



Machine Learning 07

Kihyun Shin
DMSE, HBNU

Multiclass classification



MNIST example

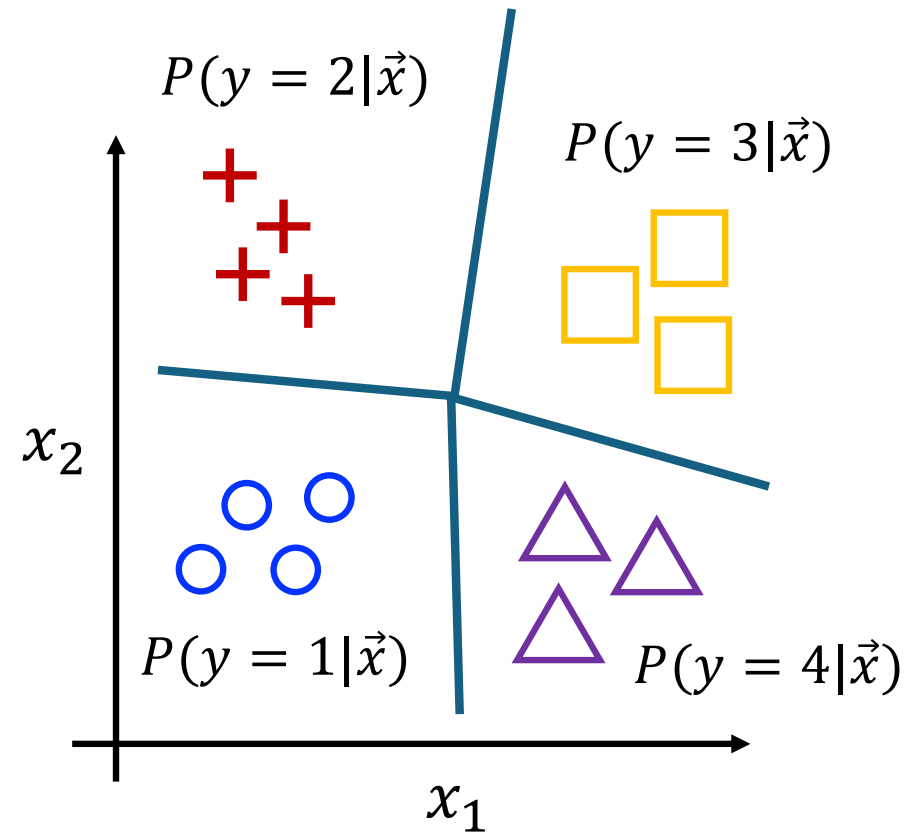
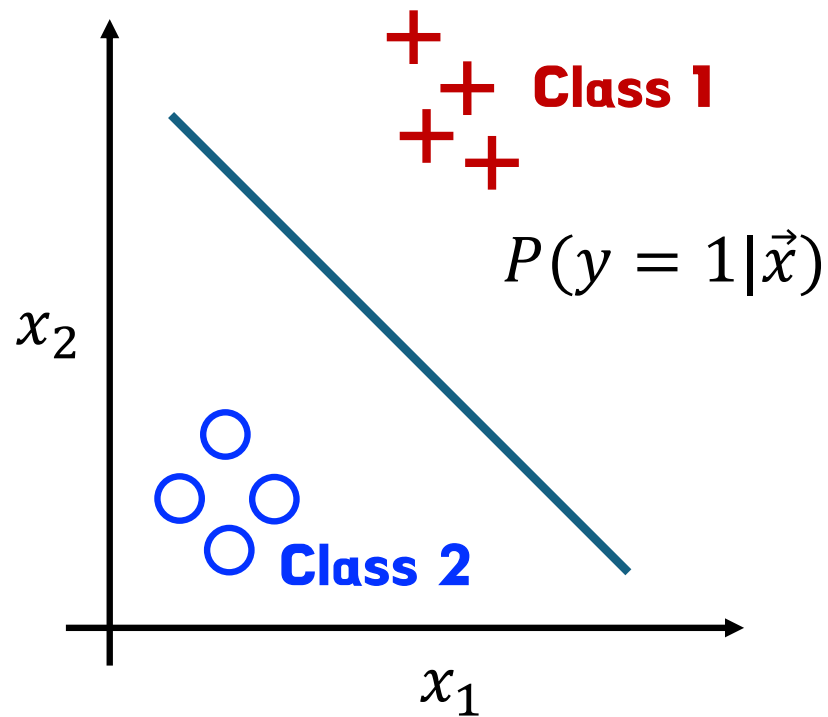
0 1 2 3 4 5 6 7 8 9
 $y = 0$ 1 2 3 4 5 6 7 8 9

$x \rightarrow 7$ $y = 7$

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)

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

$$+ a_1 = g(z) = \frac{1}{1 + e^{-z}} = P(y = 1|\vec{x}) \quad \mathbf{0.71}$$

$$\circ a_2 = 1 - a_1 = P(y = 0|\vec{x}) \quad \mathbf{0.29}$$

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: $a_1 + a_2 + a_3 + \dots + a_N = 1$

Softmax regression (4 possible outputs)

$$z_1 = \vec{w}_1 \cdot \vec{x} + b_1 \quad a_1 = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}}$$

+ + \circ \square \triangle

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

$$z_2 = \vec{w}_2 \cdot \vec{x} + b_2 \quad a_2 = \frac{e^{z_2}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}}$$

\circ

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

$$z_3 = \vec{w}_3 \cdot \vec{x} + b_3 \quad a_3 = \frac{e^{z_3}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}}$$

\square

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

$$z_4 = \vec{w}_4 \cdot \vec{x} + b_4 \quad a_4 = \frac{e^{z_4}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}}$$

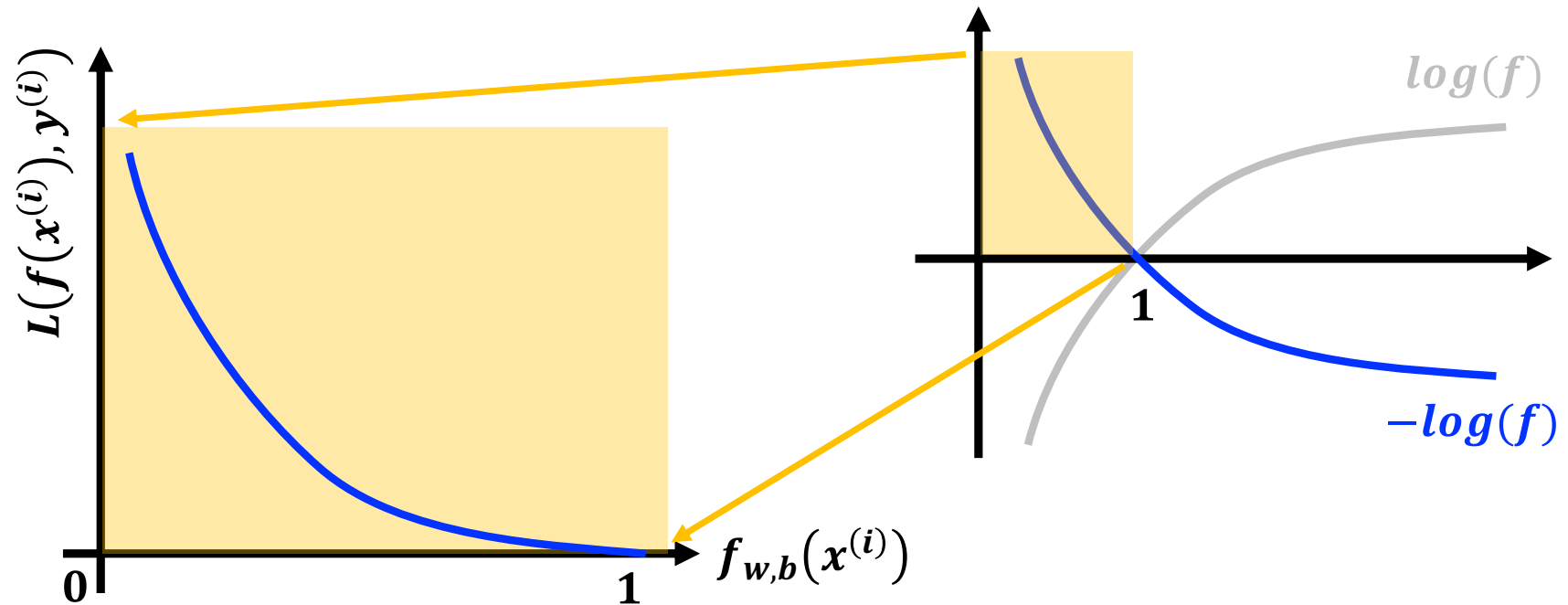
\triangle

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



Loss function

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

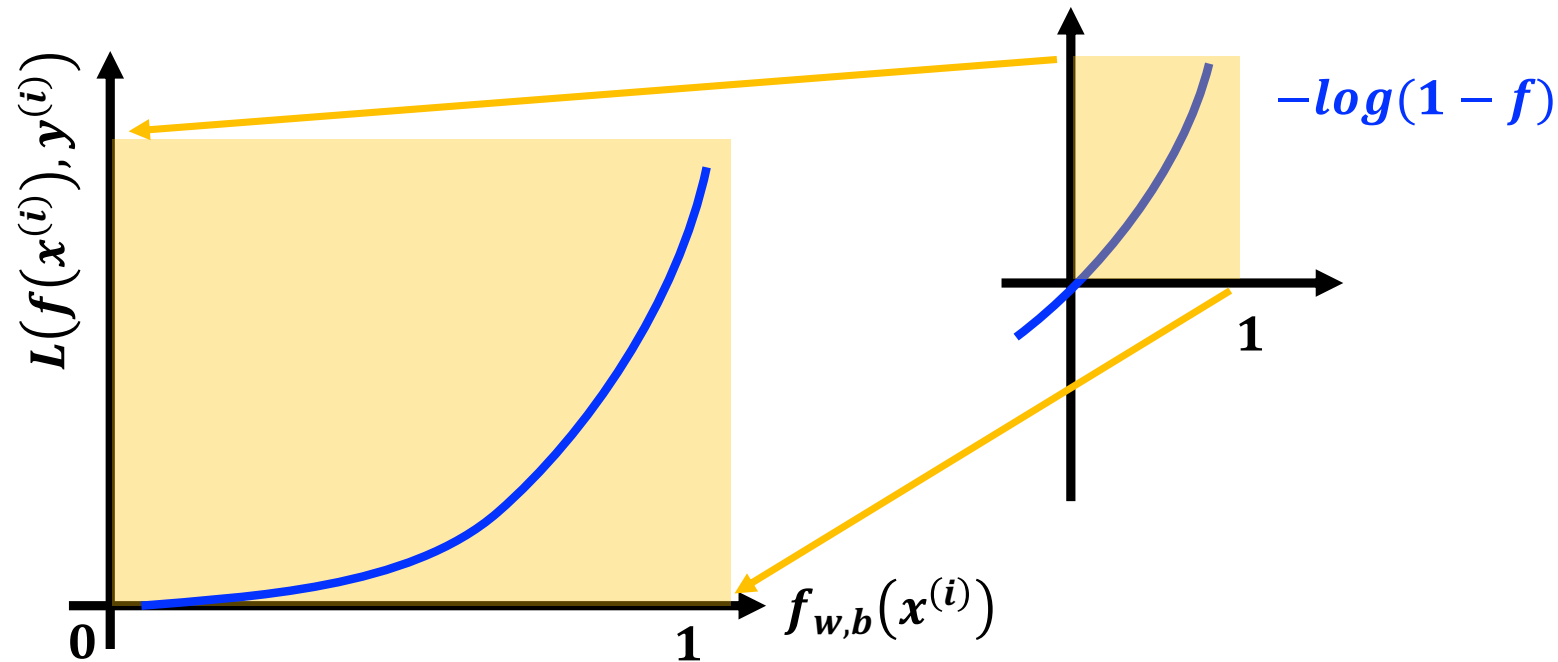


As $f_{w,b}(x^{(i)}) \rightarrow 1$ then loss $\rightarrow 0$

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

Loss function

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



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

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

Cost

Logistic regression

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

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

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

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

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

Softmax regression

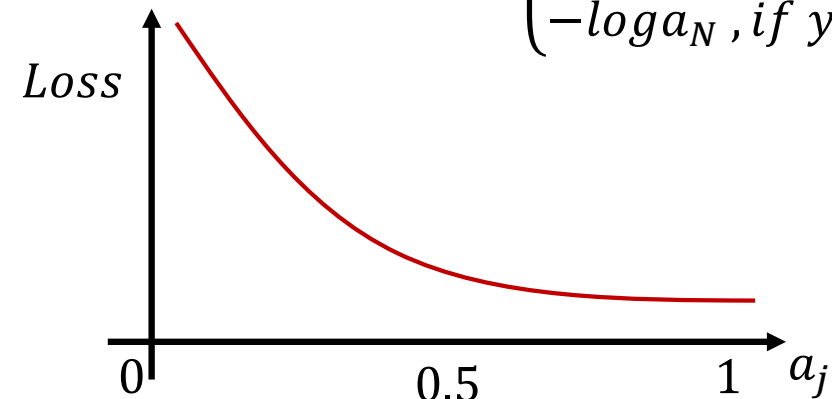
$$a_1 = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + \dots + e^{z_N}} = P(y = 1|\vec{x})$$

⋮

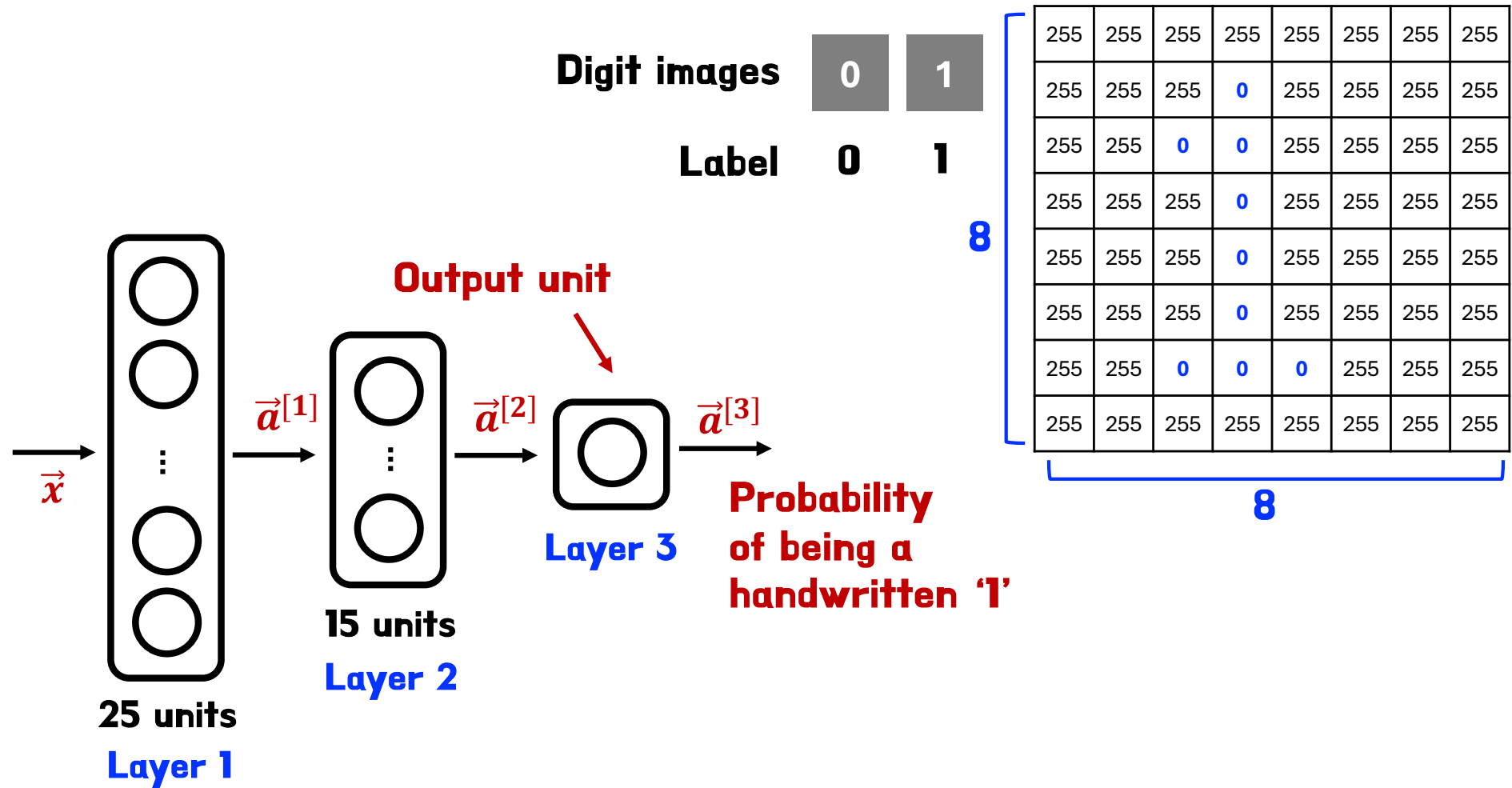
$$a_N = \frac{e^{z_N}}{e^{z_1} + e^{z_2} + \dots + e^{z_N}} = P(y = N|\vec{x})$$

Cross-entropy loss

$$\text{Loss}(a_1, a_2, \dots, a_N, y) = \begin{cases} -\log a_1, & \text{if } y = 1 \\ -\log a_2, & \text{if } y = 2 \\ \dots \\ -\log a_N, & \text{if } y = N \end{cases}$$



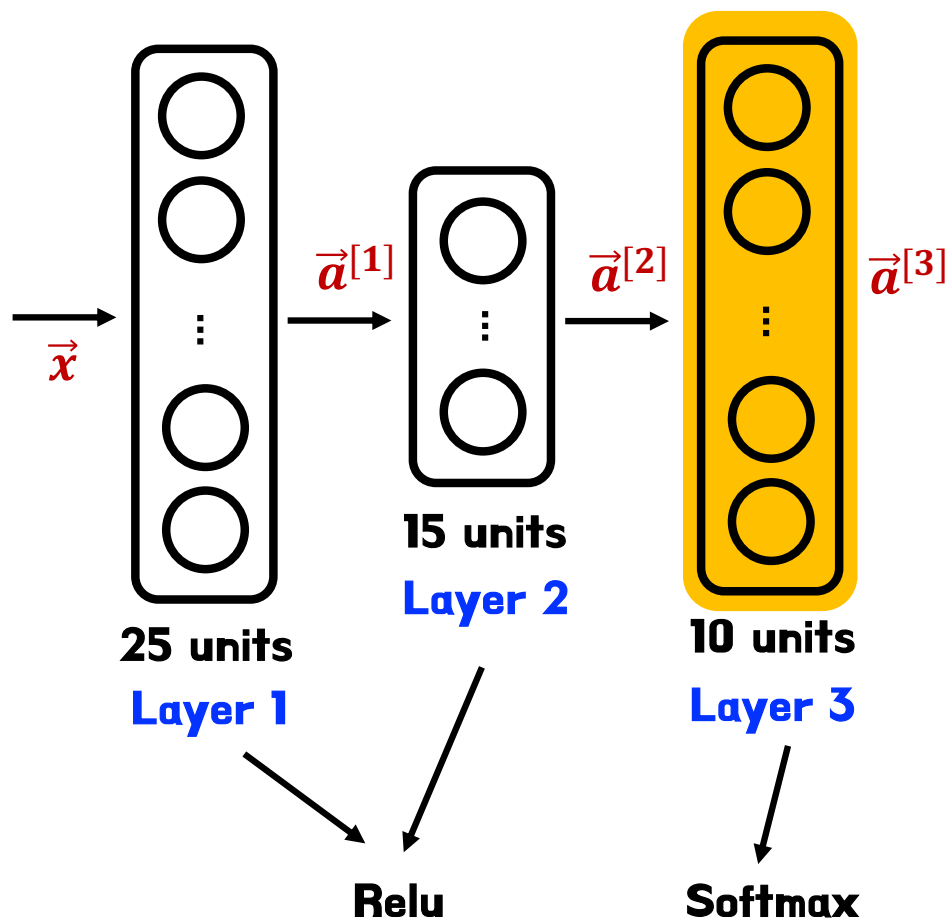
Neural Network with Softmax output



Hidden units
(neurons)



Neural Network with Softmax output



$$z_1^{[3]} = \vec{w}_1^{[3]} \cdot \vec{a}^{[2]} + b_1^{[3]}$$

$$a_1^{[3]} = \frac{e^{z_1^{[3]}}}{e^{z_1^{[3]}} + \dots + e^{z_{10}^{[3]}}}$$

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

$$z_{10}^{[3]} = \vec{w}_{10}^{[3]} \cdot \vec{a}^{[2]} + b_{10}^{[3]}$$

$$a_{10}^{[3]} = \frac{e^{z_{10}^{[3]}}}{e^{z_1^{[3]}} + \dots + e^{z_{10}^{[3]}}}$$

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

Logistic regression

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

Softmax

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



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  
0.00020000000000000000
```

```
In [2]: x2 = 1 + (1/10000) - (1 - 1/10000)  
print(f"{x2: .18f}")  
0.0001999999999999978
```

```
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

$$Loss = -y \log a - (1 - y) \log(1 - a)$$

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

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

More accurate loss (in code)

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

Logits = z



Numerically Accurate Implementation

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

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

Softmax regression

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

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

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

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)
```

4. predict

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

not a_1, \dots, a_{10}

is z_1, \dots, z_{10}



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)
```

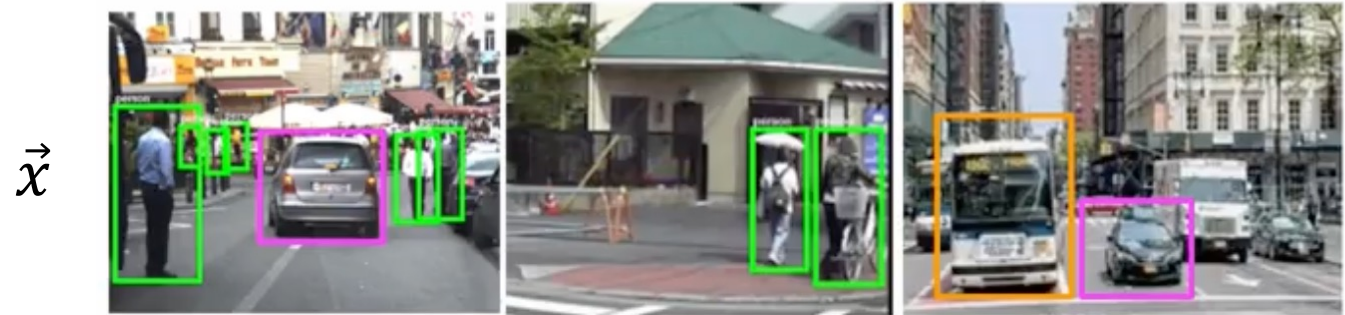
4. predict

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



z

Multi-label Classification



Is there a car ?

yes

no

yes

Is there a bus ?

no

no

yes

Is there a pedestrian ?

yes

yes

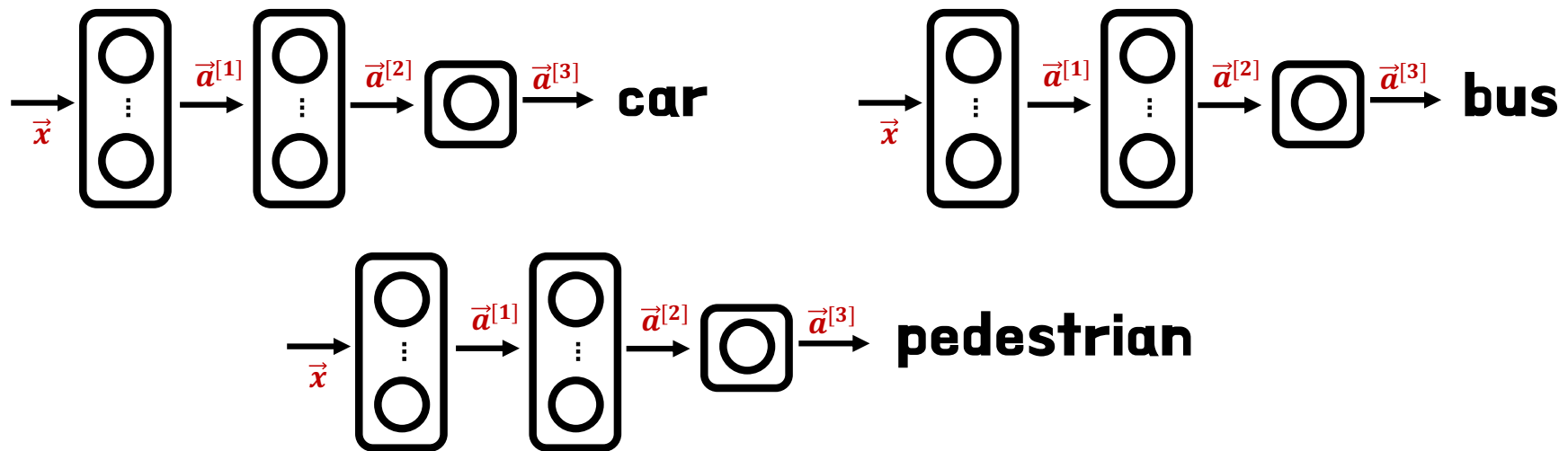
no

$$y = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$$

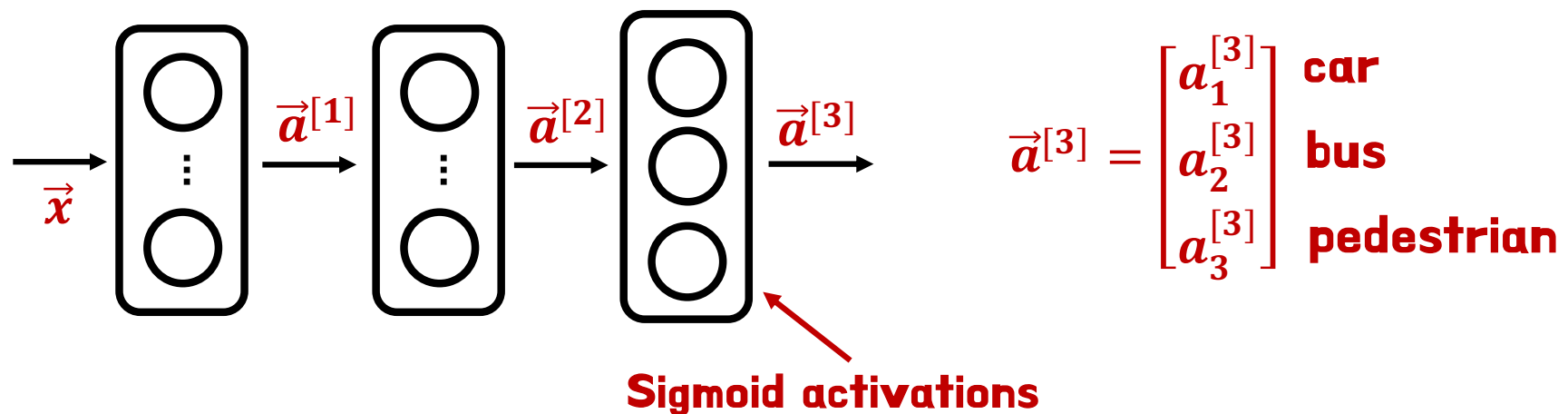
$$y = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

$$y = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$$

Multi-label Classification



Alternatively, train on neural network with three outputs



Additional Neural Network Concepts



Gradient descent algorithm

Repeat until convergence

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

Learning rate Partial derivative

In simplified equation,

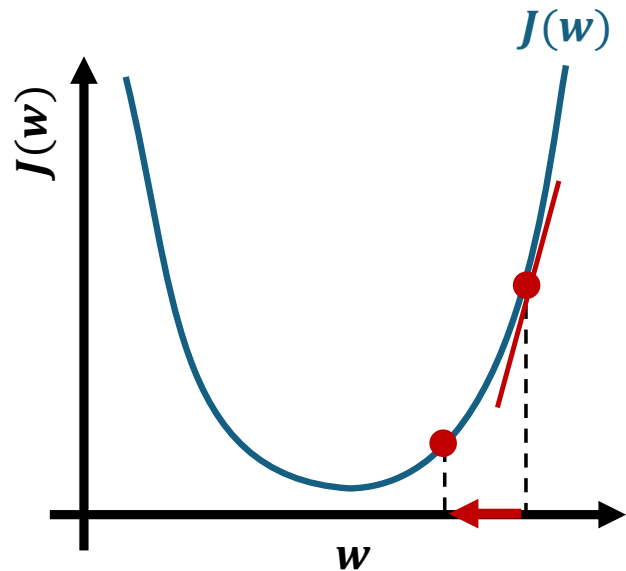
$$J(w)$$

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

$$\min_w J(w)$$



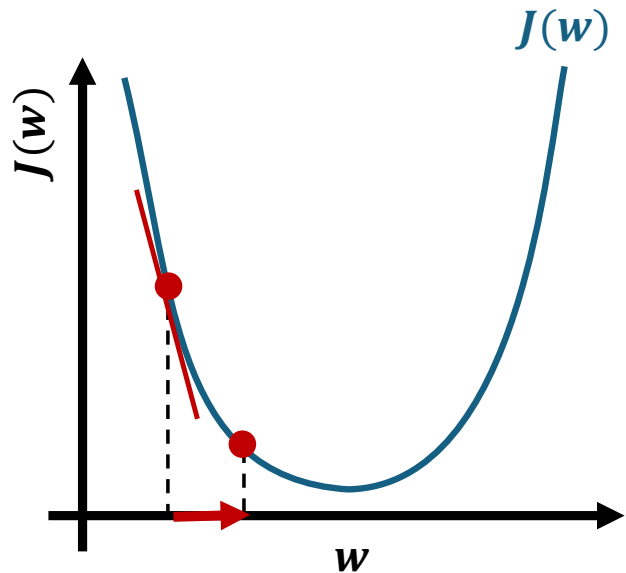
Gradient descent algorithm



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

> 0

$$= w - \alpha(\text{positive number})$$



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

< 0

$$= w - \alpha(\text{negative number})$$



Learning rate

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

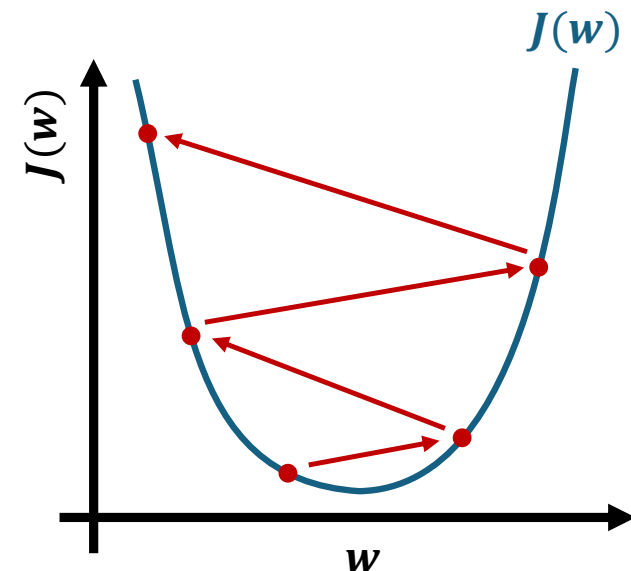
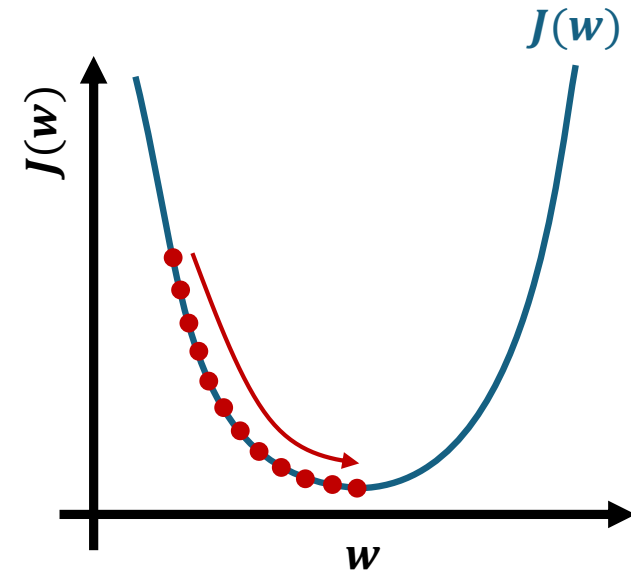
If is α too small ...

Gradient descent may be slow

If is α too large ...

Gradient descent may:

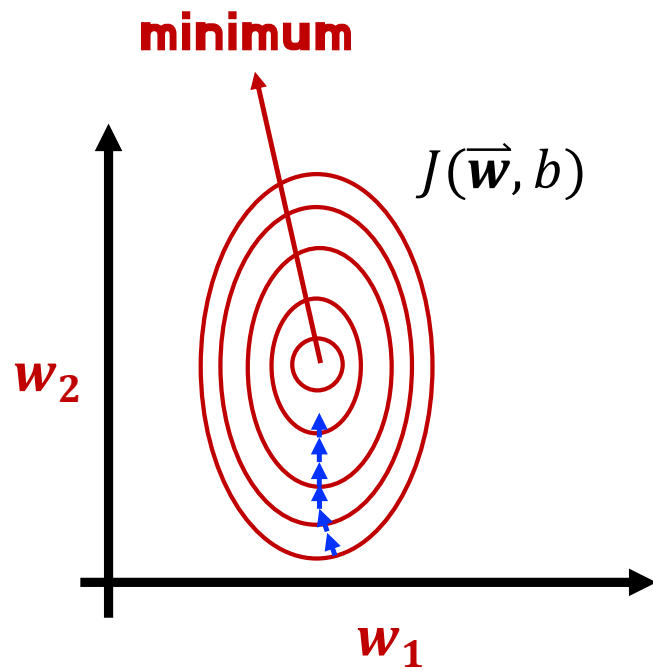
- **Overshoot, never reach minimum**
- **Fail to converge \rightarrow diverge**



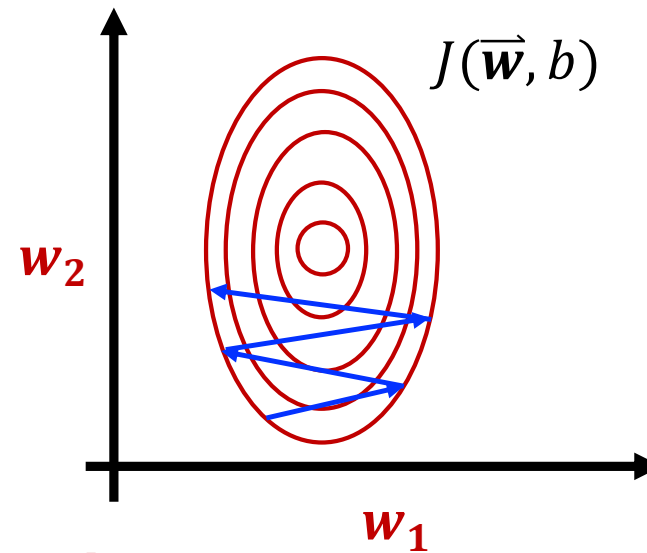
Gradient descent

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

Learning rate



Go faster - increase α



Go slower - decrease α

"Adam" algorithm



Adam algorithm intuition

Adam: **AD**Aptive **M**oment estimation

Not just one α

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

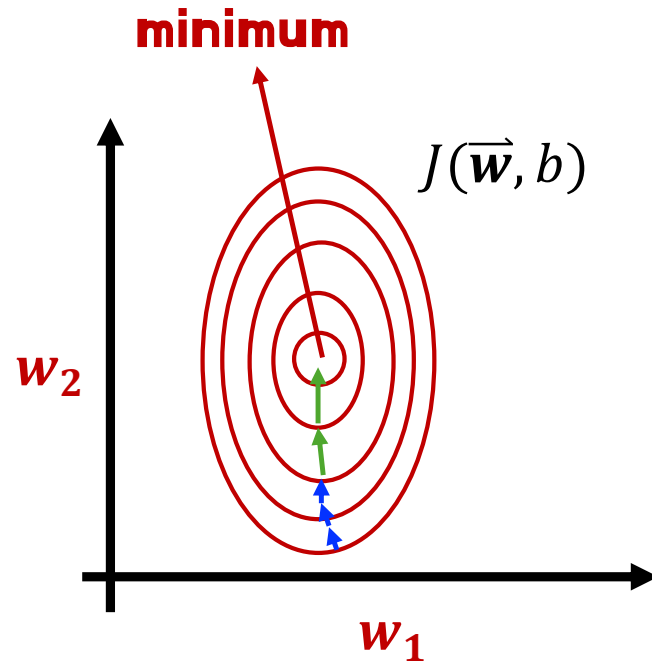
\vdots

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

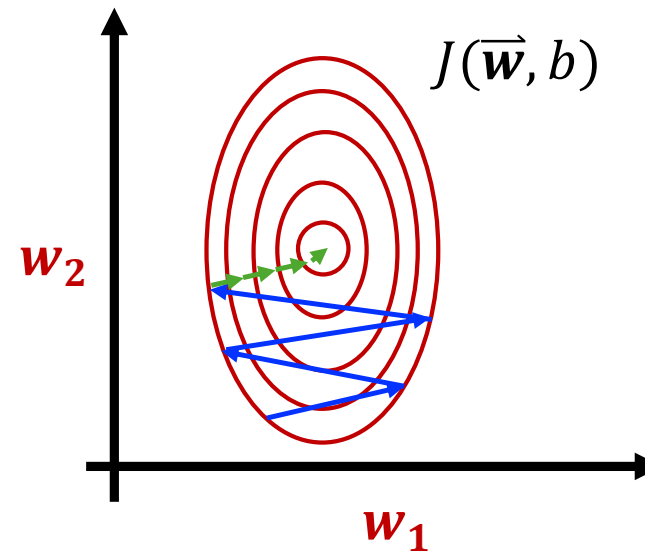
$$b = b - \alpha_{11} \frac{\partial}{\partial b} J(\vec{w}, b)$$



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))
```

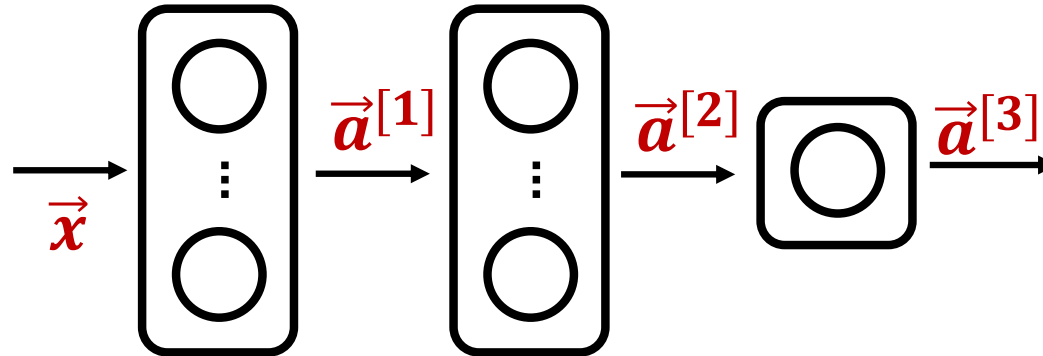
```
model.fit (X, Y, epochs=100)
```

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

Faster than general gradient descent
→ mostly used optimizer “Adam”



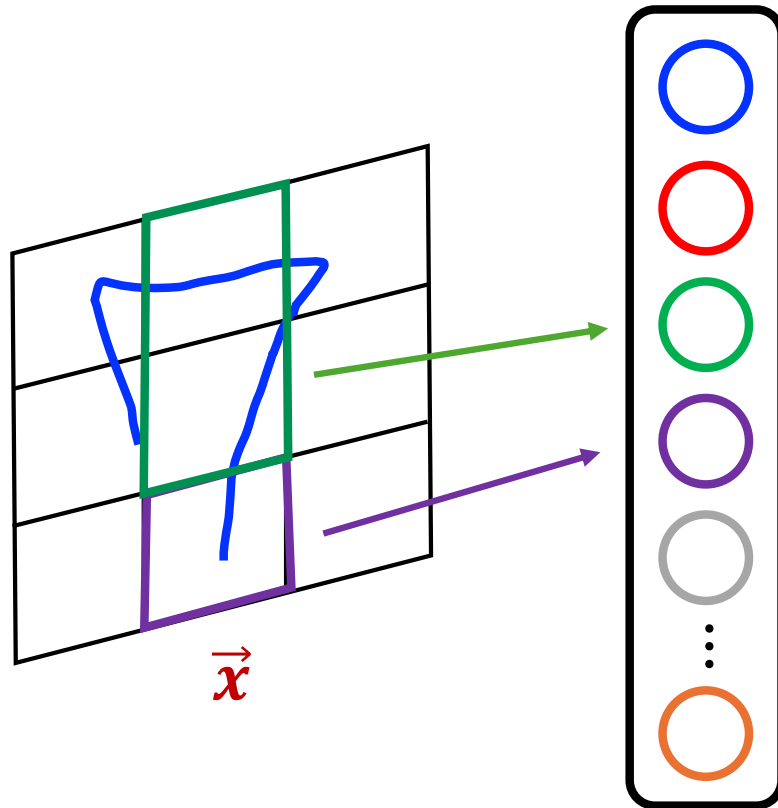
Dense Layer



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

$$\vec{a}_1^{[2]} = g(\vec{w}_1^{[2]} \cdot \vec{a}^{[1]} + b_1^{[2]})$$

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

