Deep Learning Concepts

Sergey Ivanov (617)

qbrick@mail.ru

September 16, 2019

1 Deep Learning
- Basic idea
- Supervised learning
- Unsupervised learning

Deep Learning

Basic idea

Key principle

Suppose we want to find some function $y(x)$.

Concept of learning

1. construct some model $y = f(x, \theta)$ using basic building blocks

Key principle

Suppose we want to find some function $y(x)$.

Concept of learning

1. construct some model $y = f(x, \theta)$ using basic building blocks
2. select some differentiable scalar criterion to optimize $L(f)$

Key principle

Suppose we want to find some function $y(x)$.

Concept of learning

1. construct some model $y = f(x, \theta)$ using basic building blocks
2. select some differentiable scalar criterion to optimize $L(f)$
3. select optimization procedure (i.e. gradient descent)

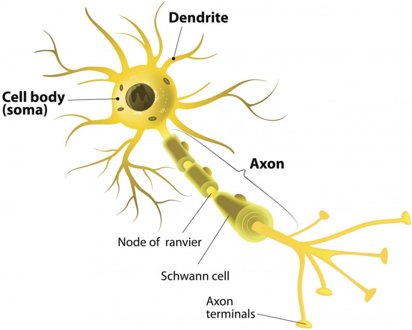
Key principle

Suppose we want to find some function $y(x)$.

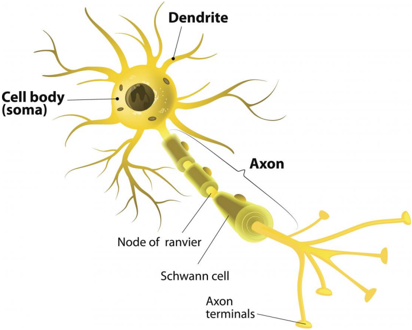
Concept of learning

1. construct some model $y = f(x, \theta)$ using basic building blocks
2. select some differentiable scalar criterion to optimize $L(f)$
3. select optimization procedure (i.e. gradient descent)
4. solve $\theta^* = \min\limits_{\theta} L(f)$

Neurons

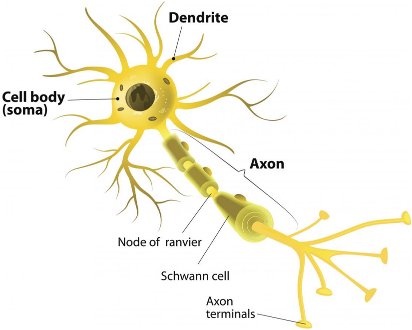

Dendrite
Cell body
(soma)
Axon
Node of ranvier
Schwann cell
Axon
terminals

Neurons



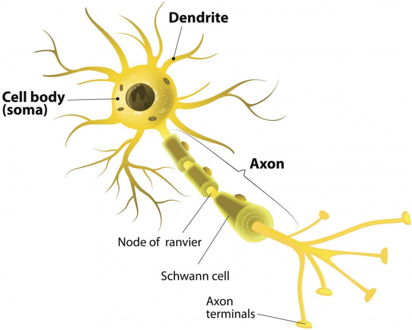
input: $x \in \{0,1\}^n$

Neurons



input: $x \in \{0, 1\}^n$
parameters: $w \in \mathbb{R}^n$, $b \in \mathbb{R}$

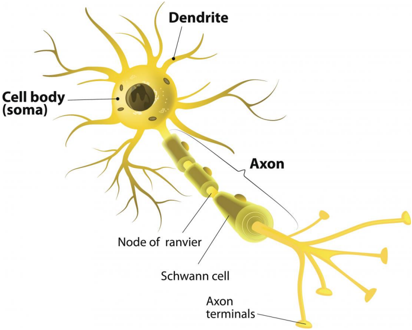
Neurons



input: $x \in \{0, 1\}^n$
parameters: $w \in \mathbb{R}^n$, $b \in \mathbb{R}$

1. ***i*-th signal:** $w_i x_i$

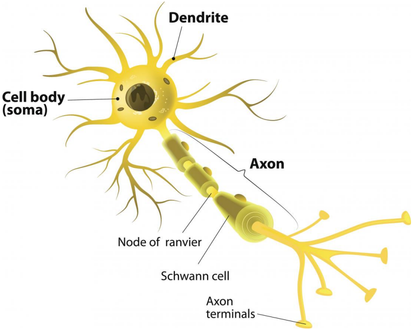
Neurons



<u>input:</u> $x \in \{0,1\}^n$
<u>parameters:</u> $w \in \mathbb{R}^n$, $b \in \mathbb{R}$

1. i-th signal: $w_i x_i$
2. accumulation: $\sum_i w_i x_i$

Neurons



input: $x \in \{0, 1\}^n$

parameters: $w \in \mathbb{R}^n$, $b \in \mathbb{R}$

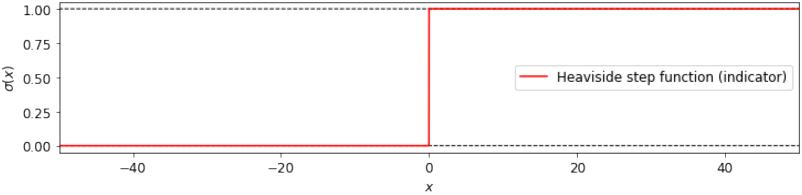
1. **i-th signal:** $w_i x_i$
2. **accumulation:** $\sum_i w_i x_i$
3. **output:** $\sum_i w_i x_i > b$

Artificial neurons

$$y(x) = \mathbb{I}[\langle w, x \rangle - b > 0]$$

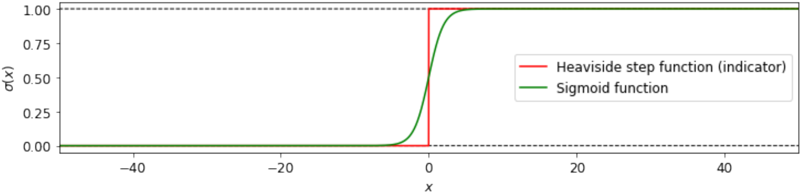
Artificial neurons

$$y(x) = \mathbb{I}[\langle w, x \rangle - b > 0]$$



Artificial neurons

$$y(x) = \mathbb{I}[\langle w, x \rangle - b > 0]$$

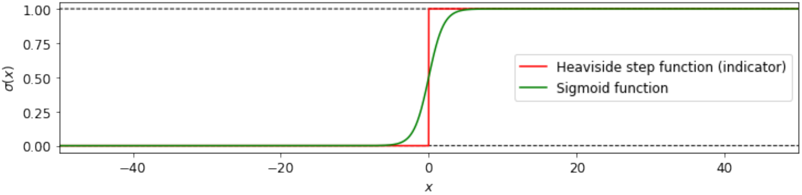


General idea

Everything discrete can be smoothed!

Artificial neurons

$$y(x) = \sigma\left(\langle w, x \rangle - b\right)$$



General idea

Everything discrete can be smoothed!

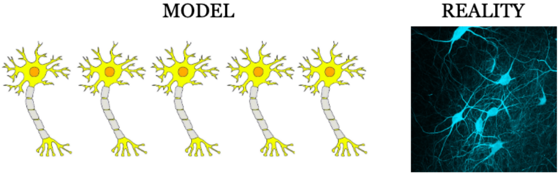
Sigmoid function:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Fully-connected layer

Standard building block for neural networks:

$$y(x) = \sigma(Wx - b)$$

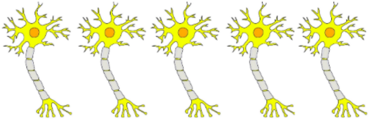


Fully-connected layer

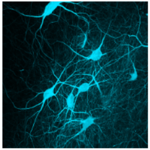
Standard building block for neural networks:

$$y(x) = \sigma(Wx - b)$$

MODEL



REALITY



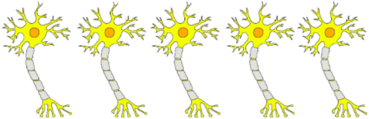
universal approximation properties!

Fully-connected layer

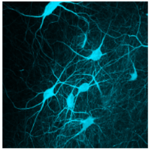
Standard building block for neural networks:

$$y(x) = \sigma(Wx - b)$$

MODEL



REALITY



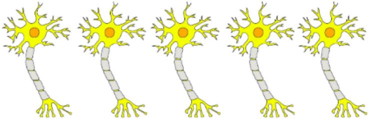
- universal approximation properties!
- if there is infinite number of neurons...

Fully-connected layer

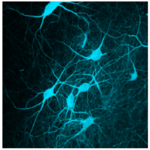
Standard building block for neural networks:

$$y(x) = \sigma(Wx - b)$$

MODEL



REALITY

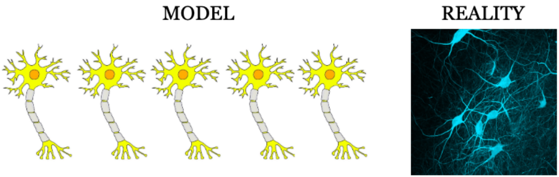


- universal approximation properties!
- if there is infinite number of neurons...
- stack more layers!

Fully-connected layer

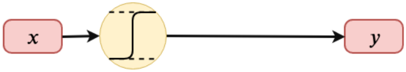
Standard building block for neural networks:

$$y(x) = \sigma(Wx - b)$$

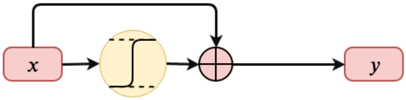


- 😇 universal approximation properties!
- 😈 if there is infinite number of neurons...
- 😇 stack more layers!
- 😈 gradient vanishing / exploding problem!

Stacking a lot of layers



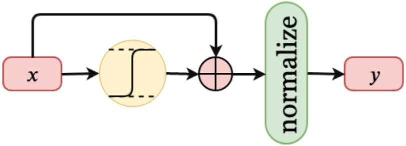
Stacking a lot of layers



Residual connections

$$y = x + \sigma(Wx - b)$$

Stacking a lot of layers



Residual connections

$$y = x + \sigma(Wx - b)$$

Layer normalization

$$\mu = \frac{1}{m}\sum_{i}^{n} x_i \qquad s^2 = \frac{1}{m}\sum_{i}^{n}(x_i - \mu) \qquad y = (x - \mu)/s$$

Typical issues

- input x may have some complex structure: how to convert it to vector in $\mathbb{R}^d$?

Typical issues

- input x may have some complex structure: how to convert it to vector in $\mathbb{R}^d$?
  - **categorical features:** one-hot encoding

Typical issues

- input x may have some complex structure: how to convert it to vector in $\mathbb{R}^d$?
    - **categorical features:** one-hot encoding
    - **images:** convolutional layers + pooling (CNN)

Typical issues

- input x may have some complex structure: how to convert it to vector in $\mathbb{R}^d$?
  - **categorical features:** one-hot encoding
  - **images:** convolutional layers + pooling (CNN)
  - **sequence:** recurrent layers (RNN, LSTM, GRU)

Typical issues

- input x may have some complex structure: how to convert it to vector in $\mathbb{R}^d$?
  - **categorical features:** one-hot encoding
  - **images:** convolutional layers + pooling (CNN)
  - **sequence:** recurrent layers (RNN, LSTM, GRU)
  - **raw audio:** ?!?

Typical issues

- input x may have some complex structure: how to convert it to vector in $\mathbb{R}^d$?
  - **categorical features:** one-hot encoding
  - **images:** convolutional layers + pooling (CNN)
  - **sequence:** recurrent layers (RNN, LSTM, GRU)
  - **raw audio:** ?!?
- output y may have some complex structure: how to build the model?

Typical issues

- input x may have some complex structure: how to convert it to vector in $\mathbb{R}^d$?
  - **categorical features:** one-hot encoding
  - **images:** convolutional layers + pooling (CNN)
  - **sequence:** recurrent layers (RNN, LSTM, GRU)
  - **raw audio:** ?!?
- output y may have some complex structure: how to build the model?
- no or little data available, how to choose criterion?

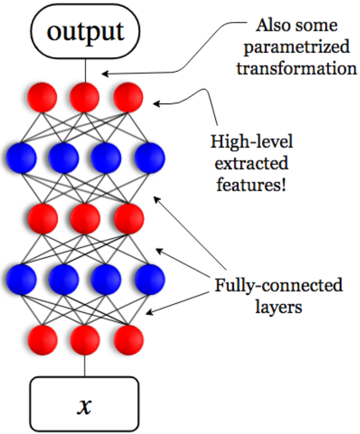
Typical issues

- input x may have some complex structure: how to convert it to vector in $\mathbb{R}^d$?
  - **categorical features:** one-hot encoding
  - **images:** convolutional layers + pooling (CNN)
  - **sequence:** recurrent layers (RNN, LSTM, GRU)
  - **raw audio:** ?!?
- output y may have some complex structure: how to build the model?
- no or little data available, how to choose criterion?
- uninterpretable («black box» model)

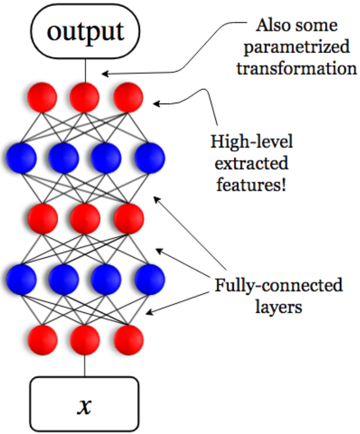
Deep Learning

Supervised learning

Supervised learning

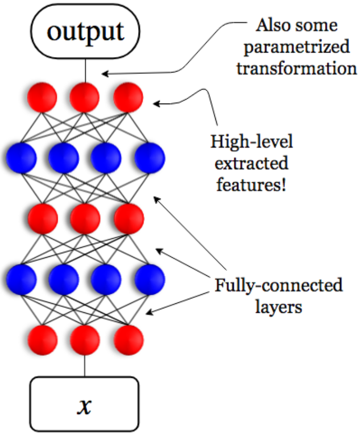


Supervised learning



Let (x_i, y_i) be our data.
$x_i \in \mathbb{R}^D$

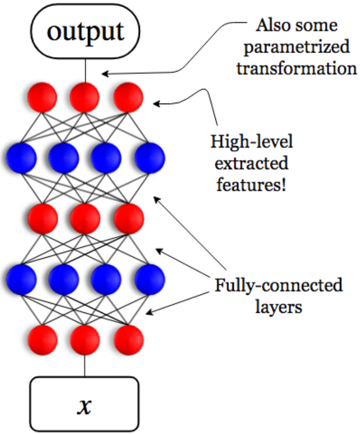
Supervised learning



Let (x_i, y_i) be our data.
$x_i \in \mathbb{R}^D$

1. stack some FC layers and get **high-level representation** $z(x) \in \mathbb{R}^d$

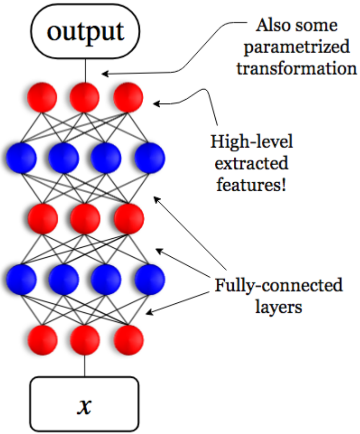
Supervised learning



Let (x_i, y_i) be our data.
$x_i \in \mathbb{R}^D$

1. stack some FC layers and get **high-level representation** $z(x) \in \mathbb{R}^d$
2. choose final decision rule $\hat{y}(z)$.

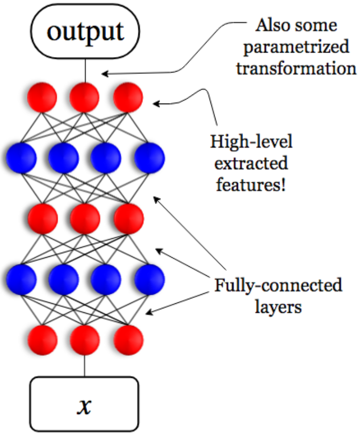
Supervised learning



Let (x_i, y_i) be our data.
$x_i \in \mathbb{R}^D$

1. stack some FC layers and get **high-level representation** $z(x) \in \mathbb{R}^d$
2. choose final decision rule $\hat{y}(z)$.
3. choose loss function $\text{Loss}(y, \hat{y})$

Supervised learning



Let (x_i, y_i) be our data.
$x_i \in \mathbb{R}^D$

1. stack some FC layers and get **high-level representation** $z(x) \in \mathbb{R}^d$
2. choose final decision rule $\hat{y}(z)$.
3. choose loss function $\text{Loss}(y, \hat{y})$
4. $L(f) = \frac{1}{N}\sum\limits_i \text{Loss}(y_i, \hat{y}(z(x_i)))$

Final decision rules

Here $z \in \mathbb{R}^d$ is high-level representation (outputs from neurons on final layer).

- $y \in \mathbb{R}$

Final decision rules

Here $z \in \mathbb{R}^d$ is high-level representation (outputs from neurons on final layer).

- $y \in \mathbb{R}$
    - **Linear layer:** $\hat{y} = \langle w, z \rangle + b$

Final decision rules

Here $z \in \mathbb{R}^d$ is high-level representation (outputs from neurons on final layer).

- $y \in \mathbb{R}$
  - **Linear layer:** $\hat{y} = \langle w, z \rangle + b$
- $y \in [0, 1]$

Final decision rules

Here $z \in \mathbb{R}^d$ is high-level representation (outputs from neurons on final layer).

- $y \in \mathbb{R}$
  - **Linear layer:** $\hat{y} = \langle w, z \rangle + b$
- $y \in [0, 1]$
  - **Linear layer + sigmoid:** $\hat{y} = \sigma\left(\langle w, z \rangle + b\right)$

Final decision rules

Here $z \in \mathbb{R}^d$ is high-level representation (outputs from neurons on final layer).

- $y \in \mathbb{R}$
  - **Linear layer:** $\hat{y} = \langle w, z \rangle + b$
- $y \in [0, 1]$
  - **Linear layer + sigmoid:** $\hat{y} = \sigma\left(\langle w, z \rangle + b\right)$
- $y \in \mathbb{R}_{++}$

Final decision rules

Here $z \in \mathbb{R}^d$ is high-level representation (outputs from neurons on final layer).

- $y \in \mathbb{R}$
  - **Linear layer:** $\hat{y} = \langle w, z \rangle + b$
- $y \in [0, 1]$
  - **Linear layer + sigmoid:** $\hat{y} = \sigma\left(\langle w, z \rangle + b\right)$
- $y \in \mathbb{R}_{++}$
  - **Linear + exp:** $\hat{y} = e^{\langle w, z \rangle + b}$

Final decision rules

Here $z \in \mathbb{R}^d$ is high-level representation (outputs from neurons on final layer).

- $y \in \mathbb{R}$
  - **Linear layer:** $\hat{y} = \langle w, z \rangle + b$
- $y \in [0, 1]$
  - **Linear layer + sigmoid:** $\hat{y} = \sigma\left(\langle w, z \rangle + b\right)$
- $y \in \mathbb{R}_{++}$
  - **Linear + exp:** $\hat{y} = e^{\langle w, z \rangle + b}$
  - **Linear + softplus:** $\hat{y} = \log\left(1 + e^{\langle w, z \rangle + b}\right)$

Final decision rules

Here $z \in \mathbb{R}^d$ is high-level representation (outputs from neurons on final layer).

- $y \in \mathbb{R}$
  - **Linear layer:** $\hat{y} = \langle w, z \rangle + b$
- $y \in [0, 1]$
  - **Linear layer + sigmoid:** $\hat{y} = \sigma\left(\langle w, z \rangle + b\right)$
- $y \in \mathbb{R}_{++}$
  - **Linear + exp:** $\hat{y} = e^{\langle w, z \rangle + b}$
  - **Linear + softplus:** $\hat{y} = \log\left(1 + e^{\langle w, z \rangle + b}\right)$
- $y \in \{1, 2, 3 \dots C\}$

Final decision rules

Here $z \in \mathbb{R}^d$ is high-level representation (outputs from neurons on final layer).

- $y \in \mathbb{R}$
  - **Linear layer:** $\hat{y} = \langle w, z \rangle + b$
- $y \in [0, 1]$
  - **Linear layer + sigmoid:** $\hat{y} = \sigma\left(\langle w, z \rangle + b\right)$
- $y \in \mathbb{R}_{++}$
  - **Linear + exp:** $\hat{y} = e^{\langle w, z \rangle + b}$
  - **Linear + softplus:** $\hat{y} = \log\left(1 + e^{\langle w, z \rangle + b}\right)$
- $y \in \{1, 2, 3 \dots C\}$
  - **Linear layer + softmax:** $\hat{y} = \text{softmax}\left(\langle w, z \rangle + b\right)$
    (softmax = exp + normalize)

Loss functions

- Regression
  - MSE, MAE

Loss functions

- Regression
  - MSE, MAE
- Classification

Loss functions

- Regression
  - MSE, MAE
- Classification
  - why cross-entropy is so popular?

Loss functions

- Regression
  - MSE, MAE
- Classification
  - why cross-entropy is so popular?

Probabilistic interpretation of supervised learning

$$x, y \sim p(x, y) = p(x)p(y \mid x)$$

$$p(y \mid x) \text{ — ?}$$

Loss functions

- Regression
  - MSE, MAE
- Classification
  - why cross-entropy is so popular?

Probabilistic interpretation of supervised learning

$$x, y \sim p(x, y) = p(x)p(y \mid x)$$

$$p(y \mid x) — ?$$

Our neural network actually defines approximating distribution $q(y \mid x, \theta)$. What to do next?

Losses derivation

- **Maximum likelihood estimation:**

$$\prod_i q(y_i \mid x_i, \theta) \to \max_\theta$$

Losses derivation

- **Maximum likelihood estimation:**

$$\prod_i q(y_i \mid x_i, \theta) \to \max_\theta$$

- **Divergence minimization:**

$$\mathcal{D}(p(y \mid x) \parallel q(y_i \mid x_i, \theta)) \to \min_\theta$$

Losses derivation

- **Maximum likelihood estimation:**

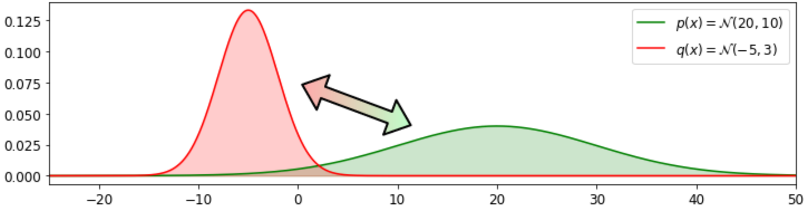
$$\prod_i q(y_i \mid x_i, \theta) \to \max_\theta$$

- **Divergence minimization:**

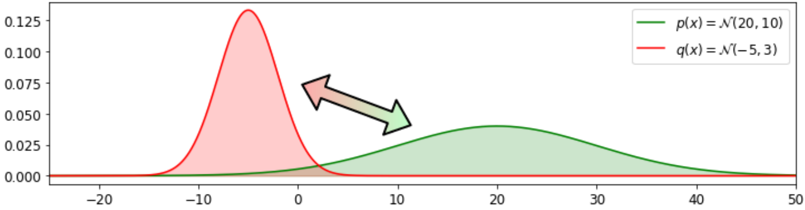
$$\mathcal{D}(p(y \mid x) \parallel q(y_i \mid x_i, \theta)) \to \min_\theta$$

- **Bayesian inference:** seek for $p(\theta \mid X, Y)$

Divergences

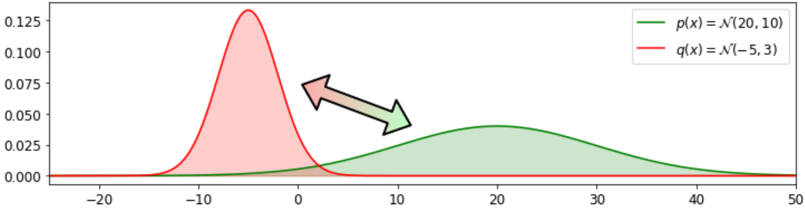


Divergences



- Kullback-Leibler divergence
- Wasserstein distance
- Jensen-Shannon divergence
- Cramer distance
- ...

Divergences



- **Kullback-Leibler divergence — the chosen one!**
- Wasserstein distance
- Jensen-Shannon divergence
- Cramer distance
- ...

Kullback-Leibler Divergence

Definition

$$\mathrm{KL}(p \parallel q) := \int_{\mathcal{Y}} p(y) \log \frac{p(y)}{q(y)} dy$$

Kullback-Leibler Divergence

Definition

$$\mathrm{KL}(p \parallel q) := \int_{\mathcal{Y}} p(y) \log \frac{p(y)}{q(y)} dy = \mathbb{E}_{p(y)} \log \frac{p(y)}{q(y)}$$

Kullback-Leibler Divergence

Definition

$$\mathrm{KL}(p \parallel q) := \int_{\mathcal{Y}} p(y) \log \frac{p(y)}{q(y)} dy = \mathbb{E}_{p(y)} \log \frac{p(y)}{q(y)}$$

Wonderful properties:

× p and q must share domain

Kullback-Leibler Divergence

Definition

$$\mathrm{KL}(p \parallel q) := \int_{\mathcal{Y}} p(y) \log \frac{p(y)}{q(y)} dy = \mathbb{E}_{p(y)} \log \frac{p(y)}{q(y)}$$

Wonderful properties:

- × p and q must share domain
- × assymetric

Kullback-Leibler Divergence

Definition

$$\mathrm{KL}(p \parallel q) := \int_{\mathcal{Y}} p(y) \log \frac{p(y)}{q(y)} dy = \mathbb{E}_{p(y)} \log \frac{p(y)}{q(y)}$$

Wonderful properties:

- × p and q must share domain
- × assymetric
- × does not satisfy the triangle inequality

Kullback-Leibler Divergence

Definition

$$\mathrm{KL}(p \parallel q) := \int_{\mathcal{Y}} p(y) \log \frac{p(y)}{q(y)} dy = \mathbb{E}_{p(y)} \log \frac{p(y)}{q(y)}$$

Wonderful properties:

- × p and q must share domain
- × assymetric
- × does not satisfy the triangle inequality

Motivation behind Kullback-Leibler

Recall our task:

$$\mathrm{KL}(p(y \mid x) \parallel q(y_i \mid x_i, \theta)) \to \min_{\theta}$$

Motivation behind Kullback-Leibler

Recall our task:

$$\mathrm{KL}(p(y \mid x) \parallel q(y_i \mid x_i, \theta)) \to \min_\theta$$

Using definition:

$$\mathbb{E}_{p(y|x)} \log p(y \mid x) - \mathbb{E}_{p(y|x)} \log q(y_i \mid x_i, \theta) \to \min_\theta$$

Motivation behind Kullback-Leibler

Recall our task:

$$\mathrm{KL}(p(y \mid x) \parallel q(y_i \mid x_i, \theta)) \to \min_{\theta}$$

Using definition:

$$\mathbb{E}_{p(y|x)} \log p(y \mid x) - \mathbb{E}_{p(y|x)} \log q(y_i \mid x_i, \theta) \to \min_{\theta}$$

Const(θ) terms can be ignored!

Motivation behind Kullback-Leibler

Recall our task:

$$\mathrm{KL}(p(y \mid x) \parallel q(y_i \mid x_i, \theta)) \to \min_{\theta}$$

Using definition:

$$-\mathbb{E}_{p(y \mid x)} \log q(y_i \mid x_i, \theta) \to \min_{\theta}$$

Const(θ) terms can be ignored!

Motivation behind Kullback-Leibler

Recall our task:

$$\mathrm{KL}(p(y \mid x) \parallel q(y_i \mid x_i, \theta)) \to \min_{\theta}$$

Using definition:

$$-\mathbb{E}_{p(y|x)} \log q(y_i \mid x_i, \theta) \to \min_{\theta}$$

Const(θ) terms can be ignored!

Implicit expectation minimization

We do not know $p(y \mid x)$, but ability to sample from it is enough!

Monte-Carlo gradient estimation

How to calculate gradient for optimization methods in such case?

$$L(f) = \mathbb{E}_{p(x,y)} \operatorname{Loss}(x, y, \theta) \to \min_{\theta}$$

Monte-Carlo gradient estimation

How to calculate gradient for optimization methods in such case?

$$L(f) = \mathbb{E}_{p(x,y)} \operatorname{Loss}(x, y, \theta) \to \min_{\theta}$$

Proposition: $\nabla_\theta L(f) = \mathbb{E}_{p(x,y)} \nabla_\theta \operatorname{Loss}(x, y, \theta)$

Monte-Carlo gradient estimation

How to calculate gradient for optimization methods in such case?

$$L(f) = \mathbb{E}_{p(x,y)} \operatorname{Loss}(x, y, \theta) \to \min_{\theta}$$

Proposition: $\nabla_\theta L(f) = \mathbb{E}_{p(x,y)} \nabla_\theta \operatorname{Loss}(x, y, \theta)$

Monte-Carlo estimation

$$\mathbb{E}_{p(x,y)} \nabla_\theta \operatorname{Loss}(x, y, \theta) \approx \frac{1}{M} \sum_{i}^{M} \nabla_\theta \operatorname{Loss}(x_i, y_i, \theta)$$

where x_i, y_i are samples from $p(x, y)$.

Monte-Carlo gradient estimation

How to calculate gradient for optimization methods in such case?

$$L(f) = \mathbb{E}_{p(x,y)} \operatorname{Loss}(x, y, \theta) \to \min_{\theta}$$

Proposition: $\nabla_\theta L(f) = \mathbb{E}_{p(x,y)} \nabla_\theta \operatorname{Loss}(x, y, \theta)$

Monte-Carlo estimation

$$\mathbb{E}_{p(x,y)} \nabla_\theta \operatorname{Loss}(x, y, \theta) \approx \frac{1}{M} \sum_{i}^{M} \nabla_\theta \operatorname{Loss}(x_i, y_i, \theta)$$

where x_i, y_i are samples from $p(x, y)$.

- ✓ an **unbiased** estimation (gives true gradient in expectation)

Stochastic gradient descent

Use unbiased estimations of gradient instead of true gradients!

Algorithm 1 SGD

Initialize θ_0 randomly
for $t = 0, 1, 2, \dots$ **do**
    Sample M pairs $x_i, y_i \sim p(x, y)$
    $g_t \leftarrow \frac{1}{M} \sum_i^M \nabla_\theta \operatorname{Loss}(x_i, y_i, \theta_t)$
    $\theta_{t+1} \leftarrow \theta_t - \alpha_t g_t$
end for

Stochastic gradient descent

Use unbiased estimations of gradient instead of true gradients!

Algorithm 2 SGD

1: Initialize θ_0 randomly
2: **for** $t = 0, 1, 2, \dots$ **do**
3: Sample M pairs $x_i, y_i \sim p(x, y)$
4: $g_t \leftarrow \frac{1}{M}\sum_i^M \nabla_\theta \operatorname{Loss}(x_i, y_i, \theta_t)$
5: $\theta_{t+1} \leftarrow \theta_t - \alpha_t g_t$
6: **end for**

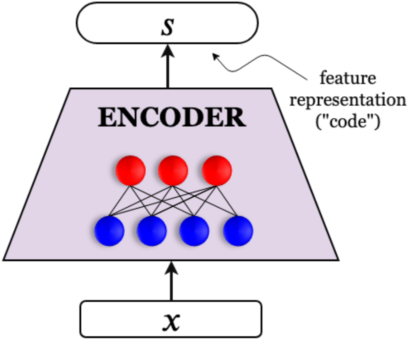
SGD converges to local optima if

$$\sum_t \alpha_t = +\infty \quad \sum_t \alpha_t^2 < +\infty$$

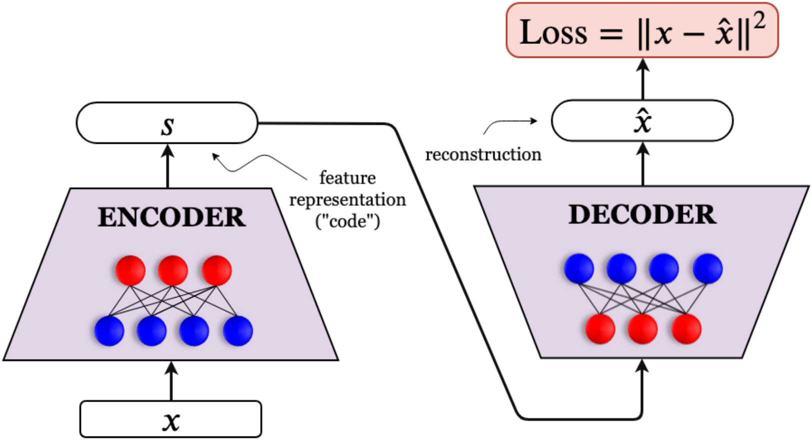
Deep Learning

Unsupervised learning

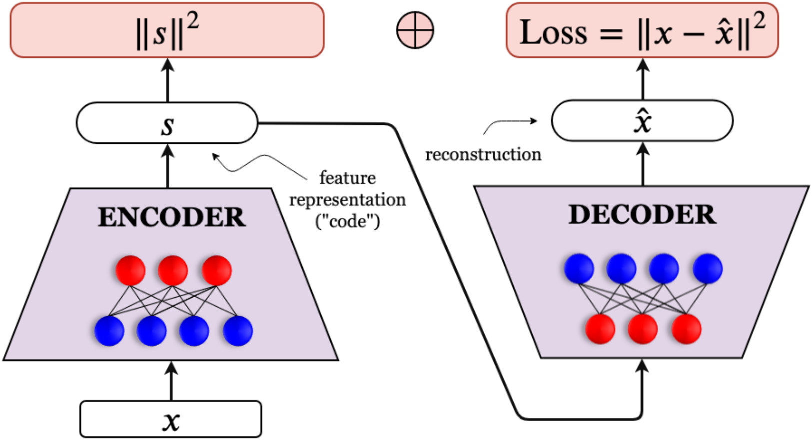
Autoencoder



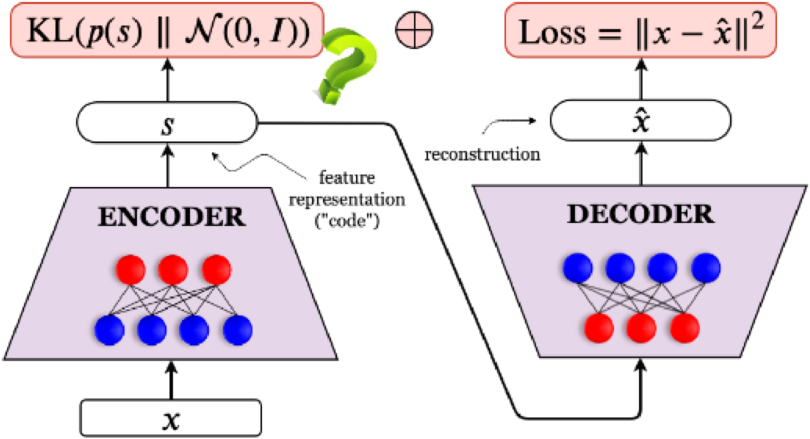
Autoencoder



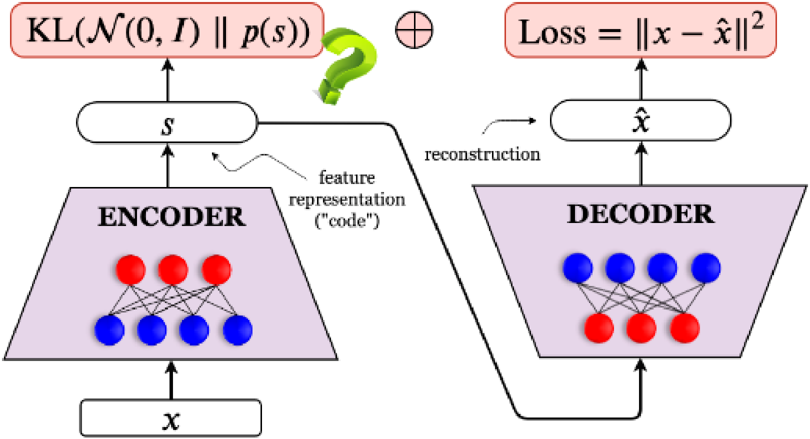
Shaping latent representation



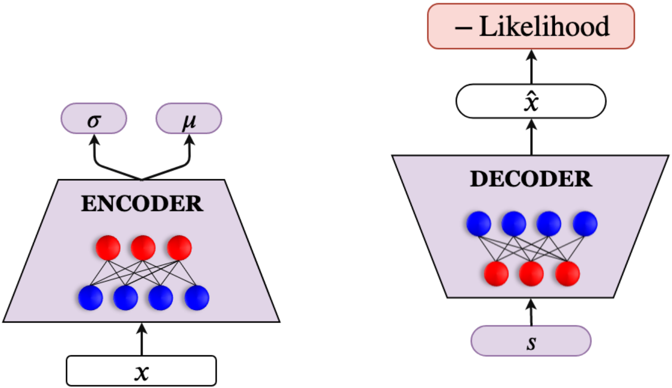
Shaping latent representation



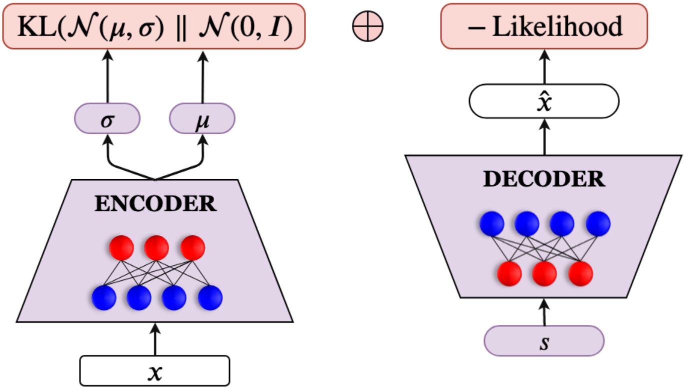
Shaping latent representation



VAE

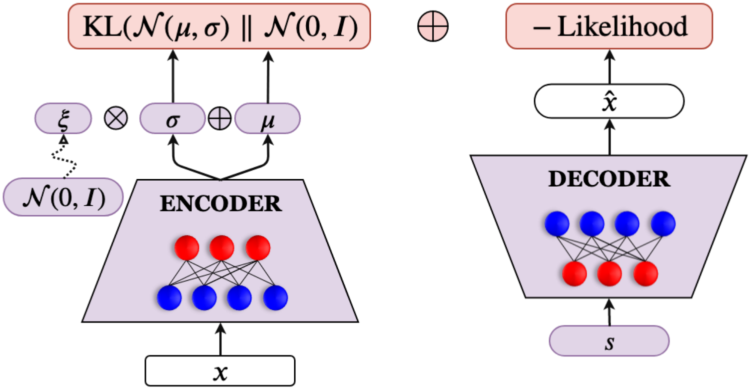


VAE



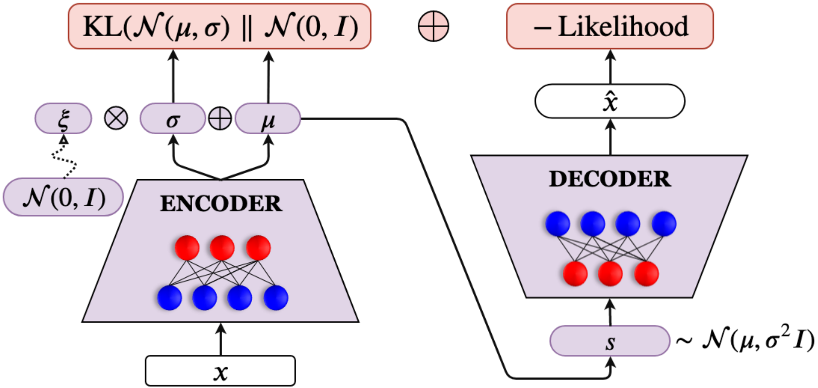
$\mathrm{KL}(\mathcal{N}(\mu, \sigma) \,||\, \mathcal{N}(0, I))$
$\oplus$
$-$Likelihood
σ
μ
ENCODER
x
$\hat{x}$
DECODER
s

VAE

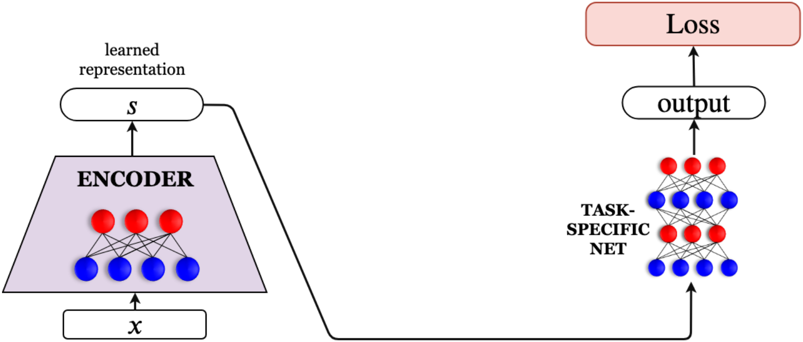


$\mathrm{KL}(\mathcal{N}(\mu, \sigma) \,||\, \mathcal{N}(0, I))$
$\oplus$
$-$ Likelihood
ξ
$\otimes$
σ
$\oplus$
μ
$\hat{x}$
$\mathcal{N}(0, I)$
ENCODER
DECODER
x
s

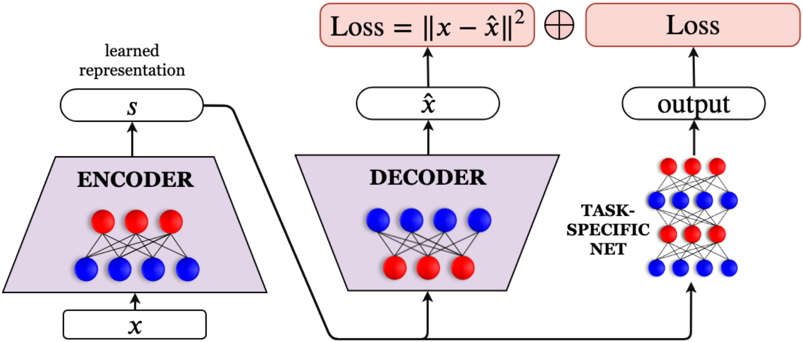
VAE


KL($\mathcal{N}(\mu, \sigma)$ || $\mathcal{N}(0, I)$)
$\oplus$
– Likelihood
$\hat{x}$
ξ
$\otimes$
σ
$\oplus$
μ
$\mathcal{N}(0, I)$
ENCODER
DECODER
x
s
$\sim \mathcal{N}(\mu, \sigma^2 I)$

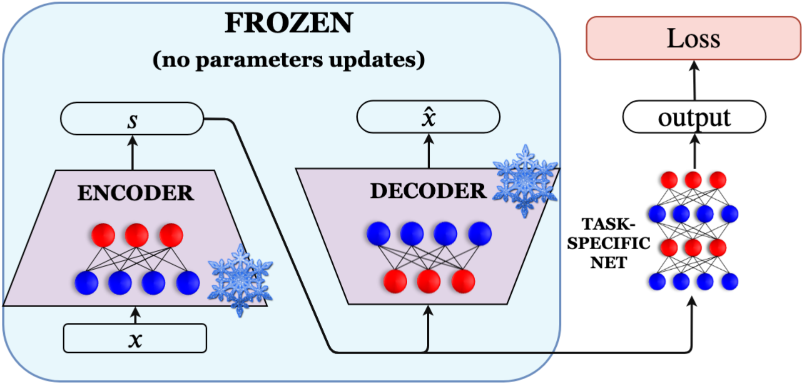
Possible usage



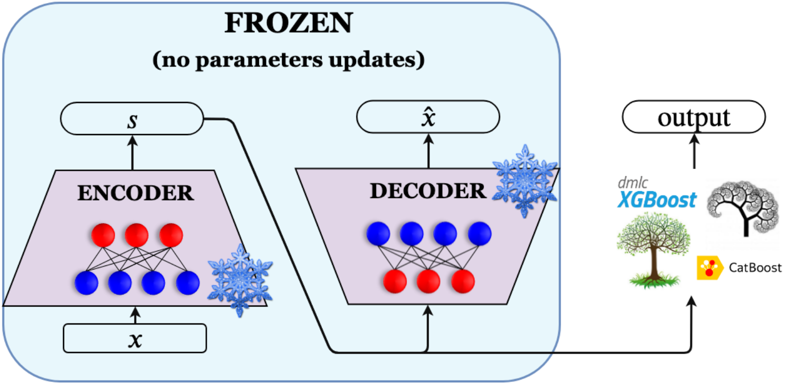
Possible usage



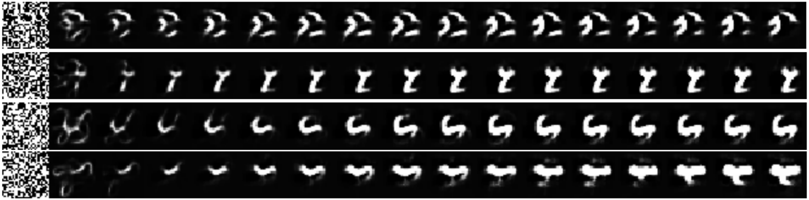
Transfer learning



Transfer learning

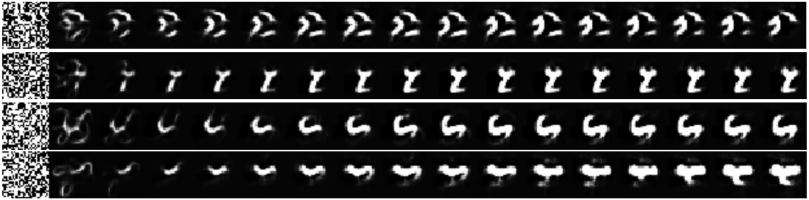


Example: digits that are not[1]



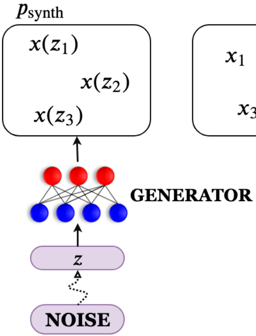
[1] https://arxiv.org/abs/1606.04345

Example: digits that are not[1]

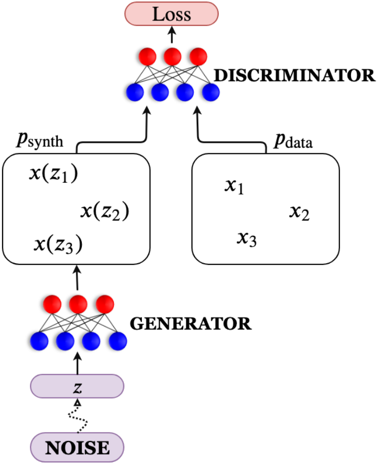


[1] https://arxiv.org/abs/1606.04345

Generative Adversarial Networks (GAN)



Generative Adversarial Networks (GAN)



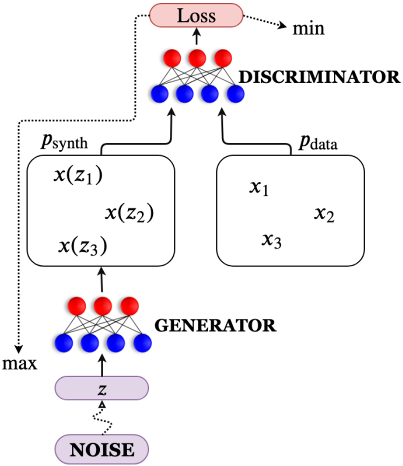
Training discriminator D:

$$\text{Loss}(D, G) :=$$

$$-\mathbb{E}_{x \sim p_{\text{real}}} \log D(x) -$$

$$-\mathbb{E}_{x \sim p_{\text{synth}}} \log(1 - D(x)) \to \min_D$$

Generative Adversarial Networks (GAN)



Training discriminator D:

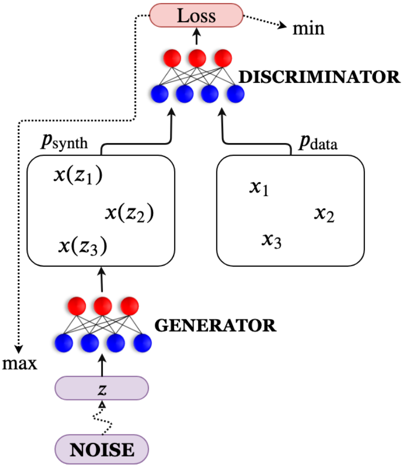
$$\text{Loss}(D, G) :=$$

$$-\mathbb{E}_{x\sim p_{\text{real}}} \log D(x) -$$
$$-\mathbb{E}_{x\sim p_{\text{synth}}} \log(1 - D(x)) \to \min_D$$

Training generator G:

$$\text{Loss}(D, G) \to \max_G$$

Generative Adversarial Networks (GAN)



Training discriminator D:

$$\text{Loss}(D, G) :=$$

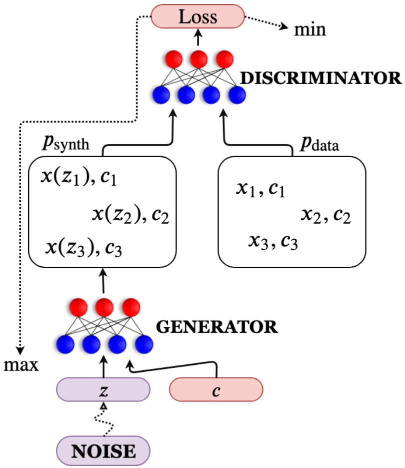
$$-\mathbb{E}_{x \sim p_{\text{real}}} \log D(x) -$$

$$-\mathbb{E}_{x \sim p_{\text{synth}}} \log(1 - D(x)) \to \min_D$$

Training generator G:

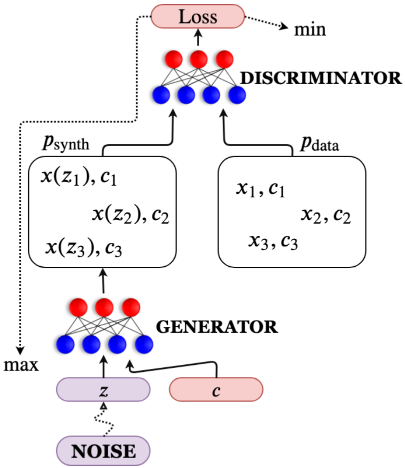
$$\text{Loss}(D, G) \to \max_G$$

Conditional GAN (cGAN)



Train $p_{\text{synth}}(x \mid c)$
to imitate $p_{\text{data}}(x \mid c)$!

Conditional GAN (cGAN)

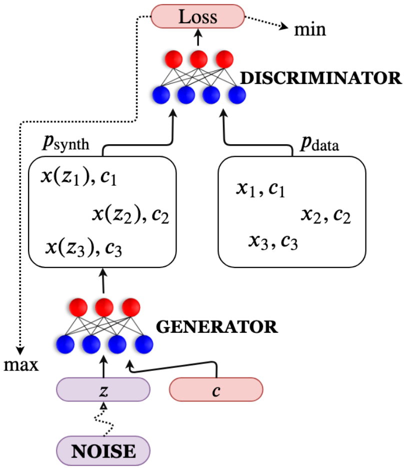


Train $p_{\text{synth}}(x \mid c)$
to imitate $p_{\text{data}}(x \mid c)$!

$$\mathbb{E}_{c \sim p(c)} \operatorname{Loss}(D, G, c) \to \min_D$$

$$\mathbb{E}_{c \sim p(c)} \operatorname{Loss}(D, G, c) \to \max_G$$

Conditional GAN (cGAN)



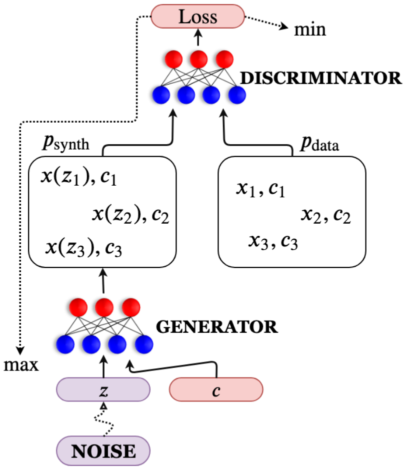
Train $p_{\text{synth}}(x \mid c)$
to imitate $p_{\text{data}}(x \mid c)$!

$$\mathbb{E}_{c \sim p(c)} \text{Loss}(D, G, c) \to \min_D$$

$$\mathbb{E}_{c \sim p(c)} \text{Loss}(D, G, c) \to \max_G$$

✓ condition can be of any complexity!

Conditional GAN (cGAN)



Train $p_{\text{synth}}(x \mid c)$
to imitate $p_{\text{data}}(x \mid c)$!

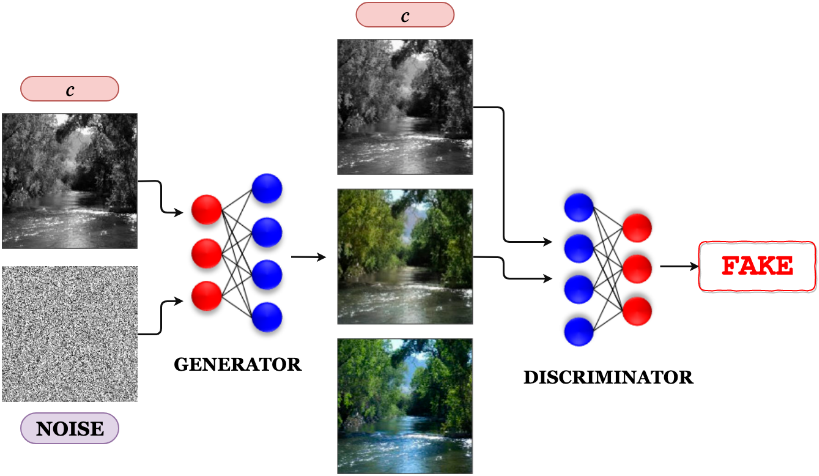
$$\mathbb{E}_{c \sim p(c)} \operatorname{Loss}(D, G, c) \to \min_D$$

$$\mathbb{E}_{c \sim p(c)} \operatorname{Loss}(D, G, c) \to \max_G$$

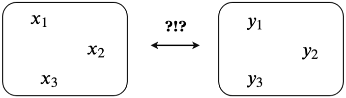
- ✓ condition can be of any complexity!
- ✓ can be viewed as **loss function learning** when output is complex

cGAN: Example

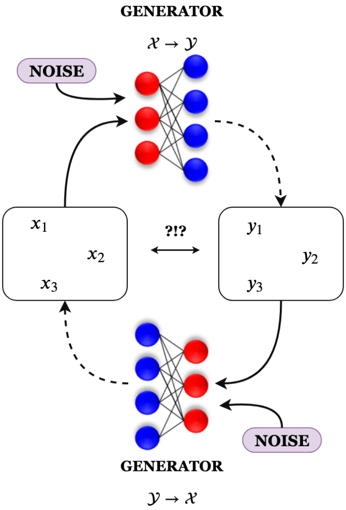
cGAN: Example



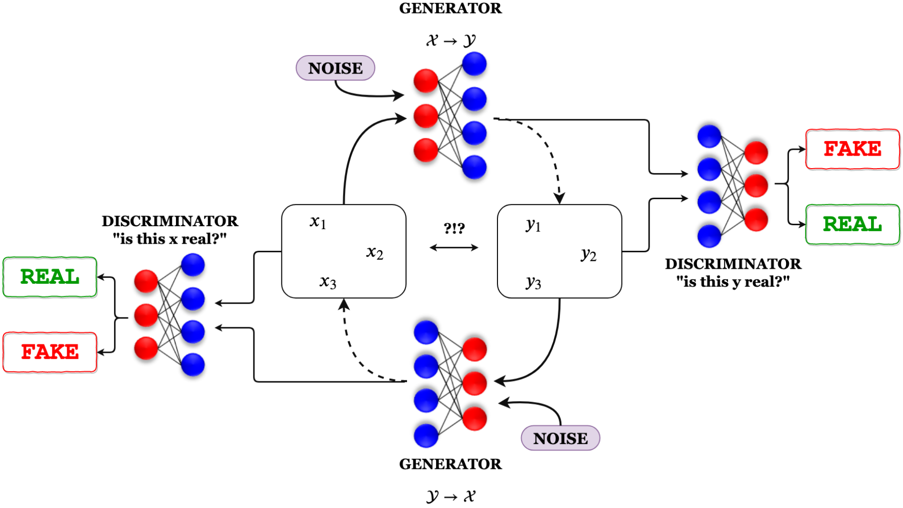
Unpaired learning



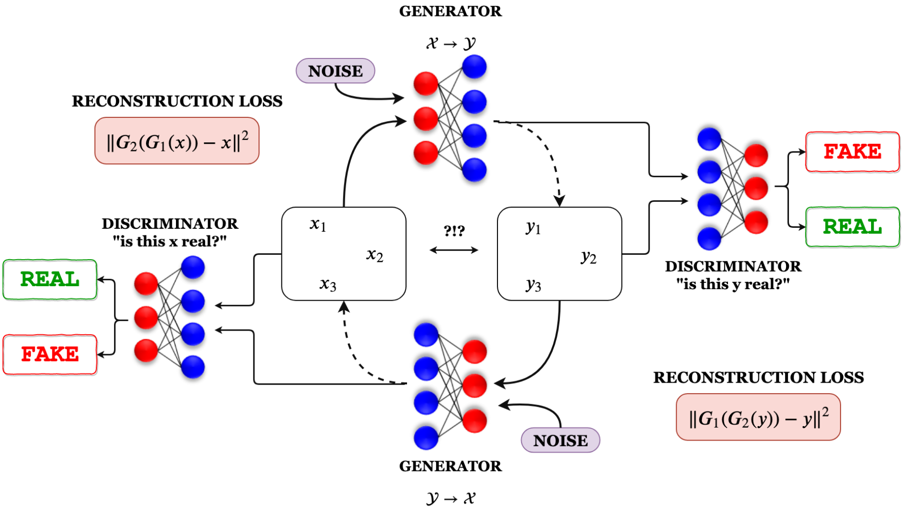
Unpaired learning



Unpaired learning



Unpaired learning



CycleGAN: Example