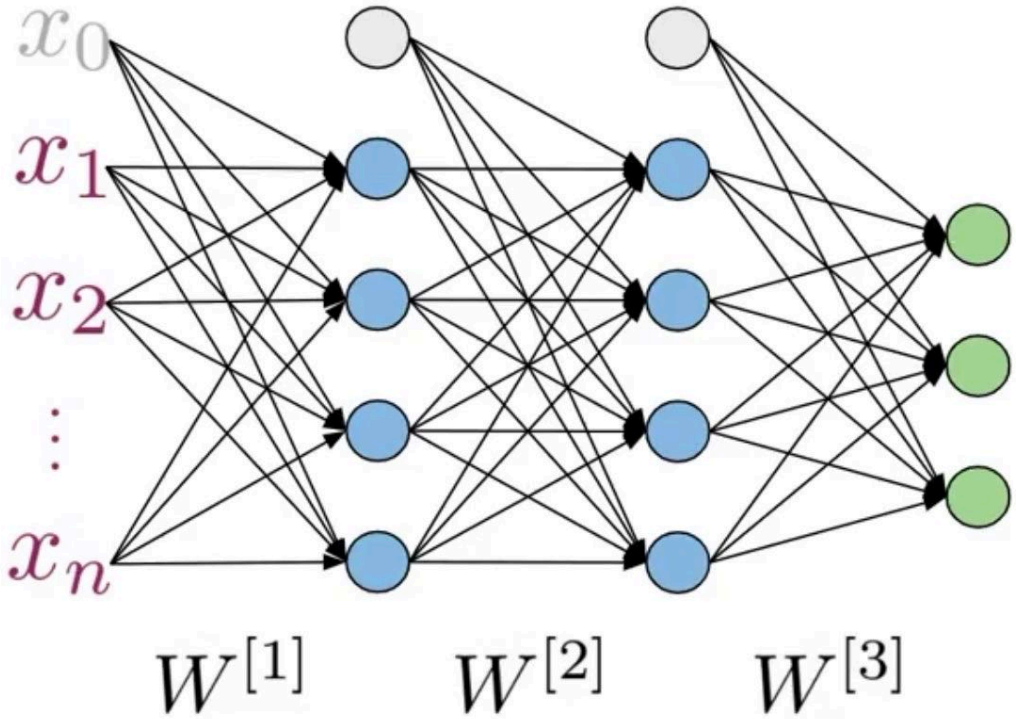
- Structure for sentiment analysis
- Classify complex tweets
- Initial representation

Outline

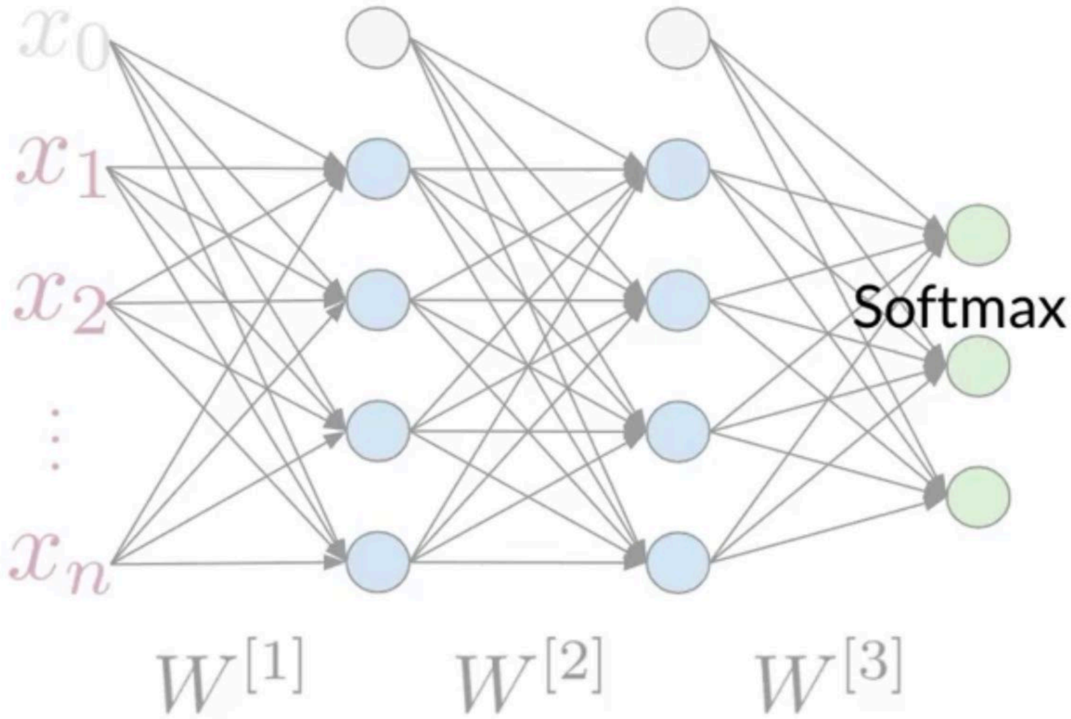
- Define a basic neural network using Trax
- Benefits of Trax



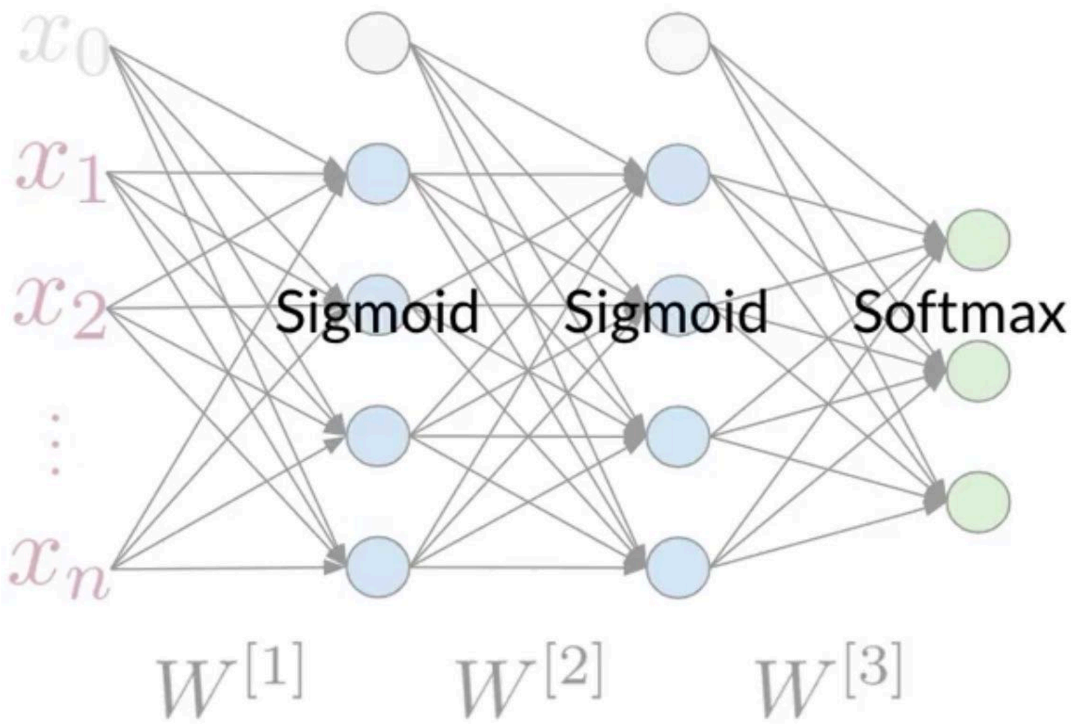
Neural Networks in Trax



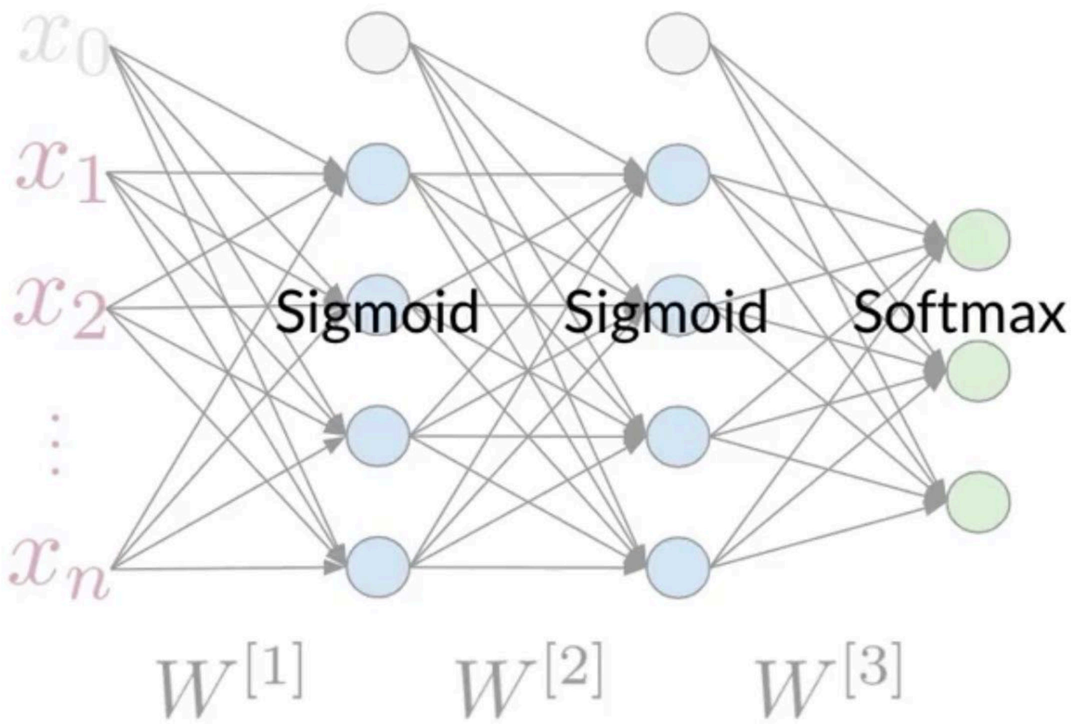
Neural Networks in Trax



Neural Networks in Trax

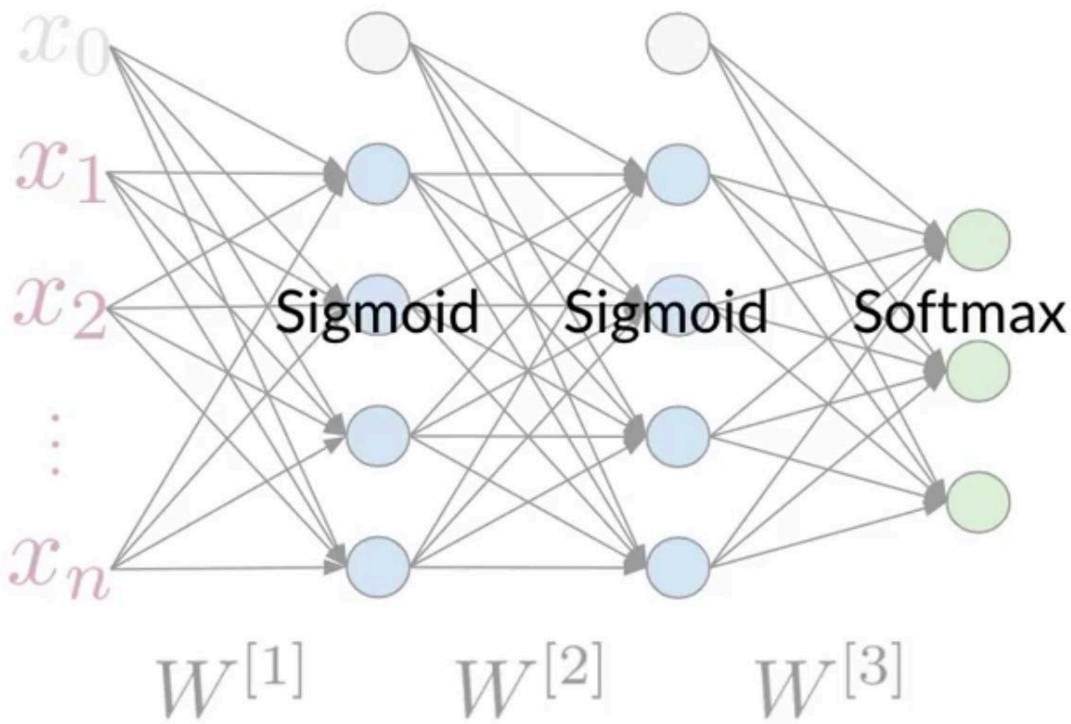


Neural Networks in Trax




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

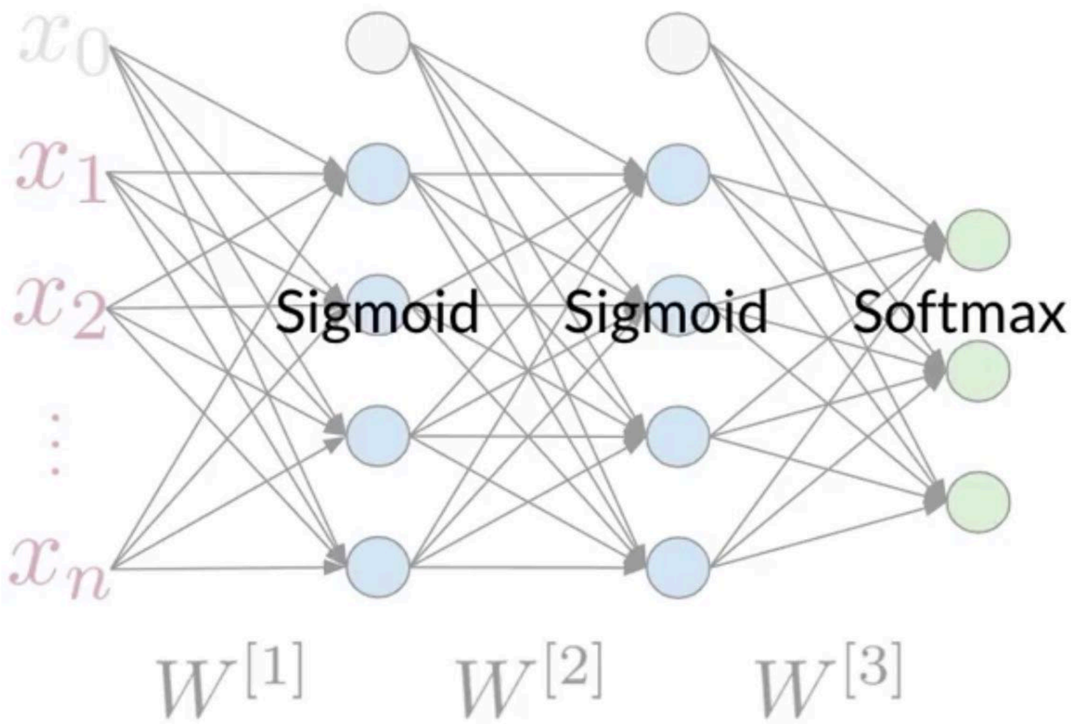
Neural Networks in Trax




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

          tl.Sigmoid(),
```

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

Advantages of using frameworks

- Run fast on CPUs, GPUs and TPUs
- Parallel computing

Advantages of using frameworks

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

Advantages of using frameworks

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

Tensorflow Pytorch

Advantages of using frameworks

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

Tensorflow Pytorch JAX

Summary

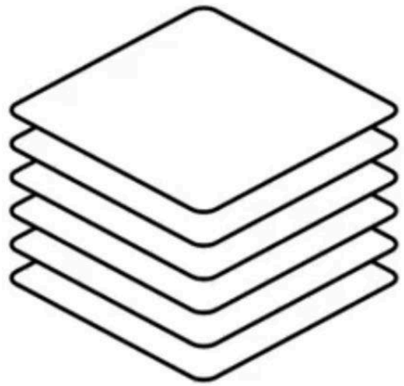
- Order of computation ⟶ Model in Trax
- Benefits from using frameworks

deeplearning.ai

Trax: Layers

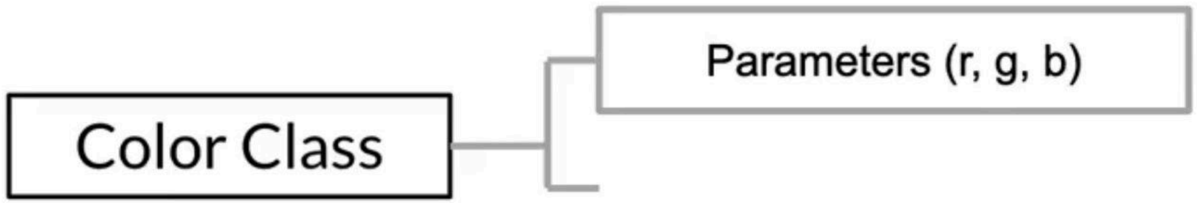
Outline

- How classes work and their implementation

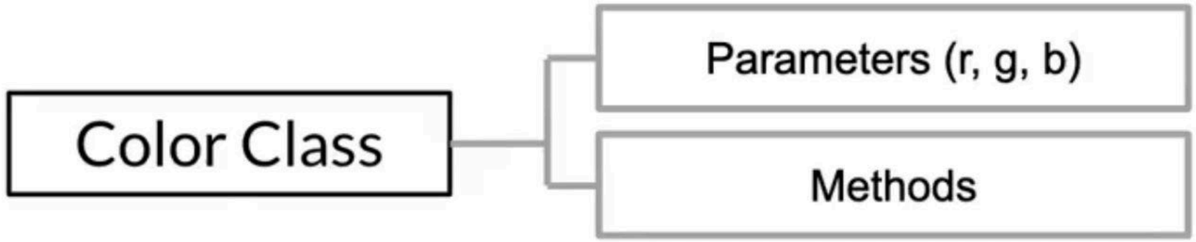


Classes

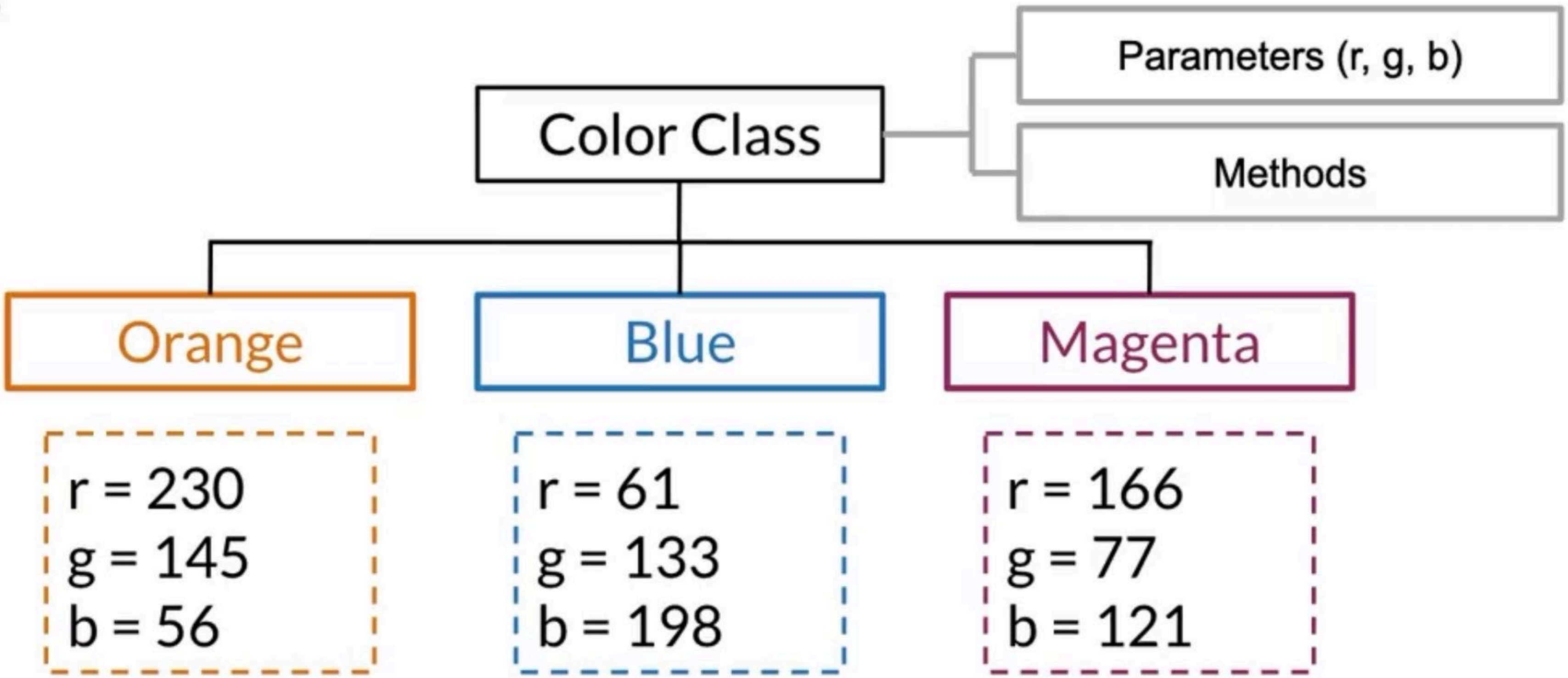
Classes



Classes



Classes


Parameters (r, g, b)
Color Class
Methods
Orange
Blue
Magenta
r = 230
g = 145
b = 56
r = 61
g = 133
b = 198
r = 166
g = 77
b = 121

Classes in Python

Classes in Python

```
class MyClass(Object):

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

Classes in Python

```
class MyClass(Object):

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

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

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)
```

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))
```

Classes in Python

```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)
```

```python
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)
```

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
```

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)
```

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))
```

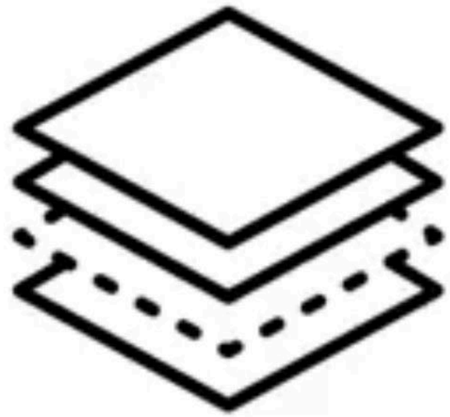
52

Summary

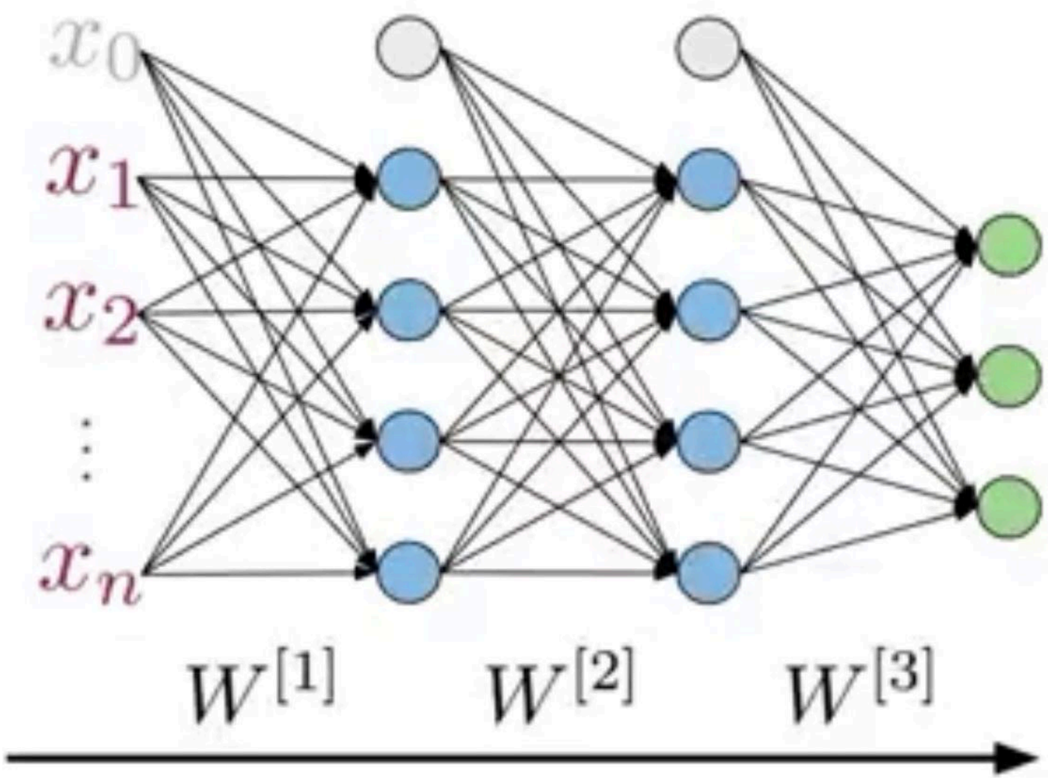
- Classes, subclasses and instances

Outline

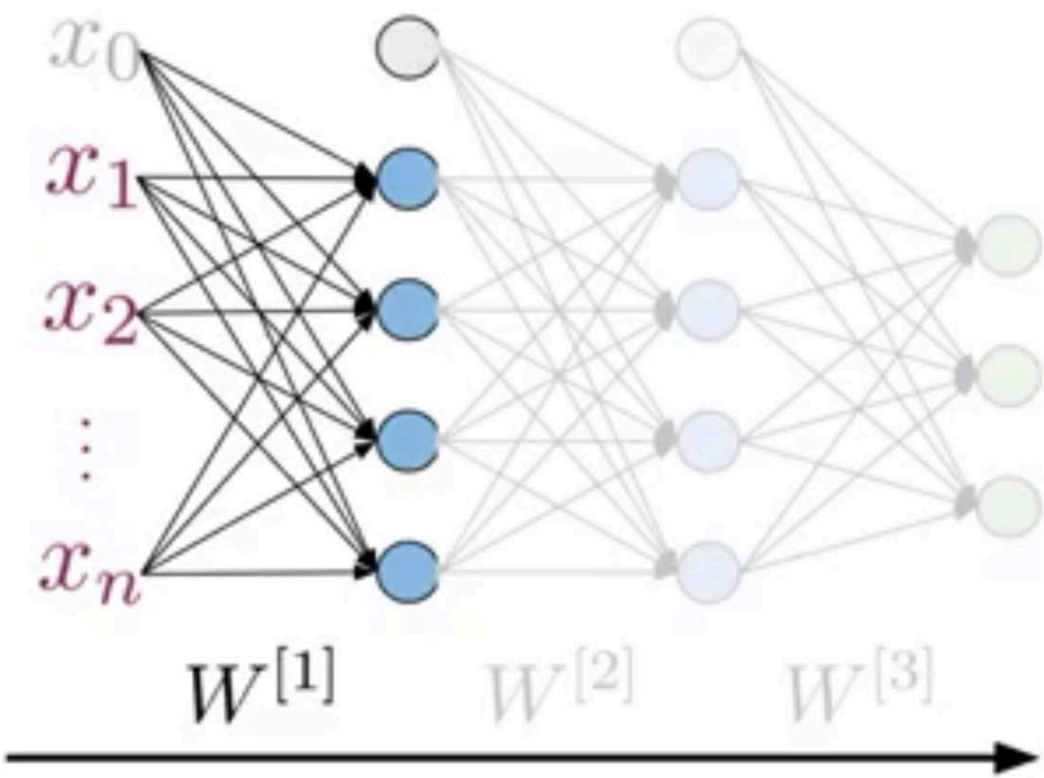
- Dense and ReLU layers



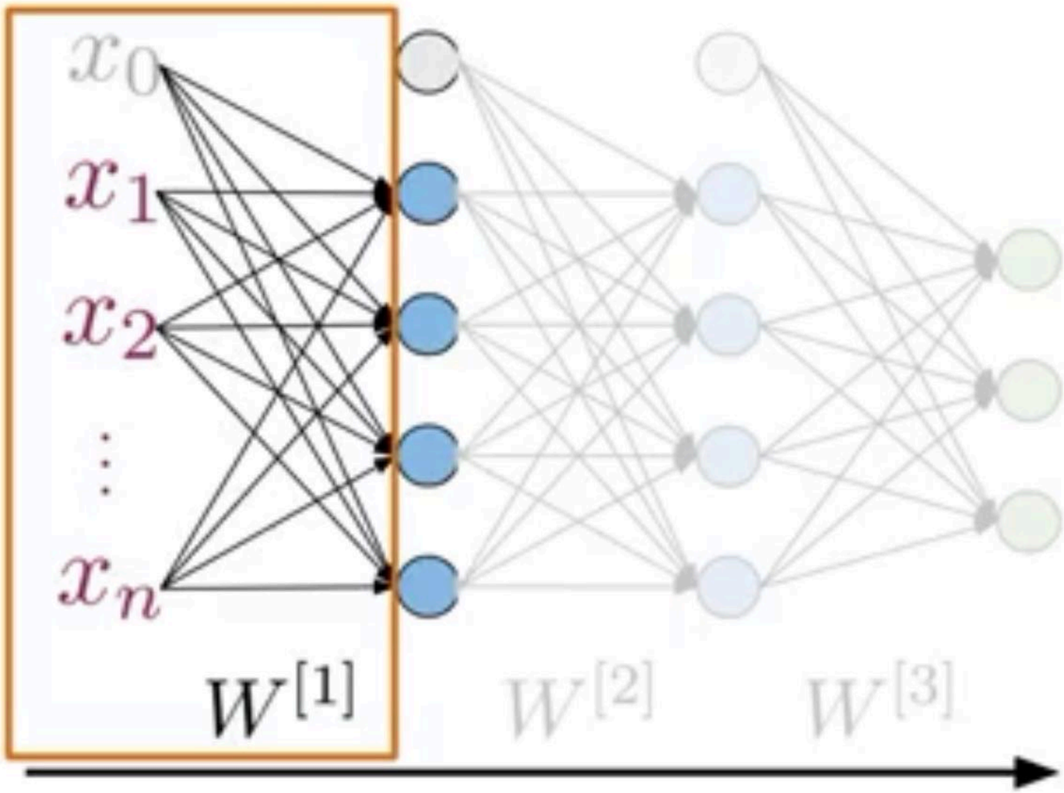
Dense Layer



Dense Layer

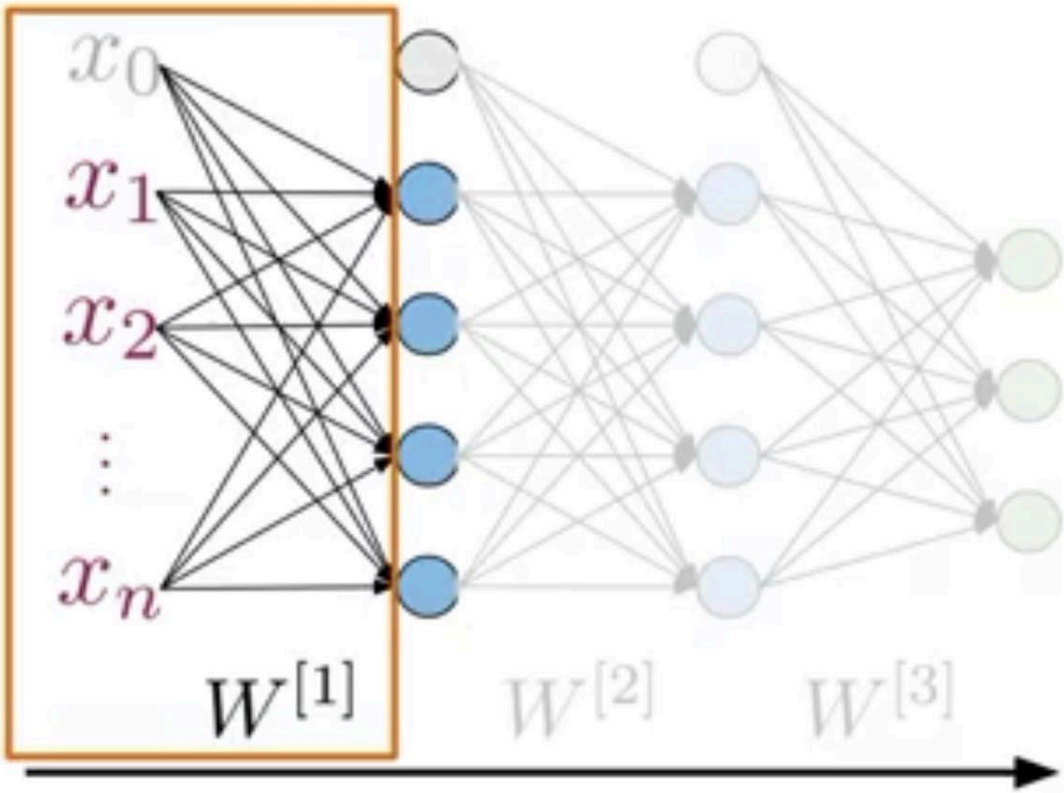


Dense Layer



Fully connected layer

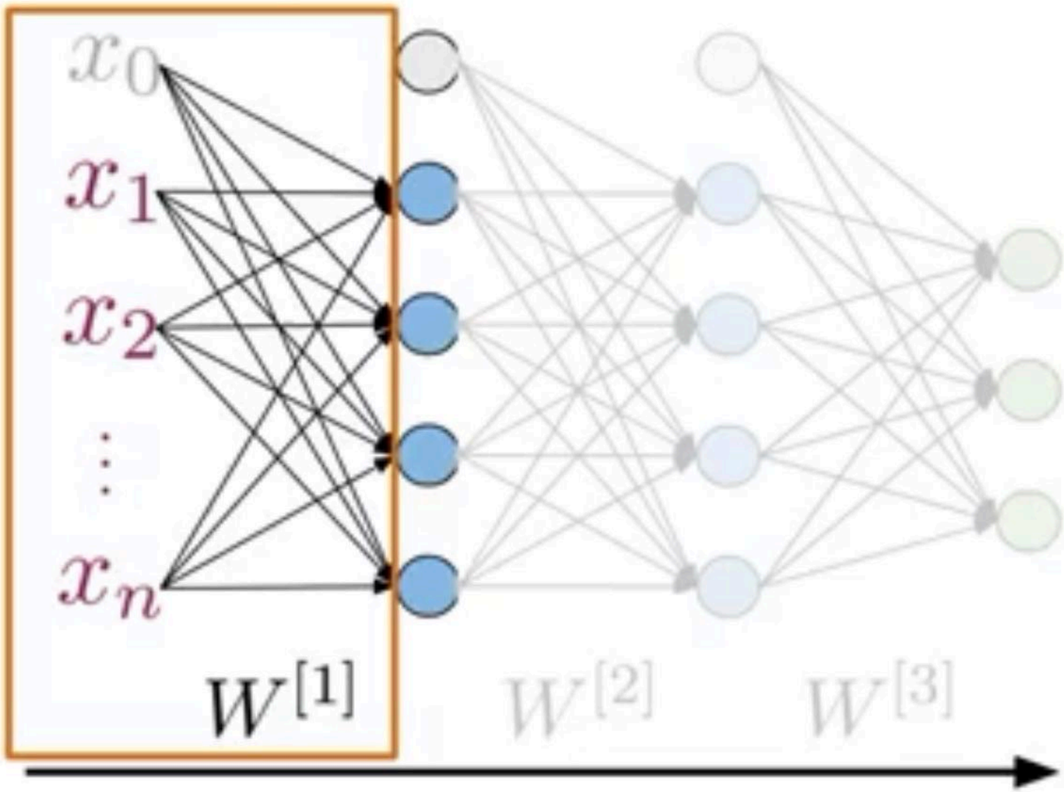
Dense Layer



Fully connected layer

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

Dense Layer

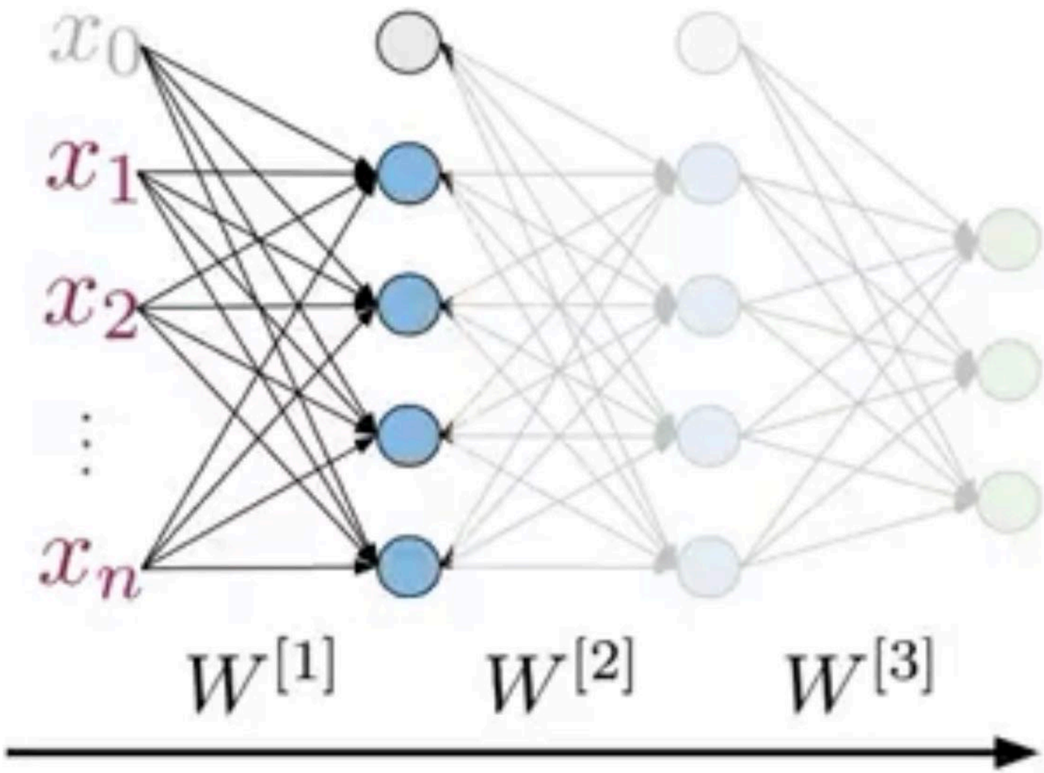


Fully connected layer

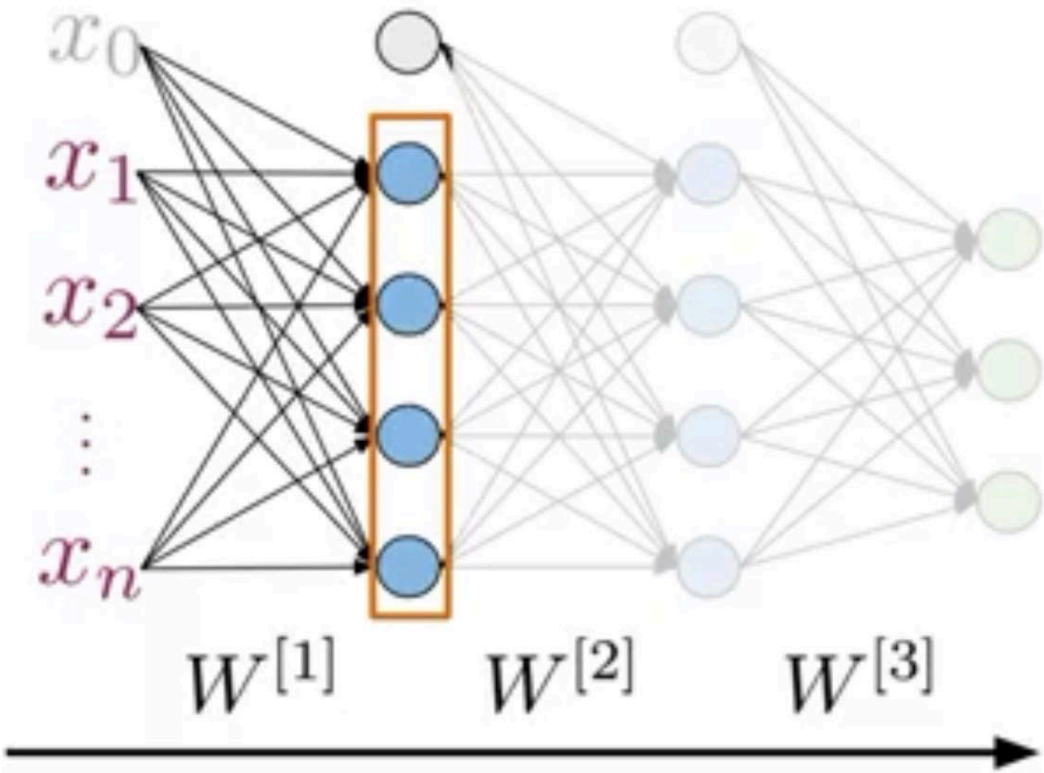
$$z^{[i]} = \boxed{W^{[i]}} a^{[i-1]}$$

Trainable parameters

ReLU Layer



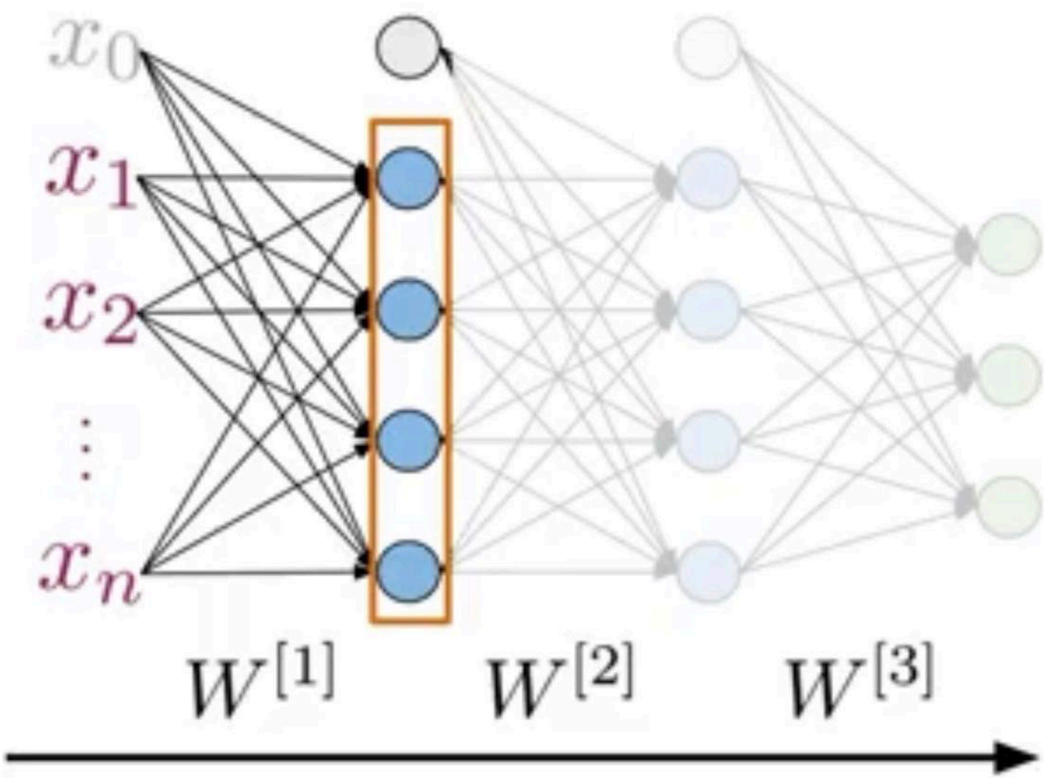
ReLU Layer



ReLU = Rectified linear unit

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

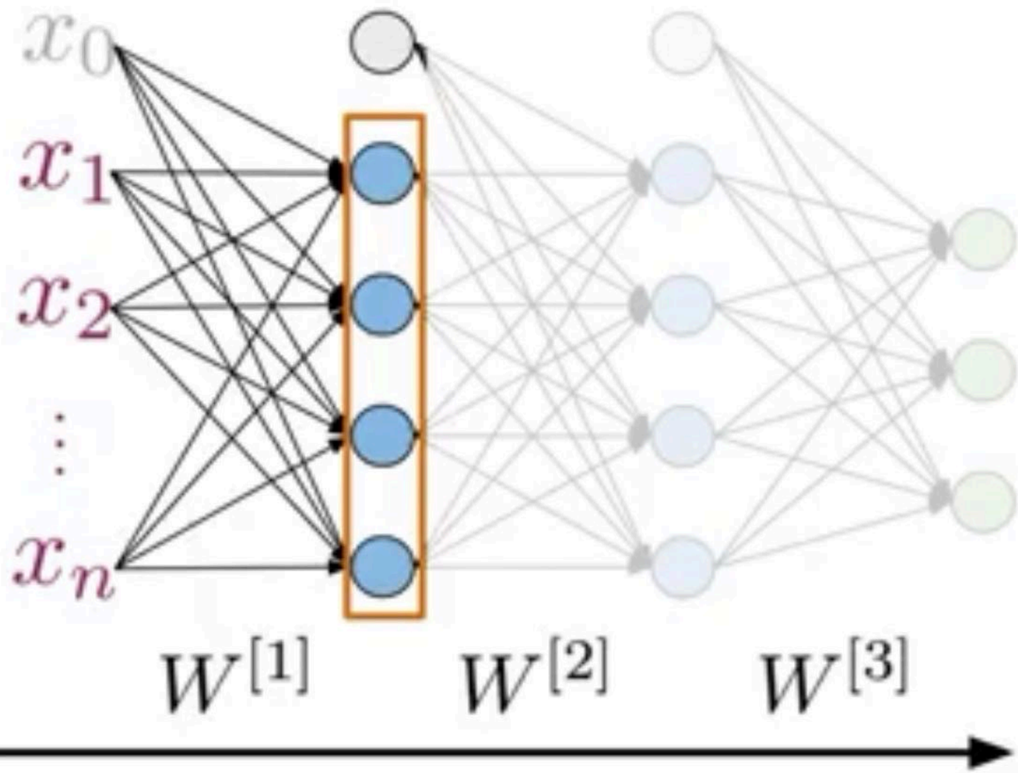
ReLU Layer



ReLU = Rectified linear unit

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

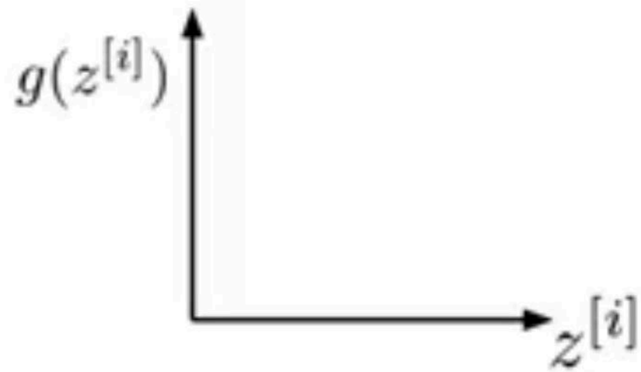
ReLU Layer



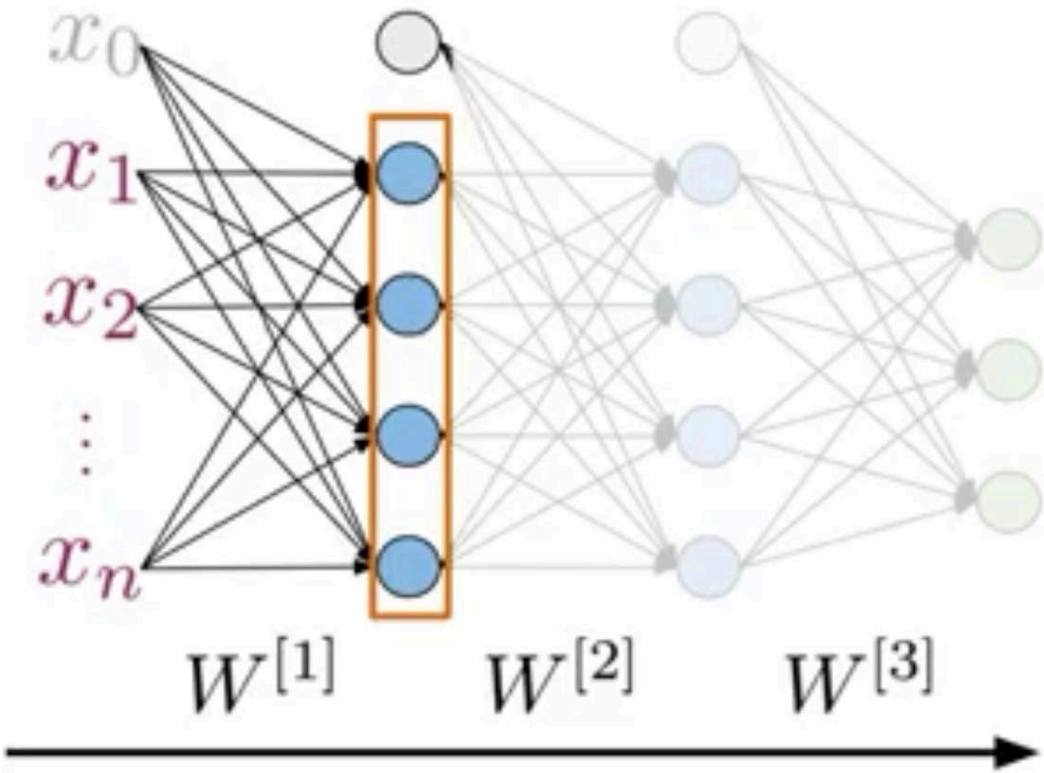
ReLU = Rectified linear unit

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

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



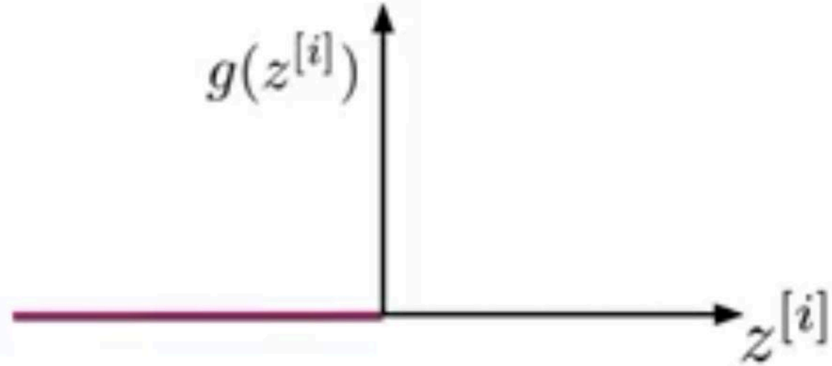
ReLU Layer



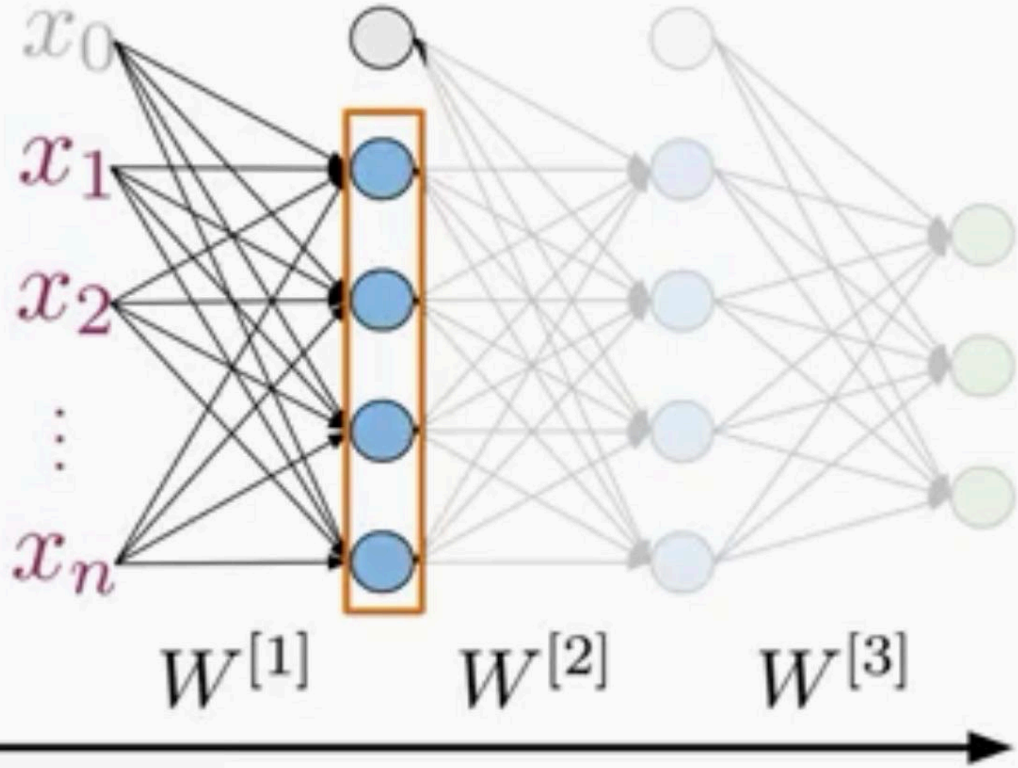
ReLU = Rectified linear unit

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

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



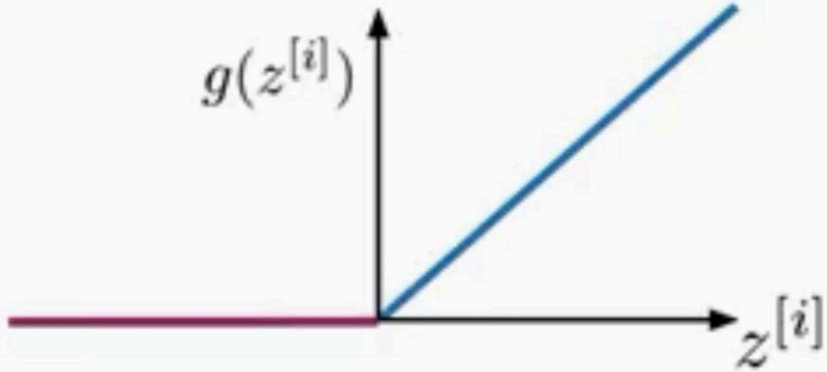
ReLU Layer



ReLU = Rectified linear unit

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

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



Summary

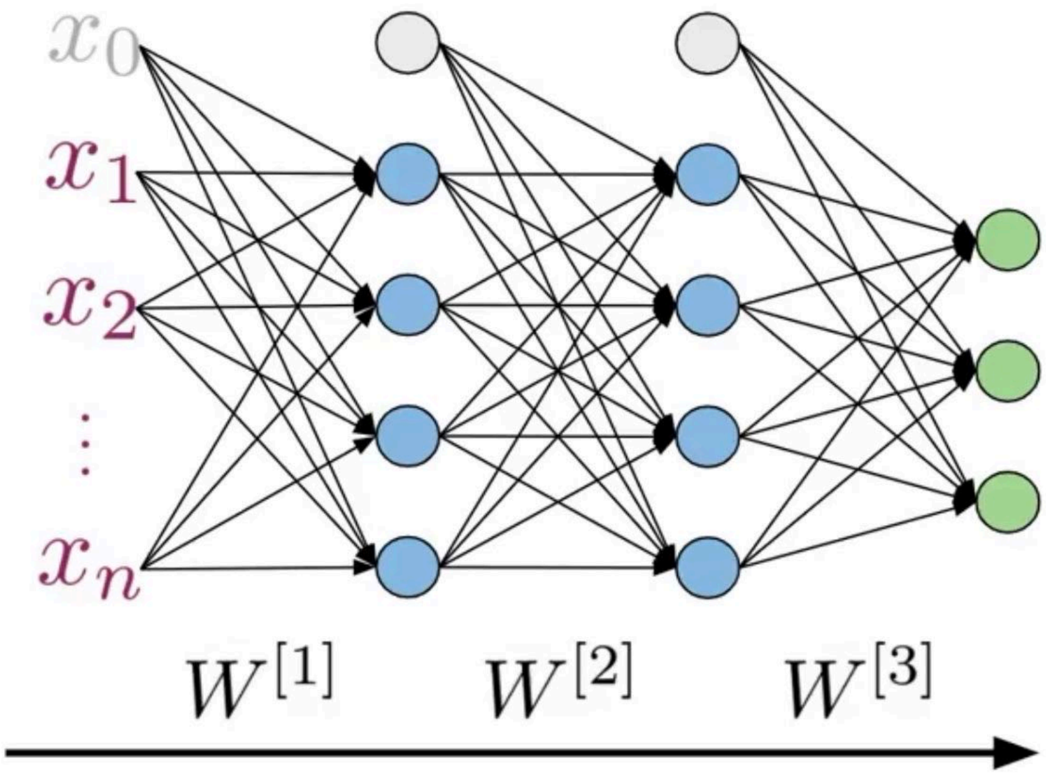
- Dense Layer ⟶ $z^{[i]} = W^{[i]} a^{[i-1]}$
- ReLU Layer ⟶ $g(z^{[i]}) = \max(0, z^{[i]})$

5:38 PM Tue Jul 28

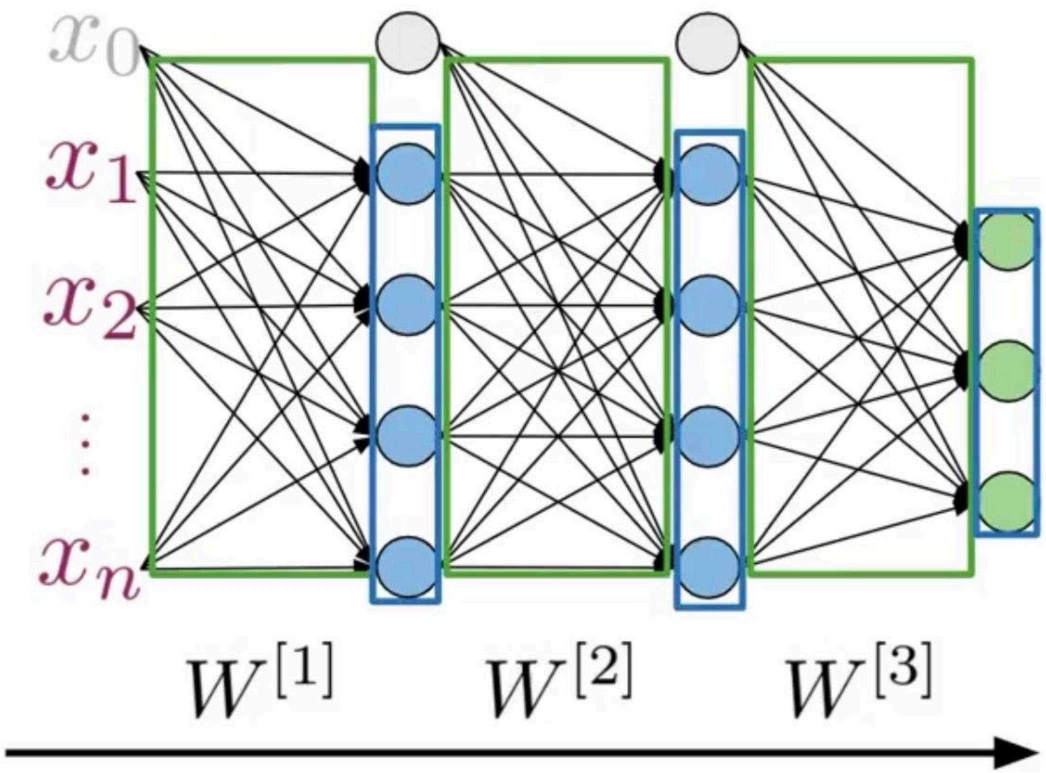
deeplearning.ai

Serial Layer

Serial Layer

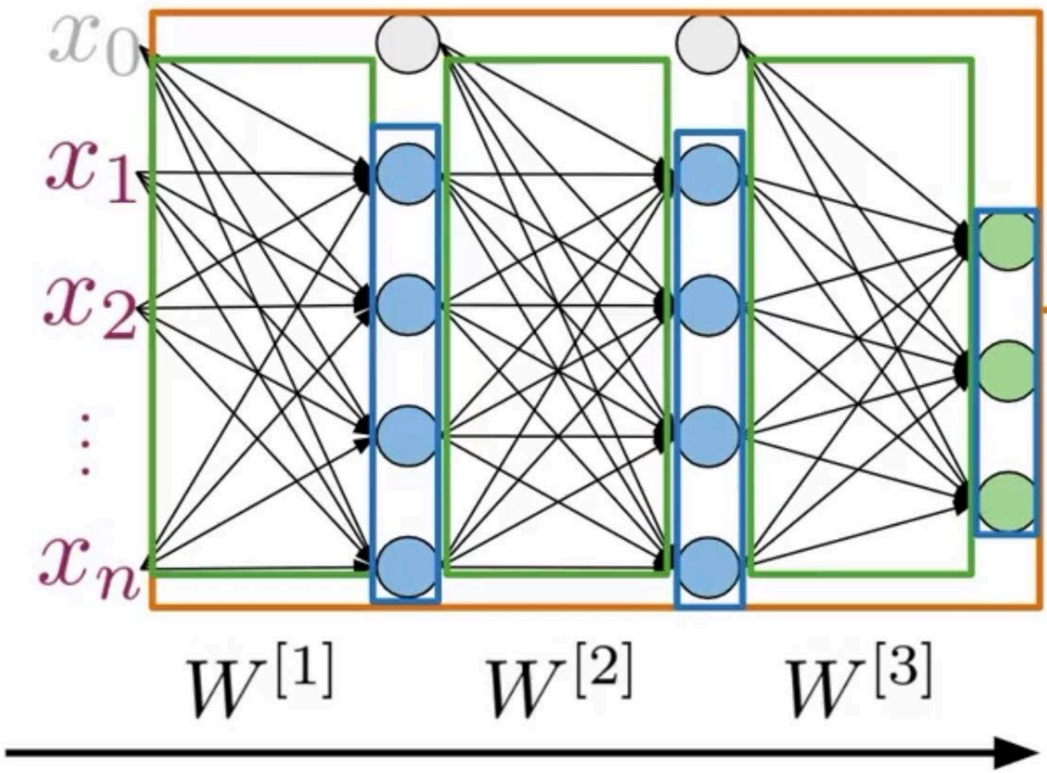


Serial Layer



- Dense Layers
- Activation Layers

Serial Layer



Composition of layers in *serial* arrangement

- Dense Layers
- Activation Layers

Summary

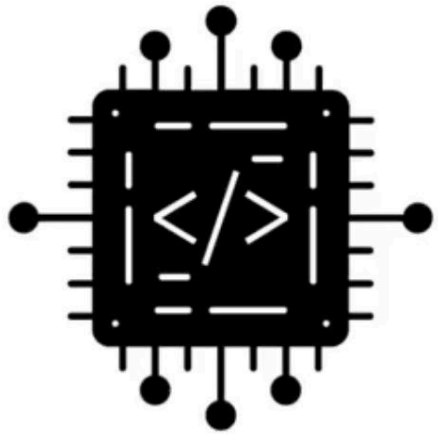
- Serial layer is a composition of sublayers

deeplearning.ai

Trax: Other Layers

Outline

- Embedding layer
- Mean layer



Embedding Layer

Vocabulary
I
am
happy
because
learning
NLP
sad
not

Embedding Layer

Vocabulary	Index
I	1
am	2
happy	3
because	4
learning	5
NLP	6
sad	7
not	8

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

Embedding Layer

Vocabulary	Index		
I	1	$\boxed{0.020}$	$\boxed{0.006}$
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

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

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

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

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

0.020	0.006
-0.003	0.010
0.009	0.010

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

$$\downarrow$$

$$\boxed{\begin{matrix} 0.020 & 0.006 \\ -0.003 & 0.010 \\ 0.009 & 0.010 \end{matrix}}$$

$$\downarrow$$

Mean of the word embeddings

$$\boxed{\begin{matrix} 0.009 \\ 0.009 \end{matrix}}$$

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

$$\downarrow$$

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

$$\downarrow$$

Mean of the word embeddings

$$\begin{bmatrix} 0.009 \\ 0.009 \end{bmatrix}$$

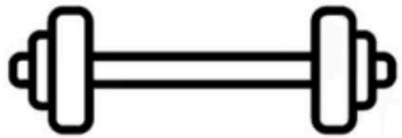
No trainable parameters

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

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

Computing gradients in Trax

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

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

Gradient

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
```

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)
```

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)
```

Training with grad()

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

In a loop

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

Training with grad()

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

In a loop

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

Forward and
Back-propagation

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!