Lecture 13

Tian Han

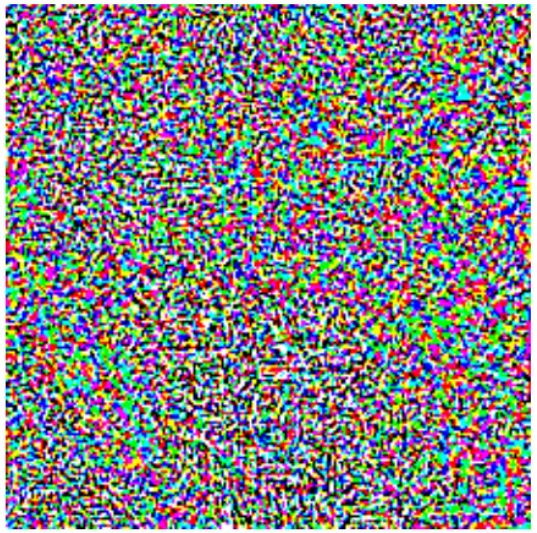
Outline

- Attack for Deep models
- Reinforcement learning Basic

Data Evasion Attacks

Data Evasion Attack

$+ .007 \times$



$=$

"panda"
57.7% confidence

"gibbon"
99.3% confidence

Reference

- Goodfellow, Shlens, and Szegedy. Explaining and harnessing adversarial examples. *arXiv:1412.6572*, 2014.

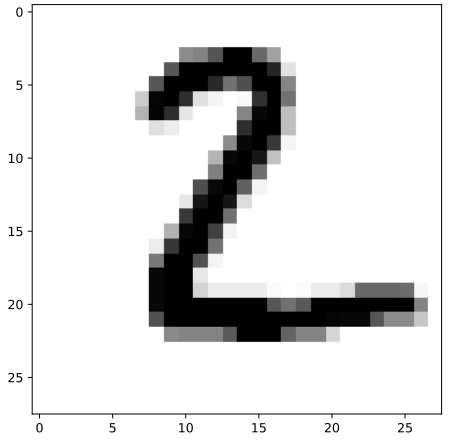
Revisit CNN for MNIST Classification

Neural Network for MNIST

- Neural network: $\mathbf{p} = \mathbf{f}(\mathbf{x}; \mathbf{W})$

Trainable parameters of the neural network.

28×28 input image.

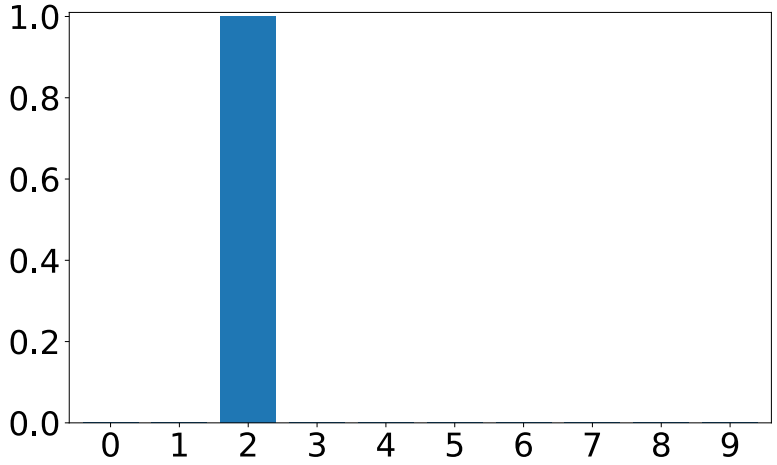


Neural Network for MNIST

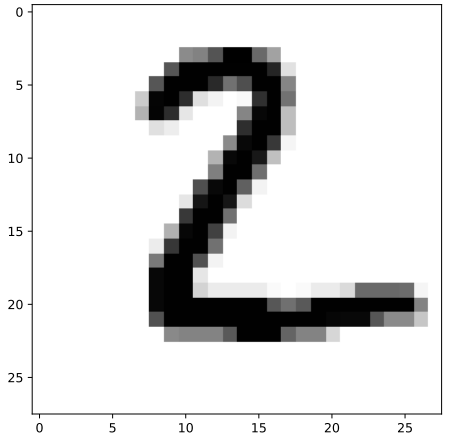
- Neural network: $\mathbf{p} = \mathbf{f}(\mathbf{x}; \mathbf{W})$

Trainable parameters of the neural network.

- The prediction --- a 10-dim vector.
- p_j indicates how likely $\mathbf{x}$ is the digit j.

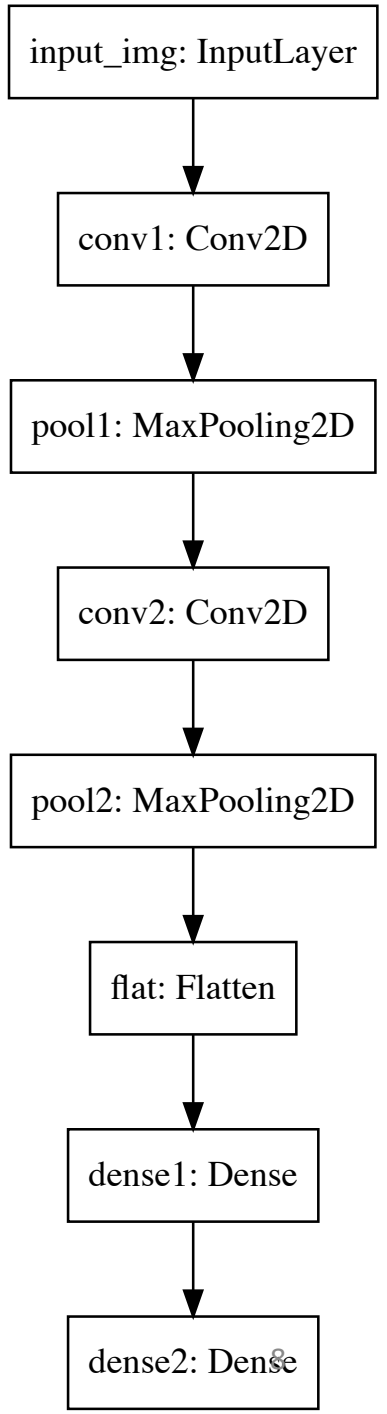


28×28 input image.



CNN for MNIST

```
from keras.layers import Input, Conv2D, MaxPooling2D, Flatten, Dense
from keras import models



# input
input_img = Input(shape=(28, 28, 1), name='input_img')

# layers
conv1 = Conv2D(10, (5, 5), activation='relu', name='conv1')(input_img)
pool1 = MaxPooling2D((2, 2), name='pool1')(conv1)
conv2 = Conv2D(20, (5, 5), activation='relu', name='conv2')(pool1)
pool2 = MaxPooling2D((2, 2), name='pool2')(conv2)
flat = Flatten(name='flat')(pool2)
dense1 = Dense(100, activation='relu', name='dense1')(flat)
pred = Dense(10, activation='softmax', name='dense2')(dense1)

# model
model = models.Model(inputs=input_img, outputs=pred)
```



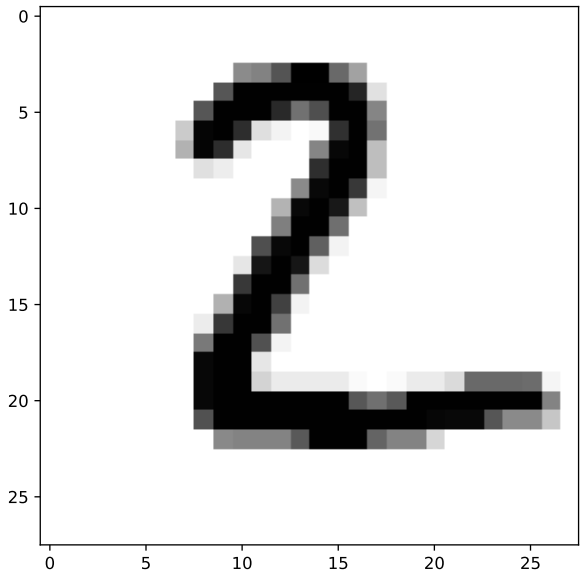
CNN for MNIST

```
model.compile(optimizer='RMSprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

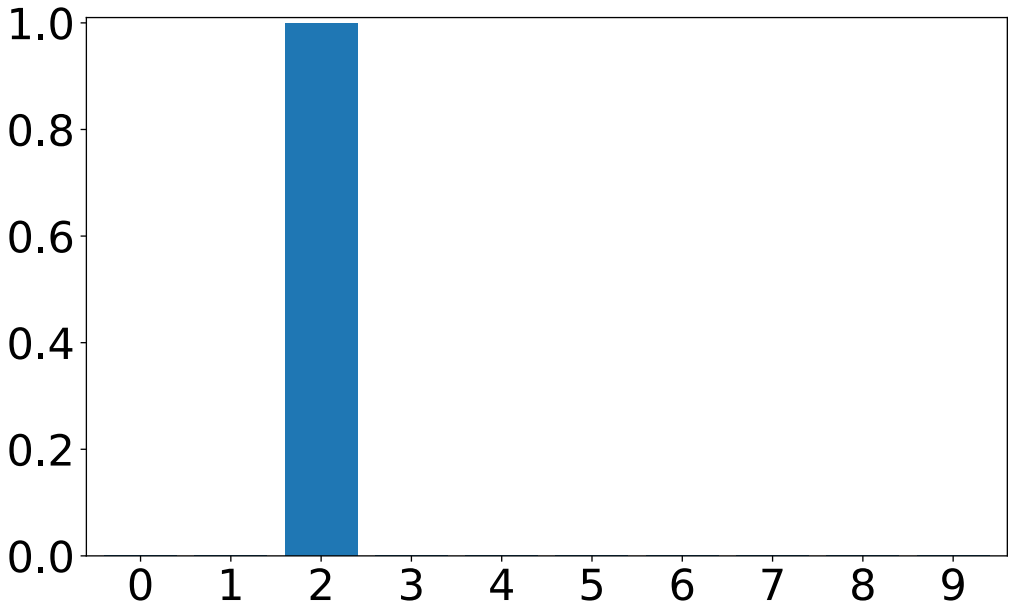
```
history = model.fit(
    x_train,
    y_train_vec,
    shuffle=True,
    epochs=10,
    batch_size=128,
    validation_data=(x_test, y_test_vec)
)
```

CNN for MNIST

- Training accuracy = 99.5%
- Validation accuracy = 99.1%



28×28 input image

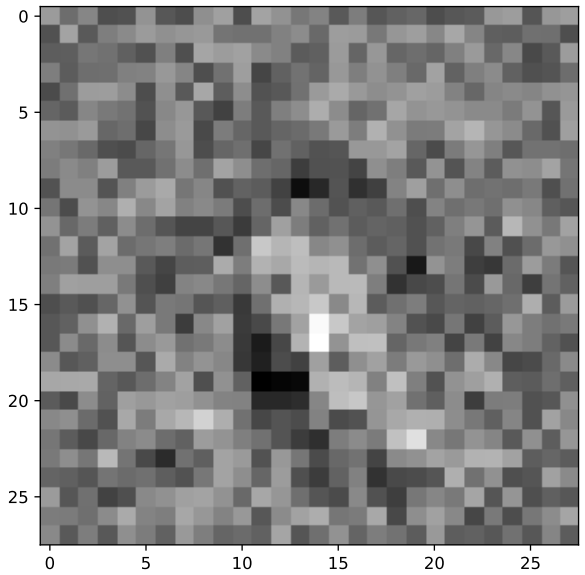


10-dim prediction vector

Optimization w.r.t. X

Fake Image

- Our trained CNN thinks the 28×28 input image is digit "**0**".

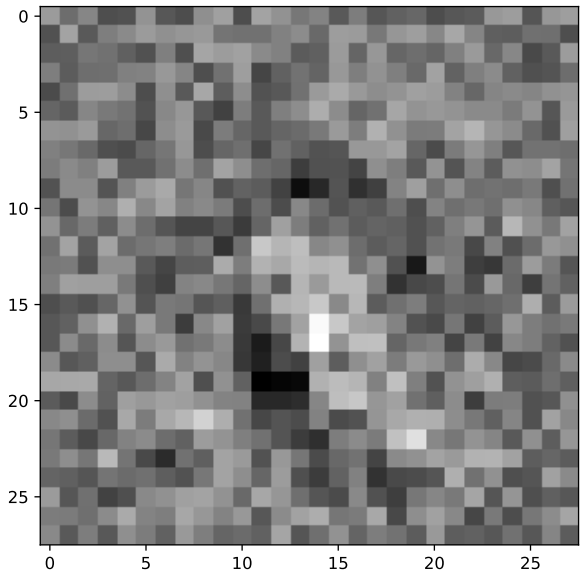


28×28 input image

10-dim prediction vector

Fake Image

Question: How is the fake image generated?



28×28 input image

10-dim prediction vector

Generate Fake Image

Question: How is the fake image generated?

- Neural network: $\mathbf{p} = \mathbf{f}(\mathbf{x}; \mathbf{W})$.
- Previously, we fix $\mathbf{x}_1, \cdots, \mathbf{x}_n$ and solve

$$\mathbf{W}^\star = \underset{\mathbf{W}}{\operatorname{argmin}} \sum_{j=1}^{n} \operatorname{Dist}\left(\mathbf{y}_j, \mathbf{f}(\mathbf{x}_j; \mathbf{W})\right).$$

Generate Fake Image

Question: How is the fake image generated?

To generate a fake image, we do the following:

1. Set a fake target, e.g., $\tilde{\mathbf{y}} = [1, 0, 0, \cdots, 0]$.
2. Fix the network parameters to $\mathbf{W}^\star$.
3. Generate a fake image $\tilde{\mathbf{x}}$ by

$$\tilde{\mathbf{x}} = \underset{\mathbf{x}}{\operatorname{argmin}} \ \mathrm{Dist}(\tilde{\mathbf{y}}, \ \mathbf{f}(\mathbf{x}; \mathbf{W}^\star)).$$

Generate Fake Image

Step 1: Set a fake target.

```
import numpy as np
from keras import backend as K
from keras.layers import Input

j = 0 # the fake lable
y_tilde = np.zeros((1, 10))
y_tilde[0, j] = 1
print(y_tilde)

fake_target = Input(tensor=K.constant(y_tilde))
```

```
[[1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
```

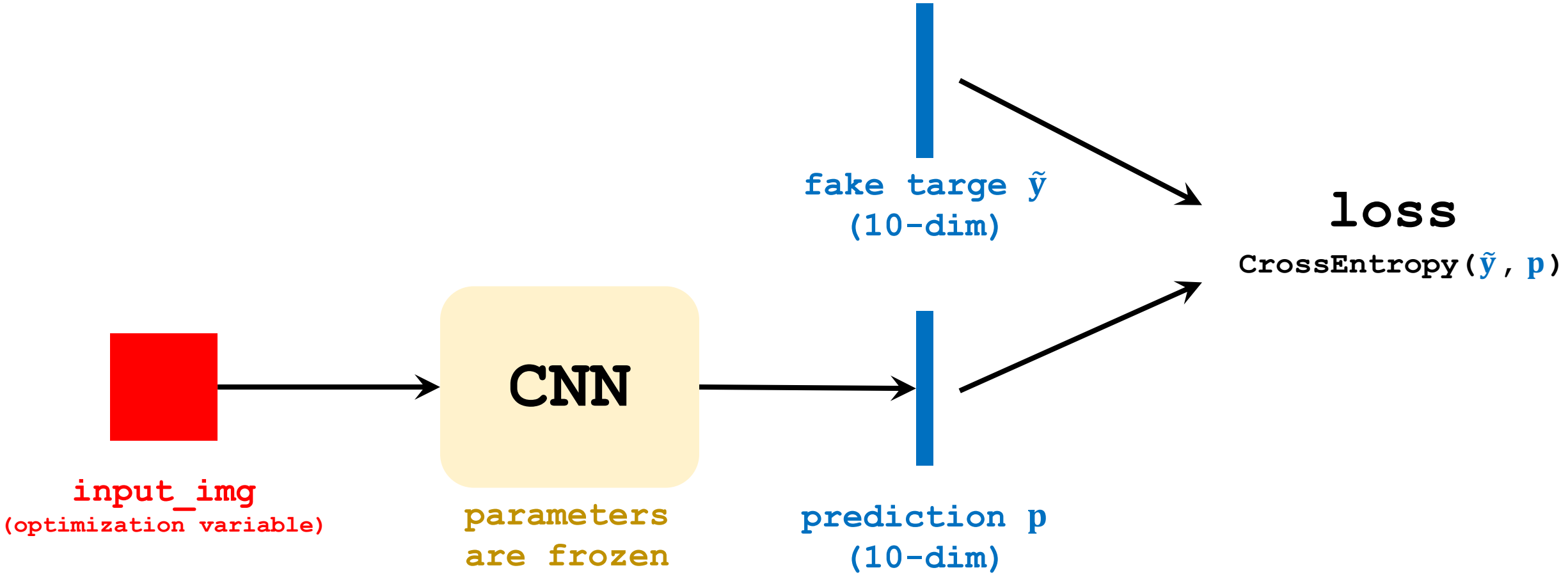
Generate Fake Image

Step 2: Fix the trained network's parameters, **W**.

```
from keras.layers import Input

model.trainable = False
input_img = Input(shape=(28, 28, 1))
pred = model(input_img)
```

Generate Fake Image

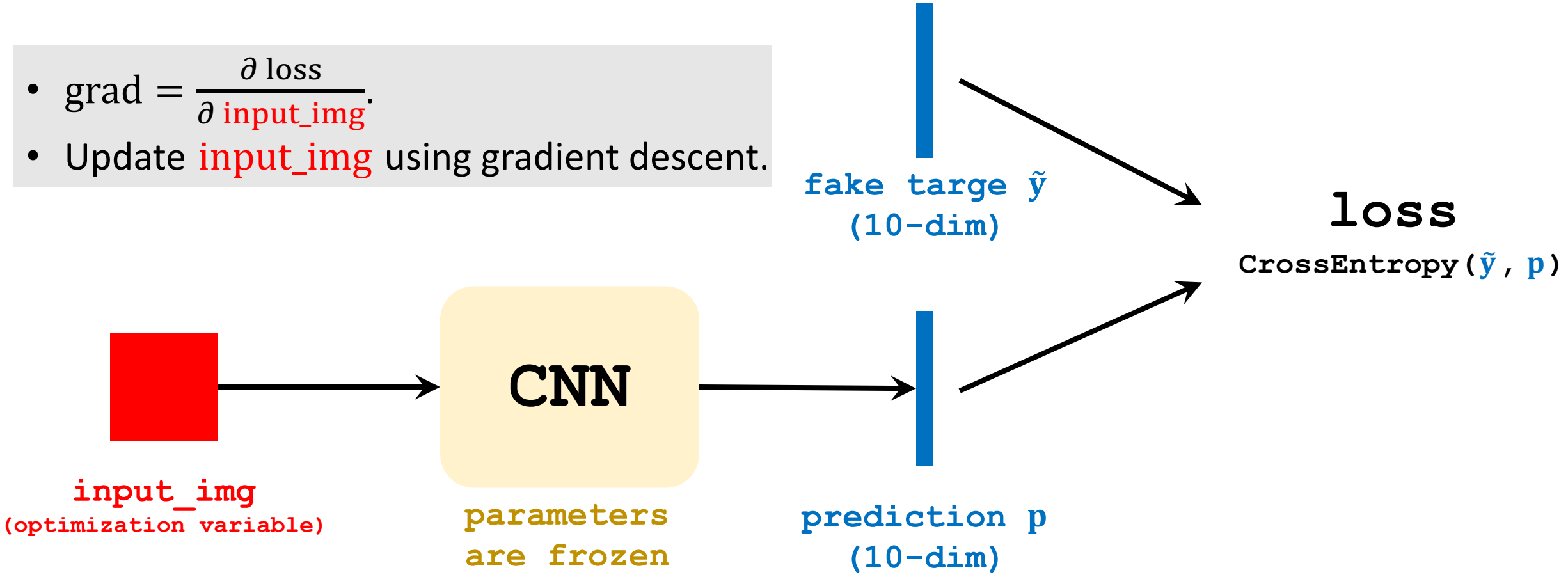
Step 3: Generate a fake image.



Generate Fake Image

Step 3: Generate a fake image.

- $\text{grad} = \frac{\partial\, \text{loss}}{\partial\, \text{input\_img}}$.
- Update input_img using gradient descent.



Generate Fake Image

Step 3: Generate a fake image.

- Define the loss function and evaluate the gradient.
  - **`pred`**: the output of the pre-trained model.
  - **`fake_target`**: $[1, 0, 0, \cdots, 0]$.
  - Loss function: `CrossEntropy(pred, fake_target)`.

```
import keras
from keras import backend as K

loss = keras.metrics.binary_crossentropy(pred, fake_target)
grads = K.gradients(loss, [input_img])[0]
fetch_loss_and_grads = K.function([input_img], [loss, grads])
```

Generate Fake Image

Step 3: Generate a fake image.

initialization

gradient descent

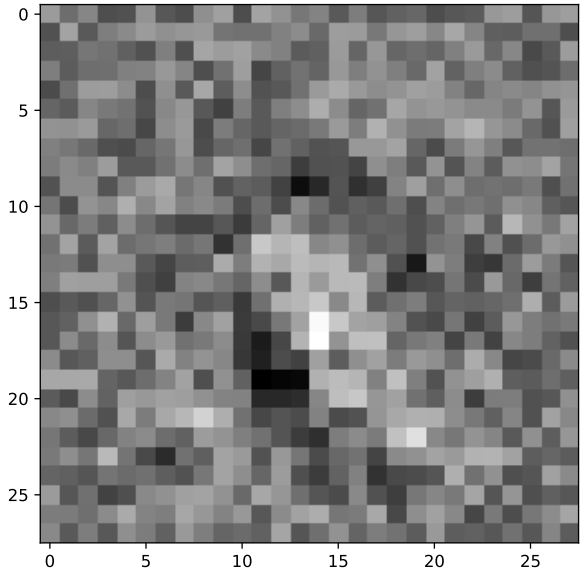
```
import numpy as np

fake_img = np.random.rand(1, 28, 28, 1)
learn_rate = 0.5

for i in range(10):
    l, g = fetch_loss_and_grads([fake_img])
    print('iter ' + str(i) + ': loss = ' + str(l))
    fake_img -= learn_rate * g
```

```
iter 0: loss = [3.039003]
iter 1: loss = [2.6458066]
iter 2: loss = [1.0068675]
iter 3: loss = [0.22192626]
iter 4: loss = [0.01752027]
iter 5: loss = [0.01353873]
iter 6: loss = [0.01130872]
iter 7: loss = [0.00976203]
iter 8: loss = [0.00869376]
iter 9: loss = [0.00787604]
```

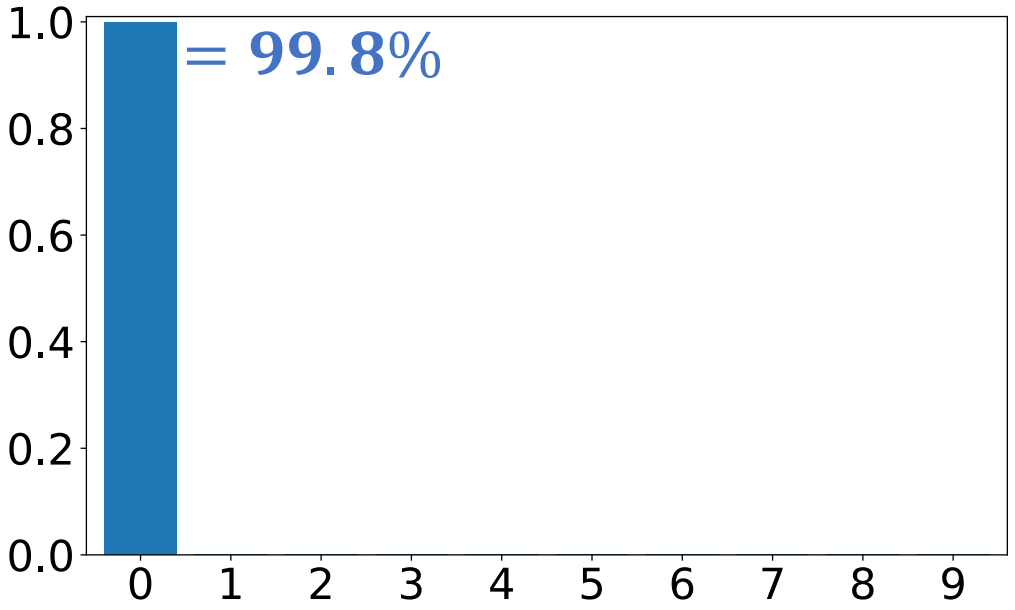
Generate Fake Image

```
print(model.predict(fake_img)[0])
```

```
[9.9776781e-01 4.1152799e-04 4.0594186e-04 8.5301363e-05 6.2220497e-05
 7.1304827e-04 2.1397672e-04 1.0672671e-04 1.9847257e-04 3.4956032e-05]
```



28×28 fake image



10-dim prediction vector

Untargeted Attack

Read the blog for details:

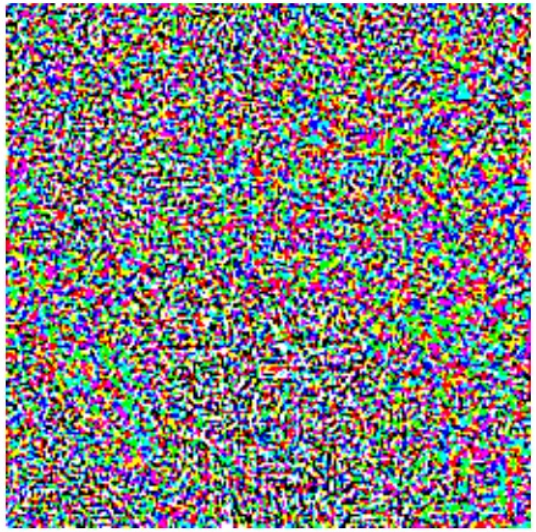
https://medium.com/onfido-tech/adversarial-attacks-and-defences-for-convolutional-neural-networks-66915ece52e7

Untargeted Attack: Goal

Goal:

- Add a small perturbation to an image. (The perturbation so small that a human can hardly notice the difference.)
- The neural network will make a wrong prediction.

$+ .007 \times$



$=$

"panda"
57.7% confidence

"gibbon"
99.3% confidence

Untargeted Attack

- $\mathbf{x}^\star$: a real image; $\mathbf{y}$: true label.
- Generate fake image $\tilde{\mathbf{x}}$ (which is close to $\mathbf{x}^\star$) such that

$$\text{Dist}(\mathbf{y},\ \mathbf{f}(\tilde{\mathbf{x}}\,;\ \mathbf{W}^\star)) \ \gg\ \text{Dist}(\mathbf{y},\ \mathbf{f}(\mathbf{x}^\star\,;\ \mathbf{W}^\star)).$$

- $\mathbf{f}\,(\tilde{\mathbf{x}}\,;\ \mathbf{W}^\star)$ is far from $\mathbf{y}$ ➔ Wrong prediction for $\tilde{\mathbf{x}}$.

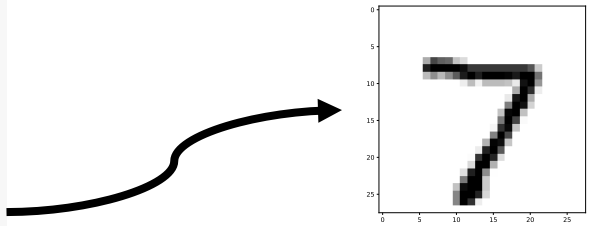
Untargeted Attack

- $\mathbf{x}^\star$: a real image; $\mathbf{y}$: true label.
- Initialize $\mathbf{x}$ by $\mathbf{x}^\star$. (So that $\tilde{\mathbf{x}}$ is close to $\mathbf{x}^\star$.)
- Repeat gradient ascent:
  - Repeat: $\mathbf{x} \leftarrow \mathbf{x} + \alpha \frac{\partial\, \mathrm{dist}(\mathbf{y},\, \mathbf{f}\,(\mathbf{x}\,;\, \mathbf{W}^\star))}{\partial\, \mathbf{x}}$.
  - Stop if the prediction, $\mathbf{f}(\mathbf{x}\,;\, \mathbf{W}^\star)$, is incorrect.
- Return $\tilde{\mathbf{x}} \leftarrow \mathbf{x}$.

Untargeted Attack

Step 1: Get an image and its true target.

```
# get the i-th test sample
i = 0
digit = x_test[i].reshape((1, 28, 28, 1))
label = y_test[i]
print('The true label is ' + str(label))

# one-hot encode the label
y_true = np.zeros((1, 10))
y_true[0, label] = 1
print('One-hot encode: ' + str(y_true))
```



```
The true label is 7
One-hot encode: [[0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]]
```

Untargeted Attack

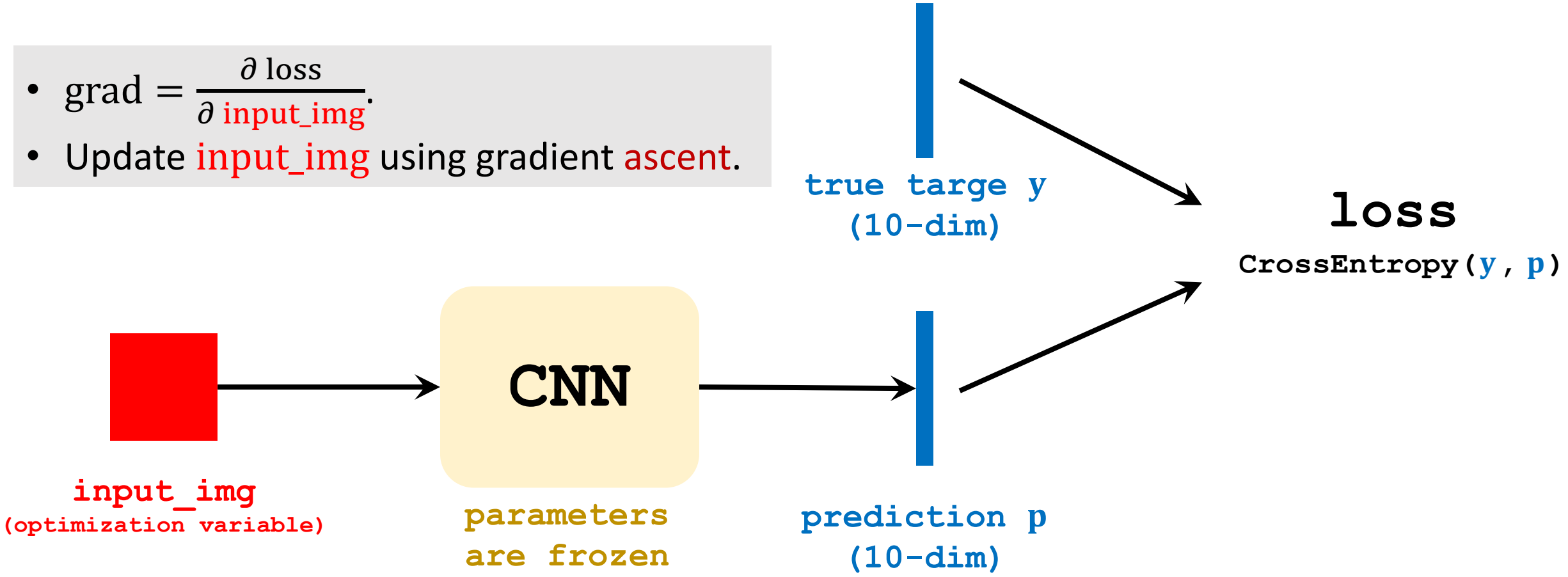
Step 2: Fix the trained network's parameters.

```
from keras.layers import Input


input_img = Input(shape=(28, 28, 1))
true_target = Input(tensor=K.constant(y_true))
pred = model(input_img)
```

Untargeted Attack

Step 3: Generate a fake image using gradient ascent.

- $\text{grad} = \frac{\partial\, \text{loss}}{\partial\, \text{input\_img}}$.
- Update input_img using gradient ascent.



Untargeted Attack

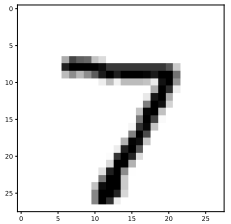
Step 3: Generate a fake image using gradient ascent.

```
import keras
from keras import backend as K

loss = keras.metrics.binary_crossentropy(pred, true_target)
grads = K.gradients(loss, [input_img])[0]
fetch_grads = K.function([input_img], [grads])
```

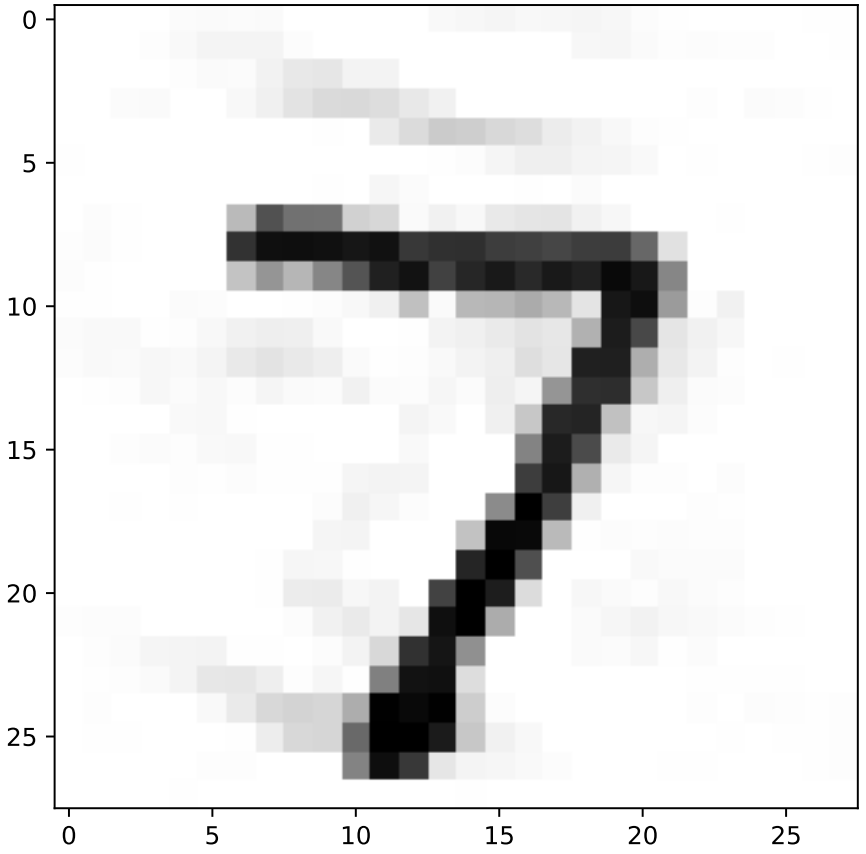
Untargeted Attack

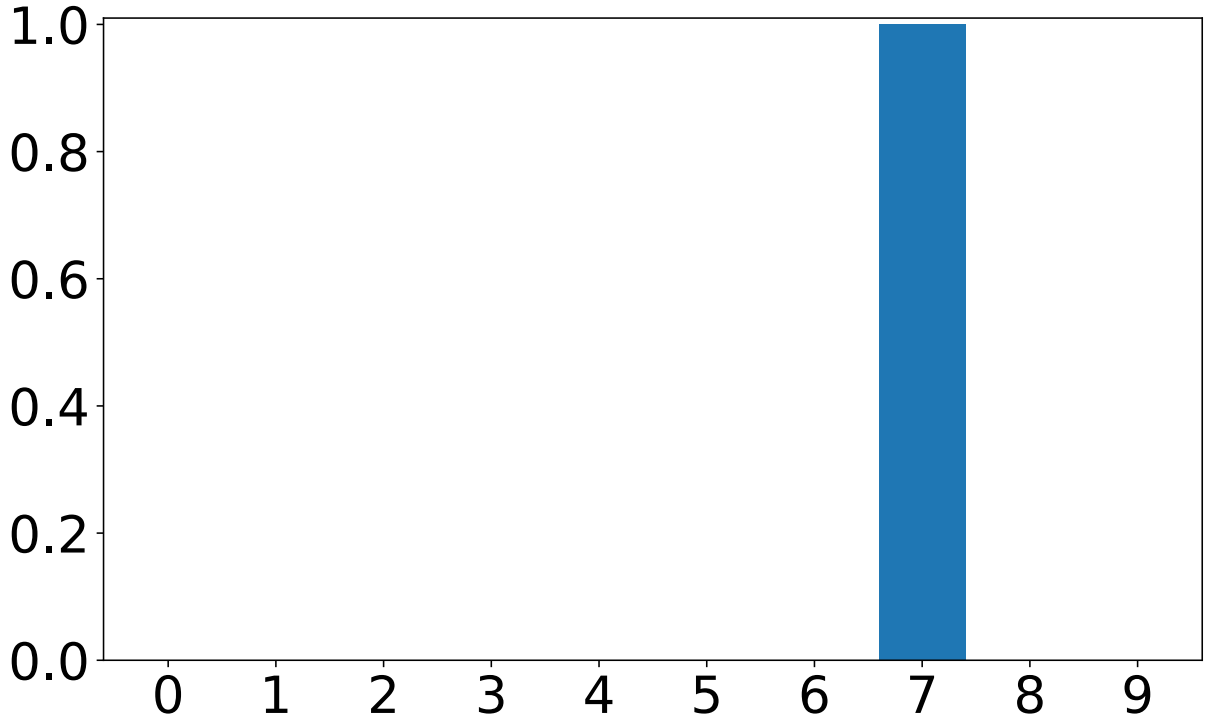
Step 3: Generate a fake image using gradient ascent.

```
import numpy

fake_img = digit.copy() # initialize

for t in range(20):
    # Gradient ascent
    grad = fetch_grads([fake_img])[0]
    g = grad / numpy.max(numpy.abs(grad))
    fake_img += 0.05 * g
    # project to [0, 1]
    fake_img[fake_img < 0] = 0
    fake_img[fake_img > 1] = 1
```

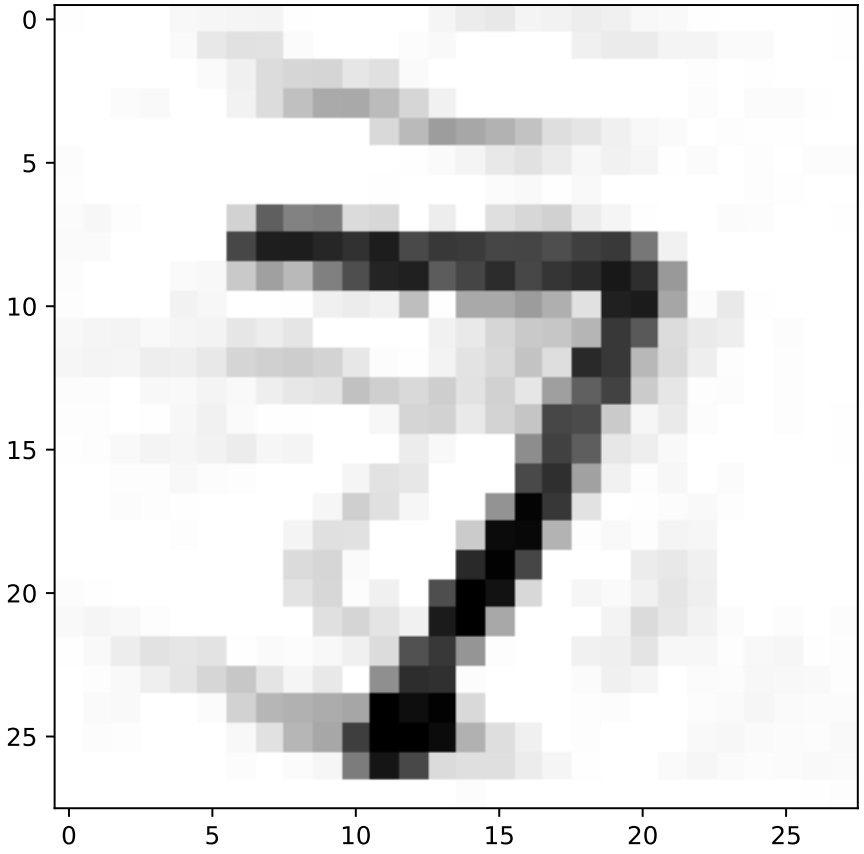


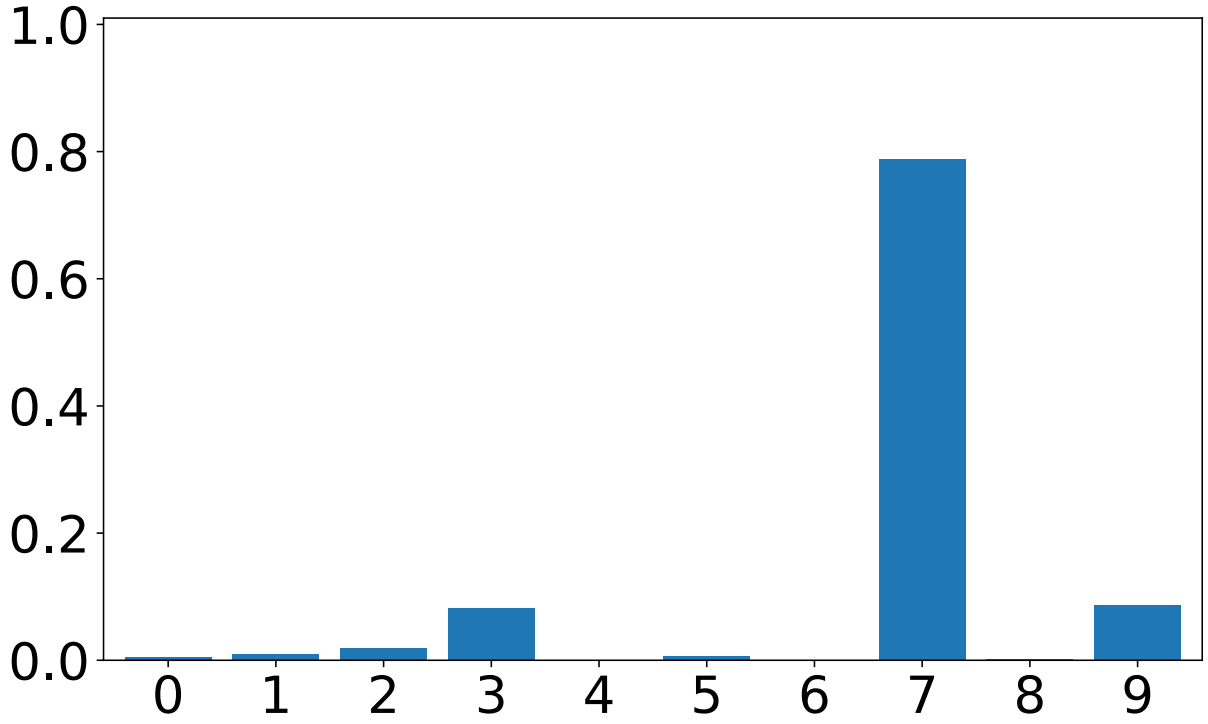
Results after 5 Steps



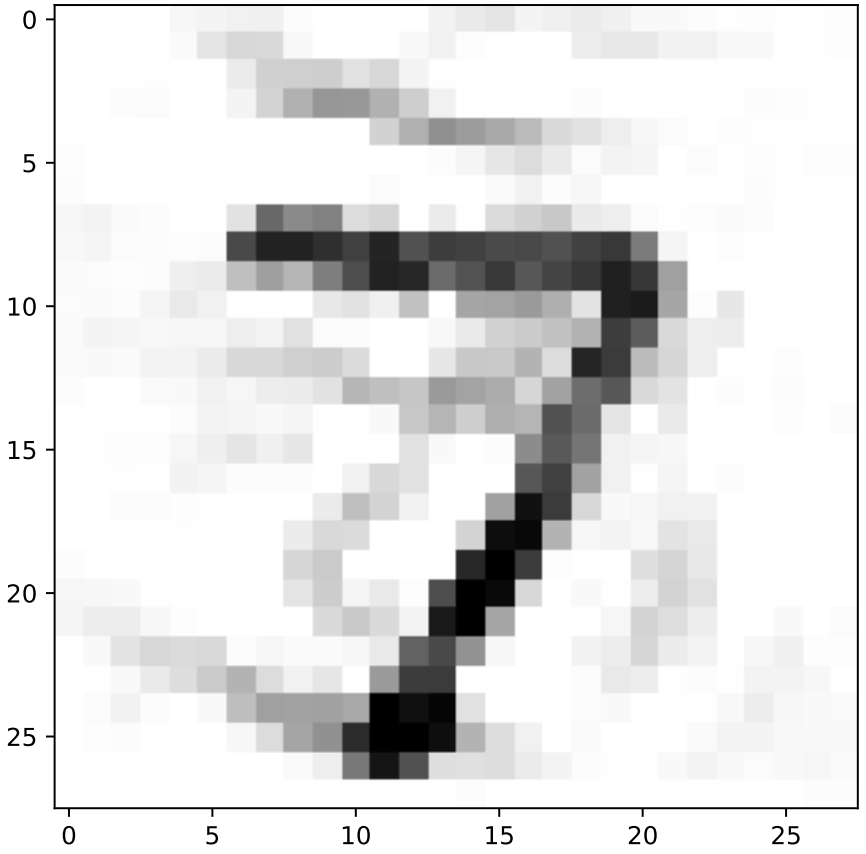


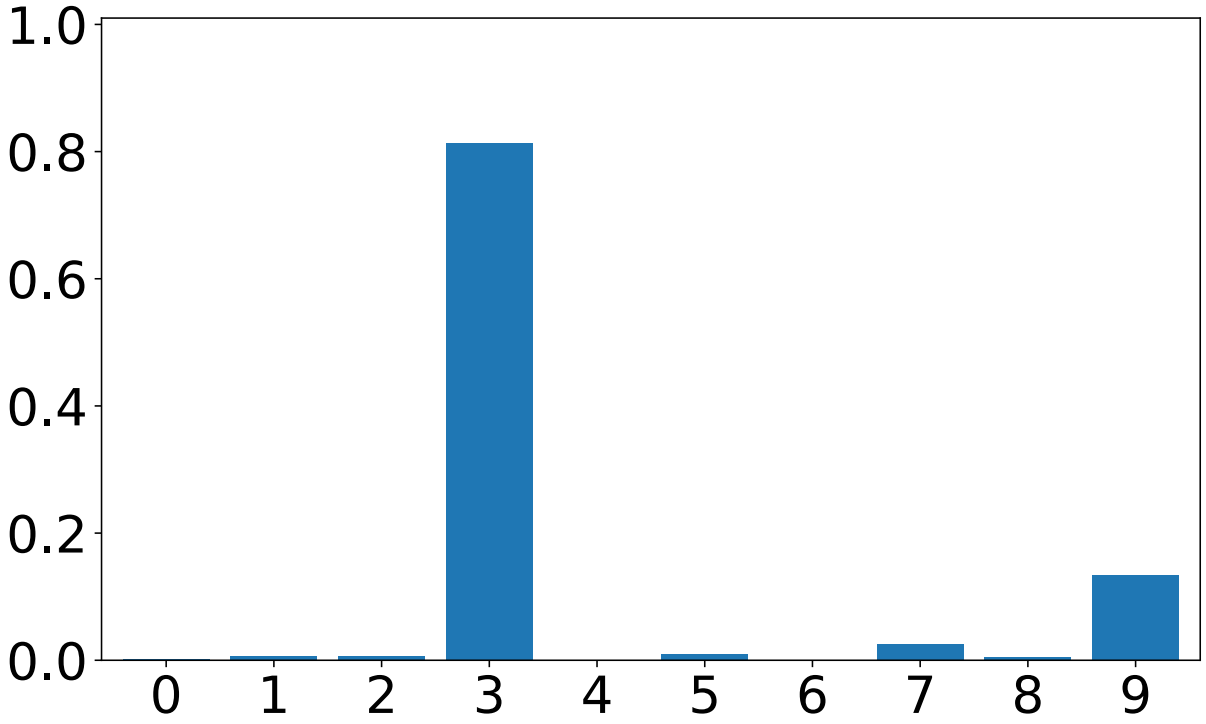
Results after 10 Steps



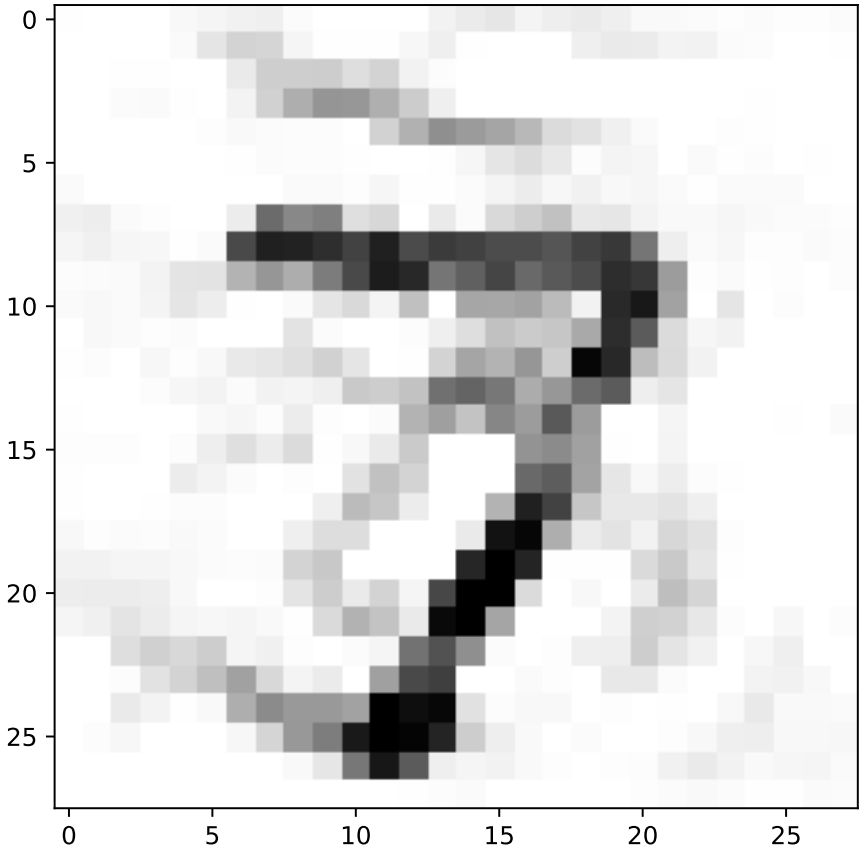


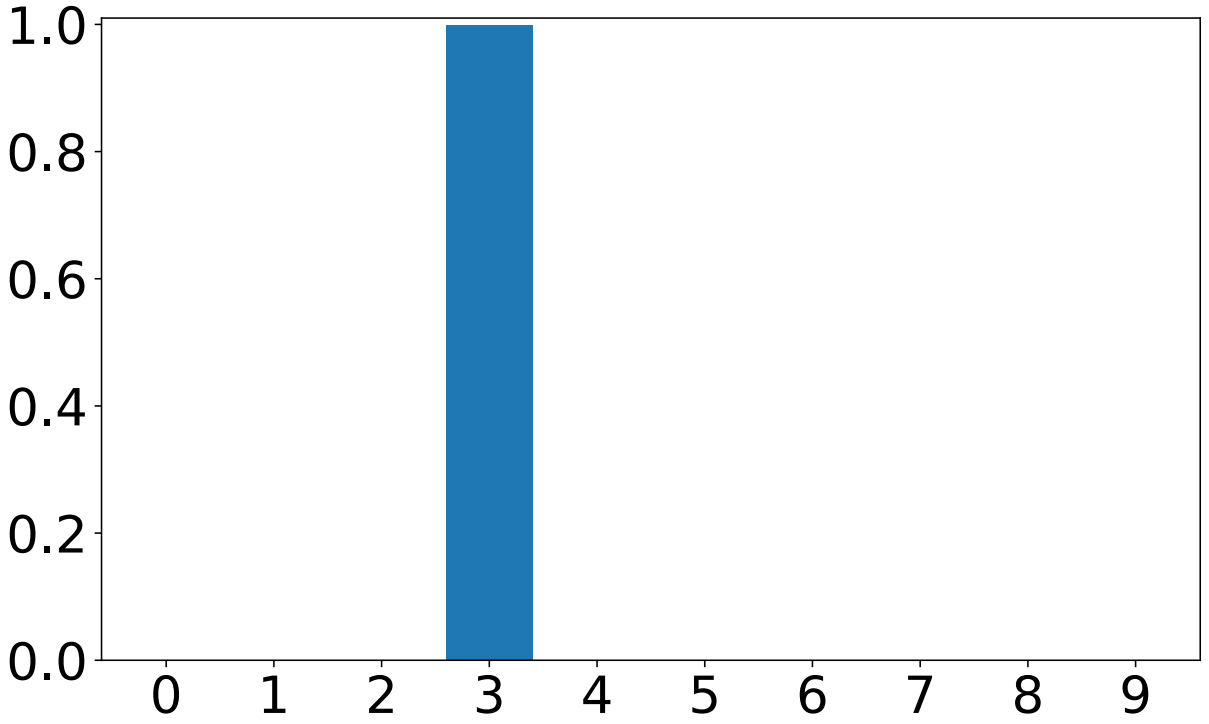
Results after 15 Steps



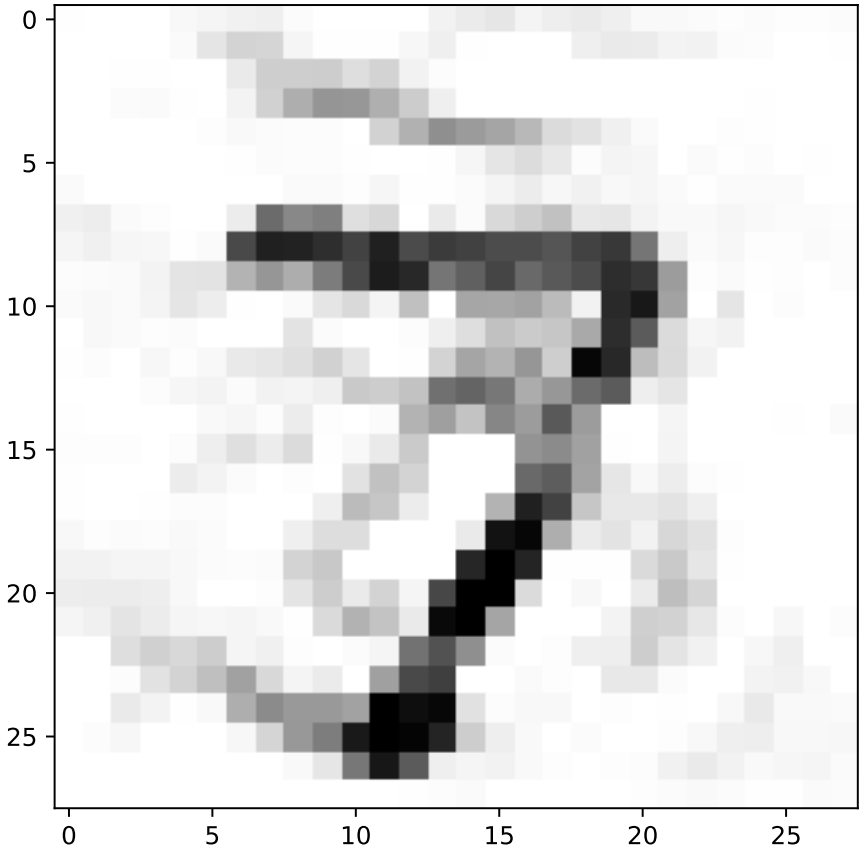


Results after 20 Steps





Results after 20 Steps



- Shallow neural nets are relatively robust.
  - Big perturbation is required.
- Deep neural nets are easier to attack.
  - A small perturbation will make the model err.

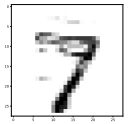
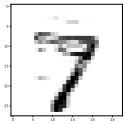
Targeted Attack

Untargeted Attack v.s. Targeted Attack

Goal of untargeted attack:

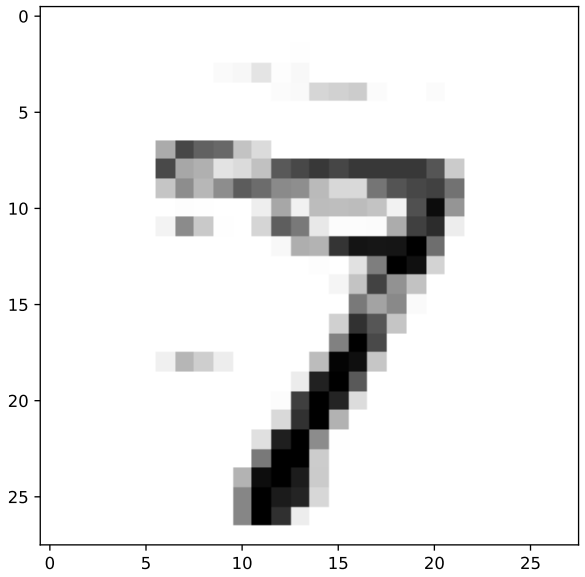
- Make the neural net err (disregarding the outcome.)
- E.g., make the neural net believe  is not digit “7”.
- No control over the prediction.

Goal of targeted attack:

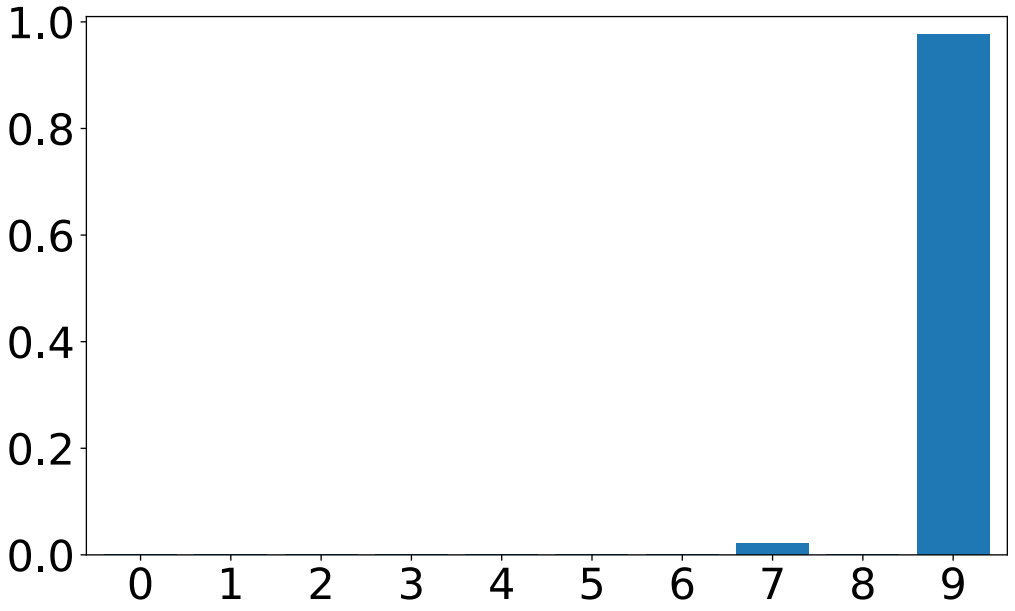
- The neural network will make the wrong prediction as we set.
- If we set the fake target to “9”, then we will get  .
- The neural net believes  is digit “9”.

Targeted Adversarial Example

- Our trained CNN thinks the 28×28 input image is digit "9".



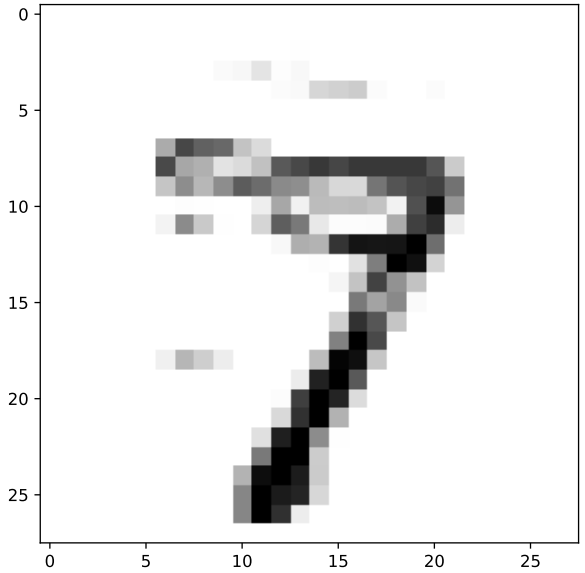
28×28 input image



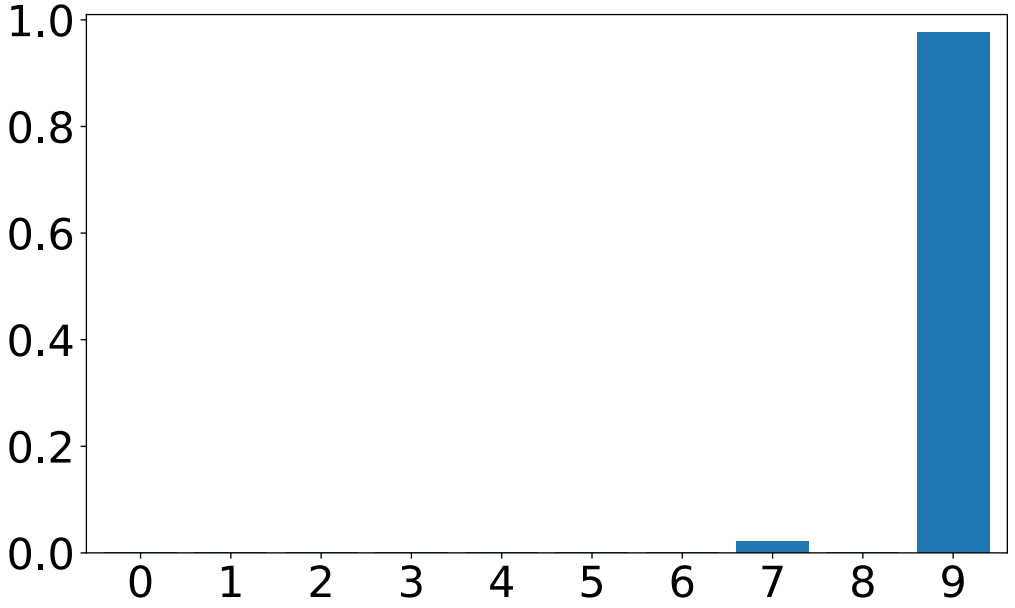
10-dim prediction vector

Targeted Adversarial Example

Question: How is the fake image generated?



`28×28 input image`



`10-dim prediction vector`

Targeted Attack

Question: How is the fake image generated?

1. Given a real image $\mathbf{x}^\star$ which we want to attack.
2. Set a fake target, e.g., $\tilde{\boldsymbol{y}} = [0, 0, 0, \cdots, 0, 1]$.
3. Fix the network parameters to $\mathbf{W}^\star$.
4. Set up the loss function

$$\text{Loss}(\tilde{\mathbf{x}}, \mathbf{x}^\star, \tilde{\mathbf{y}}) = \text{Dist}(\tilde{\mathbf{y}}, \mathbf{f}(\tilde{\mathbf{x}}; \mathbf{W}^\star)) + \lambda \left|\left|\tilde{\mathbf{x}} - \mathbf{x}^\star\right|\right|_2^2$$

make the neural net think the generated image $\tilde{\mathbf{x}}$ is digit "9".

the generated image $\tilde{\mathbf{x}}$ should look similar to the real one $\mathbf{x}^\star$.

Targeted Attack

Question: How is the fake image generated?

1. Given a real image $\mathbf{x}^\star$ which we want to attack.
2. Set a fake target, e.g., $\tilde{\mathbf{y}} = [0, 0, 0, \cdots, 0, 1]$.
3. Fix the network parameters to $\mathbf{W}^\star$.
4. Set up the loss function

$$\text{Loss}(\tilde{\mathbf{x}}, \mathbf{x}^\star, \tilde{\mathbf{y}}) = \text{Dist}(\tilde{\mathbf{y}}, \mathbf{f}(\tilde{\mathbf{x}}; \mathbf{W}^\star)) + \lambda \left|\left|\tilde{\mathbf{x}} - \mathbf{x}^\star\right|\right|_2^2$$

make the neural net think the generated image $\tilde{\mathbf{x}}$ is digit "0".

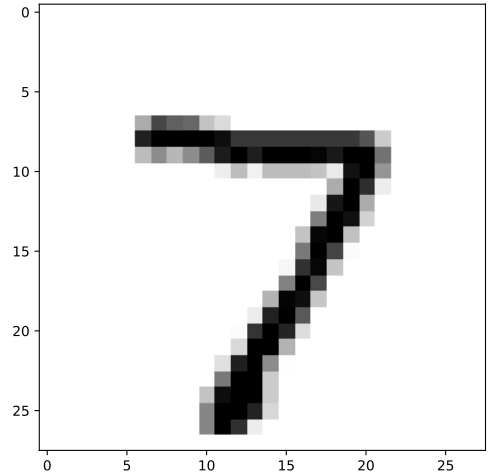
the generated image $\tilde{\mathbf{x}}$ should look similar to the real one $\mathbf{x}^\star$.

5. Generate a fake image $\tilde{\mathbf{x}}$ by $\tilde{\mathbf{x}} = \underset{\mathbf{x}}{\text{argmin}}\, \text{Loss}(\mathbf{x}, \mathbf{x}^\star, \tilde{\mathbf{y}})$.

Targeted Attack

Step 1: Given a real image which we want to attack.

```
from keras.layers import Input
from keras import backend as K

i = 0
digit = x_test[i].reshape((1, 28, 28, 1))
print('The true label is ' + str(y_test[i]))

true_img = Input(tensor=K.constant(digit))
```




```
The true label is 7
```

Targeted Attack

Step 2: Set a fake target to be digit "**9**".

```
import numpy as np
from keras.layers import Input
from keras import backend as K

j = 9 # the fake lable
y_tilde = np.zeros((1, 10))
y_tilde[0, j] = 1
print(y_tilde)

fake_target = Input(tensor=K.constant(y_tilde))
```

```
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]]
```

Targeted Attack

Step 3: Freeze the parameters. Define input image and prediction.

```
from keras.layers import Input

model.trainable = False
fake_img = Input(shape=(28, 28, 1))
pred = model(fake_img)
```

Targeted Attack

Step 4: Set the loss function.

- $\text{Loss}(\tilde{\mathbf{x}}, \mathbf{x}^\star, \tilde{\mathbf{y}}) = \text{CrossEntropy}\left(\tilde{\mathbf{y}}, \mathbf{f}\left(\tilde{\mathbf{x}}; \mathbf{W}^\star\right)\right) + \lambda\left|\left|\tilde{\mathbf{x}} - \mathbf{x}^\star\right|\right|_1$

or $\text{Loss}(\tilde{\mathbf{x}}, \mathbf{x}^\star, \tilde{\mathbf{y}}) = \text{CrossEntropy}\left(\tilde{\mathbf{y}}, \mathbf{f}\left(\tilde{\mathbf{x}}; \mathbf{W}^\star\right)\right) + \lambda\left|\left|\tilde{\mathbf{x}} - \mathbf{x}^\star\right|\right|_2^2$

make the neural net think the generated image $\tilde{\mathbf{x}}$ is digit "9".

the generated image $\tilde{\mathbf{x}}$ should look similar to the real one $\mathbf{x}^\star$.

Targeted Attack

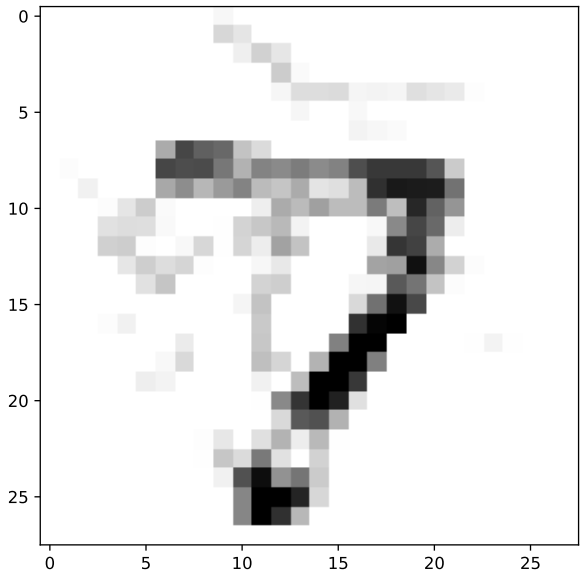
Step 4: Set the loss function.

- $\text{Loss}(\tilde{\mathbf{x}}, \mathbf{x}^\star, \tilde{\mathbf{y}}) = \text{CrossEntropy}\left(\tilde{\mathbf{y}}, \mathbf{f}\left(\tilde{\mathbf{x}};\ \mathbf{W}^\star\right)\right) + \lambda \left|\left|\tilde{\mathbf{x}} - \mathbf{x}^\star\right|\right|_1$

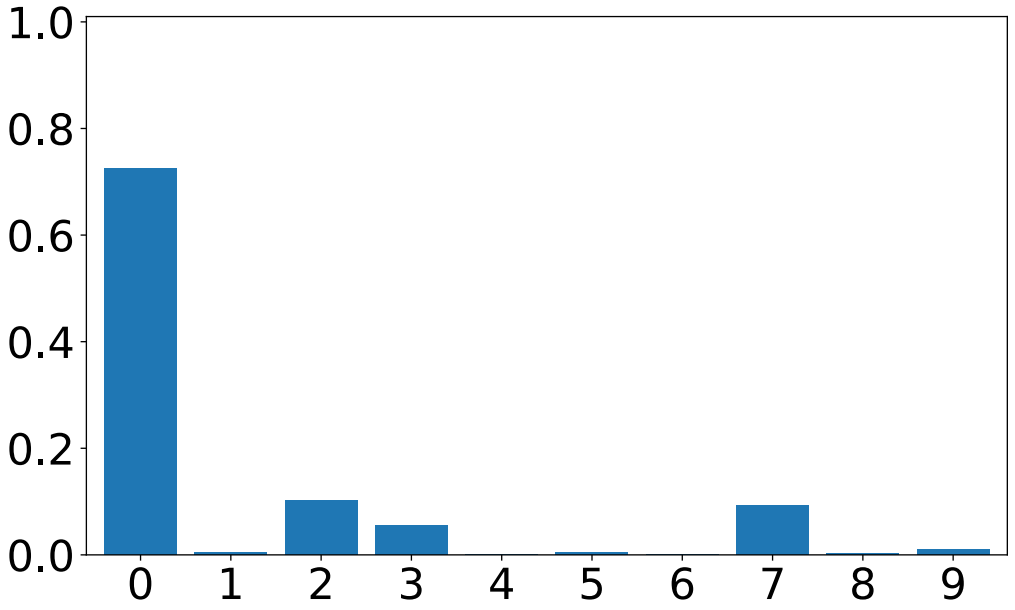
```
import keras
from keras import backend as K

param = 30

loss1 = keras.metrics.binary_crossentropy(pred, fake_target)
#loss2 = K.mean(K.square(true_img - fake_img))
loss2 = K.mean(K.abs(true_img - fake_img))

loss = loss1 + param * loss2

grads = K.gradients(loss, [fake_img])[0]
fetch_loss_and_grads = K.function([fake_img], [loss, grads])
```

Targeted Attack

- Our trained CNN thinks the 28×28 input image is digit "0".



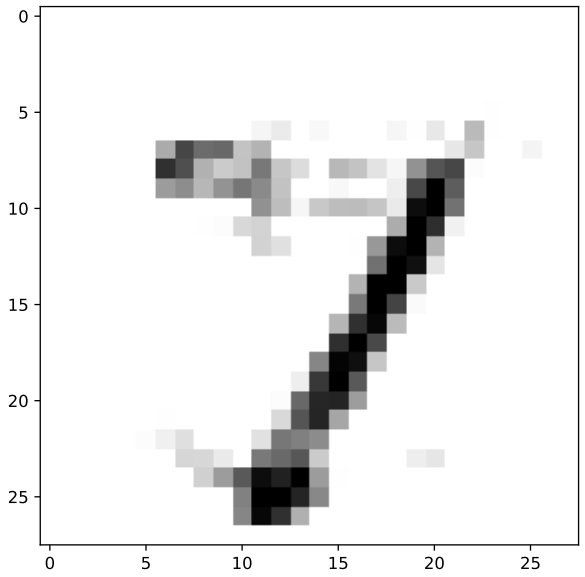
28×28 generated image



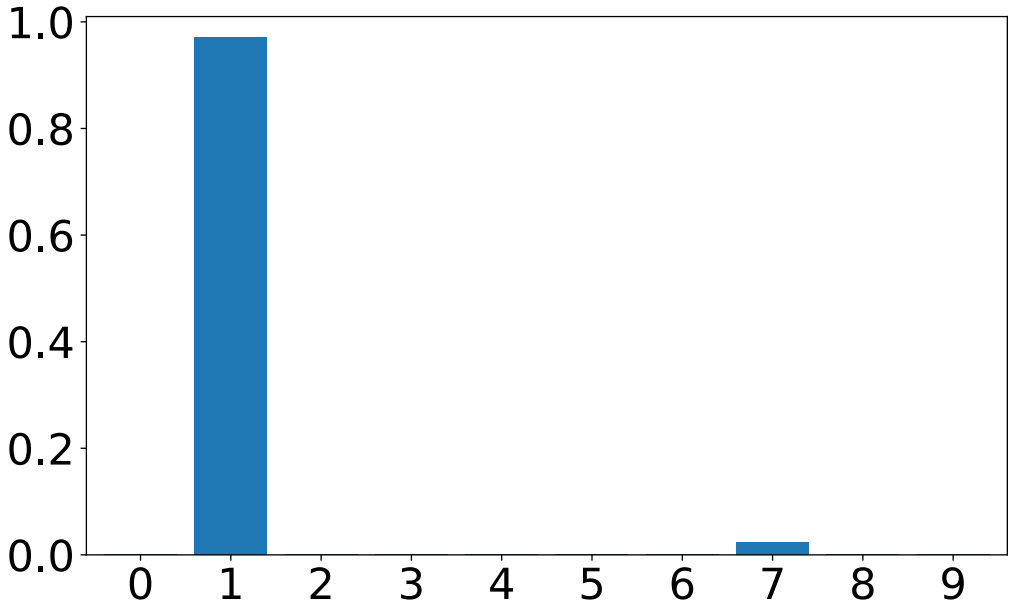
10-dim prediction vector

Targeted Attack

- Our trained CNN thinks the 28×28 input image is digit "1".



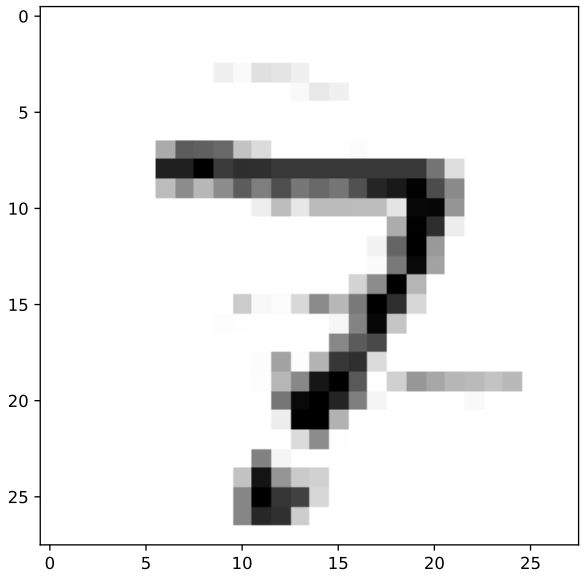
28×28 generated image



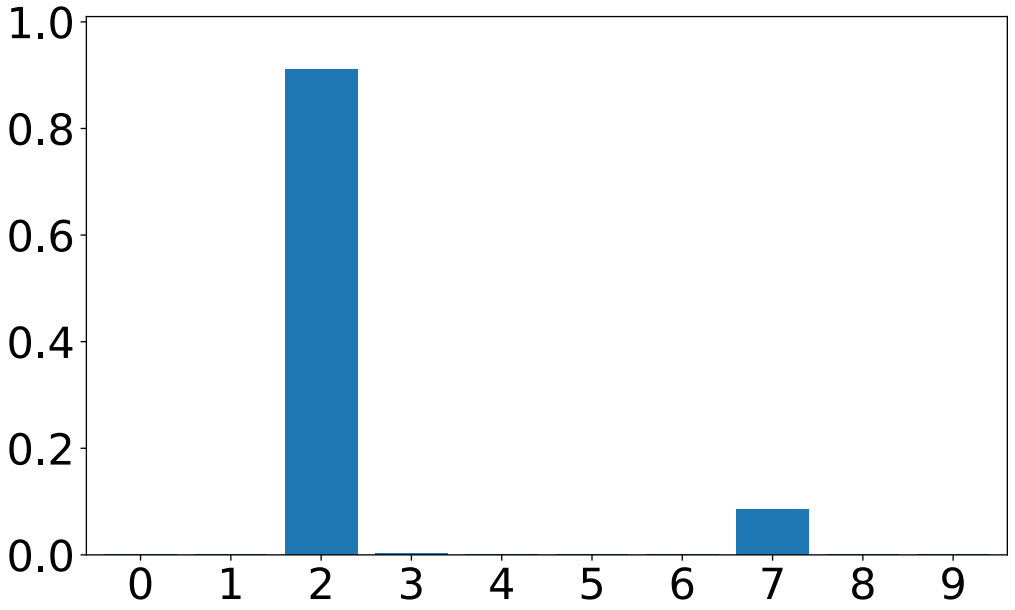
10-dim prediction vector

Targeted Attack

- Our trained CNN thinks the 28×28 input image is digit "2".



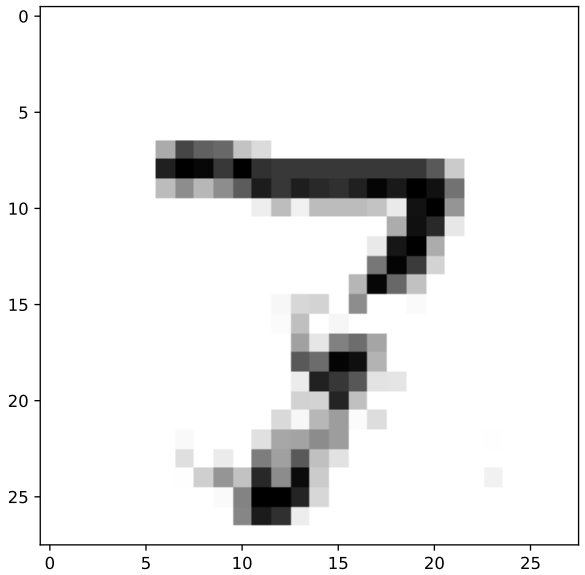
28×28 generated image



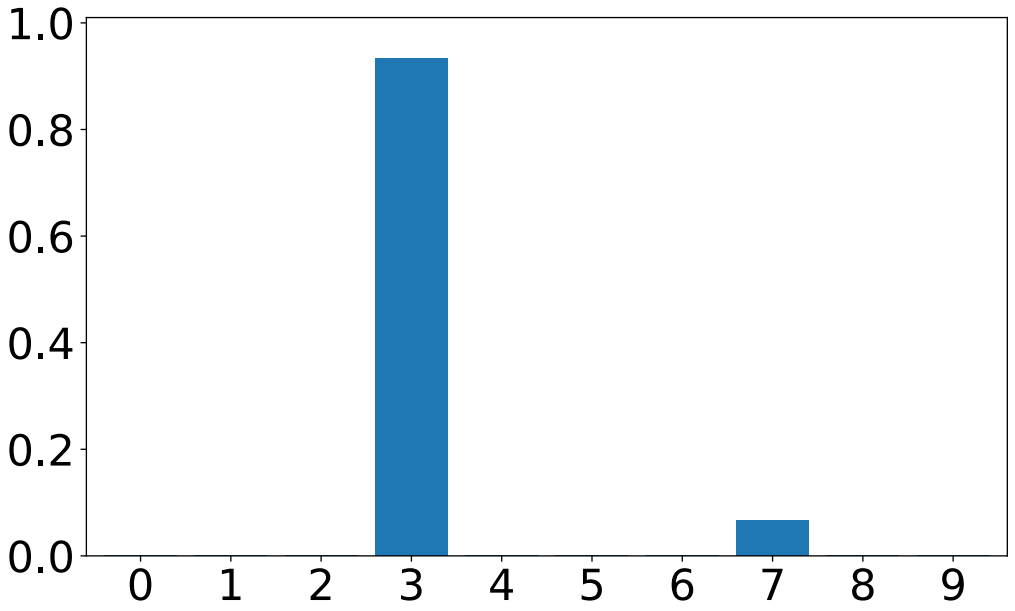
10-dim prediction vector

Targeted Attack

- Our trained CNN thinks the 28×28 input image is digit "3".



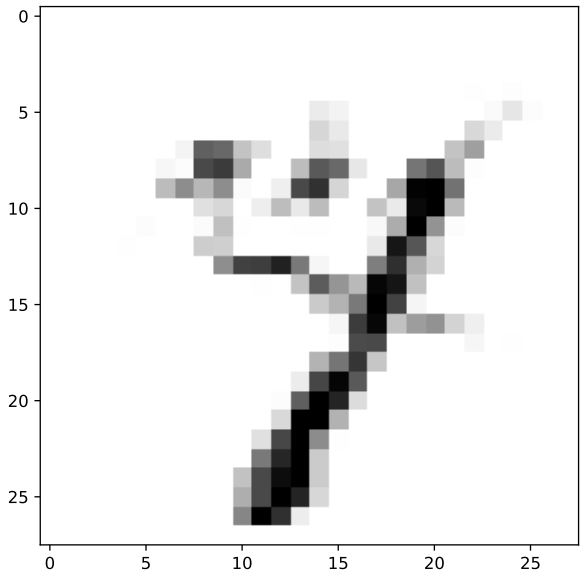
28×28 generated image



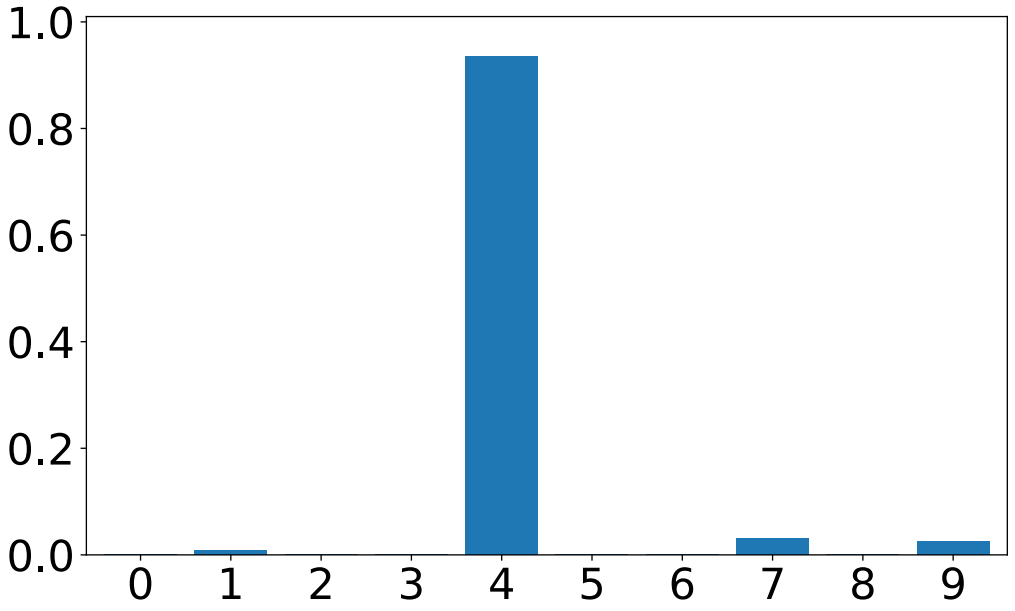
10-dim prediction vector

Targeted Attack

- Our trained CNN thinks the 28×28 input image is digit "**4**".



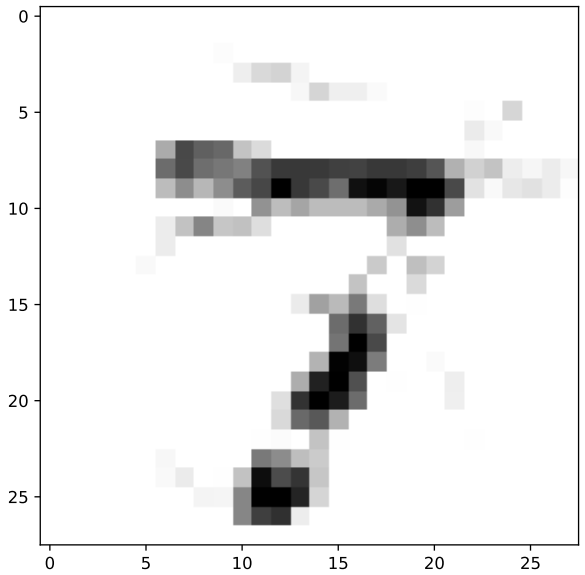
28×28 generated image



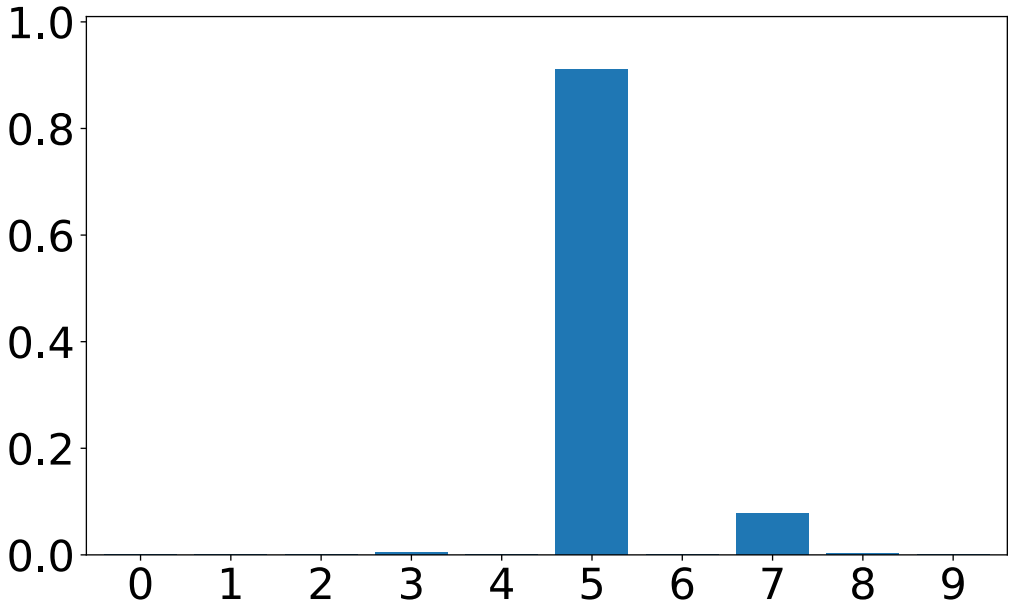
10-dim prediction vector

Targeted Attack

- Our trained CNN thinks the 28×28 input image is digit "5".



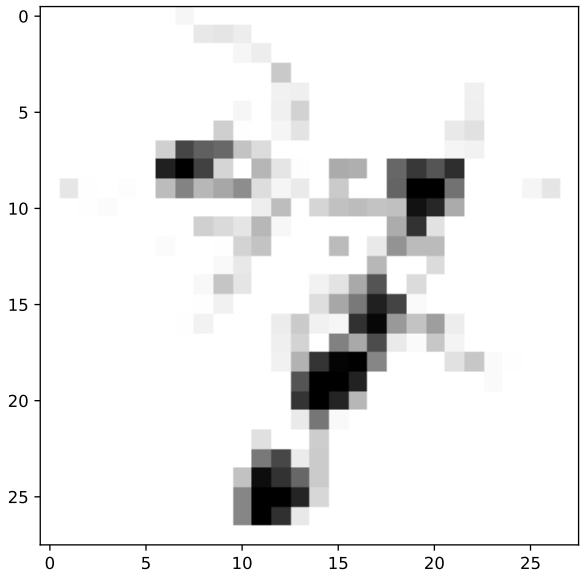
28×28 generated image



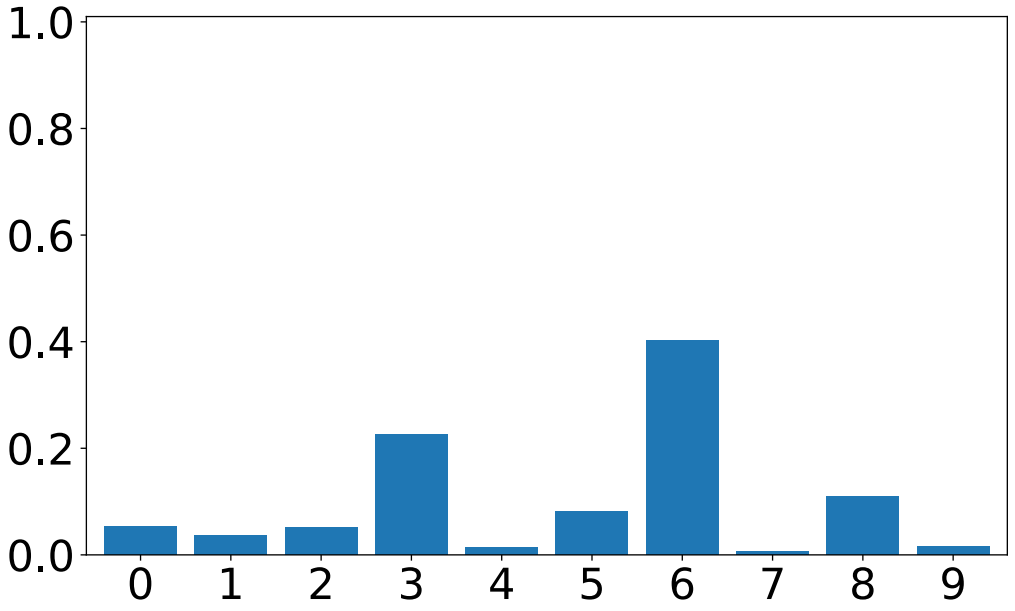
10-dim prediction vector

Targeted Attack

- Our trained CNN thinks the 28×28 input image is digit "6".



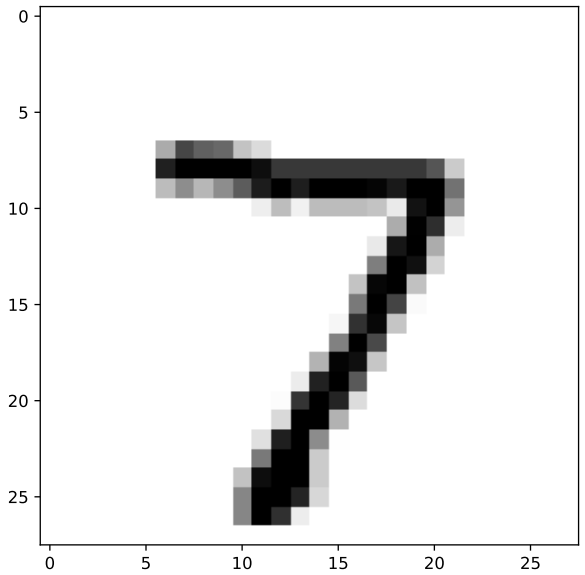
28×28 generated image



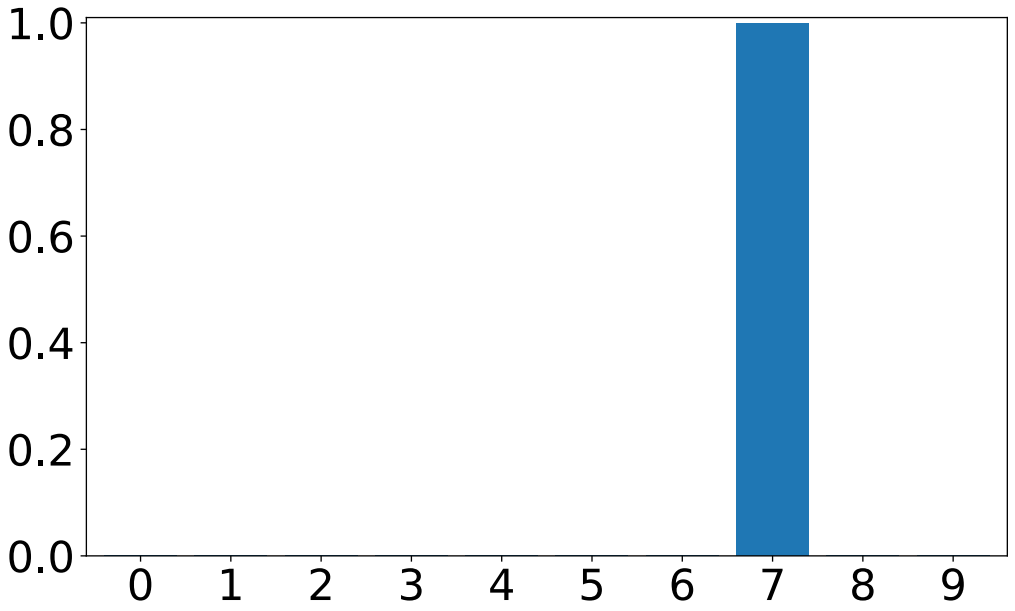
10-dim prediction vector

Targeted Attack

- Our trained CNN thinks the 28×28 input image is digit "7".



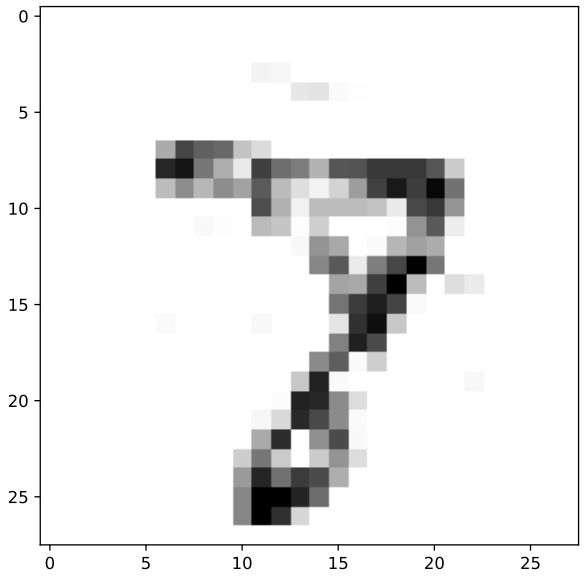
28×28 generated image



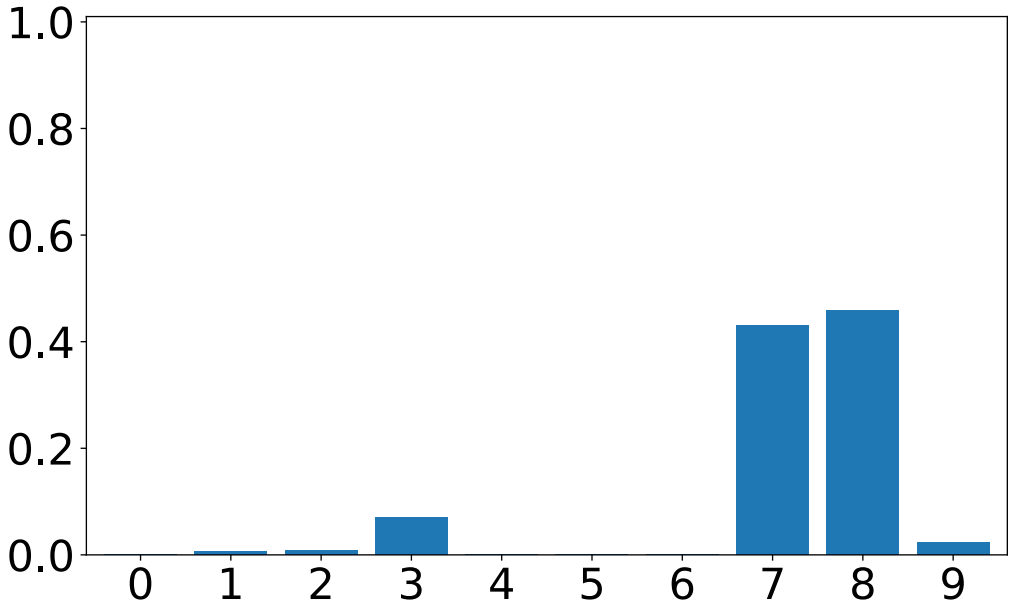
10-dim prediction vector

Targeted Attack

- Our trained CNN thinks the 28×28 input image is digit "8".



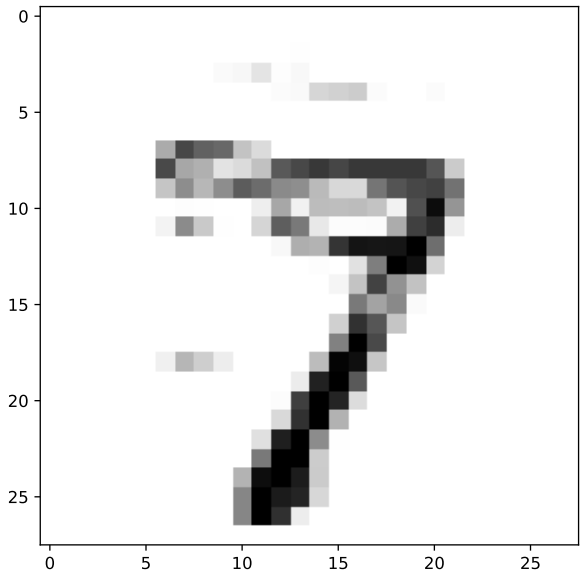
28×28 generated image

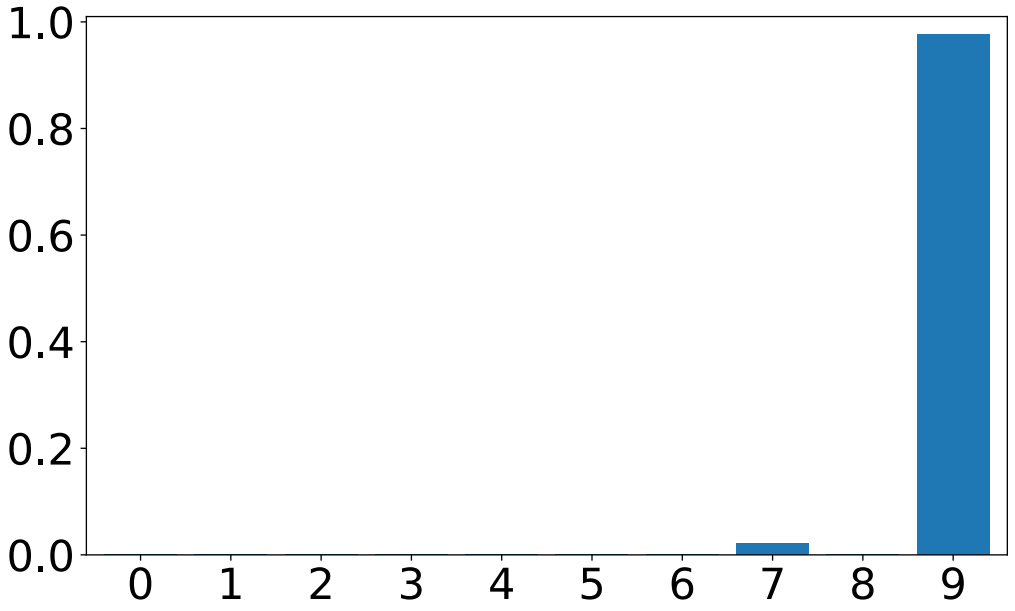


10-dim prediction vector

Targeted Attack

- Our trained CNN thinks the 28×28 input image is digit "9".

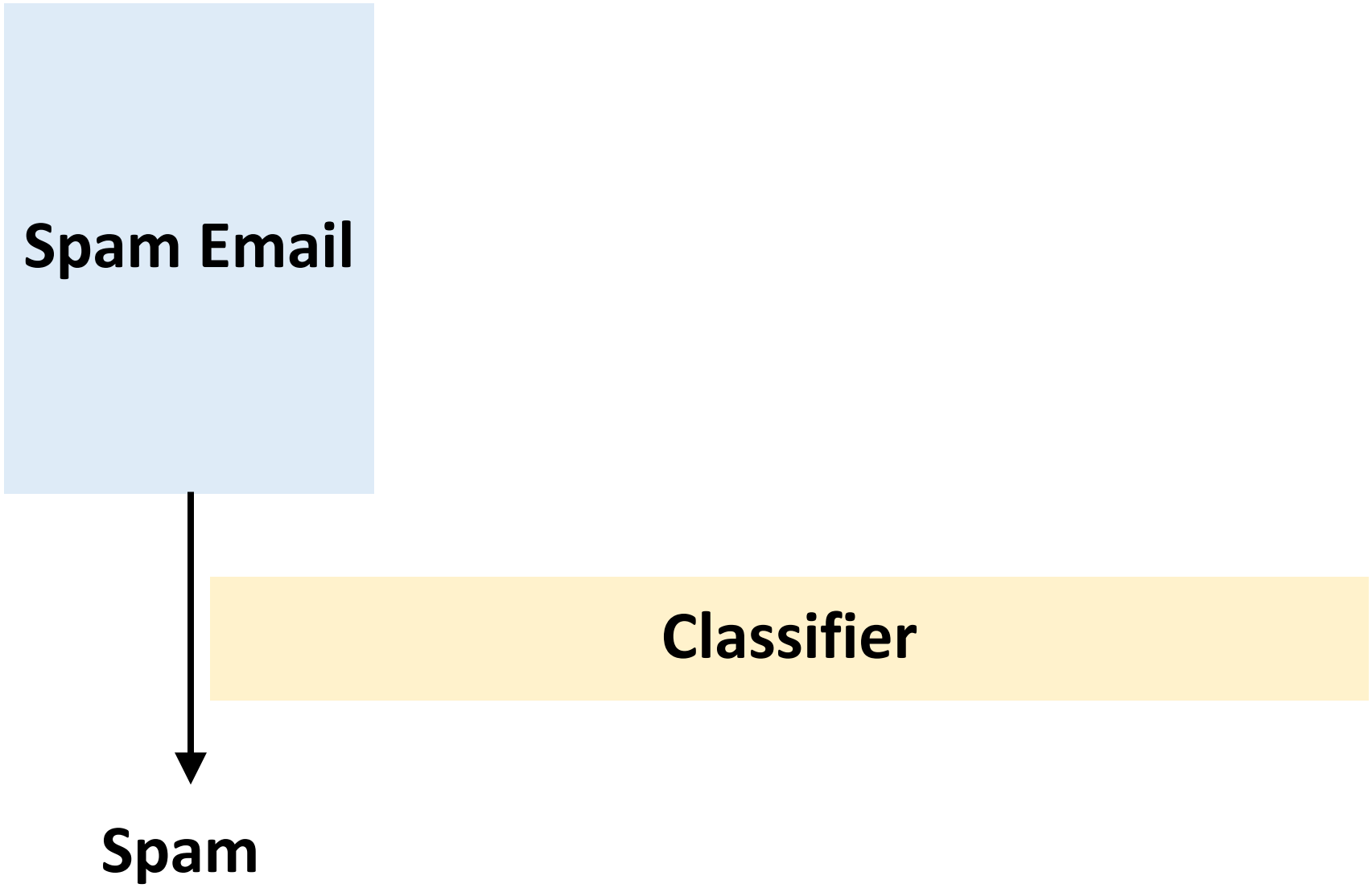




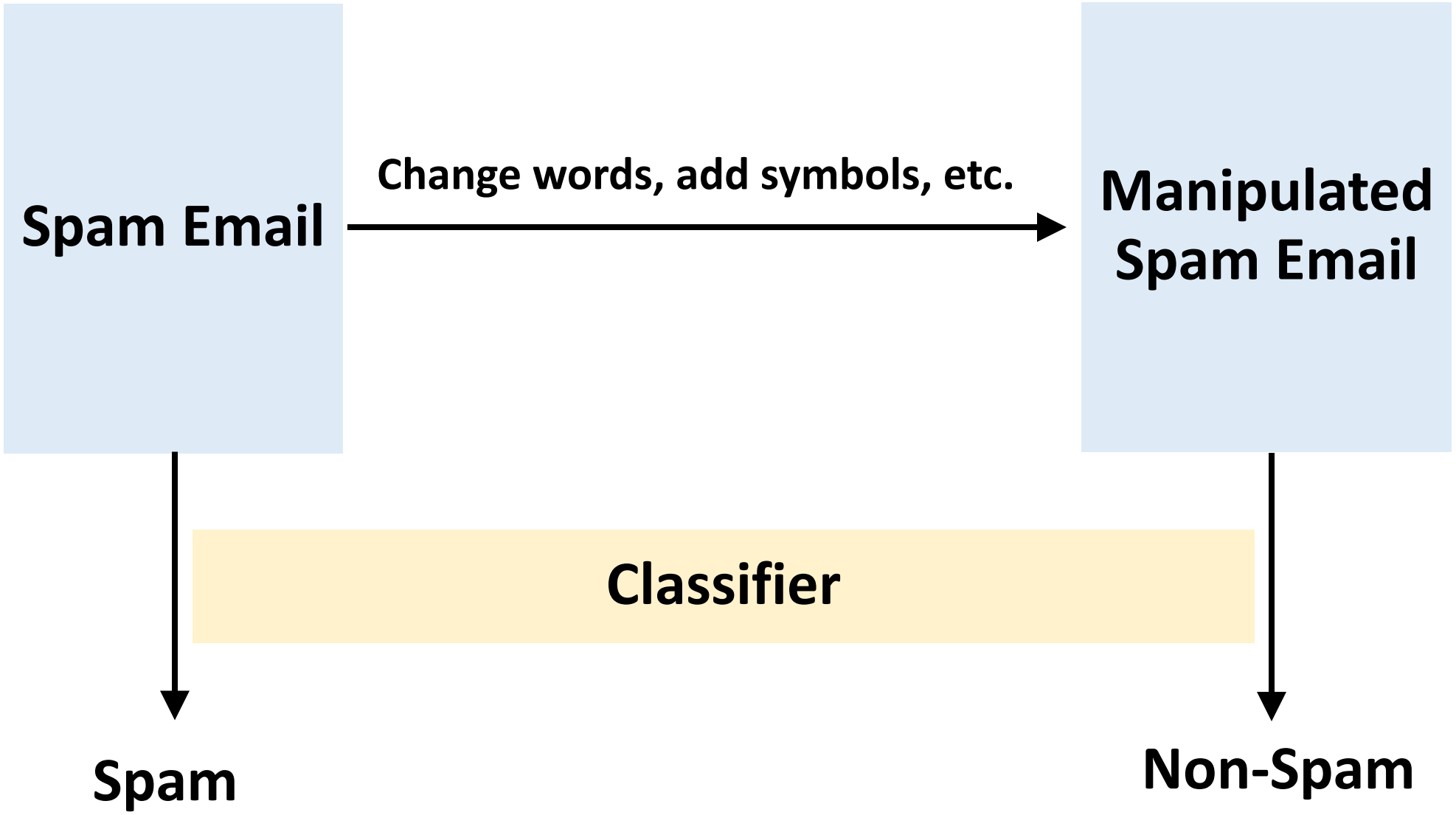
28×28 generated image

10-dim prediction vector

Targeted Attack Application



Targeted Attack Application



Adversarial Training: Min-Max Model

Reference:

- Madry et al. Towards deep learning models resistant to adversarial attacks. arXiv, 2017.

Adversarial Training

- Defense data evasion attacks.
  - Why do we care about adversarial robustness?
  - Slightly change a stop sign.
  - A self-driving car can ignore the stop sign.

Min-Max Model

- Standard model:

$$\min_{\mathbf{W}} \sum_{j=1}^{n} \text{Loss}\big(\mathbf{y}_j,\ \mathbf{f}\ (\mathbf{x}_j\ ;\ \mathbf{W})\big).$$

Min-Max Model

- Standard model:

$$\min_{\mathbf{W}} \sum_{j=1}^{n} \text{Loss}\big(\mathbf{y}_j,\ \mathbf{f}\ (\mathbf{x}_j\ ;\ \mathbf{W})\big).$$

- Min-max model (robust to adversary):

$$\min_{\mathbf{W}} \sum_{j=1}^{n} \left\{ \max_{||\boldsymbol{\delta}||<\sigma} \text{Loss}\big(\mathbf{y}_j,\ \mathbf{f}\ (\mathbf{x}_j + \boldsymbol{\delta}\ ;\ \mathbf{W})\big) \right\}.$$

- $\boldsymbol{\delta}$ is the perturbation.
- The maximization seeks to increase the loss.
- So that the model errs.

Min-Max Model

- Standard model:

$$\min_{\mathbf{W}} \sum_{j=1}^{n} \text{Loss}\big(\mathbf{y}_j,\ \mathbf{f}\ (\mathbf{x}_j\ ;\ \mathbf{W})\big).$$

- Min-max model (robust to adversary):

$$\min_{\mathbf{W}} \sum_{j=1}^{n} \Big\{ \max_{||\boldsymbol{\delta}||<\sigma} \text{Loss}\big(\mathbf{y}_j,\ \mathbf{f}\ (\mathbf{x}_j + \boldsymbol{\delta}\ ;\ \mathbf{W})\big) \Big\}.$$

Worst-case loss

Min-Max Model

- Standard model:

$$\min_{\mathbf{W}} \sum_{j=1}^{n} \text{Loss}\big(\mathbf{y}_j,\ \mathbf{f}\ (\mathbf{x}_j\ ;\ \mathbf{W})\big).$$

- Min-max model (robust to adversary):

$$\min_{\mathbf{W}} \sum_{j=1}^{n} \Big\{ \max_{||\boldsymbol{\delta}||<\sigma} \text{Loss}\big(\mathbf{y}_j,\ \mathbf{f}\ (\mathbf{x}_j + \boldsymbol{\delta}\ ;\ \mathbf{W})\big) \Big\}.$$

Worst-case loss

Minimization: Find $\mathbf{W}$ that works well even in the worst case.

Min-Max Model: Optimization

Min-max model:

$$\min_{\mathbf{W}} \sum_{j=1}^{n} \left\{ \max_{||\boldsymbol{\delta}||<\sigma} \ \text{Loss}\left(\mathbf{y}_j,\ \mathbf{f}\left(\mathbf{x}_j + \boldsymbol{\delta}\ ;\ \mathbf{W}\right)\right) \right\}.$$

Question: How to solve the model?

Min-Max Model: Optimization

Min-max model:

$$\min_{\mathbf{W}} \sum_{j=1}^{n} \left\{ \max_{||\boldsymbol{\delta}||<\sigma} \ \text{Loss}\big(\mathbf{y}_j,\ \mathbf{f}\ (\mathbf{x}_j + \boldsymbol{\delta}\ ;\ \mathbf{W})\big) \right\}.$$

Alternating maximization and minimization:

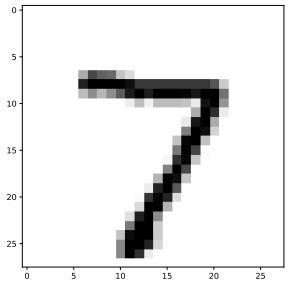
1. Randomly sample an index j from $\{1, \cdots, n\}$.
2. Adversary: $\boldsymbol{\delta}^{\star} = \underset{||\boldsymbol{\delta}||<\sigma}{\text{argmax}} \ \text{Loss}\big(\mathbf{y}_j,\ \mathbf{f}\ (\mathbf{x}_j + \boldsymbol{\delta}\ ;\ \mathbf{W})\big)$.
3. Compute $\dfrac{\partial\ \text{Loss}\big(\mathbf{y}_j,\ \mathbf{f}\ (\mathbf{x}_j + \boldsymbol{\delta}^{\star}\ ;\ \mathbf{W})\big)}{\partial\ \mathbf{W}}$ and update $\mathbf{W}$ using SGD.

Min-Max Model: Optimization

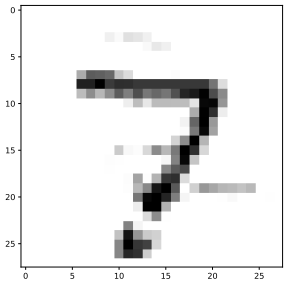
Another way to understand adversarial training.

Repeat:

1. Randomly select a real sample; generate an adversarial sample.
2. Replace real sample by adversarial sample. (Use the true label, e.g., “7”.)
3. Update the model parameters using the adversarial sample and true label.



Real sample



Adversarial sample

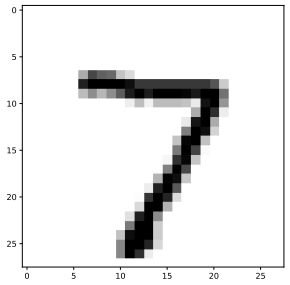
Min-Max Model: Optimization

Another way to understand adversarial training.

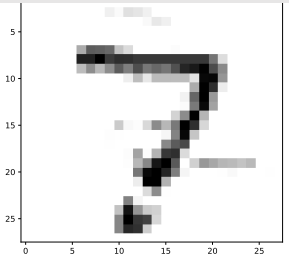
Repeat:

1. Randomly select a real sample; generate an adversarial sample.
2. Replace real sample by adversarial sample. (Use the true label, e.g., “7”.)
3. Update the model parameters using the adversarial sample and true label.

Let the model know this is “7”, not “2”.



Real sample



Adversarial sample

Adversarial Training: Gradient Regularization

Reference:

- Ross and Doshi-Velez. Improving the adversarial robustness and interpretability of deep neural networks by regularizing their input gradients. In *AAAI*, 2018.

Gradient Regularization

- Standard model:

$$\min_{\mathbf{W}} \sum_{j=1}^{n} \text{Loss}\left(\mathbf{y}_j,\ \mathbf{f}\left(\mathbf{x}_j\ ;\ \mathbf{W}\right)\right).$$

- Gradient regularization model:

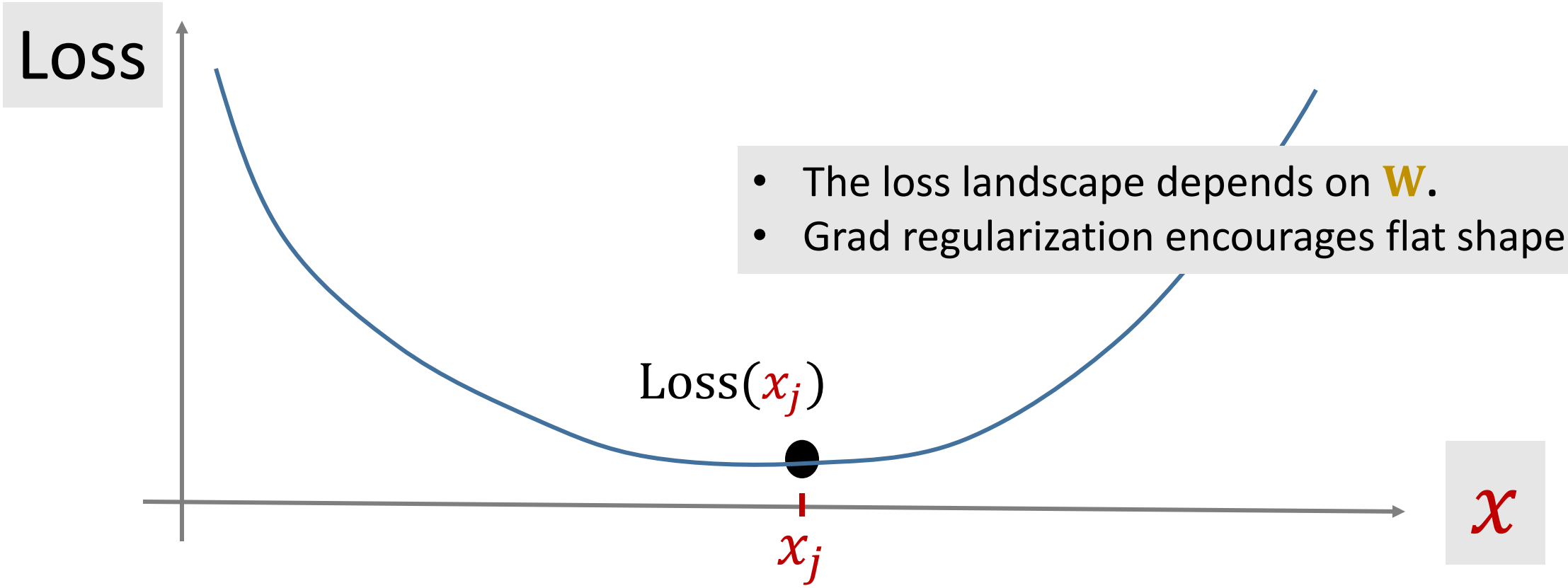
$$\min_{\mathbf{W}} \sum_{j=1}^{n} \text{Loss}\left(\mathbf{y}_j,\ \mathbf{f}\left(\mathbf{x}_j\ ;\ \mathbf{W}\right)\right) + \lambda \left|\left|g\left(\mathbf{x}_j\right)\right|\right|_2^2.$$

Here $g(\mathbf{x})$ is the derivative of Loss w.r.t. $\mathbf{x}$.

Gradient Regularization

- Gradient regularization model:

$$\min_{\mathbf{W}} \sum_{j=1}^{n} \text{Loss}\left(\mathbf{y}_j,\ \mathbf{f}\left(\mathbf{x}_j\ ;\ \mathbf{W}\right)\right)\ +\ \lambda \left|\left|g\left(\mathbf{x}_j\right)\right|\right|_2^2.$$

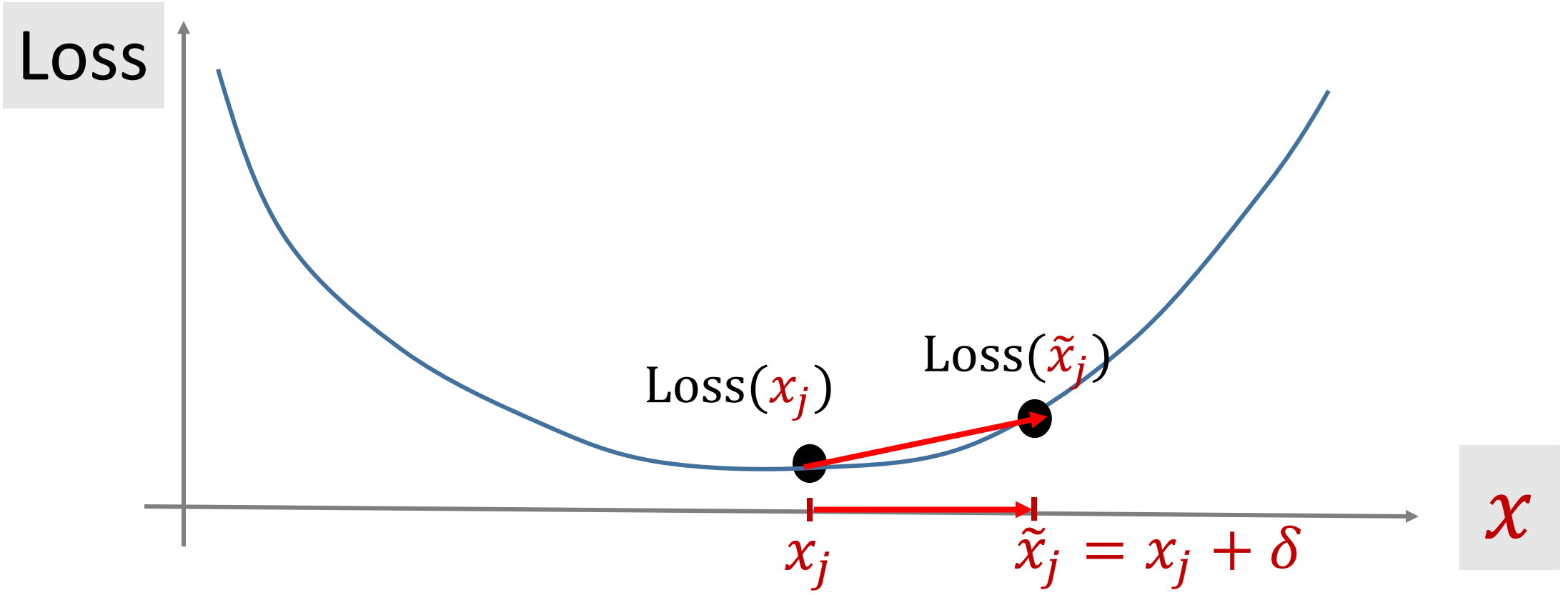


- The loss landscape depends on **W**.
- Grad regularization encourages flat shape at x_j.

Gradient Regularization

- Gradient regularization model:

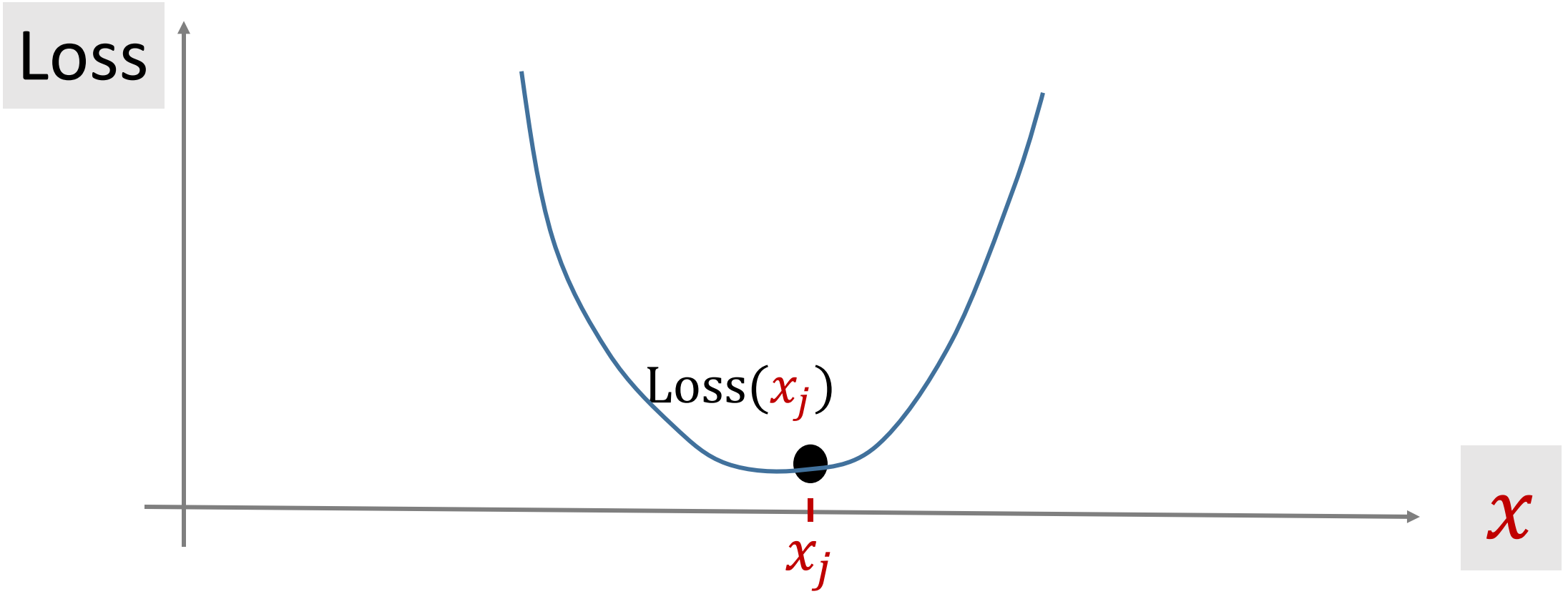
$$\min_{\mathbf{W}} \sum_{j=1}^{n} \text{Loss}\left(\mathbf{y}_j,\ \mathbf{f}\left(\mathbf{x}_j\ ;\ \mathbf{W}\right)\right)\ +\ \lambda \left|\left|g\left(\mathbf{x}_j\right)\right|\right|_2^2.$$



Gradient Regularization

- Standard model:

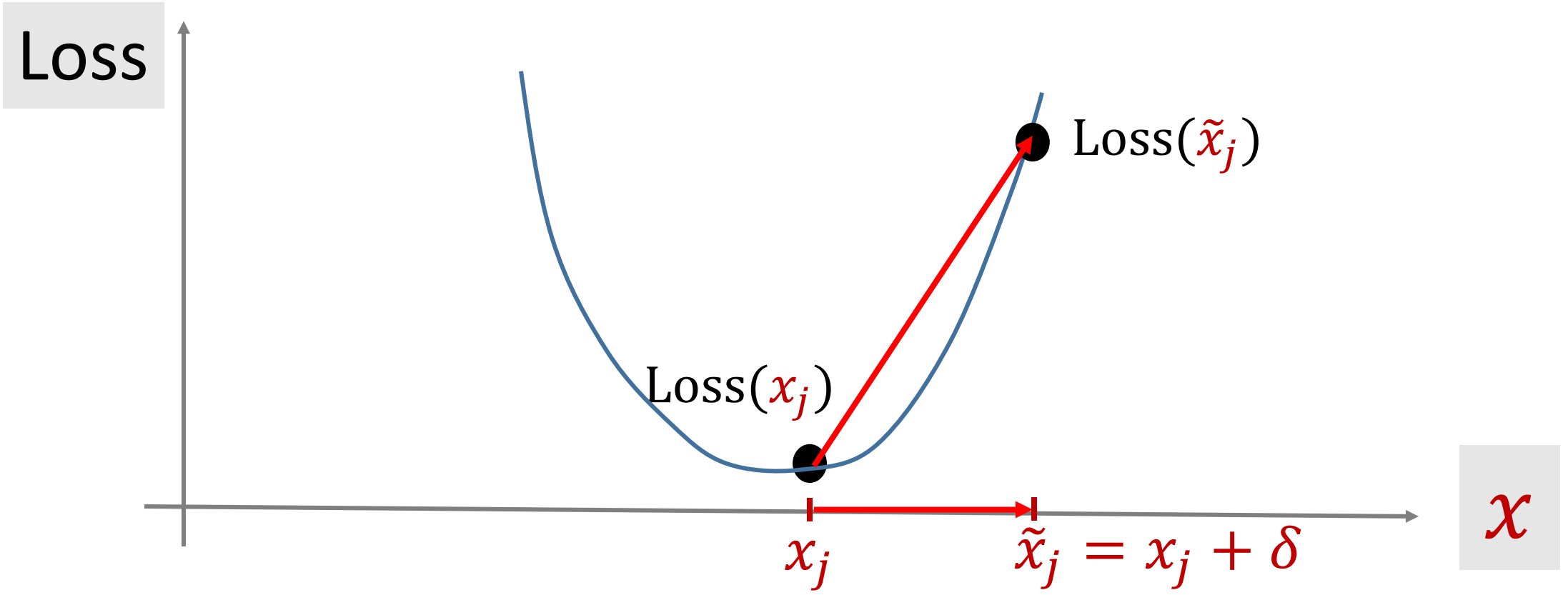
$$\min_{\mathbf{W}} \sum_{j=1}^{n} \text{Loss}\left(\mathbf{y}_j,\ \mathbf{f}\left(\mathbf{x}_j\ ;\ \mathbf{W}\right)\right).$$



Gradient Regularization

- Standard model:

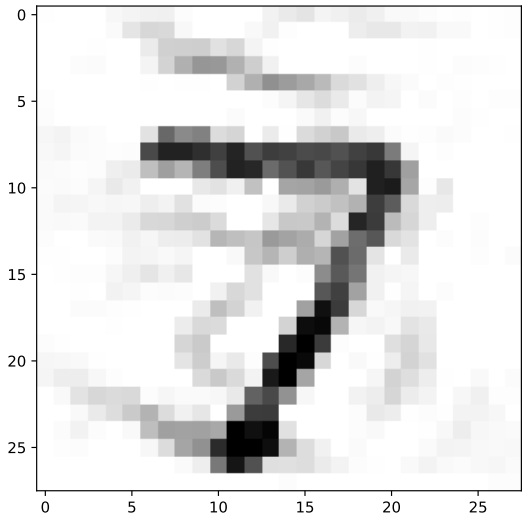
$$\min_{\mathbf{W}} \sum_{j=1}^{n} \text{Loss}\big(\mathbf{y}_j,\ \mathbf{f}\ (\mathbf{x}_j\ ;\ \mathbf{W})\big)\ .$$

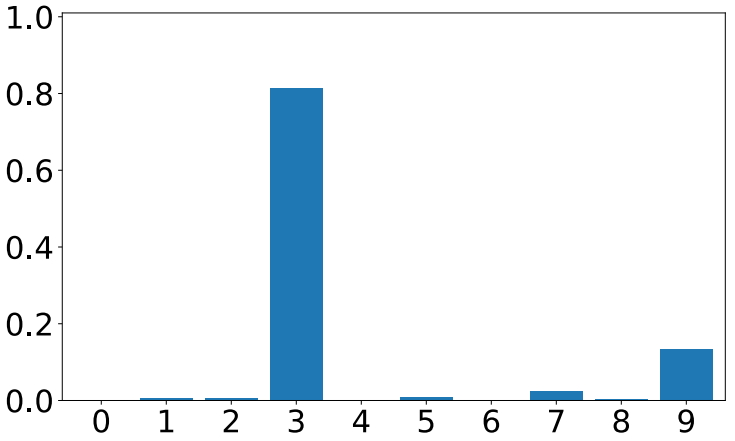


Summary

Data Evasion Attacks

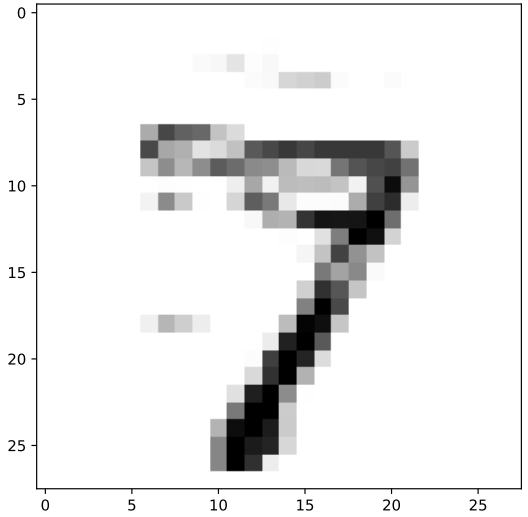
- Untargeted attack.
  - Perturb the image to increase **`Dist(true_target, prediction)`**.
  - Just let the neural network make wrong prediction.
  - Do not know what the outcome will be.

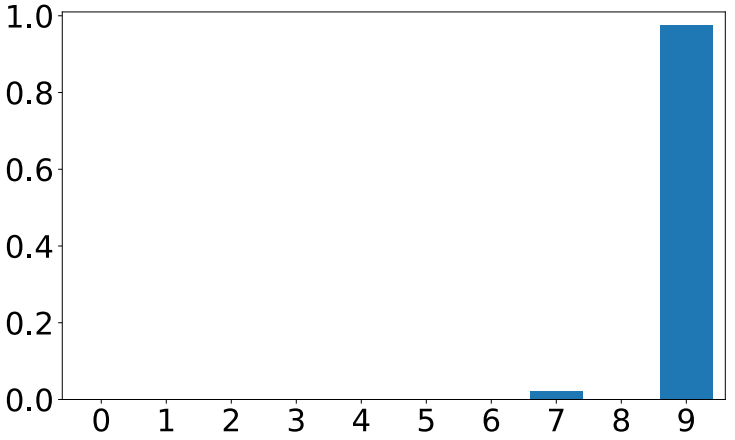




Data Evasion Attacks

- Untargeted attack.
- Targeted attack.
  - Set a **`fake_target`** and minimize **`Dist(fake_target, prediction)`**.
  - Also keep the perturbation δ small.
  - The neural network will make the wrong prediction as we set.





Adversarial Training

- Defense against attacks.
- Min-max model:

$$\min_{\mathbf{W}} \sum_{j=1}^{n} \left\{ \max_{||\boldsymbol{\delta}||<\sigma} \text{Loss}\left(\mathbf{y}_j,\ \mathbf{f}\left(\mathbf{x}_j + \boldsymbol{\delta}\ ;\ \mathbf{W}\right)\right) \right\}.$$

- Gradient regularization model:

$$\min_{\mathbf{W}} \sum_{j=1}^{n} \text{Loss}\left(\mathbf{y}_j,\ \mathbf{f}\left(\mathbf{x}_j\ ;\ \mathbf{W}\right)\right) + \lambda \left|\left| g\left(\mathbf{x}_j\right) \right|\right|_2^2,$$

where $g(\mathbf{x})$ is the derivative of Loss w.r.t. $\mathbf{x}$.

Reinforcement Learning

A little bit probability theory…

Random Variable

- Random variable: unknown; its values depends on outcomes of a random event.
- Uppercase letter X for random variable.

Random Variable	*Possible Values*	*Random Events*	*Probabilities*
$X =$	0		$\mathbb{P}(X = 0) = 0.5$
	1		$\mathbb{P}(X = 1) = 0.5$

Random Variable

- Random variable: unknown; its values depends on outcomes of a random event.
- Uppercase letter X for random variable.
- Lowercase letter x for an observed value.
- For example, I flipped a coin 4 times and observed:
  - $x_1 = 1,$
  - $x_2 = 1,$
  - $x_3 = 0,$
  - $x_4 = 1.$

Probability Density Function (PDF)

- PDF provides a relative likelihood that the value of the random variable would equal that sample.

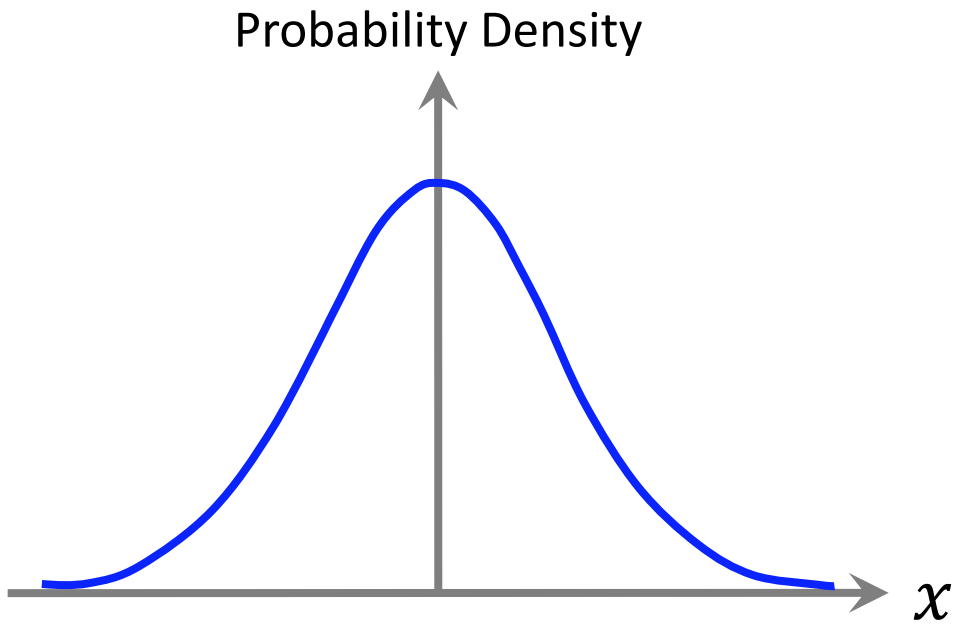
Probability Density Function (PDF)

- PDF provides a relative likelihood that the value of the random variable would equal that sample.

Example: Gaussian distribution

- It is a continuous distribution.
- PDF:

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\,\sigma^2}\right).$$



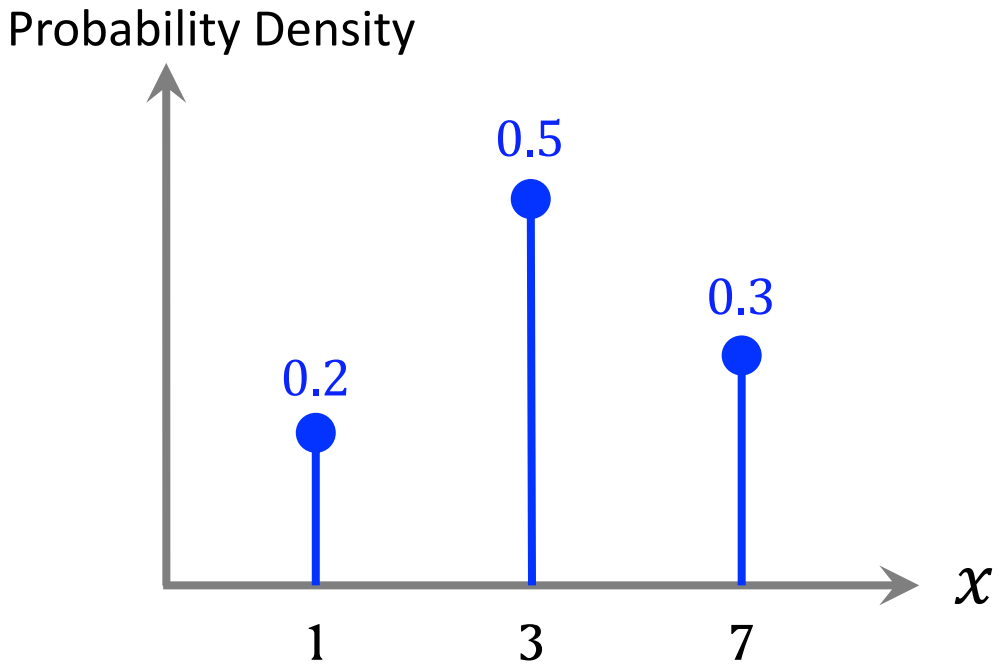
Probability Mass Function (PMF)

- PMF is a function that gives the probability that a discrete random variable is exactly equal to some value

Example

- Discrete random variable: $X \in \{1, 3, 7\}$.
- PMF:

$$p(1) = 0.2,$$
$$p(3) = 0.5,$$
$$p(7) = 0.3.$$



Properties of PDF/PMF

- Random variable X is in the domain $\mathcal{X}$.
- For continuous distribution,

$$\int_{\mathcal{X}} p(x)\, dx = 1.$$

- For discrete distribution,

$$\sum_{x \in \mathcal{X}} p(x) = 1.$$

Expectation

- Random variable X is in the domain $\mathcal{X}$.
- For continuous distribution, the expectation of $f(X)$ is:

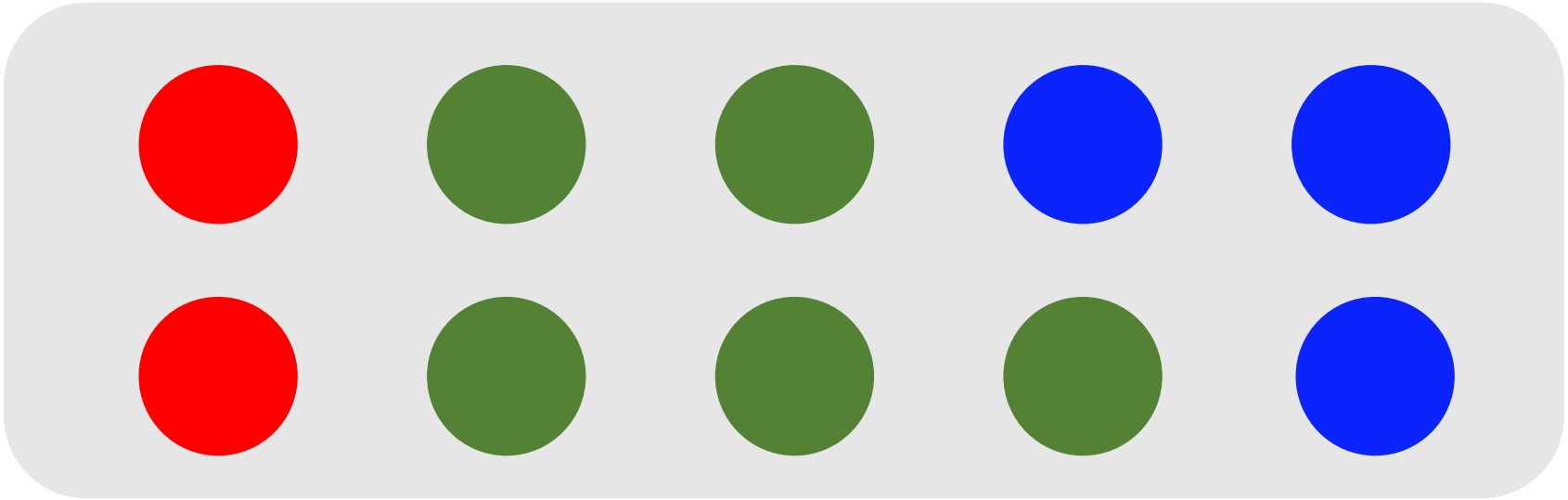
$$\mathbb{E}[f(X)] = \int_{\mathcal{X}} p(x) \cdot f(x)\, dx.$$

- For discrete distribution, the expectation of $f(X)$ is:

$$\mathbb{E}[f(X)] = \sum_{x \in \mathcal{X}} p(x) \cdot f(x)\,.$$

Random Sampling

- There are 10 balls in the bin: 2 are red, 5 are green, and 3 are blue.
- Randomly sample a ball.
- What will be the color?



Random Sampling

- Sample red ball w.p. 0.2, green ball w.p. 0.5, and blue ball w.p. 0.3.
- Randomly sample a ball.
- What will be the color?

Random Sampling

- Sample red ball w.p. 0.2, green ball w.p. 0.5, and blue ball w.p. 0.3.
- Randomly sample a ball.
- What will be the color?

```
from numpy.random import choice

samples = choice(['R', 'G', 'B'], size=100, p=[0.2, 0.5, 0.3])
print(samples)
```

```
['R' 'G' 'R' 'R' 'R' 'R' 'B' 'B' 'B' 'G' 'G' 'B' 'G' 'B' 'B' 'G' 'B' 'G'
 'B' 'B' 'G' 'B' 'G' 'B' 'B' 'G' 'B' 'B' 'G' 'B' 'G' 'G' 'G' 'G' 'G' 'B'
 'B' 'B' 'B' 'B' 'B' 'G' 'G' 'B' 'R' 'R' 'B' 'R' 'B' 'G' 'R' 'G' 'R' 'G'
 'R' 'R' 'B' 'G' 'G' 'G' 'B' 'R' 'G' 'B' 'G' 'R' 'G' 'G' 'G' 'B' 'B' 'R'
 'G' 'G' 'B' 'B' 'R' 'B' 'B' 'B' 'R' 'B' 'G' 'B' 'R' 'B' 'R' 'G' 'B' 'R'
 'B' 'B' 'G' 'G' 'G' 'R' 'R' 'B' 'R' 'G']
```

Terminologies

Terminology: state and action

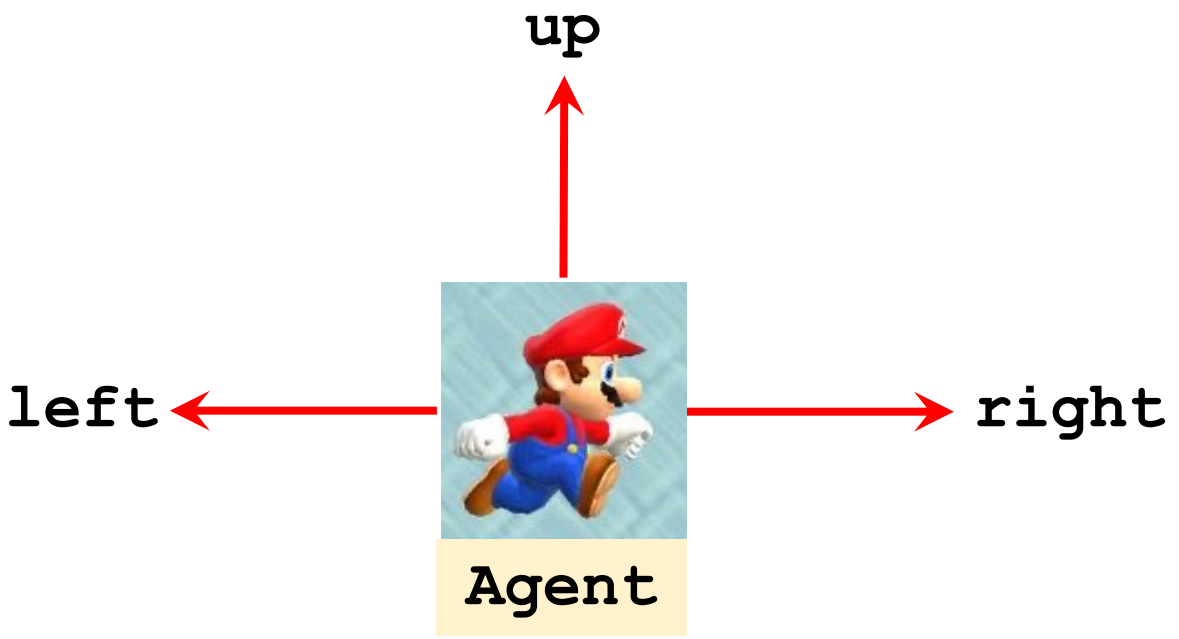
`state` s `(this frame)`

Action $a \in \{\text{left}, \text{right}, \text{up}\}$

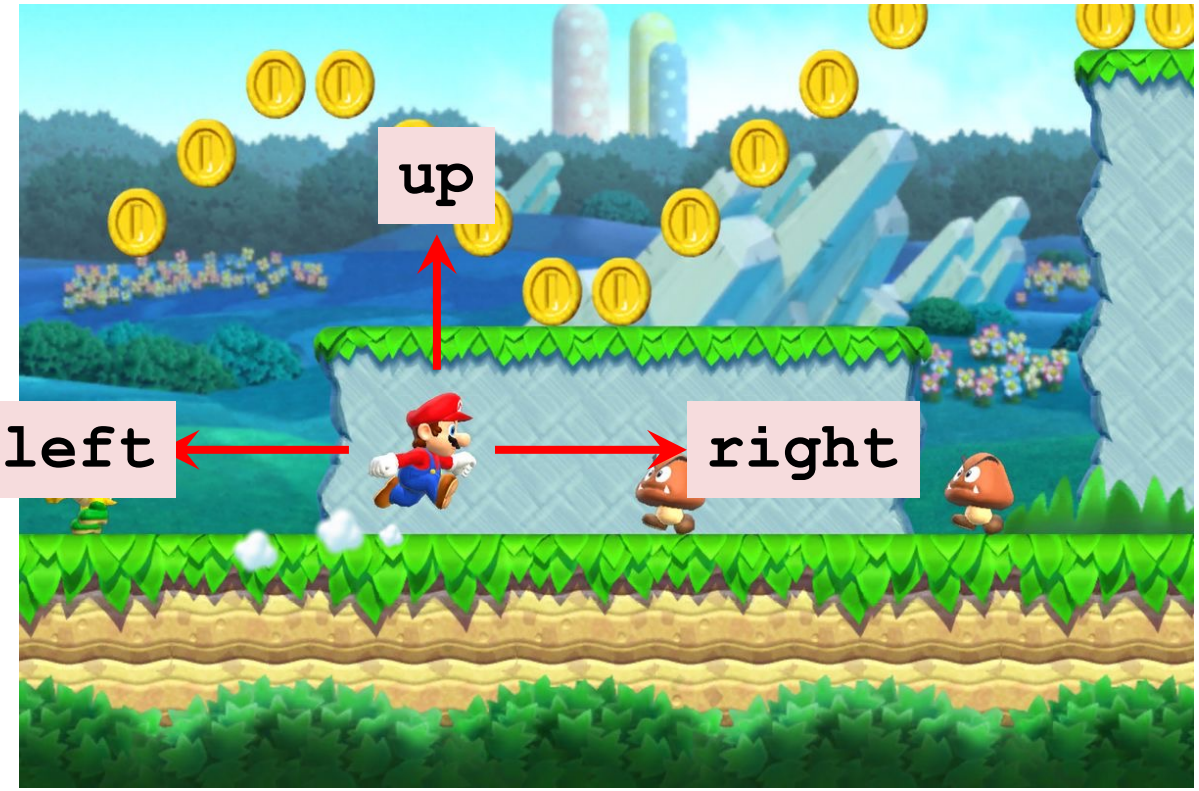
Terminology: state and action

state s **(this frame)**

Action $a \in \{\text{left}, \text{right}, \text{up}\}$



Terminology: policy



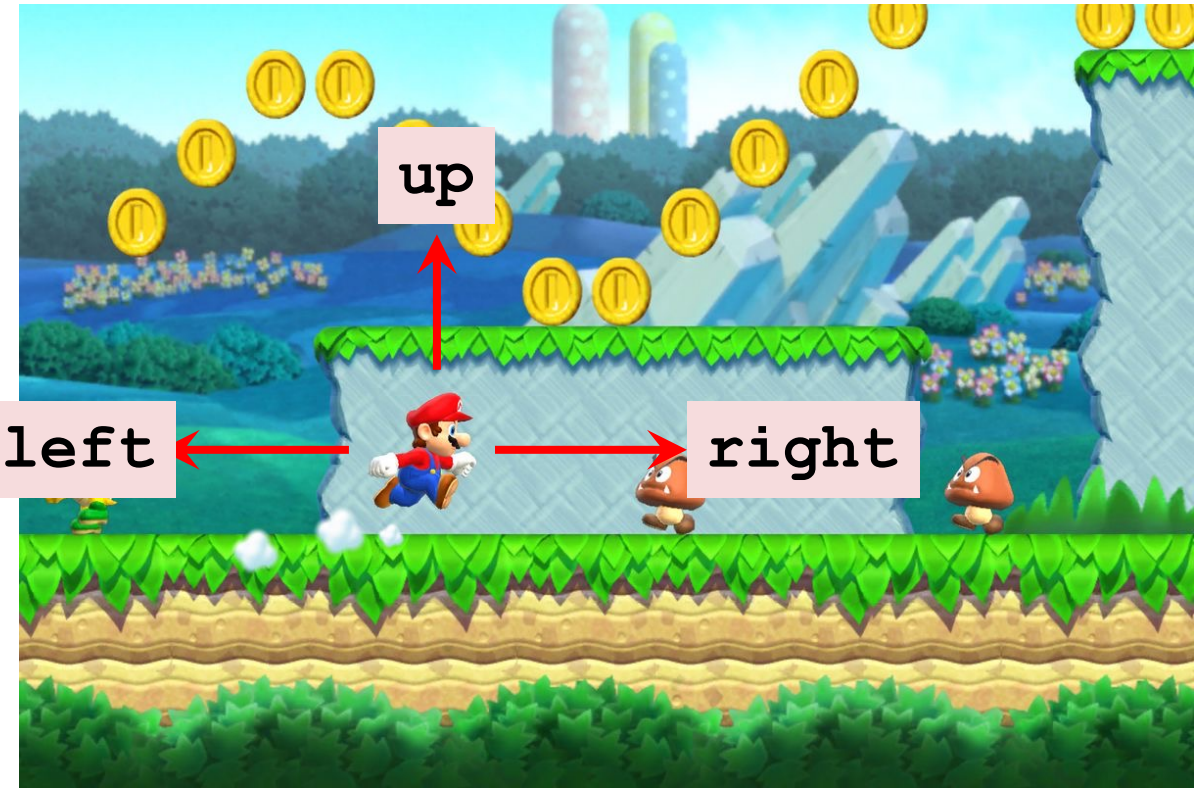
policy π

- Policy function $\pi: (s, a) \mapsto [0,1]$:

$$\pi(a \mid s) = \mathbb{P}(A = a \mid S = s).$$

- It is the probability of taking action $A = a$ given state s, e.g.,
  - $\pi(\text{left} \mid s) = 0.2$,
  - $\pi(\text{right} \mid s) = 0.1$,
  - $\pi(\text{up} \mid s) = 0.7$.
- Upon observing state $S = s$, the agent's action A can be random.

Terminology: policy



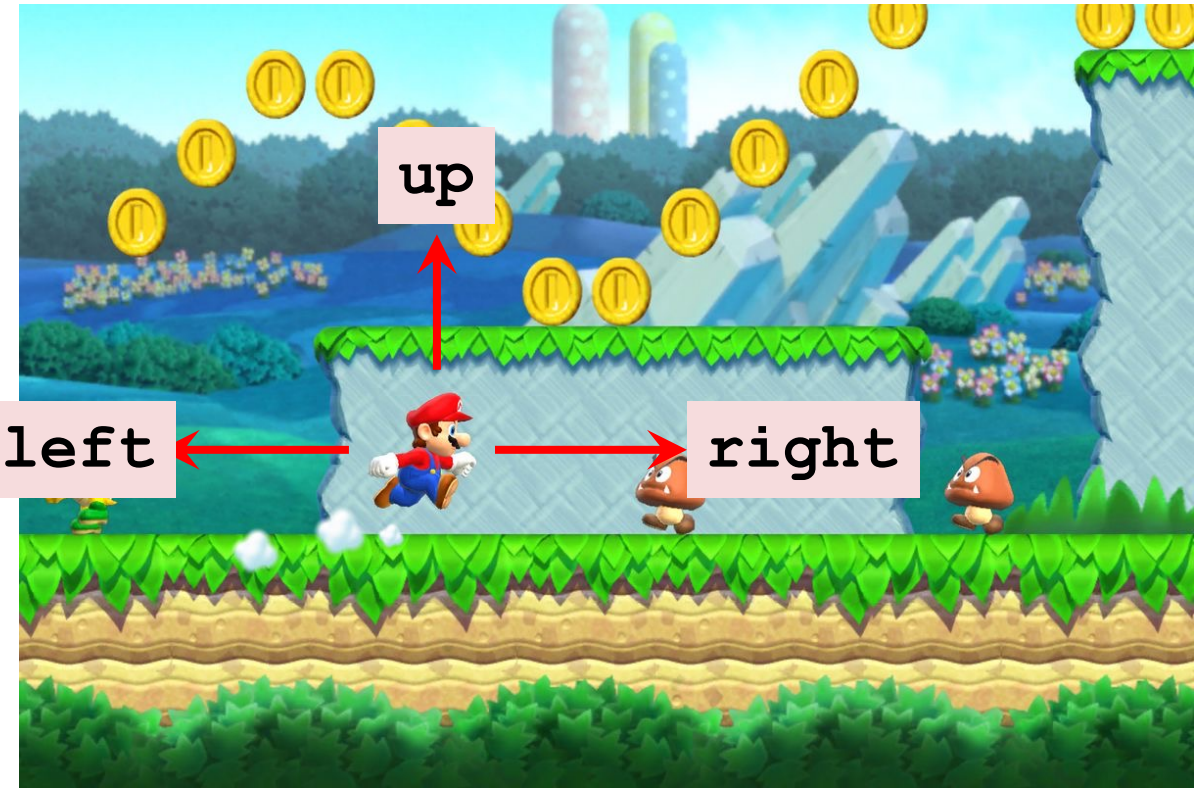
policy π

- Policy function $\pi: (s, a) \mapsto [0,1]$:

$$\pi(a \mid s) = \mathbb{P}(A = a \mid S = s).$$

- It is the probability of taking action $A = a$ given state s, e.g.,
  - $\pi(\text{left} \mid s) = 0.2$,
  - $\pi(\text{right} \mid s) = 0.1$,
  - $\pi(\text{up} \mid s) = 0.7$.
- Upon observing state $S = s$, the agent's action A can be random.

Terminology: policy



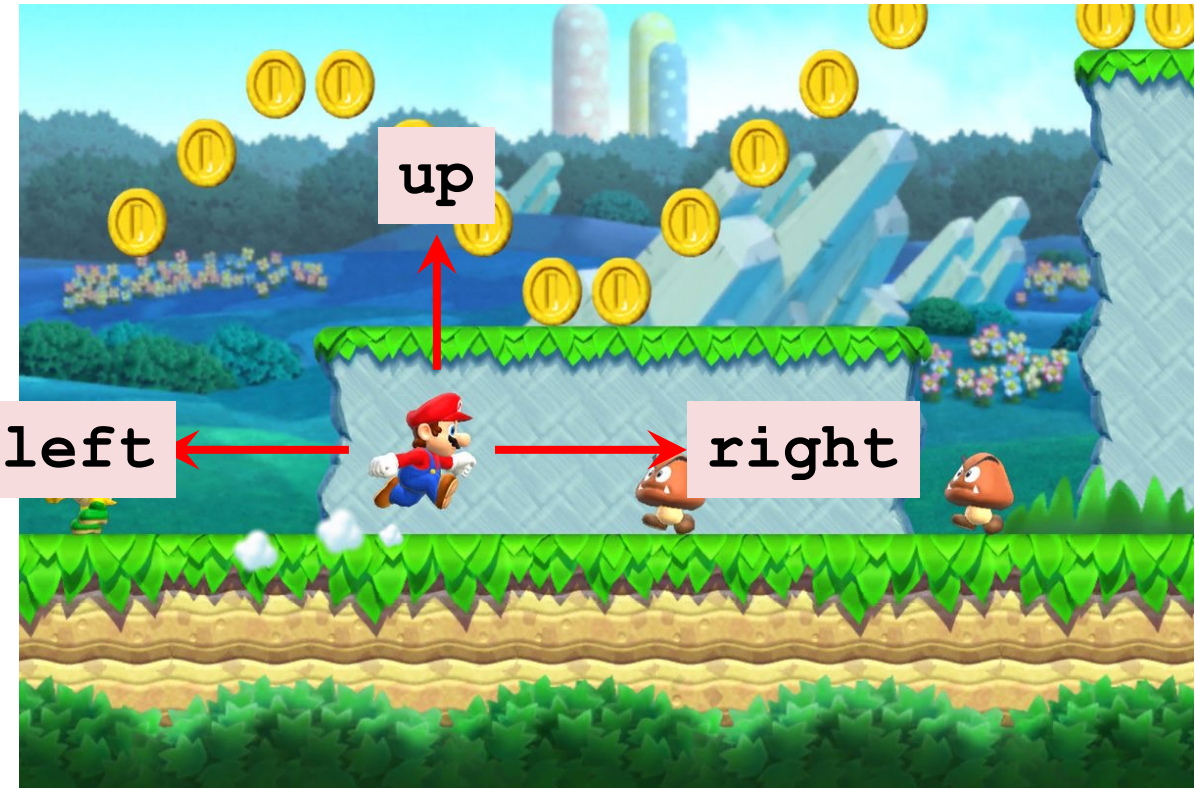
policy π

- Policy function π: $(s, a) \mapsto [0,1]$:

$$\pi(a \mid s) = \mathbb{P}(A = a \mid S = s).$$

- It is the probability of taking action $A = a$ given state s , e.g.,
  - $\pi(\text{left} \mid s) = 0.2$,
  - $\pi(\text{right} \mid s) = 0.1$,
  - $\pi(\text{up} \mid s) = 0.7$.
- Upon observing state $S = s$, the agent's action A can be random.

Terminology: policy



policy π

- Policy function π: $(s, a) \mapsto [0,1]$:

$$\pi(a \mid s) = \mathbb{P}(A = a \mid S = s).$$

- It is the probability of taking action $A = a$ given state s , e.g.,
  - $\pi(\text{left} \mid s) = 0.2$,
  - $\pi(\text{right} \mid s) = 0.1$,
  - $\pi(\text{up} \mid s) = 0.7$.
- Upon observing state $S = s$, the agent's action A can be random.

Terminology: reward

reward R

- Collect a coin: $R = +1$

Terminology: reward

reward R

- Collect a coin: $R = +1$
- Win the game: $R = +10000$

Terminology: reward

reward R

- Collect a coin: $R = +1$
- Win the game: $R = +10000$
- Touch a Goomba: $R = -10000$ (game over).

Terminology: reward

reward R

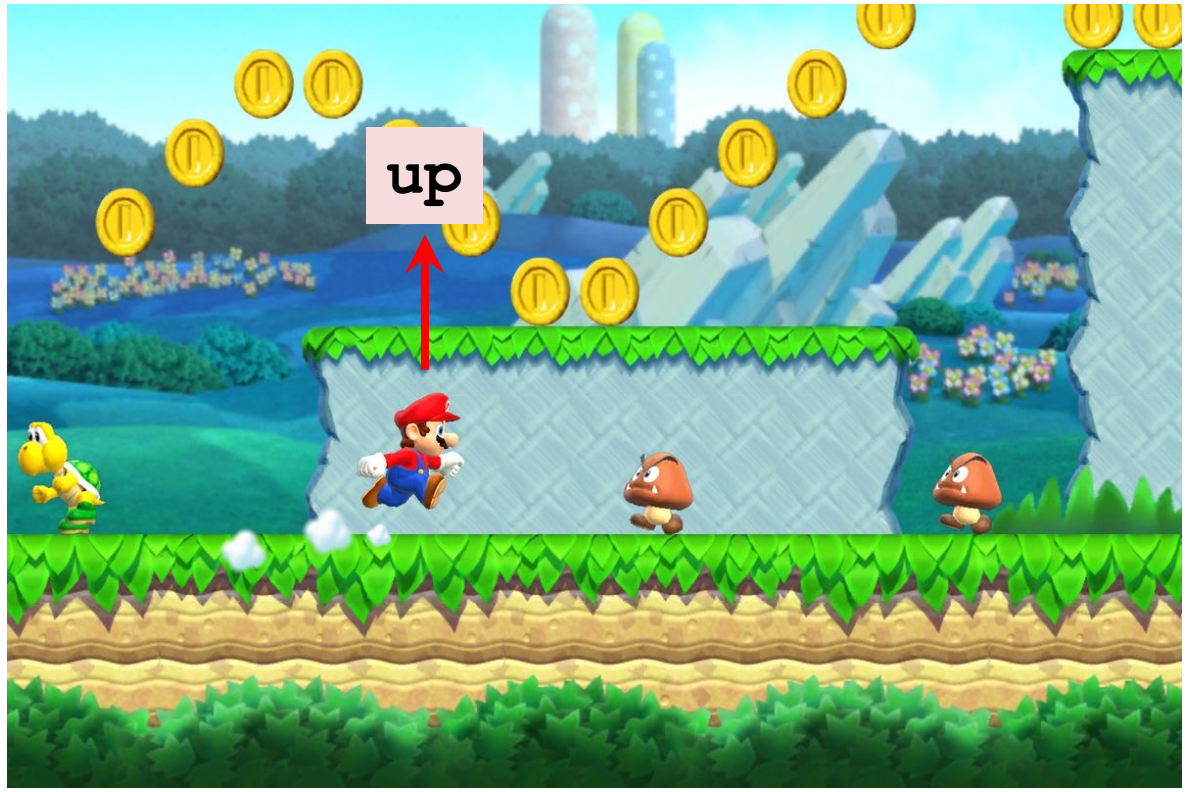
- Collect a coin: $R = +1$
- Win the game: $R = +10000$
- Touch a Goomba: $R = -10000$ (game over).
- Nothing happens: $R = 0$

Terminology: state transition

state transition

old state —action→ new state

Terminology: state transition

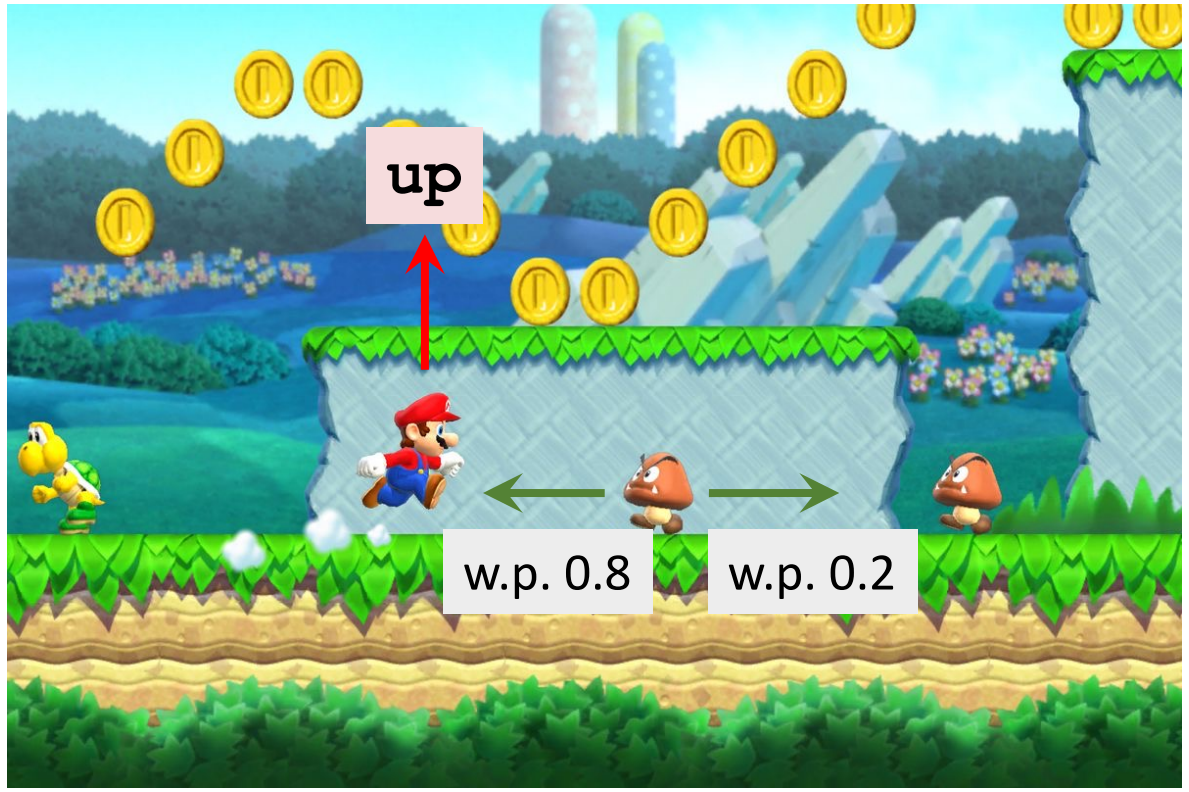


state transition

old state —action→ new state

- E.g., “up” action leads to a new state.

Terminology: state transition



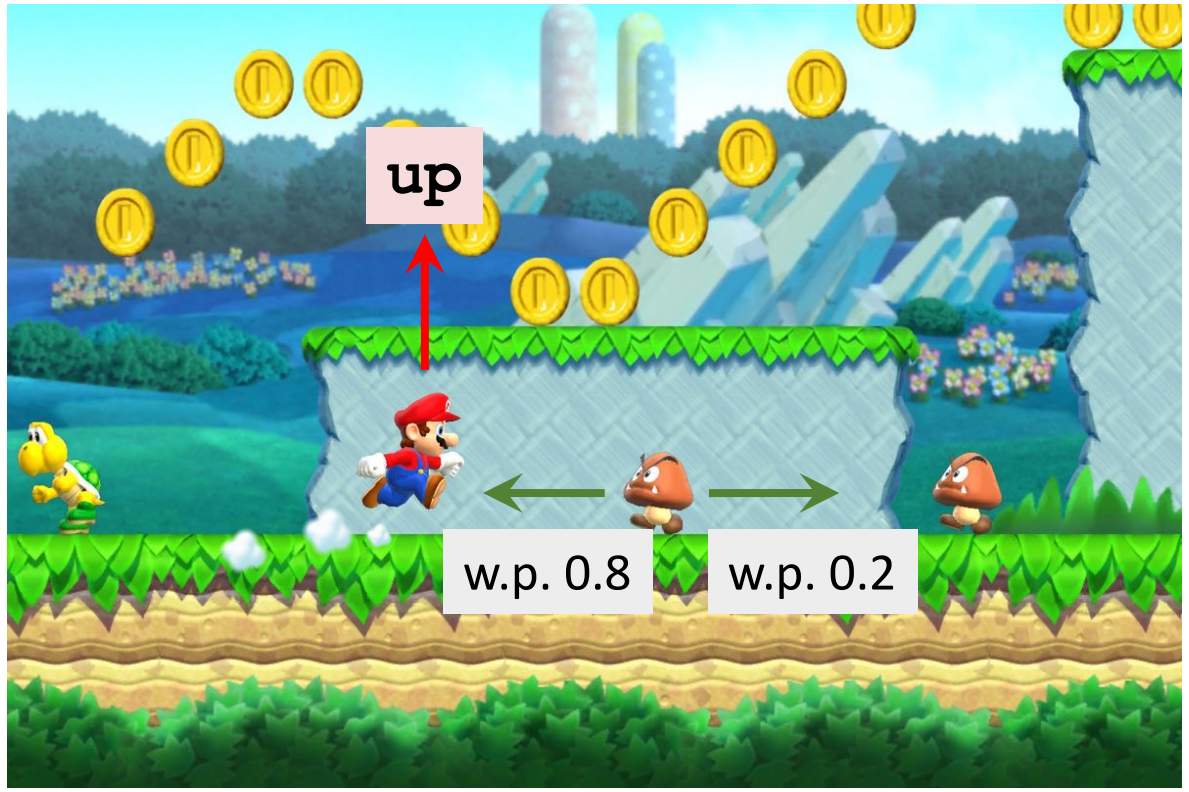
state transition

old state —action→ new state

- E.g., "up" action leads to a new state.

- State transition can be random.
- Randomness is from the environment.

Terminology: state transition



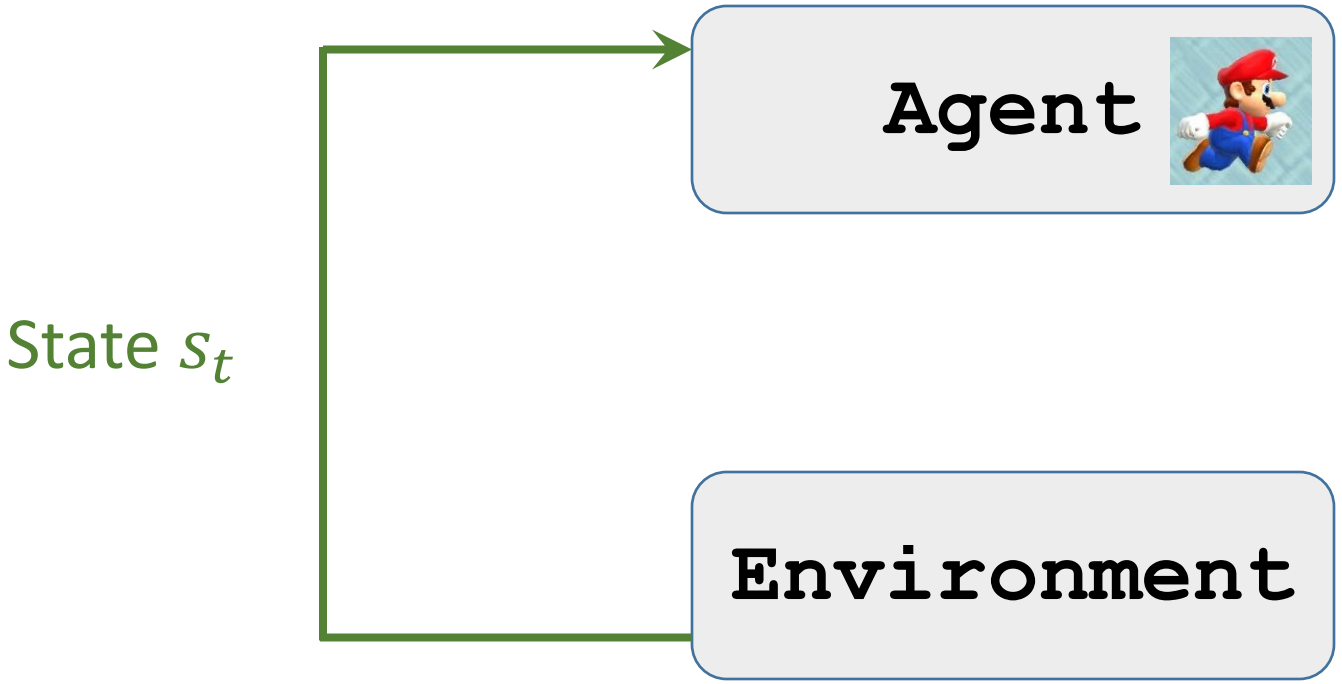
state transition

old state —action→ new state

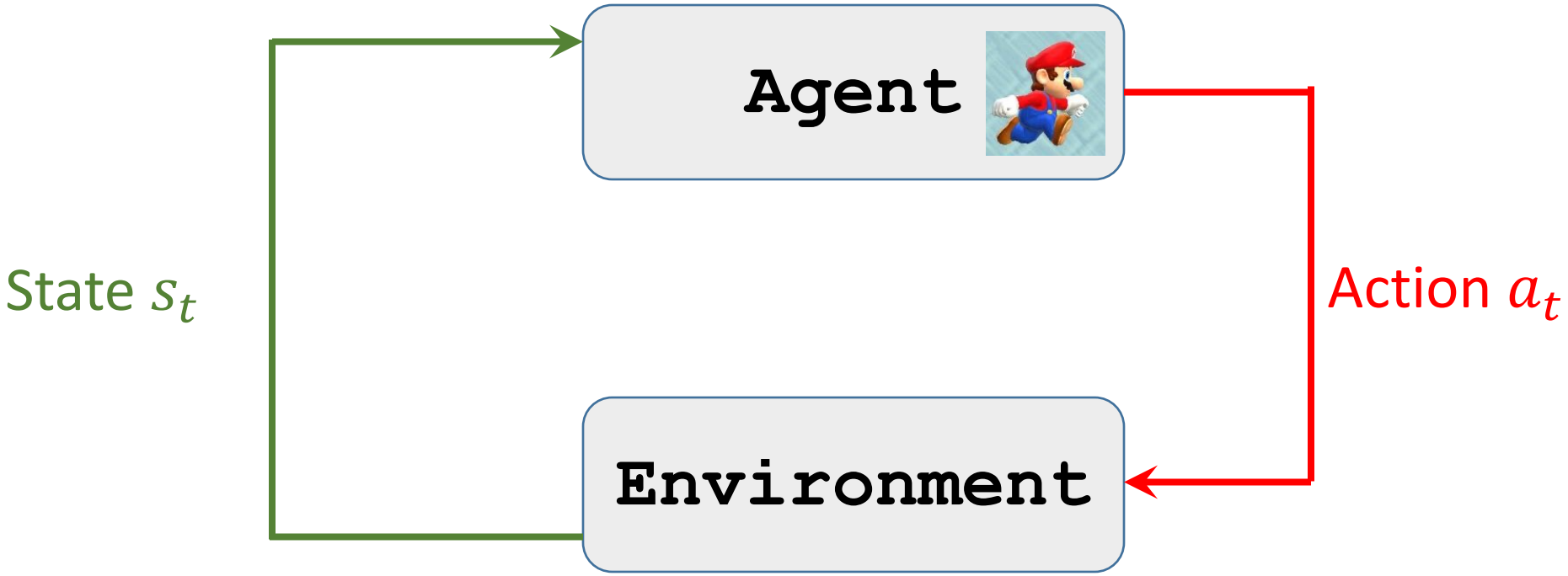
- E.g., "up" action leads to a new state.

- State transition can be random.
- Randomness is from the environment.
- $p(s'|s,a) = \mathbb{P}(S' = s'|S = s, A = a)$.

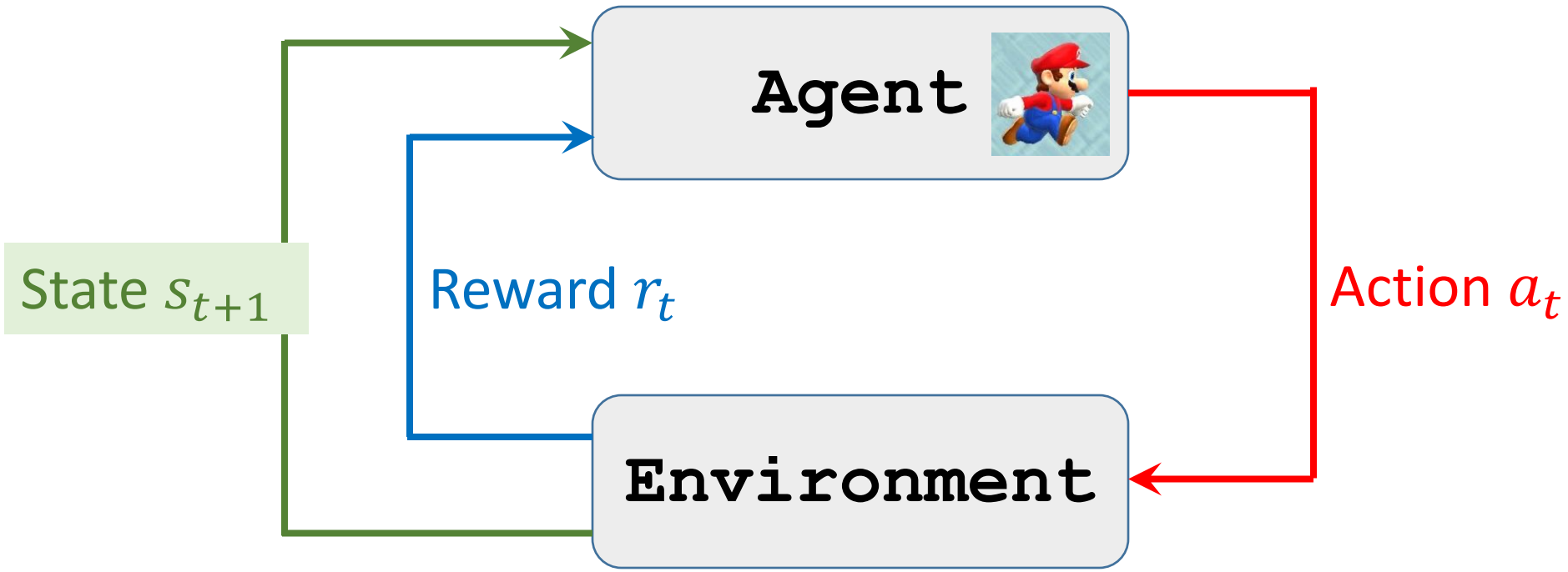
Terminology: agent environment interaction



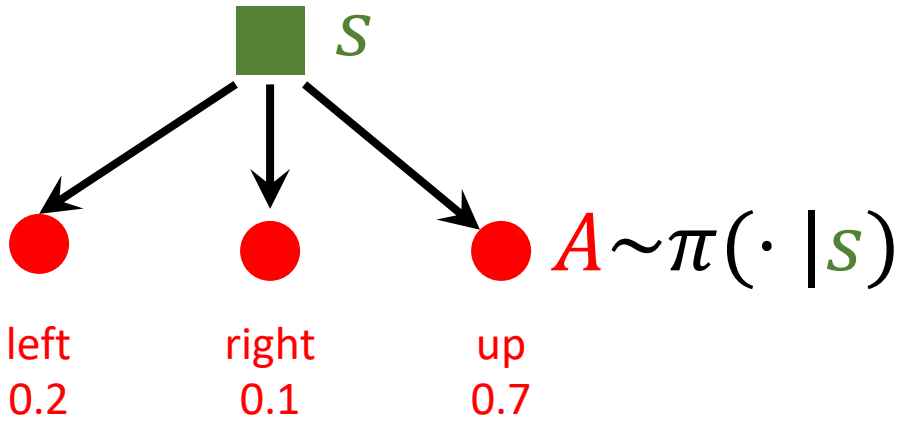
Terminology: agent environment interaction



Terminology: agent environment interaction



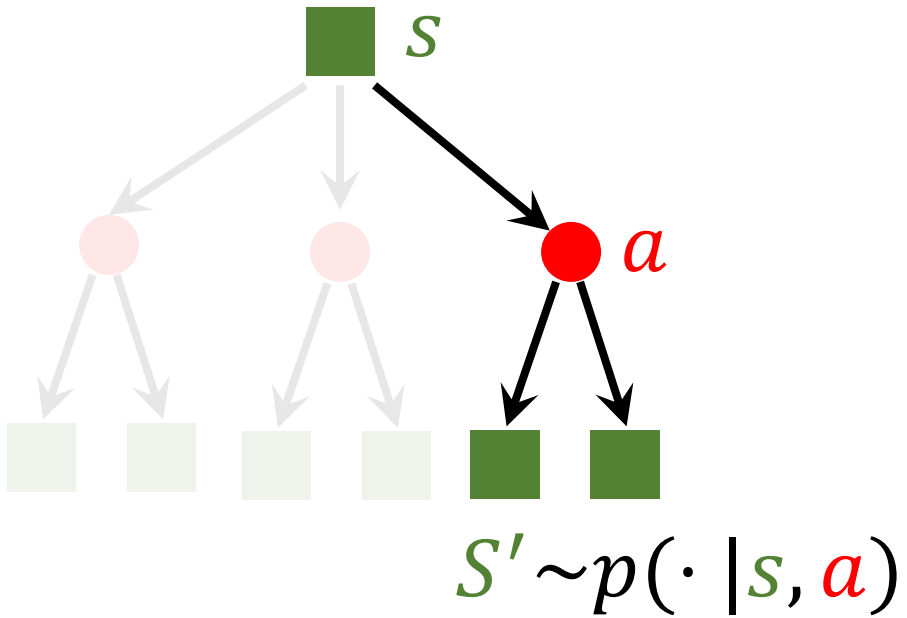
Randomness in Reinforcement Learning



Actions have randomness.

- Given state s, the action can be random, e.g., .
  - $\pi(\text{“left”}|s) = 0.2,$
  - $\pi(\text{“right”}|s) = 0.1,$
  - $\pi(\text{“up”}|s) = 0.7.$

Randomness in Reinforcement Learning



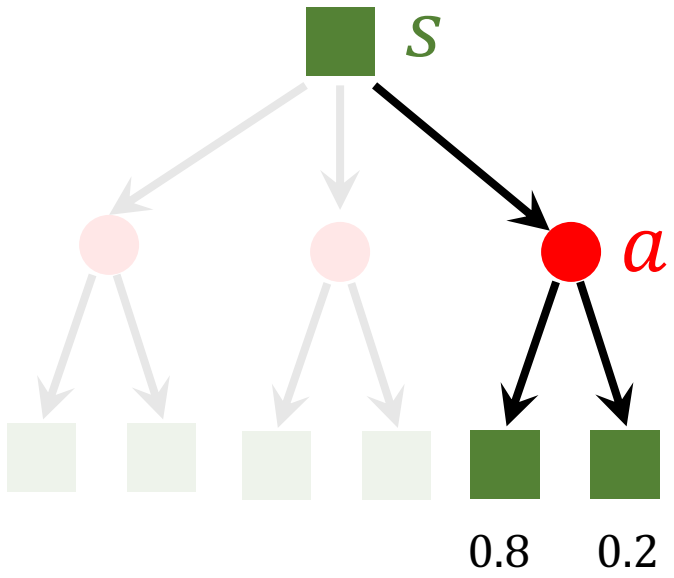
Actions have randomness.

- Given state s, the action can be random, e.g., .
  - $\pi(\text{“left”}|s) = 0.2,$
  - $\pi(\text{“right”}|s) = 0.1,$
  - $\pi(\text{“up”}|s) = 0.7.$

State transitions have randomness.

- Given state $S = s$ and action $A = a$, the environment randomly generates a new state S'.

Randomness in Reinforcement Learning



Actions have randomness.

- Given state s, the action can be random, e.g., .
  - $\pi(\text{“left”}|s) = 0.2,$
  - $\pi(\text{“right”}|s) = 0.1,$
  - $\pi(\text{“up”}|s) = 0.7.$

State transitions have randomness.

- Given state $S = s$ and action $A = a$, the environment randomly generates a new state S'.

Play the game using AI

- Observe a frame (state s_1)
- ➔ Make action a_1 (left, right, or up)
- ➔ Observe a new frame (state s_2) and reward r_1
- ➔ Make action a_2
- ➔ …

Play the game using AI

- Observe a frame (state s_1)
- ➔ Make action a_1 (left, right, or up)
- ➔ Observe a new frame (state s_2) and reward r_1
- ➔ Make action a_2
- ➔ …

- (state, action, reward) trajectory:

$$s_1, a_1, r_1, s_2, a_2, r_2, \quad \cdots \quad , s_T, a_T, r_T.$$

Rewards and Returns

Return

Definition: Return (aka cumulative future reward).

- $U_t = R_t + R_{t+1} + R_{t+2} + R_{t+3} + \cdots$

Return

Definition: Return (aka cumulative future reward).

- $U_t = R_t + R_{t+1} + R_{t+2} + R_{t+3} + \cdots$

Question: Are R_t and R_{t+1} equally important?

- Which of the followings do you prefer?
  - I give you $100 right now.
  - I will give you $100 one year later.

Return

Definition: Return (aka cumulative future reward).

- $U_t = R_t + R_{t+1} + R_{t+2} + R_{t+3} + \cdots$

Question: Are R_t and R_{t+1} equally important?

- Which of the followings do you prefer?
  - I give you $100 right now.
  - I will give you $100 one year later.

- Future reward is less valuable than present reward.
- R_{t+1} should be given less weight than R_t.

Return

Definition: Return (aka cumulative future reward).

- $U_t = R_t + R_{t+1} + R_{t+2} + R_{t+3} + \cdots$

Definition: Discounted return (aka cumulative discounted future reward).

- γ: discount rate (tuning hyper-parameter).
- $U_t = R_t + \gamma\, R_{t+1} + \gamma^2\, R_{t+2} + \gamma^3\, R_{t+3} + \cdots$

Randomness in Returns

Definition: Discounted return (at time step t).

- $U_t = R_t + \gamma\, R_{t+1} + \gamma^2\, R_{t+2} + \gamma^3\, R_{t+3} + \cdots$

At time step t, the return U_t is random.

- Two sources of randomness:
  1. Action can be random: $\mathbb{P}[A = a \mid S = s] = \pi(a|s)$.
  2. New state can be random: $\mathbb{P}[S' = s'|S = s, A = a] = p(s'|s, a)$.

Randomness in Returns

Definition: Discounted return (at time step t).

- $U_t = R_t + \gamma\, R_{t+1} + \gamma^2\, R_{t+2} + \gamma^3\, R_{t+3} + \cdots$

At time step t, the return U_t is random.

- Two sources of randomness:
  1. Action can be random: $\mathbb{P}[A = a \mid S = s] = \pi(a|s)$.
  2. New state can be random: $\mathbb{P}[S' = s'|S = s, A = a] = p(s'|s, a)$.
- For any $i \geq t$, the reward R_i depends on S_i and A_i.
- Thus, given s_t, the return U_t depends on the random variables:
  - $A_t, A_{t+1}, A_{t+2}, \cdots$ and $S_{t+1}, S_{t+2}, \cdots$.

Value Functions

Action-Value Function $Q(s, a)$

Definition: Discounted return (aka cumulative discounted future reward).

- $U_t = R_t + \gamma\, R_{t+1} + \gamma^2\, R_{t+2} + \gamma^3\, R_{t+3} + \cdots$

Action-Value Function $Q(s, a)$

Definition: Discounted return (aka cumulative discounted future reward).

- $U_t = R_t + \gamma\, R_{t+1} + \gamma^2\, R_{t+2} + \gamma^3\, R_{t+3} + \cdots$

Definition: Action-value function for policy π.

- $Q_\pi(s_t, a_t) = \mathbb{E}\,[U_t | S_t = s_t, A_t = a_t]$.

- Return U_t depends on states $S_t, S_{t+1}, S_{t+2}, \cdots$ and actions $A_t, A_{t+1}, A_{t+2}, \cdots$.

Action-Value Function $Q(s, a)$

Definition: Discounted return (aka cumulative discounted future reward).

- $U_t = R_t + \gamma\, R_{t+1} + \gamma^2\, R_{t+2} + \gamma^3\, R_{t+3} + \cdots$

Definition: Action-value function for policy π.

- $Q_\pi(s_t, a_t) = \mathbb{E}\,[U_t | S_t = s_t, A_t = a_t]$.

- Return U_t depends on states $S_t, S_{t+1}, S_{t+2}, \cdots$ and actions $A_t, A_{t+1}, A_{t+2}, \cdots$.
- Actions are random: $\mathbb{P}[A = a \mid S = s] = \pi(a|s)$. (Policy function.)
- States are random: $\mathbb{P}[S' = s' | S = s, A = a] = p(s'|s, a)$. (State transition.)

Action-Value Function $Q(s, a)$

Definition: Discounted return (aka cumulative discounted future reward).

- $U_t = R_t + \gamma\, R_{t+1} + \gamma^2\, R_{t+2} + \gamma^3\, R_{t+3} + \cdots$

Definition: Action-value function for policy π.

- $Q_\pi(s_t, a_t) = \mathbb{E}\left[U_t | S_t = s_t, A_t = a_t\right]$.

Definition: Optimal action-value function.

- $Q^\star(s_t, a_t) = \max_\pi Q_\pi(s_t, a_t)$.

State-Value Function $V(s)$

Definition: Discounted return (aka cumulative discounted future reward).

- $U_t = R_t + \gamma\, R_{t+1} + \gamma^2\, R_{t+2} + \gamma^3\, R_{t+3} + \cdots$

Definition: Action-value function for policy π.

- $Q_\pi(s_t, a_t) = \mathbb{E}\,[U_t | S_t = s_t, A_t = a_t]$.

Definition: State-value function.

- $V_\pi(s_t) = \mathbb{E}_A\,[Q_\pi(s_t, A)]$

State-Value Function $V(s)$

Definition: Discounted return (aka cumulative discounted future reward).

- $U_t = R_t + \gamma\, R_{t+1} + \gamma^2\, R_{t+2} + \gamma^3\, R_{t+3} + \cdots$

Definition: Action-value function for policy π.

- $Q_\pi(s_t, a_t) = \mathbb{E}\,[U_t | S_t = s_t, A_t = a_t]$.

Definition: State-value function.

- $V_\pi(s_t) = \mathbb{E}_A\,[Q_\pi(s_t, A)] = \sum_a \pi(a|s_t) \cdot Q_\pi(s_t, a)$. (Actions are discrete.)

Taken w.r.t. the action $A \sim \pi(\cdot\,|s_t)$.

State-Value Function $V(s)$

Definition: Discounted return (aka cumulative discounted future reward).

- $U_t = R_t + \gamma\, R_{t+1} + \gamma^2\, R_{t+2} + \gamma^3\, R_{t+3} + \cdots$

Definition: Action-value function for policy π.

- $Q_\pi(s_t, a_t) = \mathbb{E}\,[U_t | S_t = s_t, A_t = a_t]$.

Definition: State-value function.

- $V_\pi(s_t) = \mathbb{E}_A\,[Q_\pi(s_t, A)] = \sum_a \pi(a|s_t) \cdot Q_\pi(s_t, a)$. (Actions are discrete.)
- $V_\pi(s_t) = \mathbb{E}_A\,[Q_\pi(s_t, A)] = \int \pi(a|s_t) \cdot Q_\pi(s_t, a)\, da$. (Actions are continuous.)

Understanding the Value Functions

- Action-value function: $Q_\pi(s_t, a_t) = \mathbb{E}\left[U_t | S_t = s_t, A_t = a_t\right]$.
- Given policy π, $Q_\pi(s, a)$ evaluates how good it is for an agent to pick action a while being in state s.

Understanding the Value Functions

- Action-value function: $Q_\pi(s_t, a_t) = \mathbb{E}\left[U_t | S_t = s_t, A_t = a_t\right]$.
- Given policy π, $Q_\pi(s, a)$ evaluates how good it is for an agent to pick action a while being in state s.

- State-value function: $V_\pi(s) = \mathbb{E}_A\left[Q_\pi(s, A)\right]$
- For fixed policy π, $V_\pi(s)$ evaluates how good the situation is in state s.
- $\mathbb{E}_S[V_\pi(S)]$ evaluates how good the policy π is.

Play games using reinforcement learning

How does AI control the agent?

Suppose we have a good policy $\pi(a|s)$.

- Upon observing the state s_t,
- random sampling: $a_t \sim \pi(\cdot\,|s_t)$.

How does AI control the agent?

Suppose we have a good policy $\pi(a|s)$.

- Upon observing the state s_t,
- random sampling: $a_t \sim \pi(\cdot\,|s_t)$.

Suppose we know the optimal action-value function $Q^\star(s, a)$.

- Upon observe the state s_t,
- choose the action that maximizes the value: $a_t = \text{argmax}_a\, Q^\star(s_t, a)$.

Summary

Summary

Terminologies

- Agent 
- Environment
- State s.
- Action a.
- Reward r.
- Policy $\pi(a|s)$
- State transition $p(s'|s,a)$.

Summary

Terminologies

- Agent 
- Environment
- State s.
- Action a.
- Reward r.
- Policy $\pi(a|s)$
- State transition $p(s'|s,a)$.

Return and Value

- Return:

$$U_t = R_t + \gamma\, R_{t+1} + \gamma^2\, R_{t+2} + \cdots$$

- Action-value function:

$$Q_\pi(s_t, a_t) = \mathbb{E}\,[U_t | s_t, a_t].$$

- Optimal action-value function:

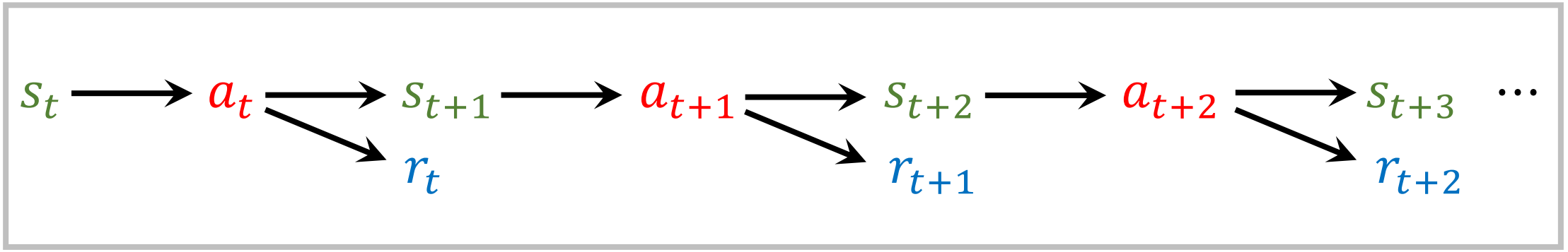
$$Q^\star(s_t, a_t) = \max_\pi Q_\pi(s_t, a_t).$$

- State-value function:

$$V_\pi(s_t) = \mathbb{E}_A[Q_\pi(s_t, A)].$$

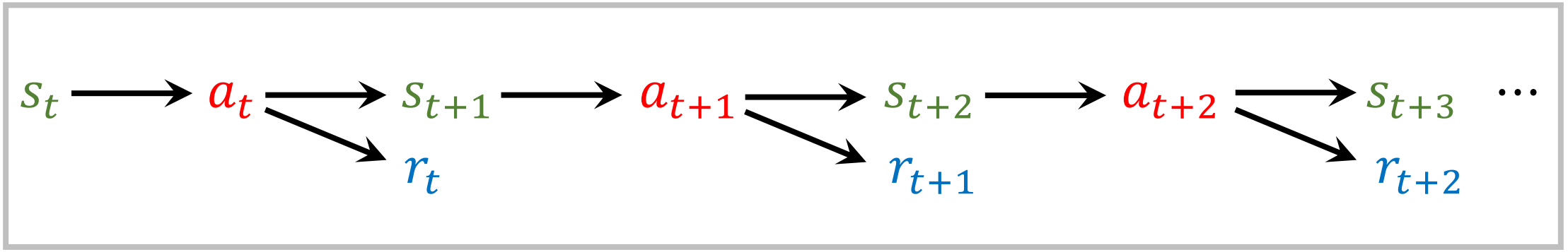
Play game using reinforcement learning

- Observe state s_t, make action a_t, environment gives s_{t+1} and reward r_t.



Play game using reinforcement learning

- Observe state s_t, make action a_t, environment gives s_{t+1} and reward r_t.



- The agent can be controlled by either $\pi(a|s)$ or $Q^\star(s,a)$.

Thank you!