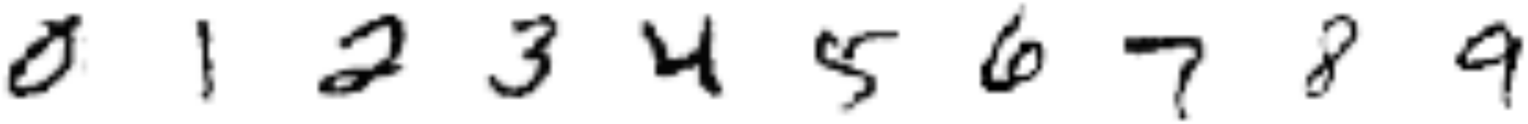
Machine Learning 07

Kihyun Shin
DMSE, HBNU

TheMAD

Multiclass classification

MNIST example



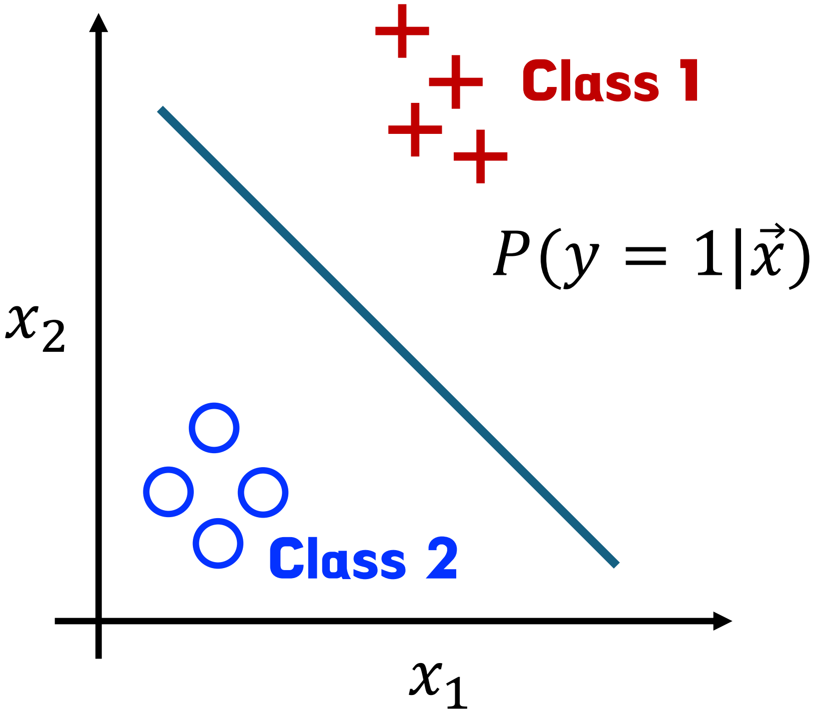
$y = 0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8 \quad 9$

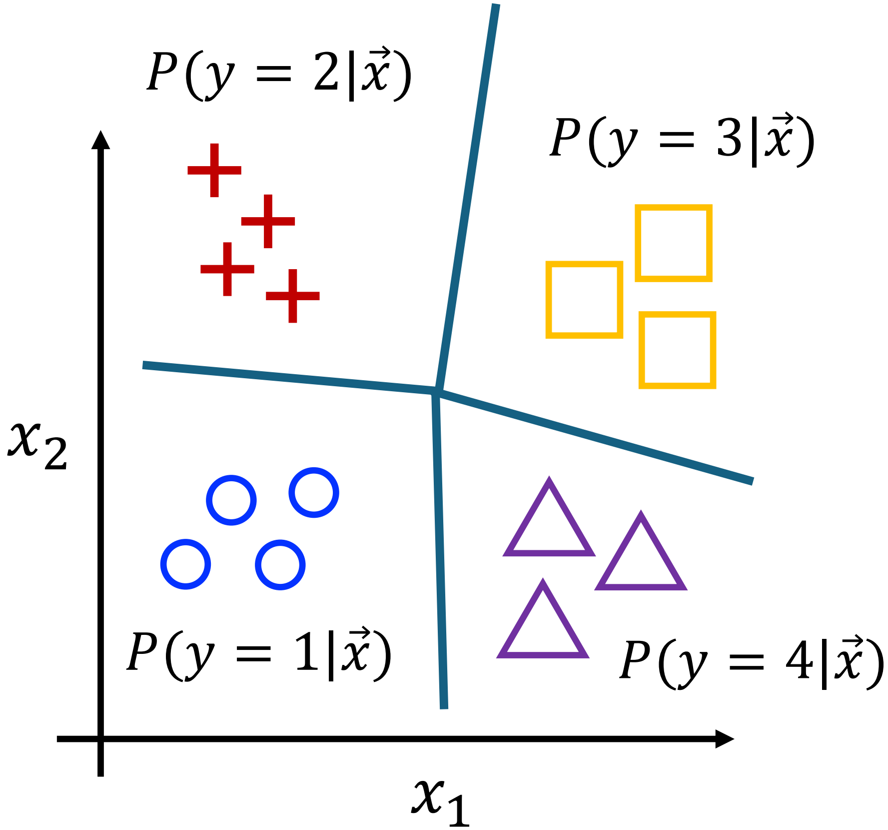
$x \rightarrow$  $\quad y = 7$

Multiclass classification problem

Target y can take more than two possible values

Multiclass classification example





Softmax regression algorithm

Softmax regression

Logistic regression (2 possible output values)

$$z = \vec{w} \cdot \vec{x} + b$$

0.71

$$+ \quad a_1 = g(z) = \frac{1}{1+e^{-z}} = P(y=1|\vec{x})$$

$$\bigcirc \quad a_2 = 1 - a_1 = P(y=0|\vec{x})$$

0.29

Softmax regression (N possible outputs)

$$z_j = \vec{w_j} \cdot \vec{x} + b_j$$

$$a_j = \frac{e^{z_j}}{\sum_{k=1}^{N} e^{z_k}} = P(y=j|\vec{x})$$

Note: $a_1 + a_2 + a_3 + \cdots + a_N = 1$

Softmax regression (4 possible outputs)

$$z_1 = \vec{w_1} \cdot \vec{x} + b_1 \qquad a_1 = \frac{e^{z_1}}{e^{z_1}+e^{z_2}+e^{z_3}+e^{z_4}}$$

$+$ $\bigcirc$ $\square$ $\triangle$

$$= P(y=1|\vec{x})$$ 0.30

$$z_2 = \vec{w_2} \cdot \vec{x} + b_2 \qquad a_2 = \frac{e^{z_2}}{e^{z_1}+e^{z_2}+e^{z_3}+e^{z_4}}$$

$$= P(y=2|\vec{x})$$ 0.20

$$z_3 = \vec{w_3} \cdot \vec{x} + b_3 \qquad a_3 = \frac{e^{z_3}}{e^{z_1}+e^{z_2}+e^{z_3}+e^{z_4}}$$

$$= P(y=3|\vec{x})$$ 0.15

$$z_4 = \vec{w_4} \cdot \vec{x} + b_4 \qquad a_4 = \frac{e^{z_4}}{e^{z_1}+e^{z_2}+e^{z_3}+e^{z_4}}$$

$$= P(y=4|\vec{x})$$

Simplified cost function

$$L\left(f\left(x^{(i)}\right), y^{(i)}\right) = -y^{(i)} \log\left(f_{w,b}\left(x^{(i)}\right)\right) - \left(1 - y^{(i)}\right) \log\left(1 - f_{w,b}\left(x^{(i)}\right)\right)$$

$$\underbrace{J(w,b)}_{\text{cost}} = \frac{1}{m}\sum_{i=1}^{m} \underbrace{L\left(f\left(x^{(i)}\right), y^{(i)}\right)}_{\text{loss}}$$

$$= -\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \log\left(f_{w,b}\left(x^{(i)}\right)\right) + \left(1 - y^{(i)}\right) \log\left(1 - f_{w,b}\left(x^{(i)}\right)\right)$$

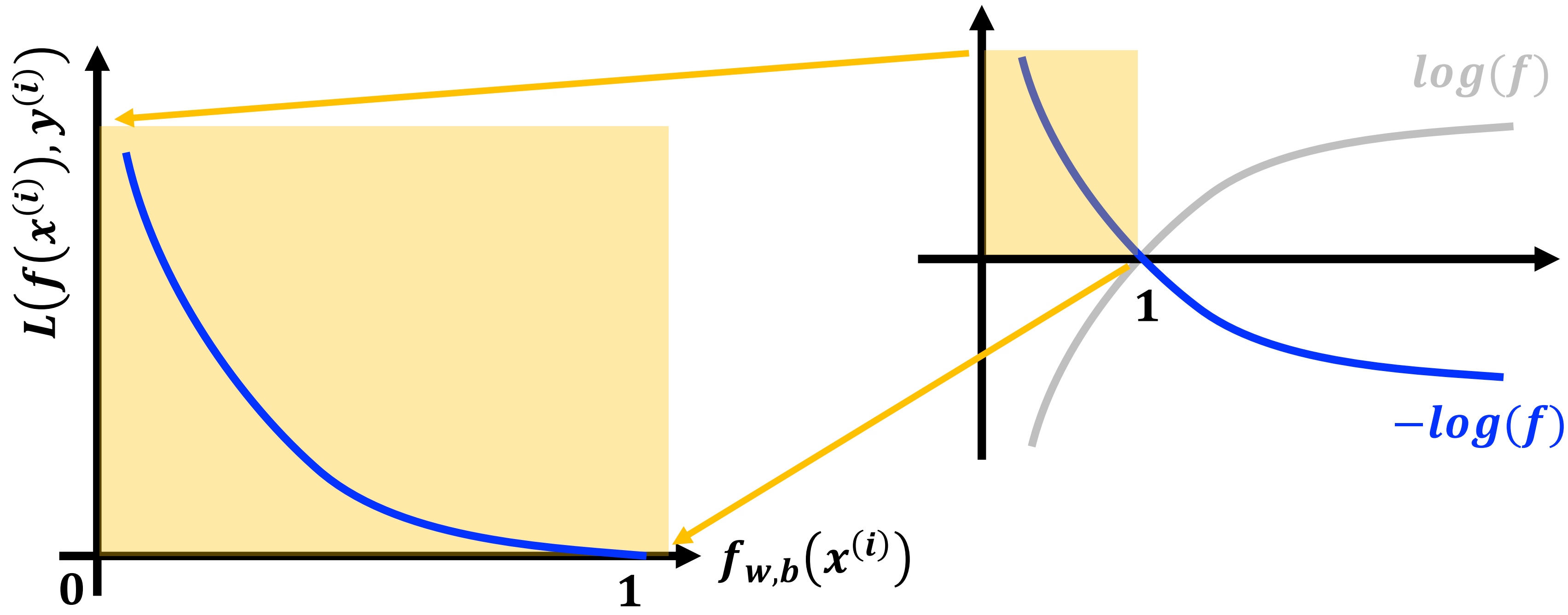
This particular cost function derived from statistics using a statistical principle called maximum likelihood estimation (idea from statistics on how to efficiently find parameters for different models)

→ Pretty much everyone uses to train logistic regression

TheMAD

Loss function

$$L\left(f\left(x^{(i)}\right), y^{(i)}\right) = \begin{cases} -log\left(f_{w,b}\left(x^{(i)}\right)\right) & if \;\; y^{(i)} = 1 \\ -log\left(1 - f_{w,b}\left(x^{(i)}\right)\right) & if \;\; y^{(i)} = 0 \end{cases}$$

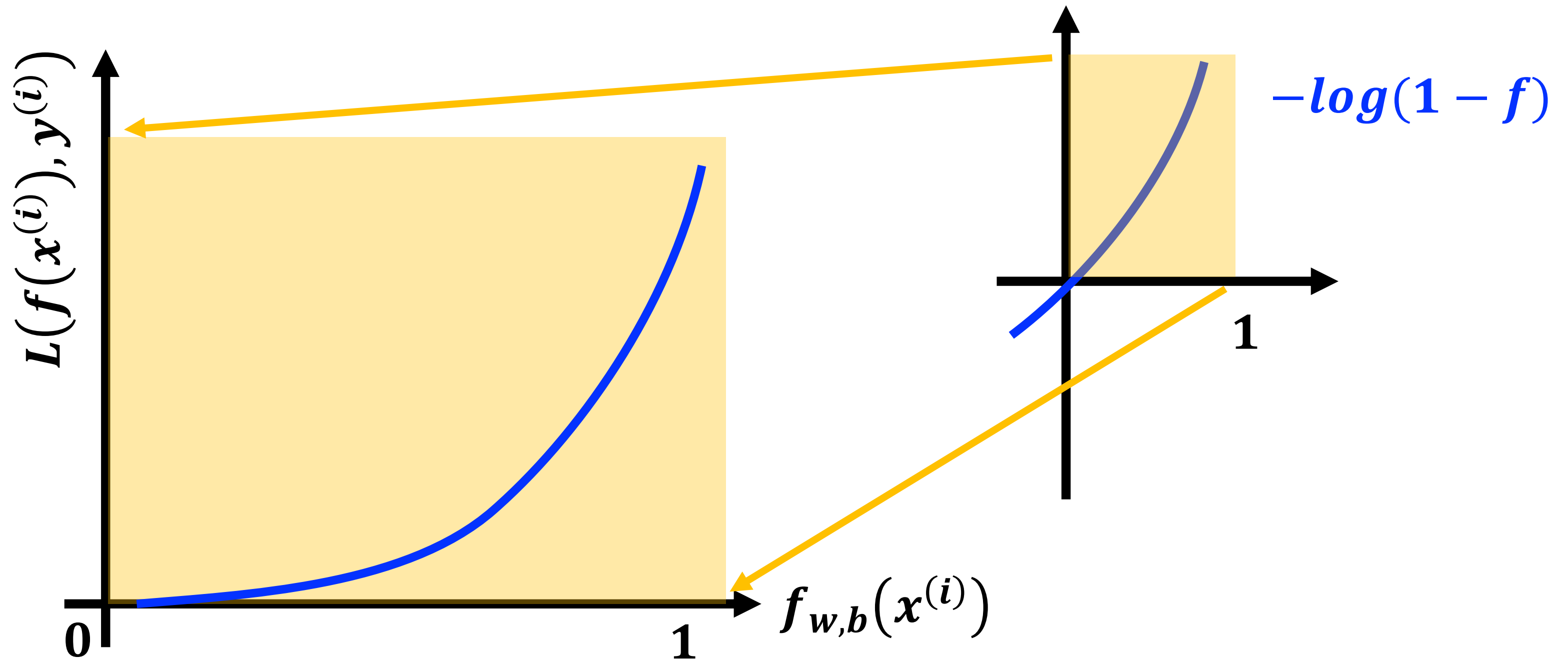


As $f_{w,b}\left(x^{(i)}\right)$ → 1 then loss → 0

As $f_{w,b}\left(x^{(i)}\right)$ → 0 then loss → infinite

Loss function

$$L\left(f\left(x^{(i)}\right), y^{(i)}\right) = \begin{cases} -log\left(f_{w,b}\left(x^{(i)}\right)\right) & if \;\; y^{(i)} = 1 \\ -log\left(1 - f_{w,b}\left(x^{(i)}\right)\right) & if \;\; y^{(i)} = 0 \end{cases}$$



As $f_{w,b}\left(x^{(i)}\right)$ → 1 then loss → infinite

As $f_{w,b}\left(x^{(i)}\right)$ → 0 then loss → 0

TheMAD

Cost

Logistic regression

$$z = \vec{w} \cdot \vec{x} + b$$

$$a_1 = g(z) = \frac{1}{1 + e^{-z}} = P(y = 1|\vec{x})$$

$$a_2 = 1 - a_1 \qquad = P(y = 0|\vec{x})$$

$$Loss = -yloga_1 - (1 - \mathrm{y})\log(1 - a_1)$$

$$J(\vec{w}, b) = average\ loss$$

Softmax regression

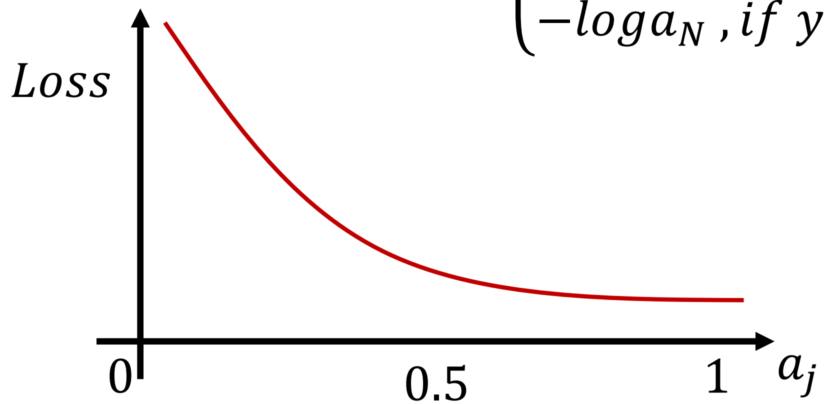
$$a_1 = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + \cdots + e^{z_N}} = P(y = 1|\vec{x})$$

$$\vdots$$

$$a_N = \frac{e^{z_N}}{e^{z_1} + e^{z_2} + \cdots + e^{z_N}} = P(y = N|\vec{x})$$

Cross-entropy loss

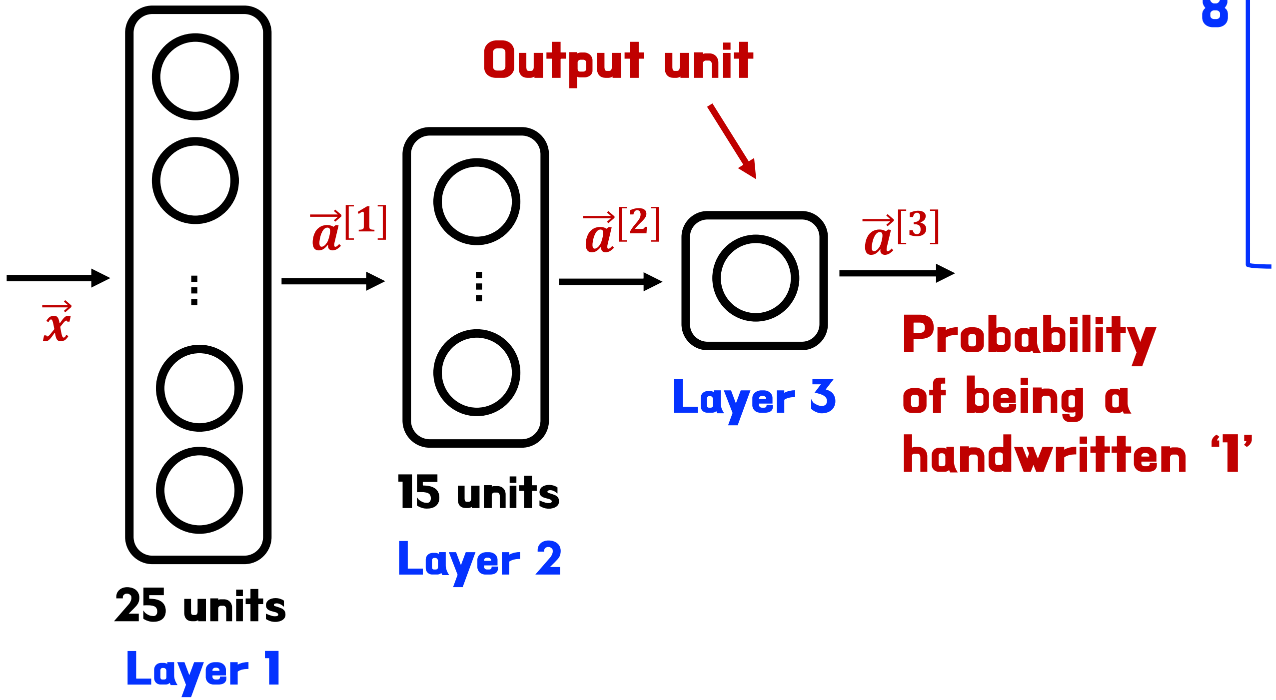
$$Loss(a_1, a_2, \dots, a_N, y) = \begin{cases} -loga_1, if\ y = 1 \\ -loga_2, if\ y = 2 \\ \cdots \\ -loga_N, if\ y = N \end{cases}$$



Neural Network with Softmax output

Digit images 0 1

Label 0 1

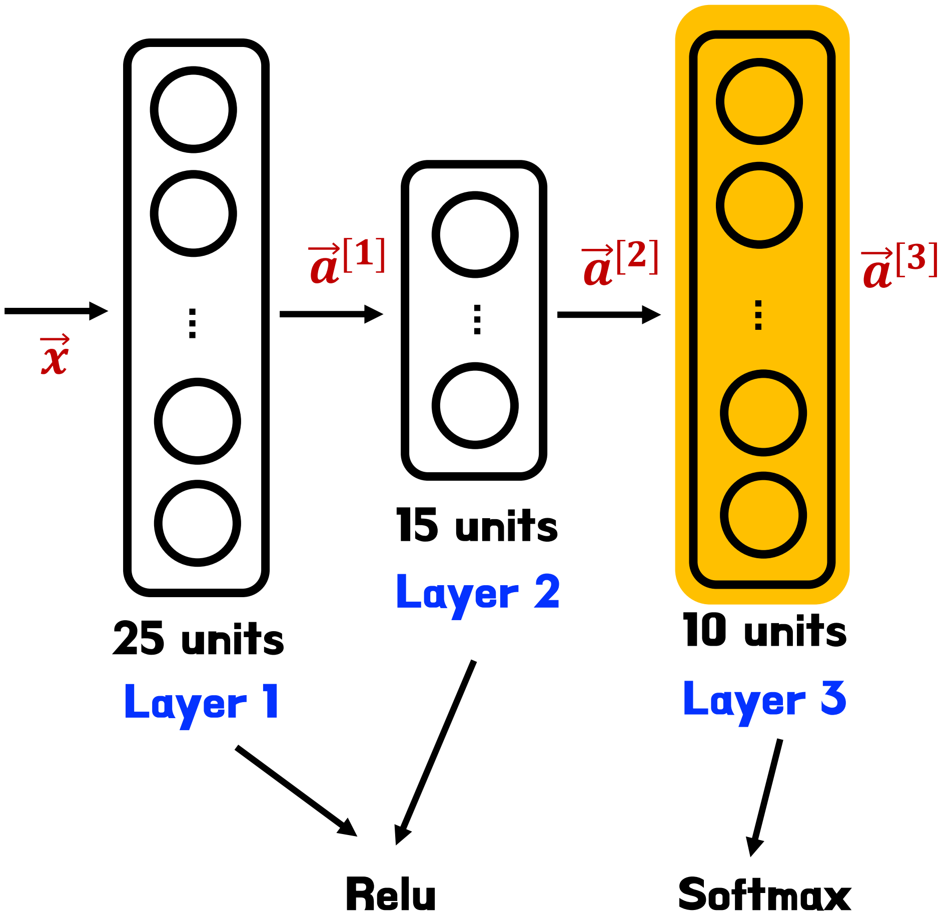


8 × 8

255	255	255	255	255	255	255	255
255	255	255	0	255	255	255	255
255	255	0	0	255	255	255	255
255	255	255	0	255	255	255	255
255	255	255	0	255	255	255	255
255	255	255	0	255	255	255	255
255	255	0	0	0	255	255	255
255	255	255	255	255	255	255	255

Hidden units (neurons)

Neural Network with Softmax output



$$z_1^{[3]} = \overrightarrow{\boldsymbol{w_1}}^{[3]} \cdot \vec{\boldsymbol{a}}^{[2]} + b_1^{[3]} \qquad a_1^{[3]} = \frac{e^{z_1^{[3]}}}{e^{z_1^{[3]}} + \cdots + e^{z_{10}^{[3]}}}$$

$$= P(y = 1|\vec{x})$$

$$\vdots$$

$$z_{10}^{[3]} = \overrightarrow{\boldsymbol{w_{10}}}^{[3]} \cdot \vec{\boldsymbol{a}}^{[2]} + b_{10}^{[3]} \qquad a_{10}^{[3]} = \frac{e^{z_{10}^{[3]}}}{e^{z_1^{[3]}} + \cdots + e^{z_{10}^{[3]}}}$$

$$= P(y = 10|\vec{x})$$

Logistic regression

$$a_1^{[3]} = g(z_1^{[3]}) \qquad a_2^{[3]} = g(z_2^{[3]})$$

Softmax

$$\vec{\boldsymbol{a}}^{[3]} = (a_1^{[3]}, \dots, a_{10}^{[3]}) = g(z_1^{[3]}, \dots, z_{10}^{[3]})$$

MNIST with softmax

1. **Specify the model**

```
import tensorflow as tf
from tensorflow.keras import Sequential
From tensorflow.keras.layers import Dense
model = Sequential([
  Dense (units=25, activation=‘relu’),
  Dense (units=15, activation=‘relu’),
  Dense (units=10, activation=‘softmax’)])
```

2. **Specify loss and cost**

```
from tensorflow.keras.losses import SparseCategoricalCrossentropy
model.compile (loss = SparseCategoricalCrossentropy())
```

3. **Train on data to minimize $J(\vec{w}, b)$**

```
model.fit (X, Y, epochs=100)
```

Note: better (recommended) version later

Don’t use the version shown here !

Numerical Roundoff Errors

Option 1

$$x = \frac{2}{10{,}000} = ?$$

Option 2

$$x = \left(1 + \frac{1}{10{,}000}\right) - \left(1 - \frac{1}{10{,}000}\right) = ?$$

Numerical Roundoff Errors

```
In [1]: x1 = 2.0 / 10000
        print(f"{x1:.18f}") # print 18 digits to the right of decimal point

        0.000200000000000000

In [2]: x2 = 1 + (1/10000) - (1 - 1/10000)
        print(f"{x2: .18f}")

         0.000199999999999978

In [ ]:
```

Numerical Roundoff Errors

Logistic regression:

$$a = g(z) = \frac{1}{1+e^{-z}}$$

```
model = Sequential([
  Dense (units=25, activation='relu'),
  Dense (units=15, activation='relu'),
  Dense (units=1, activation='sigmoid')])
```

```
model.compile (loss = BinaryCrossentropy())
```

Original loss

$$Loss = -yloga - (1-\mathrm{y})\log(1-a)$$

```
model = Sequential([
  Dense (units=25, activation='relu'),
  Dense (units=15, activation='relu'),
  Dense (units=1, activation='linear')])
```

```
model.compile (loss = BinaryCrossentropy(from_logits=True))
```

Logits = z

More accurate loss (in code)

$$Loss = -ylog(\frac{1}{1+e^{-z}}) - (1-\mathrm{y})\log(1-\frac{1}{1+e^{-z}})$$

Numerically Accurate Implementation

$$(a_1, \dots, a_{10}) = g(z_1, \dots, z_{10})$$

$$Loss = L(\vec{a}, y) = \begin{cases} -\log(a_1)\,, if\ y = 1 \\ \dots \\ -\log(a_2)\,, if\ y = 10 \end{cases}$$

Softmax regression

```
model = Sequential([
  Dense (units=25, activation='relu'),
  Dense (units=15, activation='relu'),
  Dense (units=10, activation='softmax')])
```

```
model.compile (loss = SparseCategoricalCrossentropy())
```

$$Loss = L(\vec{a}, y) = \begin{cases} -\log(\frac{e^{z_1}}{e^{z_1} + \cdots + e^{z_{10}}})\,, if\ y = 1 \\ \cdots \\ -\log(\frac{e^{z_{10}}}{e^{z_1} + \cdots + e^{z_{10}}})\,, if\ y = 10 \end{cases}$$

More accurate (but, difficult to understand)

```
model = Sequential([
  Dense (units=25, activation='relu'),
  Dense (units=15, activation='relu'),
  Dense (units=10, activation='linear')])
```

```
model.compile (loss = SparseCategoricalCrossentropy(from_logits=True))
```

MNIST (numerically accurate)

1. model

```
import tensorflow as tf
from tensorflow.keras import Sequential
From tensorflow.keras.layers import Dense
model = Sequential([
  Dense (units=25, activation=‘relu’),
  Dense (units=15, activation=‘relu’),
  Dense (units=10, activation=‘linear’)])
```

2. loss

```
from tensorflow.keras.losses import SparseCategoricalCrossentropy
model.compile (loss = SparseCategoricalCrossentropy(from_logits=True))
```

3. fit

```
model.fit (X, Y, epochs=100)
```

$not\ a_1, \dots, a_{10}$

$is\ z_1, \dots, z_{10}$

4. predict

```
logits = model (X)
f_x = tf.nn.softmax(logits)
```

 TheMAD

Logistic regression (numerically accurate)

1. model

```
import tensorflow as tf
from tensorflow.keras import Sequential
From tensorflow.keras.layers import Dense
model = Sequential([
  Dense (units=25, activation='relu'),
  Dense (units=15, activation='relu'),
  Dense (units=1, activation='linear')])
```

2. loss

```
from tensorflow.keras.losses import BinaryCrossentropy
model.compile (loss = BinaryCrossentropy(from_logits=True))
```

3. fit

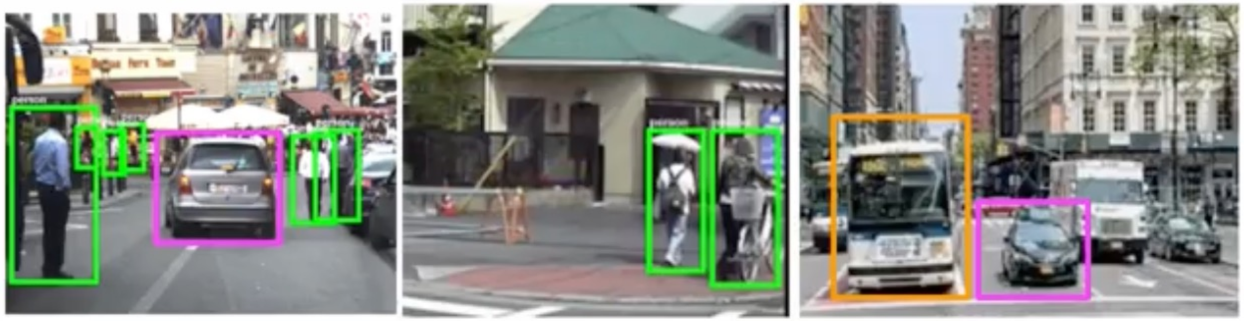
```
model.fit (X, Y, epochs=100)
```

4. predict

z

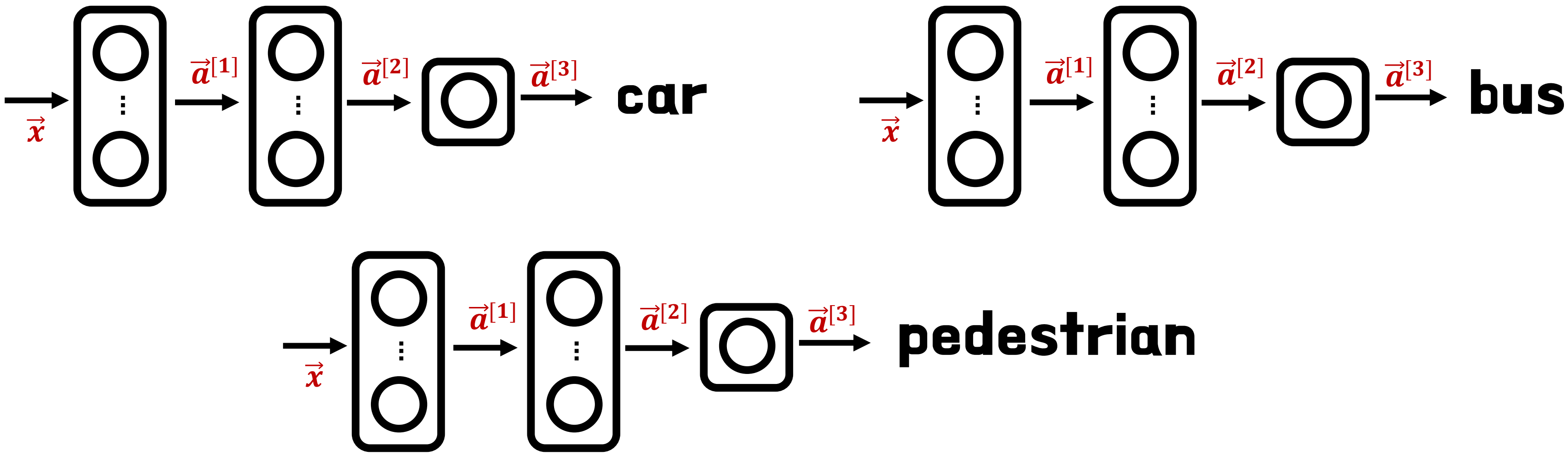
```
logits = model (X)
f_x = tf.nn.sigmoid(logit)
```

Multi-label Classification

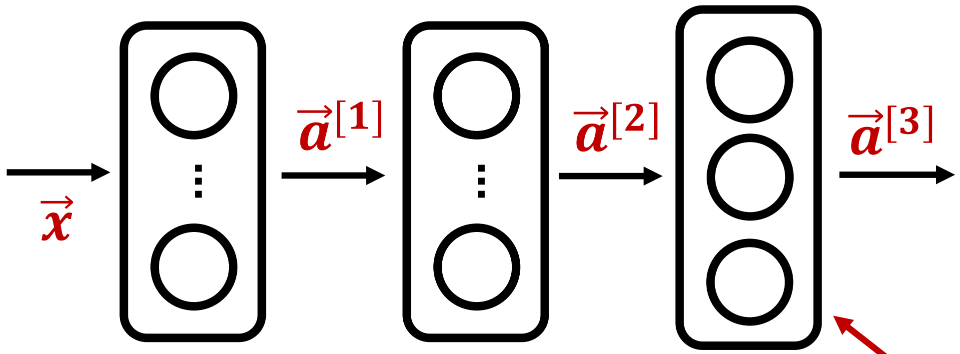
$\vec{x}$ 

Is there a car ?	yes	no	yes
Is there a bus ?	no	no	yes
Is there a pedestrian ?	yes	yes	no
	$y = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$	$y = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$	$y = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$

Multi-label Classification



Alternatively, train on neural network with three outputs



$$\vec{a}^{[3]} = \begin{bmatrix} a_1^{[3]} \\ a_2^{[3]} \\ a_3^{[3]} \end{bmatrix} \begin{array}{l} \text{car} \\ \text{bus} \\ \text{pedestrian} \end{array}$$

Sigmoid activations

TheMAD

Additional Neural Network Concepts

Gradient descent algorithm

Repeat until convergence

$$w = w - \alpha \frac{\partial}{\partial w} J(w, b) \qquad b = b - \alpha \frac{\partial}{\partial b} J(w, b)$$

Partial derivative

Learning rate

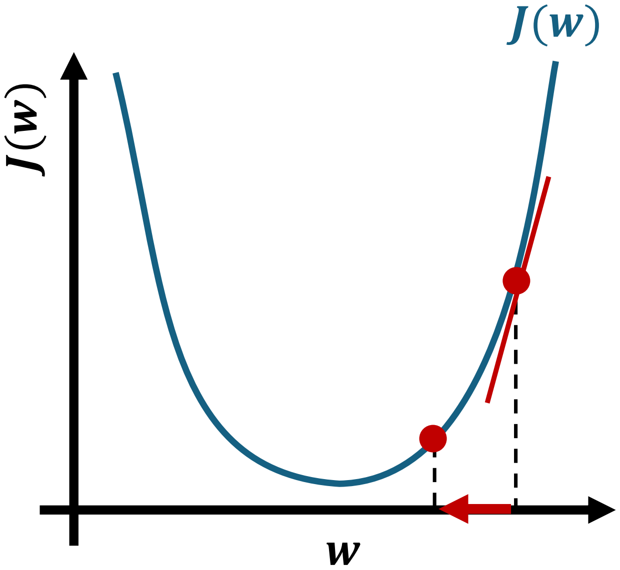
In simplified equation,

$$J(w)$$

$$w = w - \alpha \frac{d}{dw} J(w)$$

$$\min_{\boldsymbol{w}} \boldsymbol{J}(\boldsymbol{w})$$

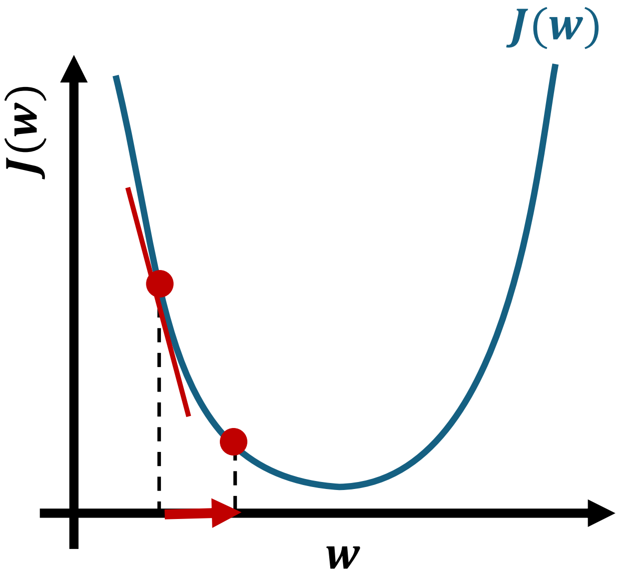
Gradient descent algorithm



$$w = w - \alpha \frac{d}{dw} J(w)$$

> 0

$$= w - \alpha(positive\ number)$$



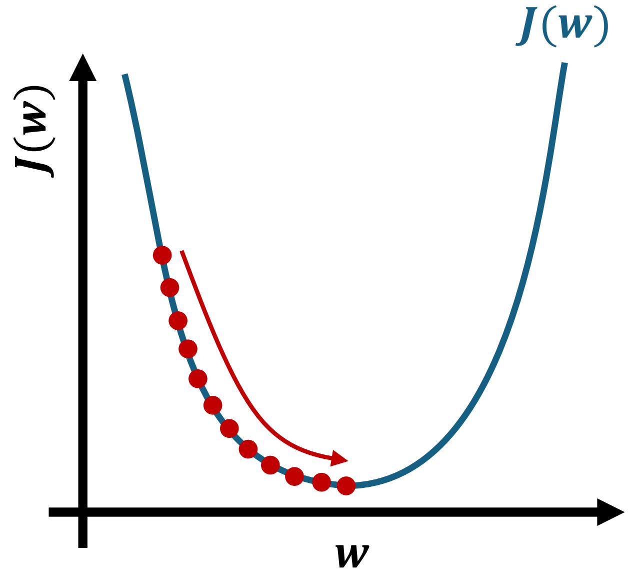
$$w = w - \alpha \frac{d}{dw} J(w)$$

< 0

$$= w - \alpha(negative\ number)$$

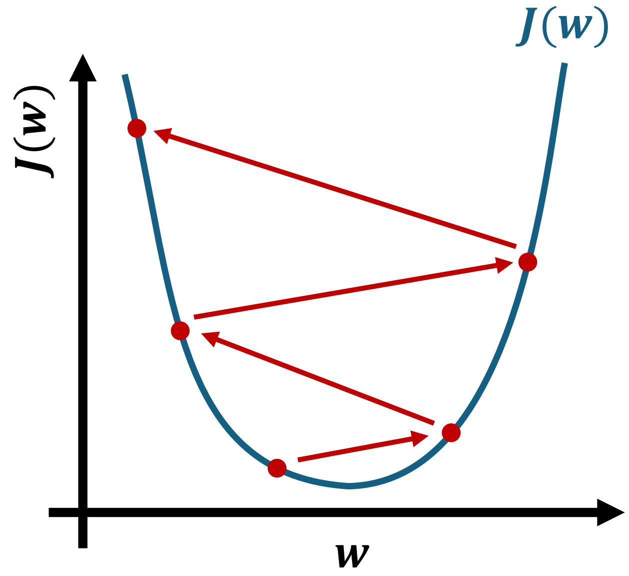
Learning rate

$$w = w - \alpha \frac{d}{dw} J(w)$$



If is α too small …

Gradient descent may be slow



If is α too large …

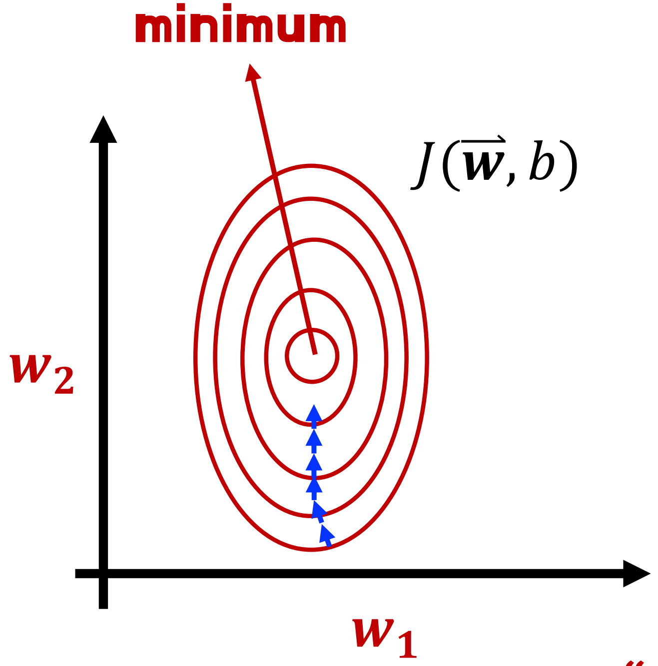
Gradient descent may:

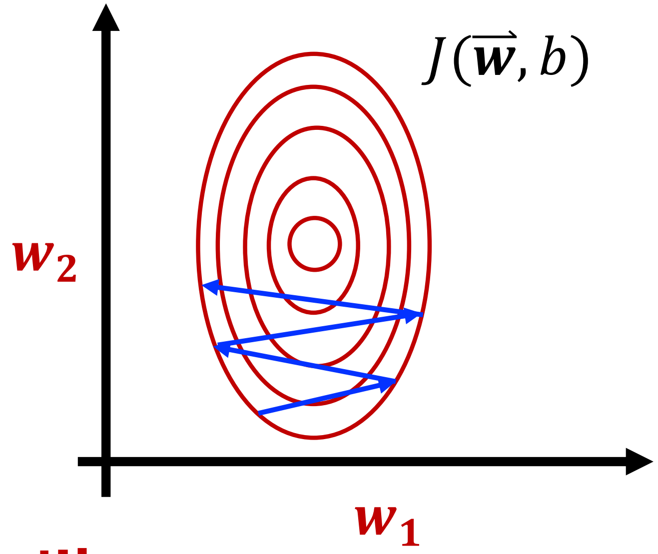
- **Overshoot, never reach minimum**
- **Fail to converge → diverge**

Gradient descent

$$w_j = w_j - \alpha \frac{\partial}{\partial w_j} J(\vec{\mathbf{w}}, b)$$

Learning rate





"Adam" algorithm

Go faster - increase α

Go slower - decrease α

Adam algorithm intuition

Adam: ADAptive Moment estimation

Not just one α

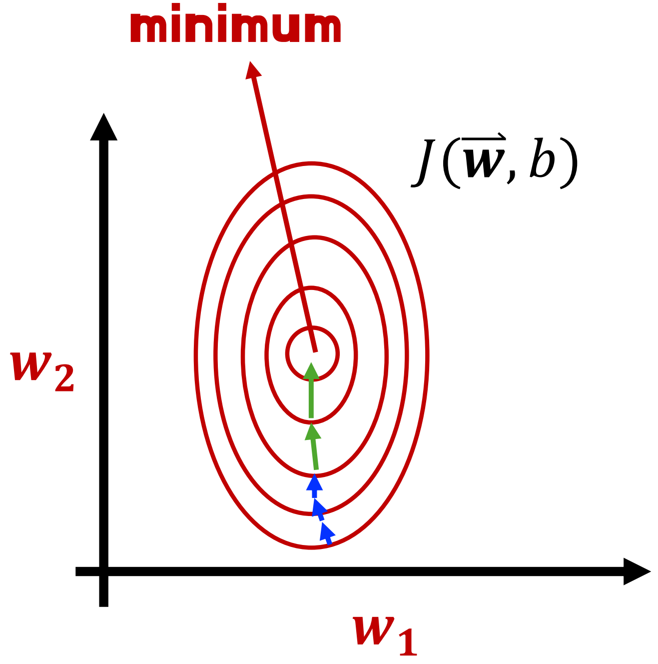
$$w_1 = w_1 - \alpha_1 \frac{\partial}{\partial w_1} J(\vec{\mathbf{w}}, b)$$

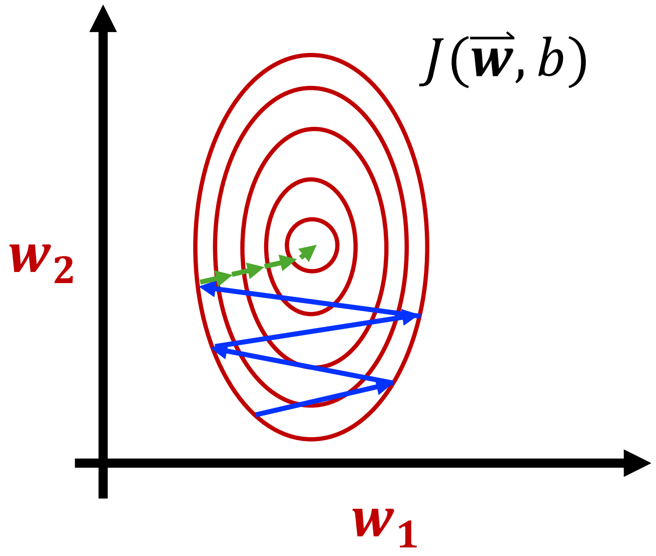
$$\vdots$$

$$w_{10} = w_{10} - \alpha_{10} \frac{\partial}{\partial w_{10}} J(\vec{\mathbf{w}}, b)$$

$$b = b - \alpha_{11} \frac{\partial}{\partial b} J(\vec{\mathbf{w}}, b)$$

Adam algorithm intuition





If w_j $(or\ b)$ keeps moving in same direction,
→ increase α_j

If w_j $(or\ b)$ keeps oscillating,
→ decrease α_j

MNIST Adam

```
import tensorflow as tf
from tensorflow.keras import Sequential
From tensorflow.keras.layers import Dense
model = Sequential([
  Dense (units=25, activation='relu'),
  Dense (units=15, activation='relu'),
  Dense (units=10, activation='linear')])
```

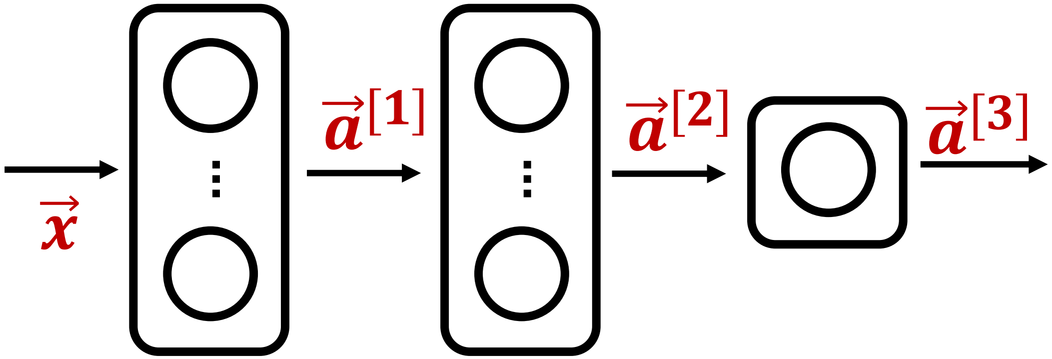
```
from tensorflow.keras.losses import SparseCategoricalCrossentropy
model.compile (optimizer=tf.keras.optimizers.Adam(learning_rate=1e-3),
loss = SparseCategoricalCrossentropy(from_logits=True))
```

```
model.fit (X, Y, epochs=100)
```

```
logits = model (X)
f_x = tf.nn.softmax(logits)
```

Faster than general gradient descent
→ mostly used optimizer "Adam"

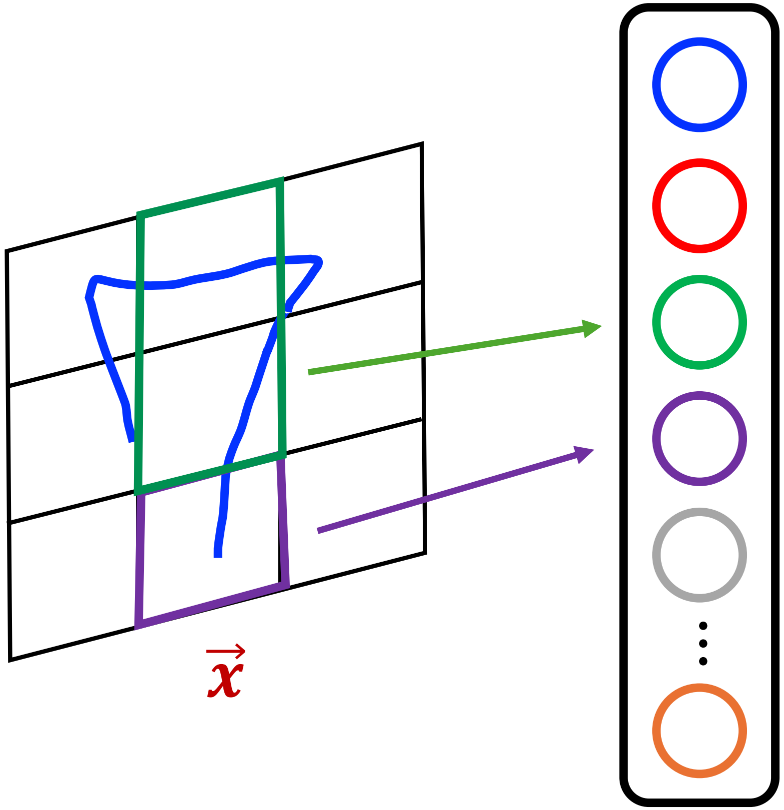
Dense Layer



Each neuron output is a function of all the activation outputs of the previous layer

$$\overrightarrow{a_1}^{[2]} = g(\overrightarrow{w_1}^{[2]} \cdot \vec{a}^{[1]} + b_1^{[2]})$$

Convolutional Layer

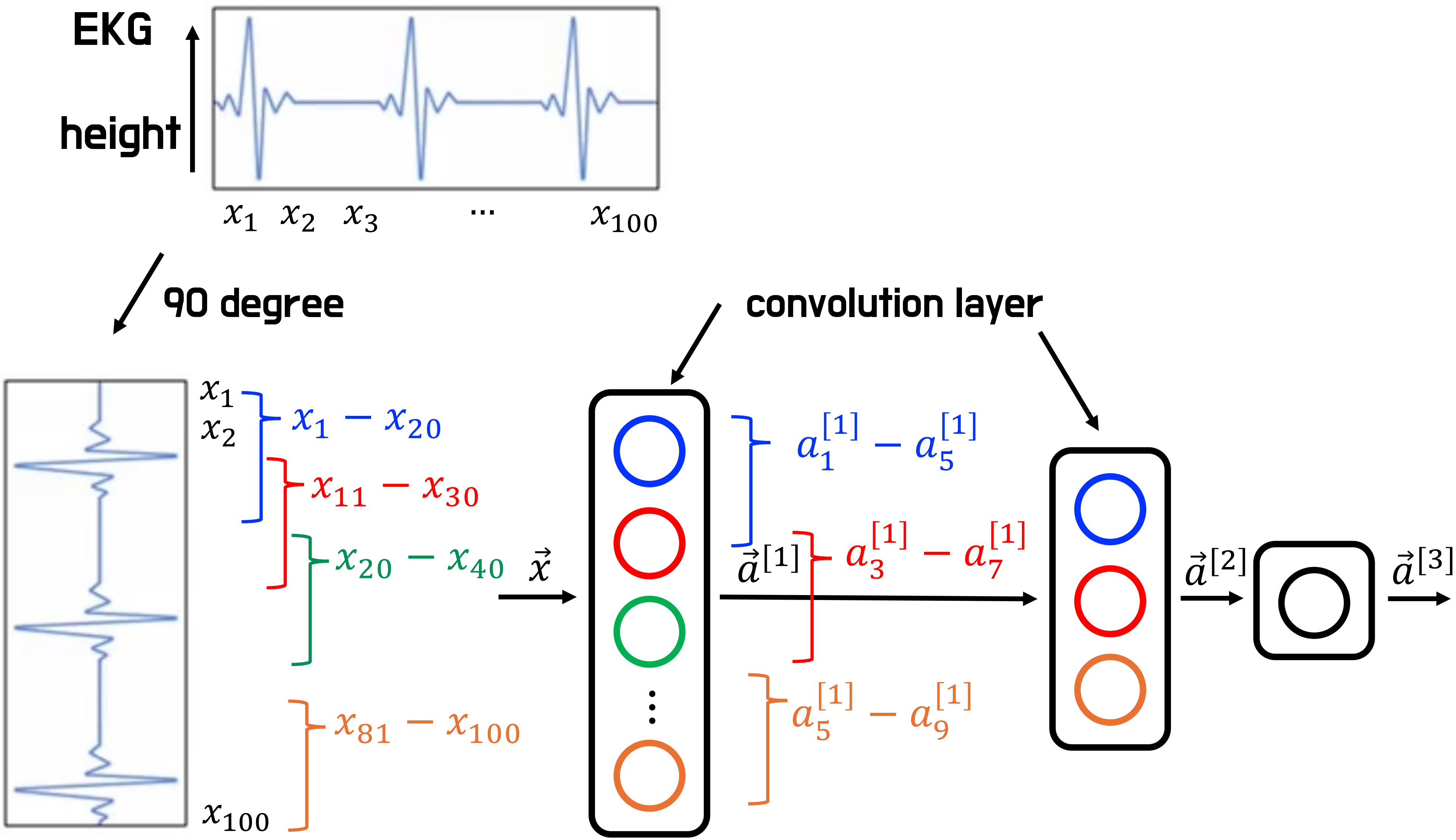


Each neuron only looks at part of the previous layer's outputs

Why?

- Faster computation
- Need less training data
  (less prone to overfitting)

Convolutional Neural Network



 TheMAD