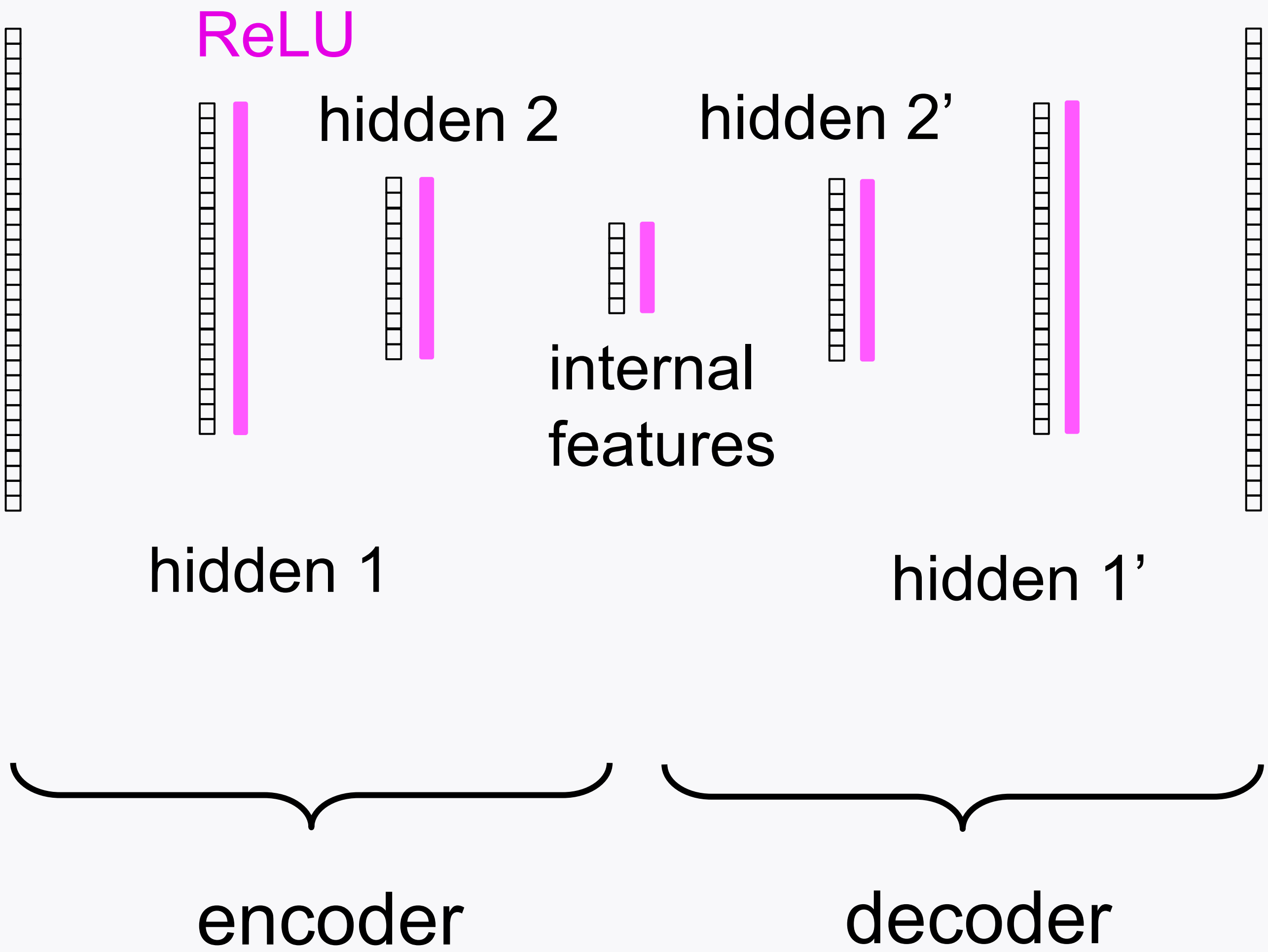
Generative Adversarial Networks (GANs) & Restricted Boltzmann Machines (RBMs)

Lecture 22

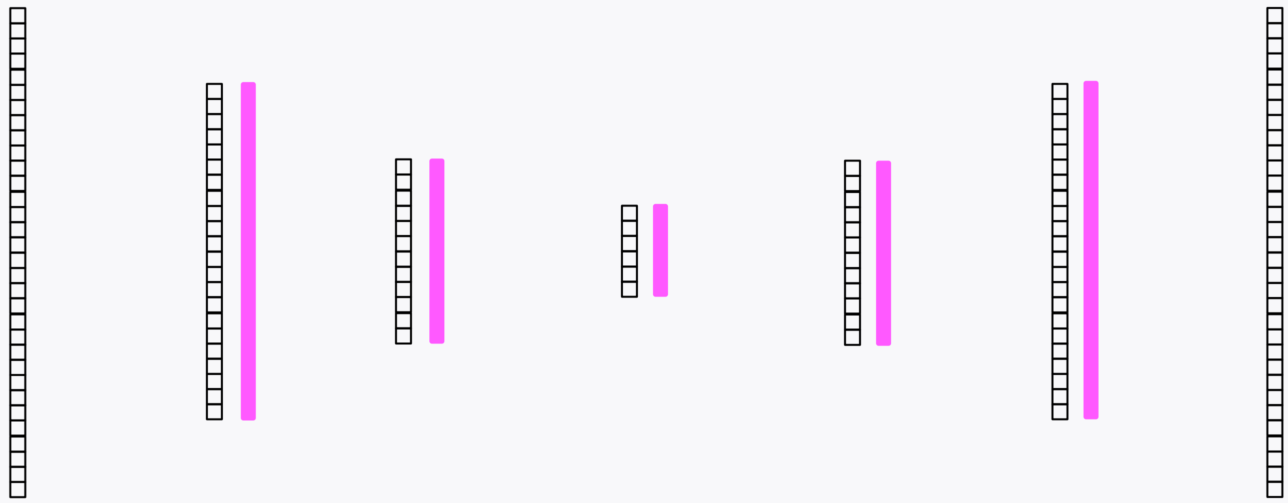
Changho Suh

October 7, 2021

Recap: Autoencoder



Recap: Autoencoder



1. Naive method

2. Standard method

3. Standard method with tying weights

Recap: Roles of autoencoder

1. Encoder: Dimensionality reduction
2. Encoder: Semi-supervised learning
3. Decoder: A generative model (with a random input)
4. Matrix completion
5. Anomaly detection

Recap: Coding for autoencoder

```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Input
```

encoder

```
Inputs_ = Input(shape = (784, ))
x = Dense(128, activation = 'relu')(Inputs_)
x = Dense(64, activation = 'relu')(x)
encoded = Dense(32, activation = 'relu')(x)
```

decoder

```
x = Dense(64, activation = 'relu')(encoded)
x = Dense(128, activation = 'relu')(x)
Outputs_ = Dense(784, activation = 'sigmoid')(x)
```

```
AutoEncoder = Model(Inputs_, Outputs_)

AutoEncoder.compile(loss='binary_crossentropy',optimizer ='adam')

AutoEncoder.fit(X_train, X_train, epochs=20)
```

Recap: Matrix completion

1.0	2.5	*
1.5	*	0.1
*	2.5	0.5

1.0	2.5	0.5
1.5	-2.1	0.1
2.0	2.5	0.5

Plays a significant role in estimating missing entries that is often needed in fusion learning.

Studied an AE-based matrix completion method.

Recap: Coding for matrix completion

```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Input

input_ = Input(shape=(610,))

x = Dense(128, activation='relu')(input_)
x = Dense(64, activation='relu')(x)
encoded = Dense(32, activation='relu')(x)

x = Dense(64, activation='relu')(encoded)
x = Dense(128, activation='relu')(x)
output_= Dense(610)(x)

Autoencoder = Model(input_, output_)
```

encoder

decoder

Recap: Coding for matrix completion

a set of observed entries

Customize an MSE loss function on $(i, j) \in \Omega$

```
def masked_mse(y_true, y_pred):
    mask = tensorflow.cast(y_true!=0,dtype=tf.float32)
    loss = keras.backend.square(mask*(y_pred - y_true))
    loss = keras.backend.mean(loss)
    return loss
```

```
Autoencoder.compile(optimizer = 'adam', loss = masked_mse)
Autoencoder.fit(rating_matrix, rating_matrix, epochs=10)
```

Next topics?

Note: Autoencoder can serve as a generative model.

There is a more powerful generative model based on:

Generative Adversarial Networks (**GANs**)

Prior to GANs, a classical method was often employed:

Restricted Boltzmann Machines (**RBMs**)

Outline of today's lecture

Will explore GANs & RBMs in depth:

1. Investigate the GAN architecture together with its rationale.
2. Study a corresponding opt. and how to solve it.
3. Figure out Boltzmann Machine (BM) and then RBM.
4. Study how RBM can serve as a generative model.
5. Study a couple of concepts regarding RBM.
6. Explore a training method for RBM.

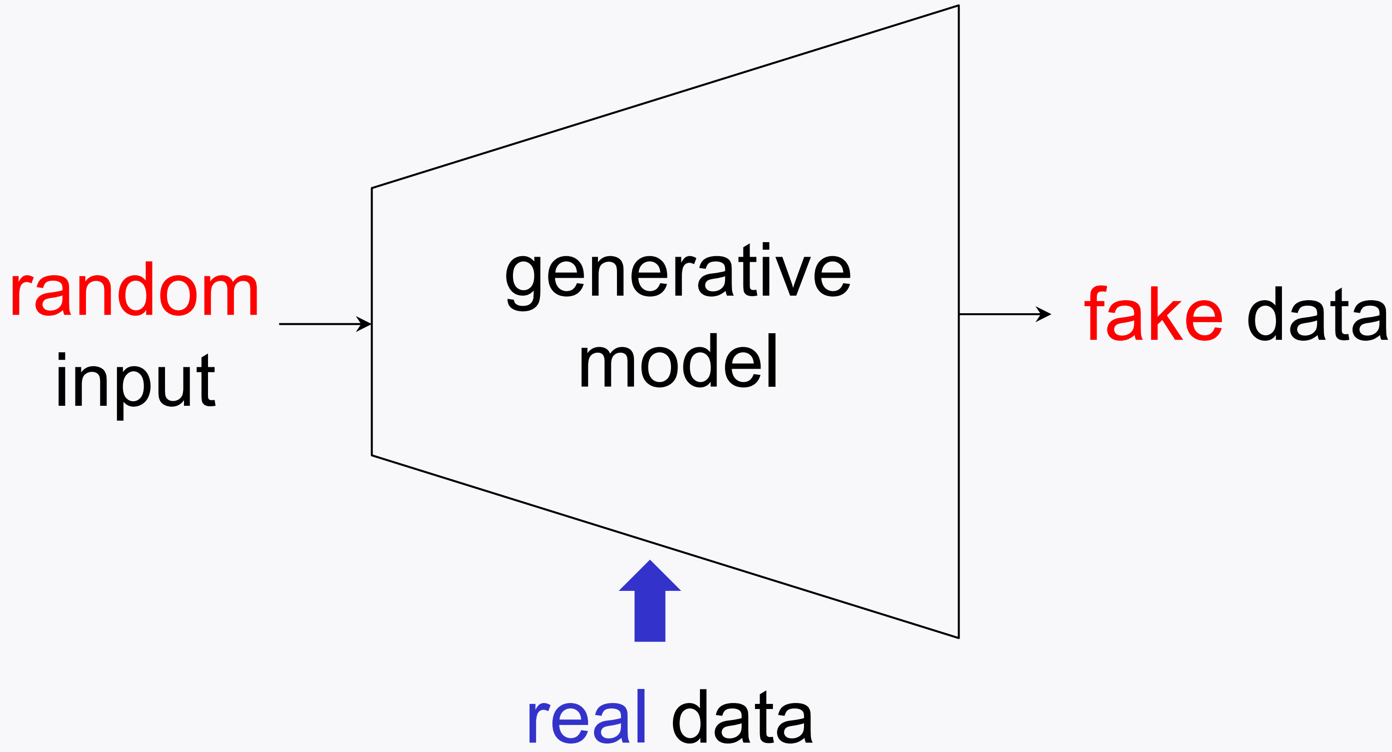
Focus of Lecture 22

Will explore GANs & RBMs in depth:

1. Investigate the GAN architecture together with its rationale.
2. Study a corresponding opt. and how to solve it.
3. Figure out Boltzmann Machine (BM) and then RBM.
4. Study how RBM can serve as a generative model.
5. Study a couple of concepts regarding RBM.
6. Explore a training method for RBM.

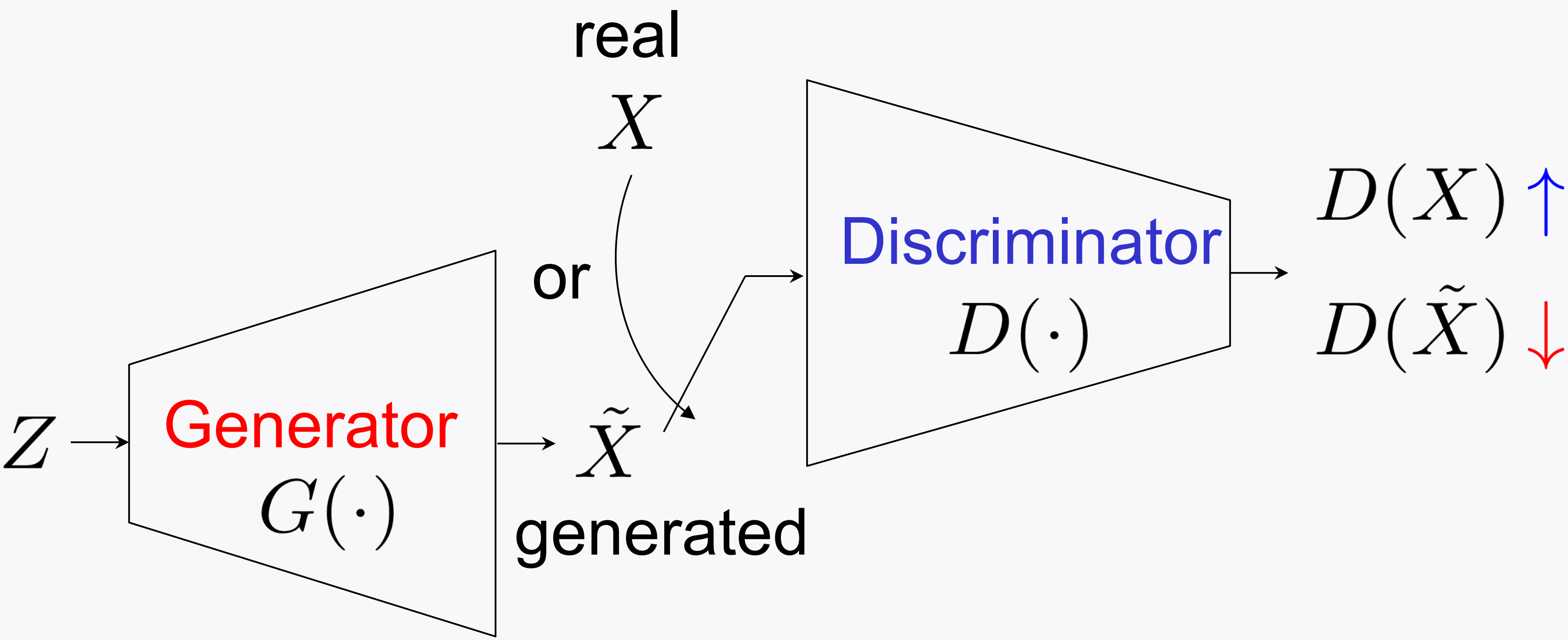
A generative model

A model that generates fake data which has a similar distribution as that of real data.



Generative Adversarial Networks

Goodfellow et al. NeurIPS14

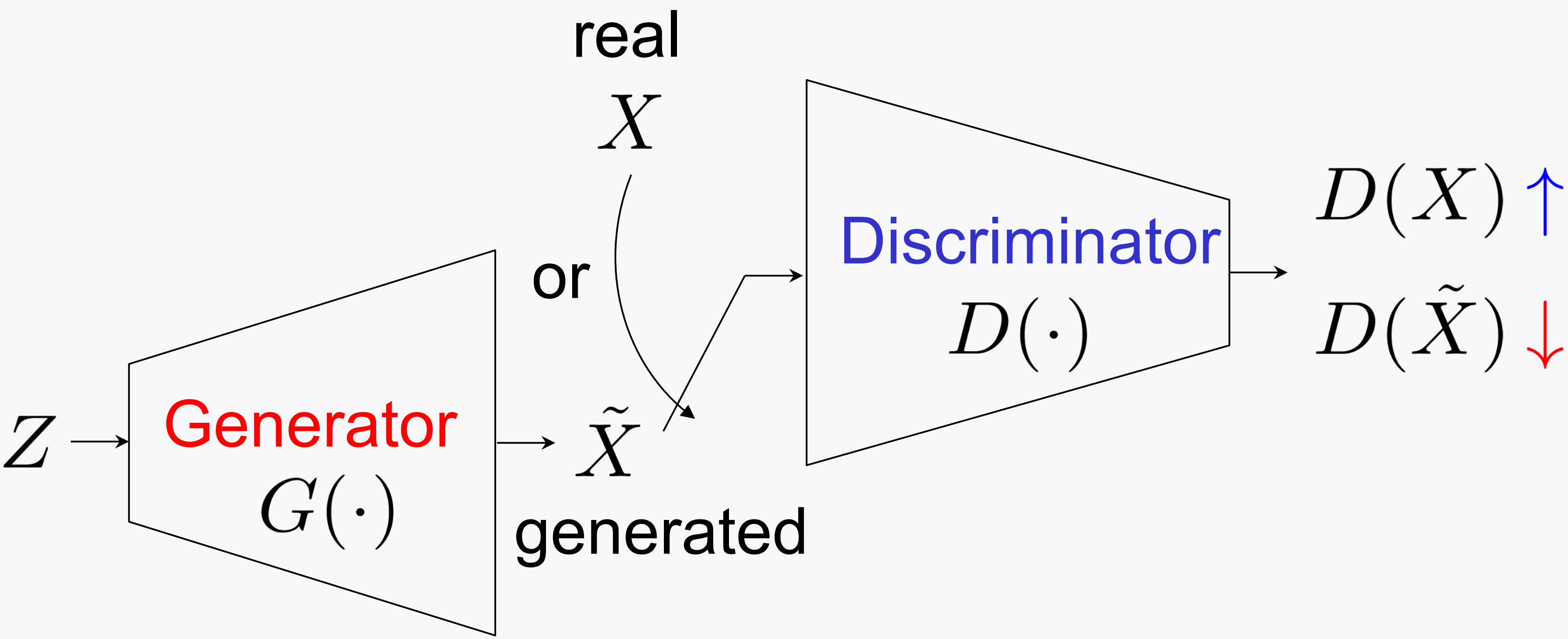


Role: Discriminate real from generated fake samples

Intend to yield a large $D(\cdot)$ if the input is real data;

a small $D(\cdot)$ for generated data.

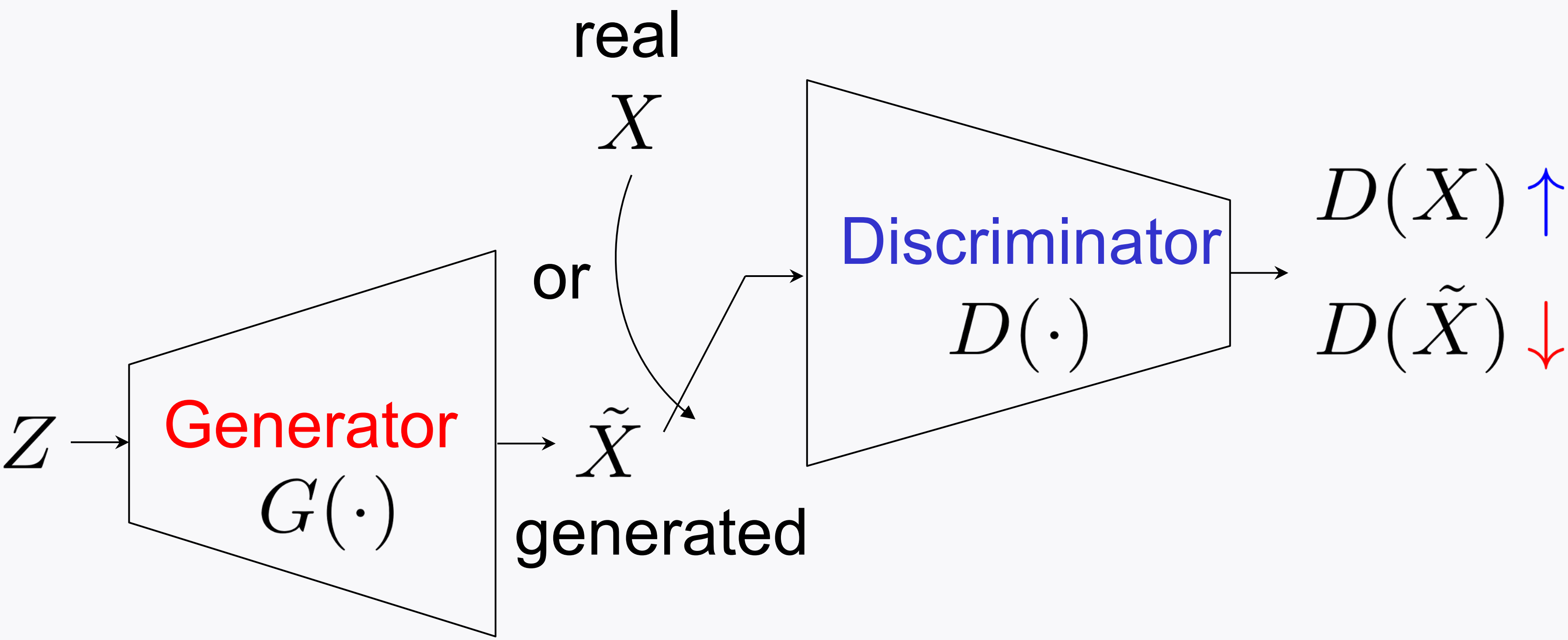
A reasonable interpretation on $D(\cdot)$



Probability of the input being real:

$$D(\cdot\quad) = \mathbb{P}((\cdot) = \text{real})$$

A reasonable interpretation on $D(\cdot)$

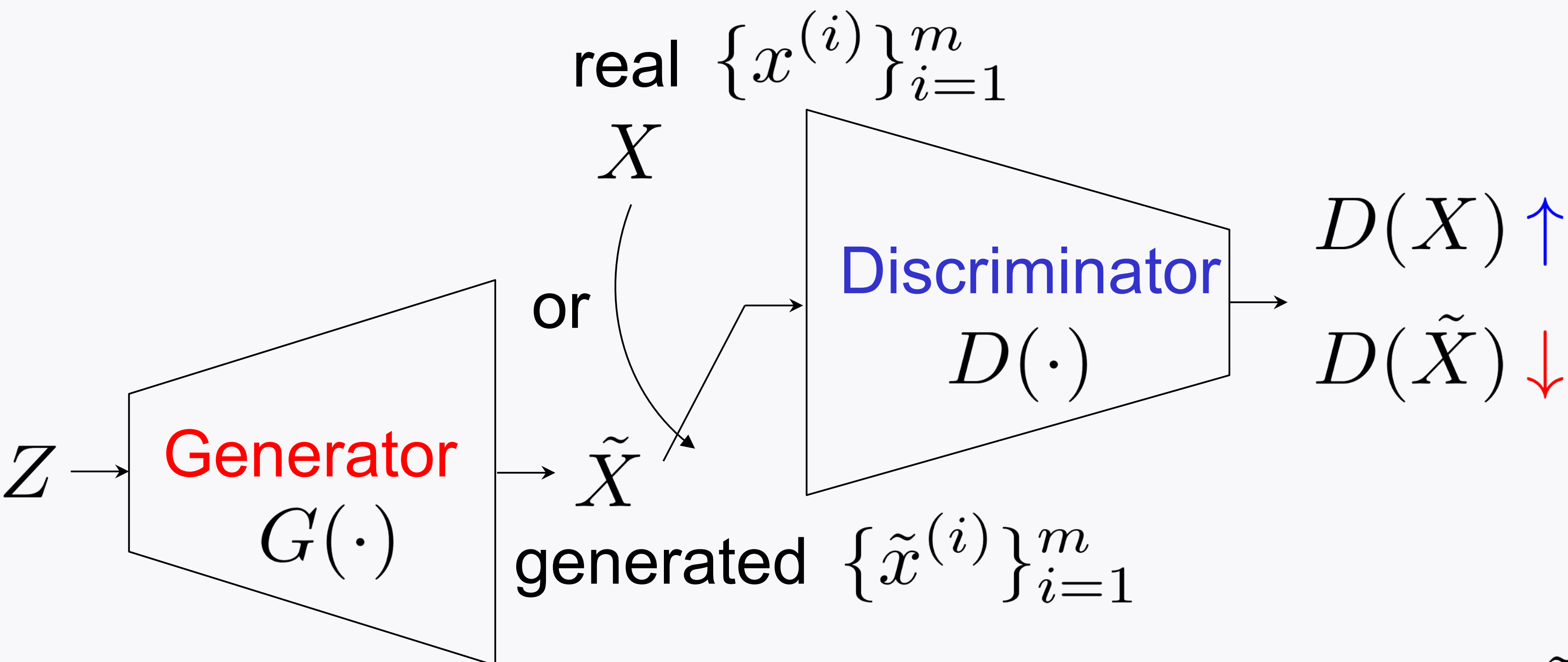


Probability of the input being real:

$$\uparrow \quad D(X) = \mathbb{P}(X = \text{real}) = 1$$

$$\downarrow \quad D(\tilde{X}) = \mathbb{P}(\tilde{X} = \text{real}) = 0$$

Optimization?

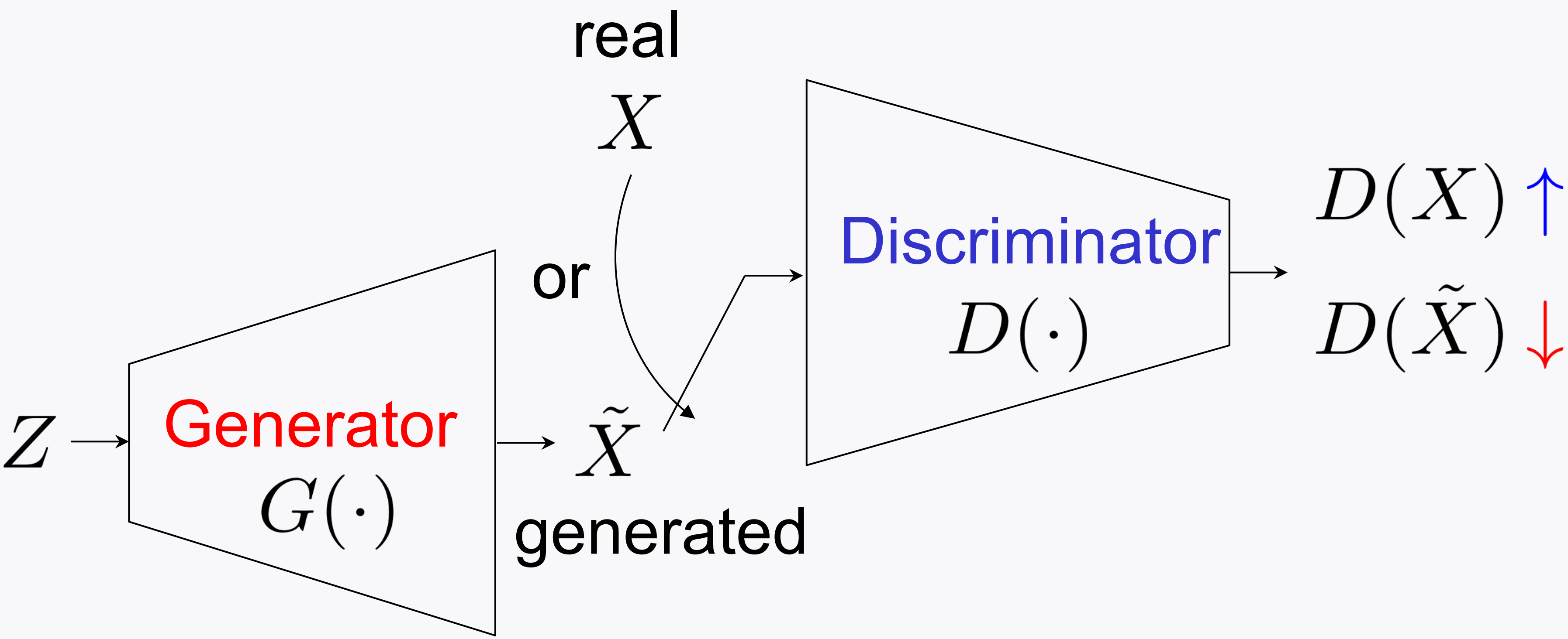


Discriminator wishes to maximize: $D(X)$ & $1 - D(\tilde{X})$

A natural optimization:

$$\max_D \frac{1}{m}\sum_{i=1}^m D(x^{(i)}) + \frac{1}{m}\sum_{i=1}^m 1 - D(\tilde{x}^{(i)})$$

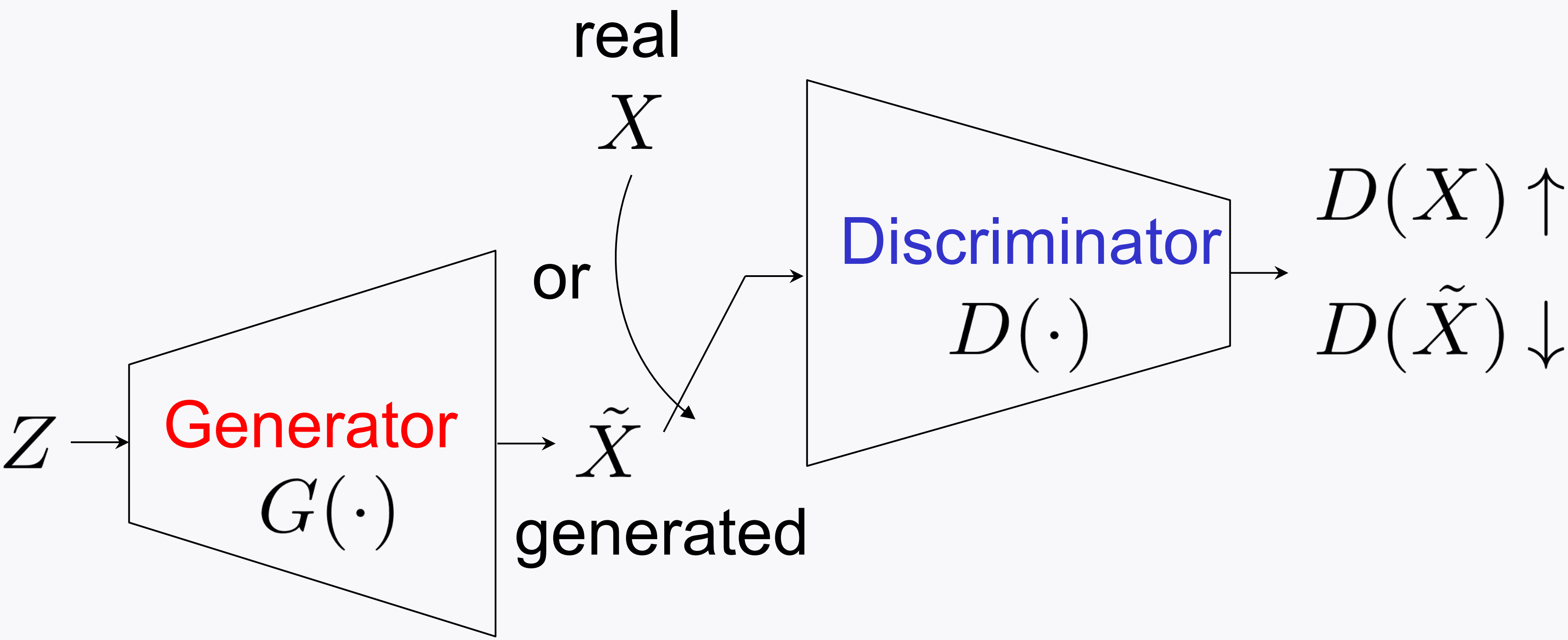
Log loss



Goodfellow employed log loss instead:

$$\max_D \frac{1}{m}\sum_{i=1}^{m} \log D(x^{(i)}) + \frac{1}{m}\sum_{i=1}^{m} \log(1 - D(\tilde{x}^{(i)}))$$

Optimization



Discriminator: $\max_{D} \frac{1}{m}\sum_{i=1}^{m} \log D(x^{(i)}) + \frac{1}{m}\sum_{i=1}^{m} \log(1 - D(\tilde{x}^{(i)}))$

Generator: $\min_{G} \frac{1}{m}\sum_{i=1}^{m} \log D(x^{(i)}) + \frac{1}{m}\sum_{i=1}^{m} \log(1 - D(\tilde{x}^{(i)}))$

$\tilde{x}^{(i)} = G(z^{(i)})$

Optimization

$$\min_{\underline{G(\cdot)}} \max_{\underline{D(\cdot)}} \frac{1}{m} \sum_{i=1}^{m} \log D(x^{(i)}) + \log(1 - D(G(z^{(i)})))$$

Question: How to solve?

Note: function optimization!

Neural net optimization

$$\min_{G(\cdot)\in\mathcal{N}} \max_{D(\cdot)\in\mathcal{N}} \frac{1}{m}\sum_{i=1}^{m} \log D(x^{(i)}) + \log(1 - D(G(z^{(i)})))$$

Take function class as neural networks.

And then parameterize them:

$$G_w(\cdot) \quad D_\theta(\cdot)$$

Optimization with parameters (w, θ)

$$\min_{w} \max_{\theta} \frac{1}{m} \sum_{i=1}^{m} \log D_\theta(x^{(i)}) + \log(1 - D_\theta(G_w(z^{(i)})))$$

Question: How to deal with min-max?

Theorem

$$\min_{w} \max_{\theta} \frac{1}{m} \sum_{i=1}^{m} \log D_\theta(x^{(i)}) + \log(1 - D_\theta(G_w(z^{(i)})))$$

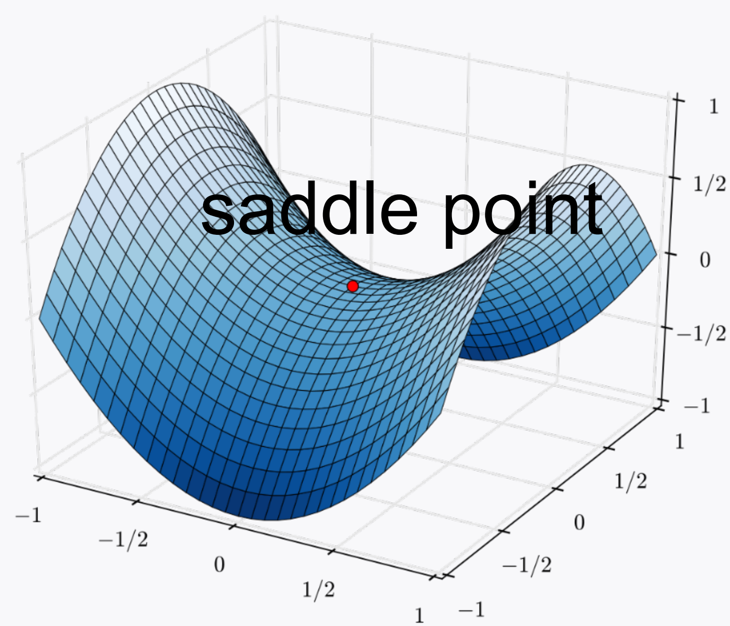
$$:= J(w, \theta)$$

Suppose:

$J(w, \theta)$ convex in w

$J(w, \theta)$ concave in θ

→ The saddle point is the optimal solution.



$J(w,\theta)$ convex-concave?

$$\min_{w} \max_{\theta} \frac{1}{m}\sum_{i=1}^{m} \log D_\theta(x^{(i)}) + \log(1 - D_\theta(G_w(z^{(i)})))$$

$$:= J(w,\theta)$$

No! In general, it is highly non-convex in w
and highly non-concave in θ

What can we do then?

$$\min_{w} \max_{\theta} \frac{1}{m}\sum_{i=1}^{m} \log D_\theta(x^{(i)}) + \log(1 - D_\theta(G_w(z^{(i)})))$$

$$:= J(w, \theta)$$

Nonetheless: Find a stationary point such that

$$\nabla_w J(w^*, \theta^*) = 0, \ \nabla_\theta J(w^*, \theta^*) = 0$$

Hope: Such point yields a near optimal performance.

Turns out: It is often the case in reality.

How to find a stationary point?

$$\min_{w} \max_{\theta} \frac{1}{m} \sum_{i=1}^{m} \log D_\theta(x^{(i)}) + \log(1 - D_\theta(G_w(z^{(i)})))$$

$$:= J(w, \theta)$$

One practical method:

Alternating gradient descent

Alternating gradient descent

1. Update Generator's weight:

$$w^{(t+1)} \leftarrow w^{(t)} - \alpha_1 \nabla_w J(w^{(t)}, \theta^{(t)})$$

2. Given $(w^{(t+1)}, \theta^{(t)})$: update Discriminator's weight:

$$\underbrace{\theta^{(t+1)} \leftarrow \theta^{(t)} + \alpha_2 \nabla_\theta J(w^{(t+1)}, \theta^{(t)})}_{\text{may repeat } k \text{ times}}$$

3. Repeat the above.

k:1 alternating gradient descent

1. Update Generator's weight:

$$w^{(t+1)} \leftarrow w^{(t)} - \alpha_1 \nabla_w J(w^{(t)}, \theta^{(t \cdot k)})$$

2. Update Discriminator's weight *k* times:

```
for i=1:k
```

$$\theta^{(t \cdot k+i)} \leftarrow \theta^{(t \cdot k+i-1)} + \alpha_2 \nabla_\theta J(w^{(t+1)}, \theta^{(t \cdot k+i-1)})$$

3. Repeat the above.

In practice: Often use Batch version & Adam.

A practical tip on Generator

Given Discriminator's parameter θ :

$$\min_{w} \frac{1}{m_{\mathcal{B}}} \sum_{i \in \mathcal{B}} \underbrace{\log D_{\theta}(x^{(i)})}_{\text{irrelevant of } w} + \log(1 - D_{\theta}(G_{w}(z^{(i)})))$$

Suffice to consider: "generator loss"

$$\min_{w} \frac{1}{m_{\mathcal{B}}} \sum_{i \in \mathcal{B}} \log(1 - D_{\theta}(G_{w}(z^{(i)})))$$

In practice, consider a *proxy*:

$$\min_{w} \frac{1}{m_{\mathcal{B}}} \sum_{i \in \mathcal{B}} -\log D_{\theta}(G_{w}(z^{(i)}))$$

Look ahead

1. Figure out Boltzmann Machine (BM) and then RBM.

2. Study how RBM can serve as a generative model.