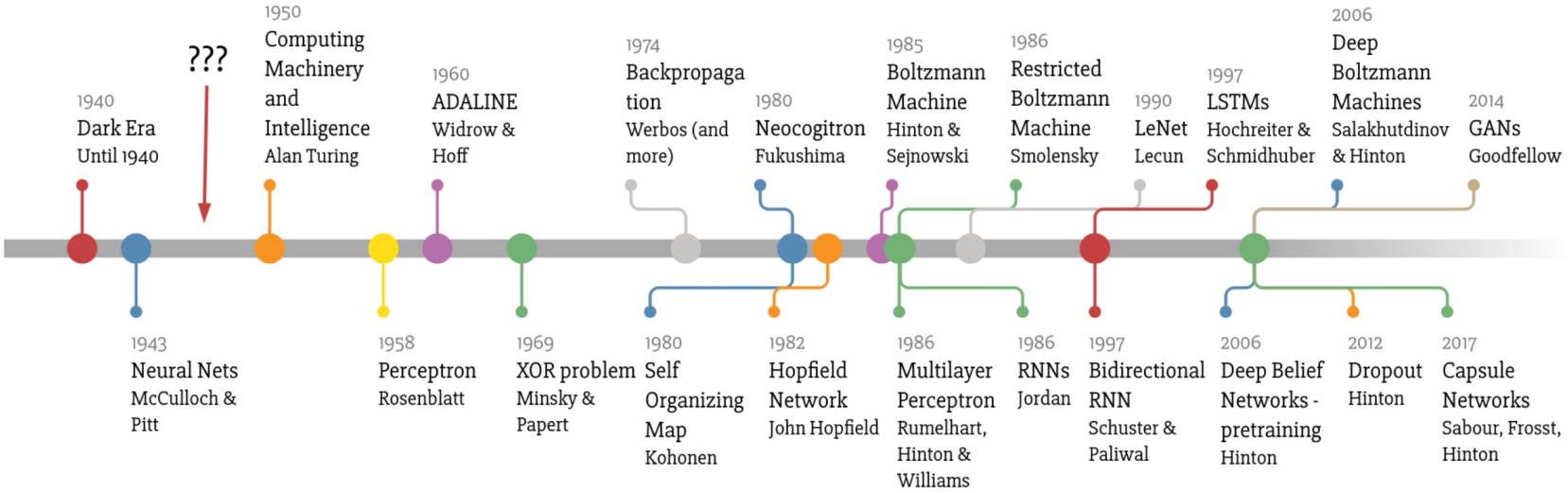
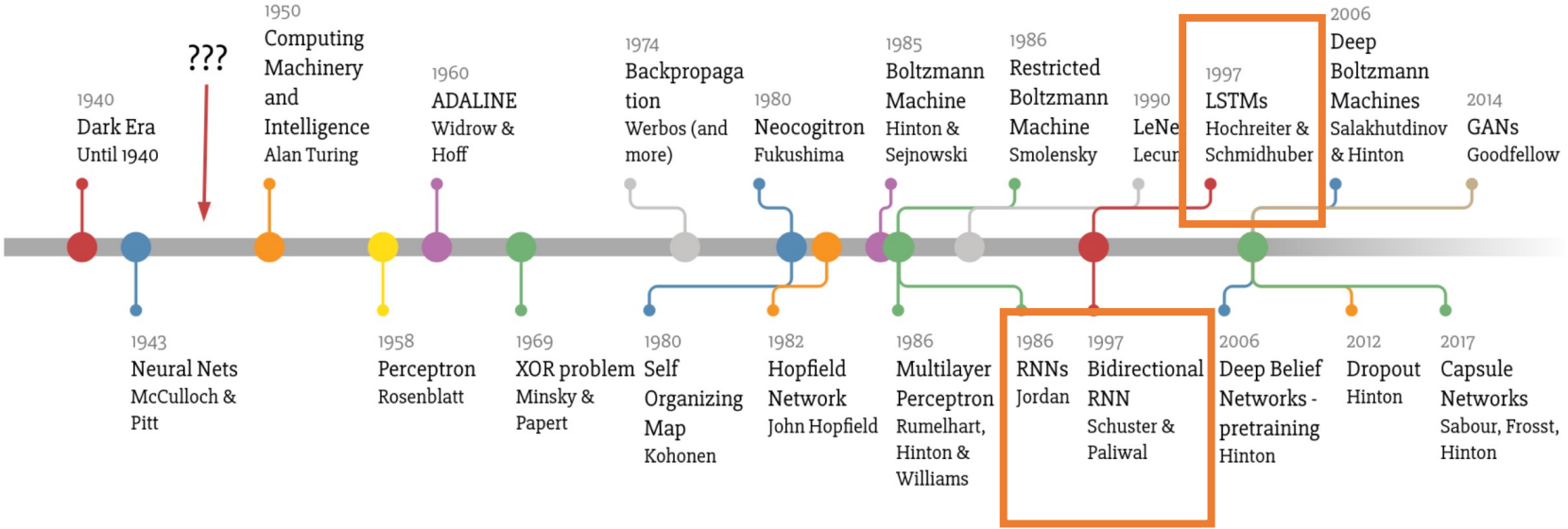
Deep Learning Timeline

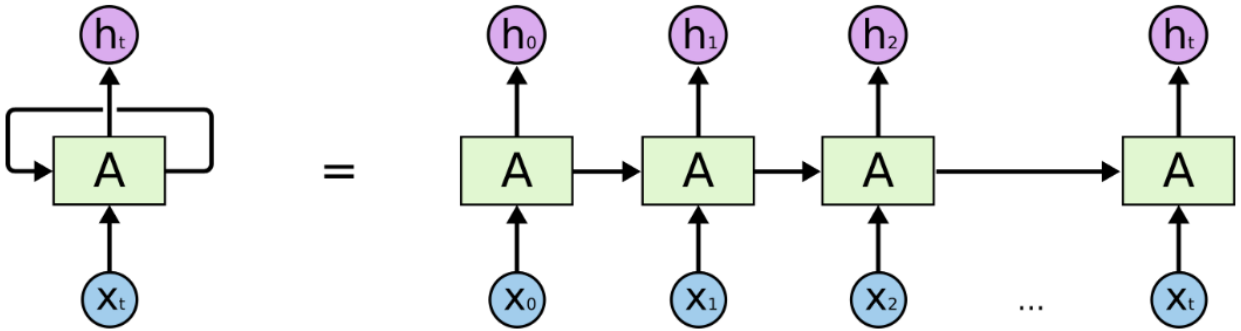


Deep Learning Timeline



Recurrent Neural Networks

A recurrent neural network (RNN) is a class of artificial neural network where connections between nodes form a directed graph along a sequence

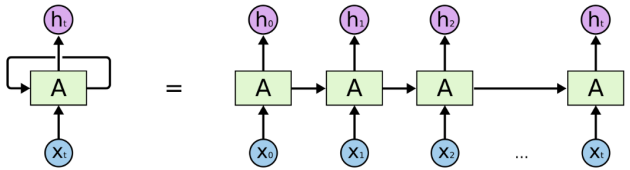


An unrolled recurrent neural network.


One to One
One to Many
Many to One
Many to Many
Many to Many
Image Classification
Image Captioning
Sentiment Analysis
Video classification
Machine Translation
Thanks for a great Party, enjoyed it!.
Positive
Emploi super in French translation of Awesome Job

Recurrent Neural Networks

```
import torch.nn as nn

class RNN(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(RNN, self).__init__()

        self.hidden_size = hidden_size

        self.i2h = nn.Linear(input_size + hidden_size, hidden_size
        self.i2o = nn.Linear(input_size + hidden_size, output_size
        self.softmax = nn.LogSoftmax(dim=1)

    def forward(self, input, hidden):
        combined = torch.cat((input, hidden), 1)
        hidden = self.i2h(combined)
        output = self.i2o(combined)
        output = self.softmax(output)
        return output, hidden

    def initHidden(self):
        return torch.zeros(1, self.hidden_size)

n_hidden = 128
rnn = RNN(n_letters, n_hidden, n_categories)
```



An unrolled recurrent neural network.

CLASS `torch.nn.Linear(in_features, out_features, bias=True)`

Applies a linear transformation to the incoming data: $y = xA^T + b$

Parameters

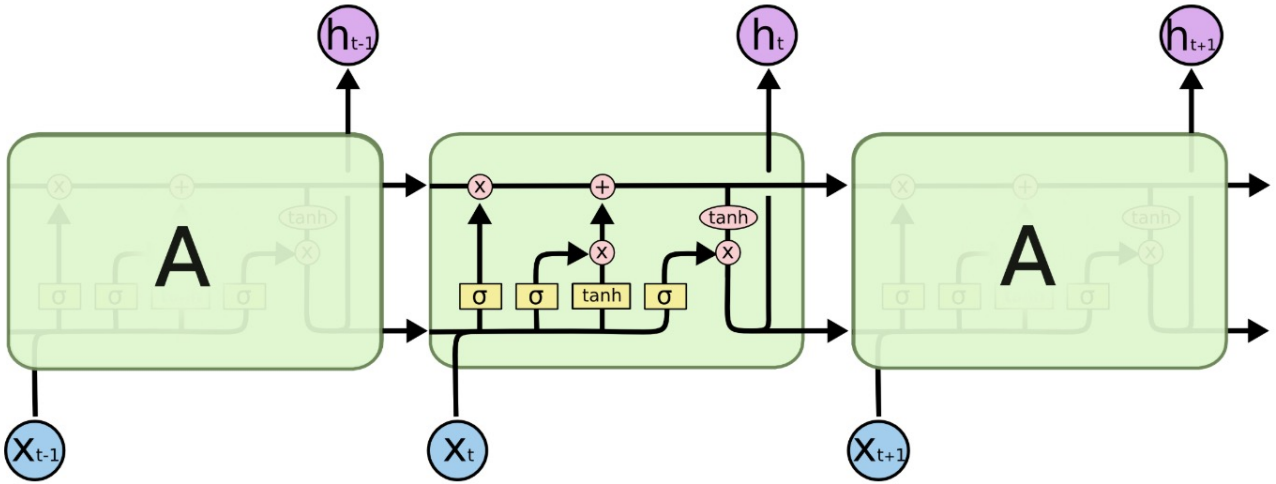
- **in_features** – size of each input sample
- **out_features** – size of each output sample
- **bias** – If set to `False`, the layer will not learn an additive bias. Default: `True`

$$Y_{n\times o} = X_{n\times i}W_{i\times o} + b$$

Recurrent Neural Networks

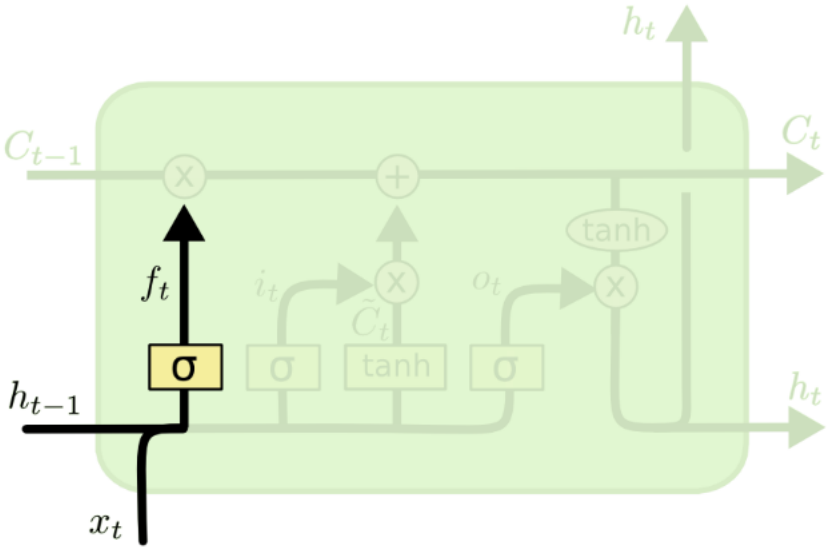
↓ The Problem of Long-Term Dependencies

Long Short Term Memory networks



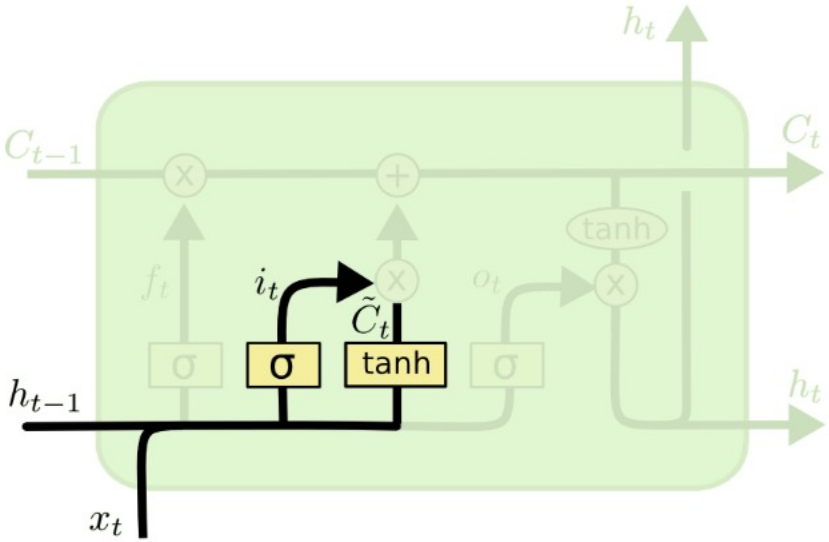
The repeating module in an LSTM contains four interacting layers.

X_0 X_1 X_2 ... video frame



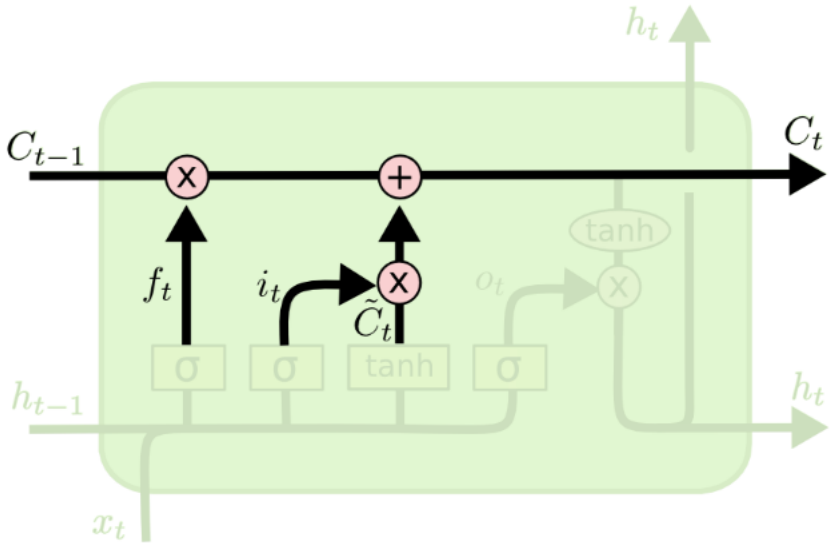
forget ;]

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$



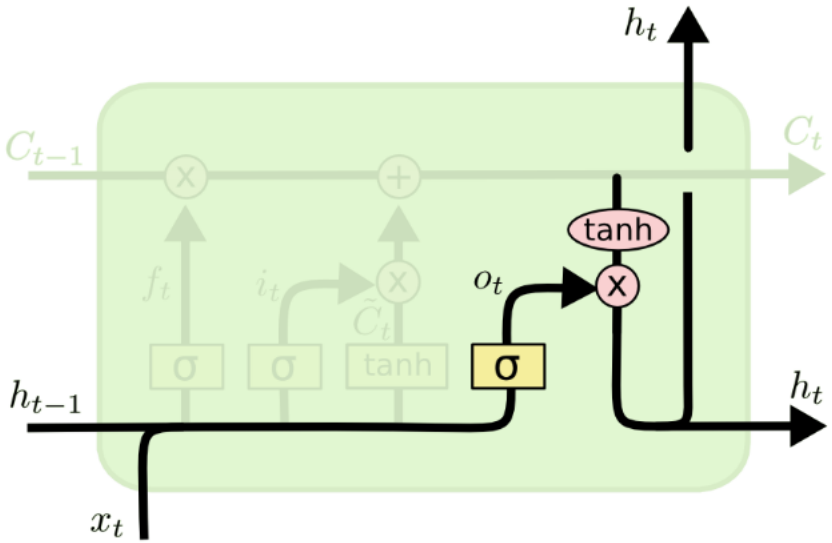
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$



长期内部存储器

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$



output

$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

hidden 更新

A boy sits on the grass

Translate: 小男孩 =

output 决定 hidden

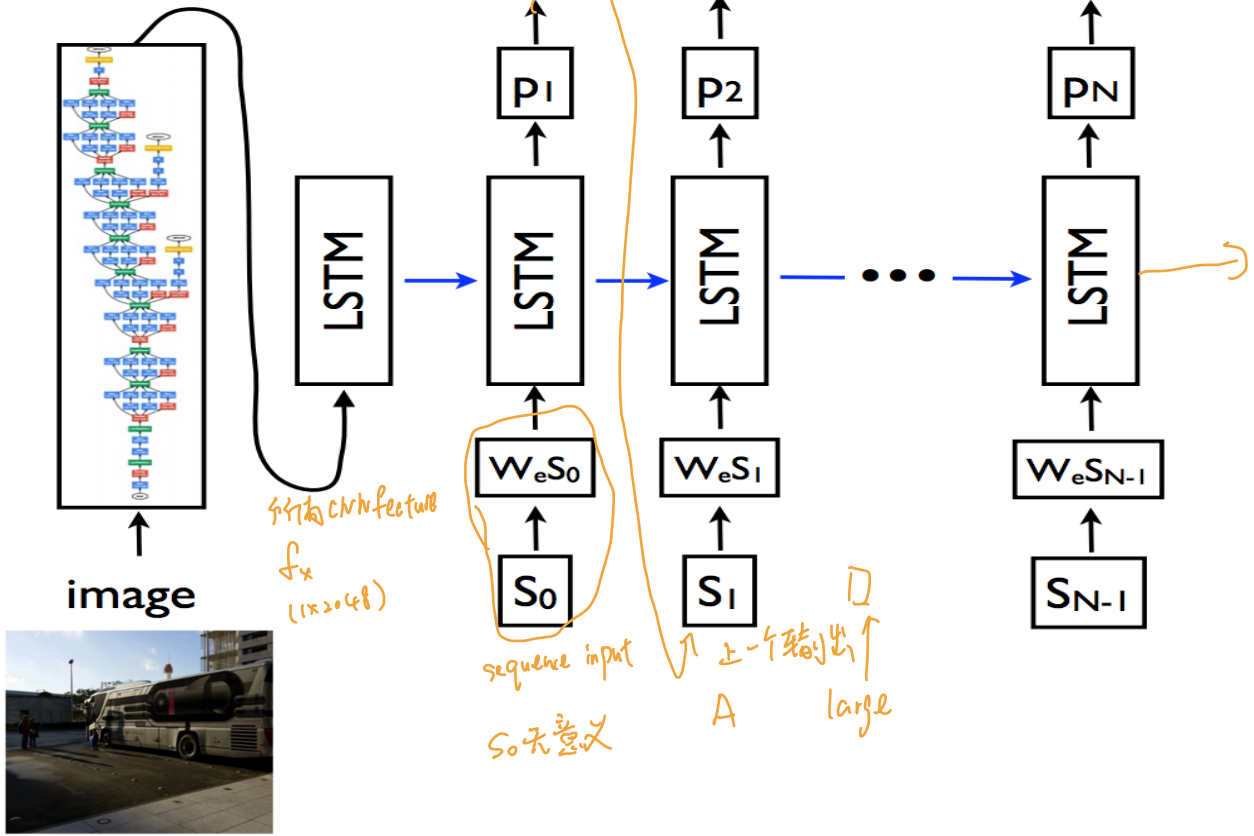
(有些已经输出过)

Image caption framework

output: A large bus sitting next to a very tall building. 描述图

提feature

One-to-many

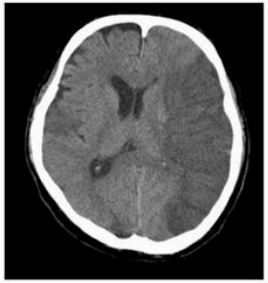


Case Study : LSTM for medical image

0) a man riding a wave on top of a surfboard. (p=0.036508)
1) a person riding a surf board on a wave. (p=0.021727)
2) a man riding a wave on a surfboard in the ocean. (p=0.004277)

0) a man riding a skateboard up the side of a ramp . (p=0.008467)
1) a man riding a skateboard on a ramp . (p=0.001182)
2) a man riding a skateboard up the side of a cement ramp . (p=0.000948)

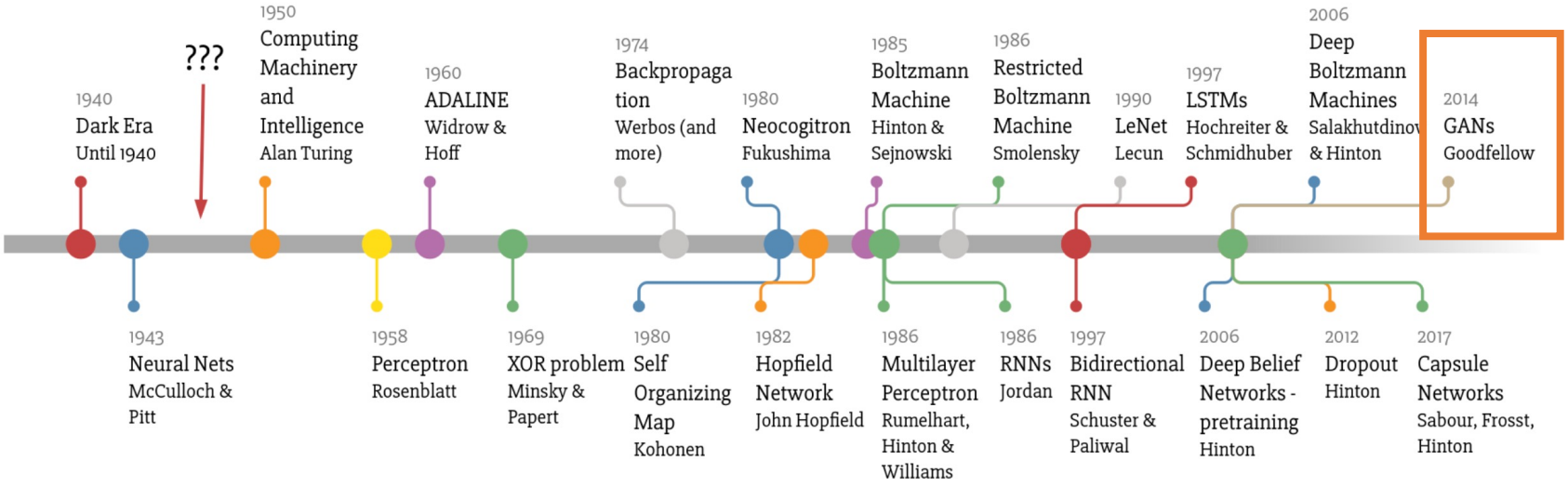
0) a ct showing a acute subdural hemorrhage with midline shift. (p=0.001608)
1) a ct showing a acute hemorrhage with left to right midline shift. (p=0.004277)
2) a ct showing a subdural hemorrhage with ventricular effacement. (p=0.021727)



0) a frontotemporal infarction with mild ventricular effacement. (p0.001608)
1) a left frontotemporal infarction with mild left ventricular effacement. (p=0.004112)
2) a left fronto temporal infarction with ventricular effacement. (p=0.012928)

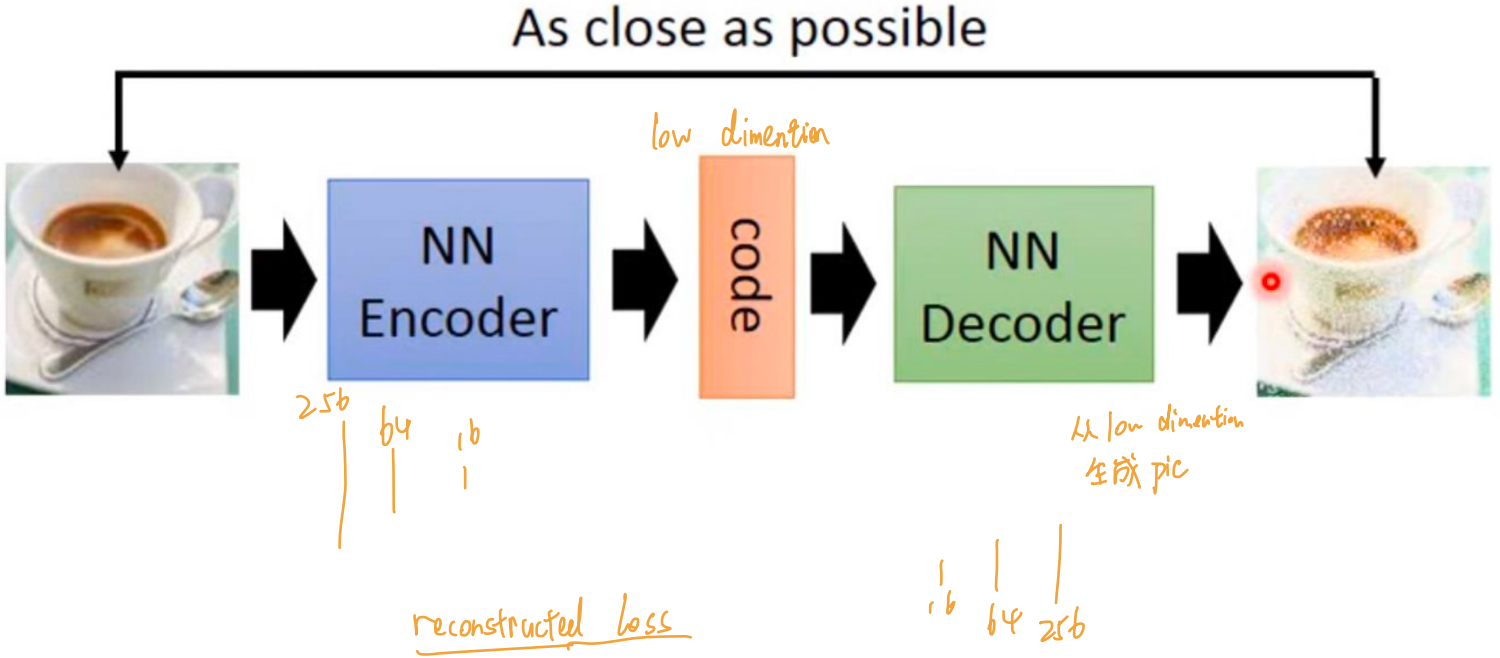
Feng, Rui, et al. "Deep learning guided stroke management: a review of clinical applications." *Journal of neurointerventional surgery* 10.4 (2018): 358-362.

Deep Learning Timeline

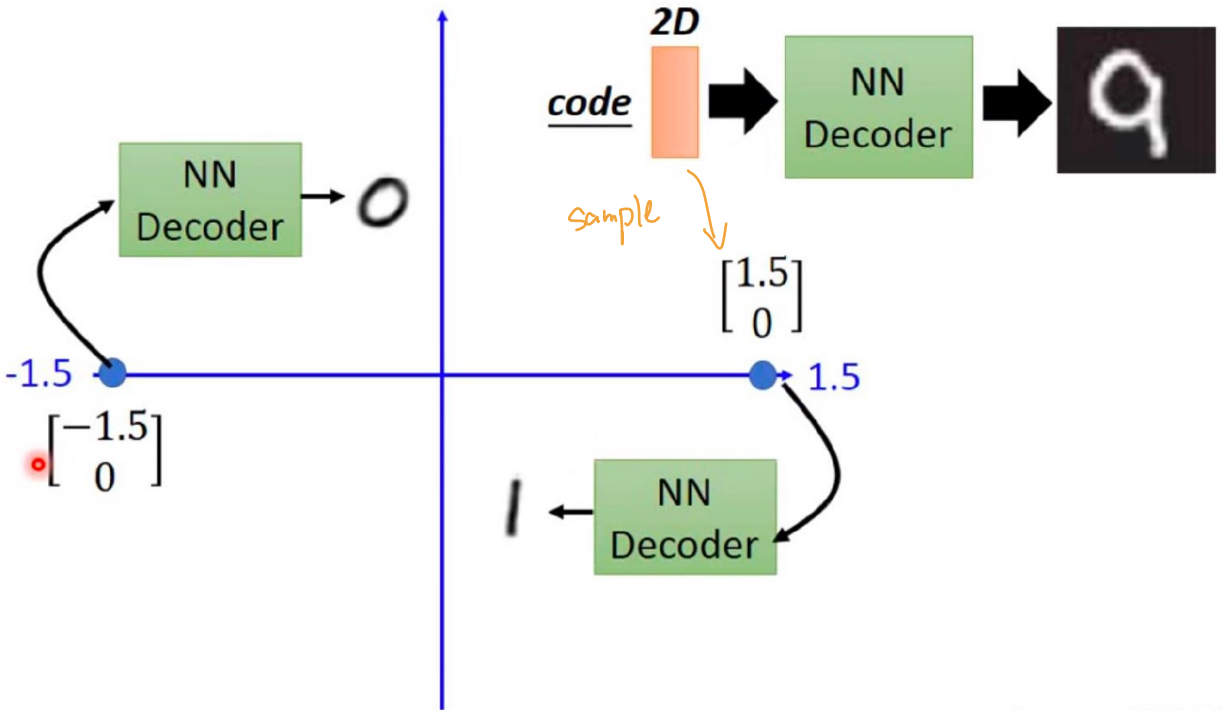


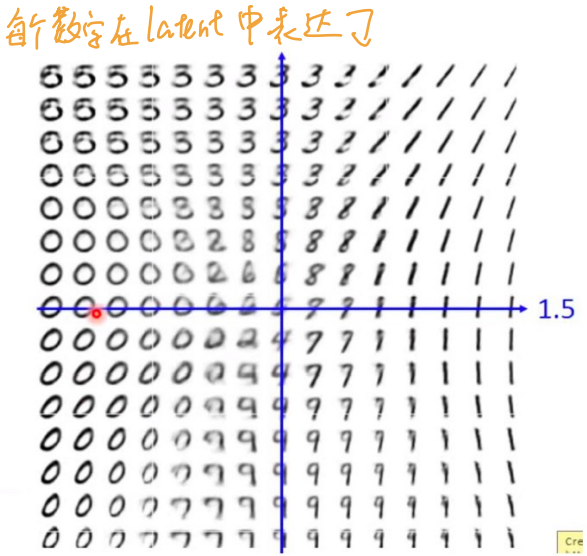
Review: Auto-encoder

∈ unsupervised learning



Review: Auto-encoder

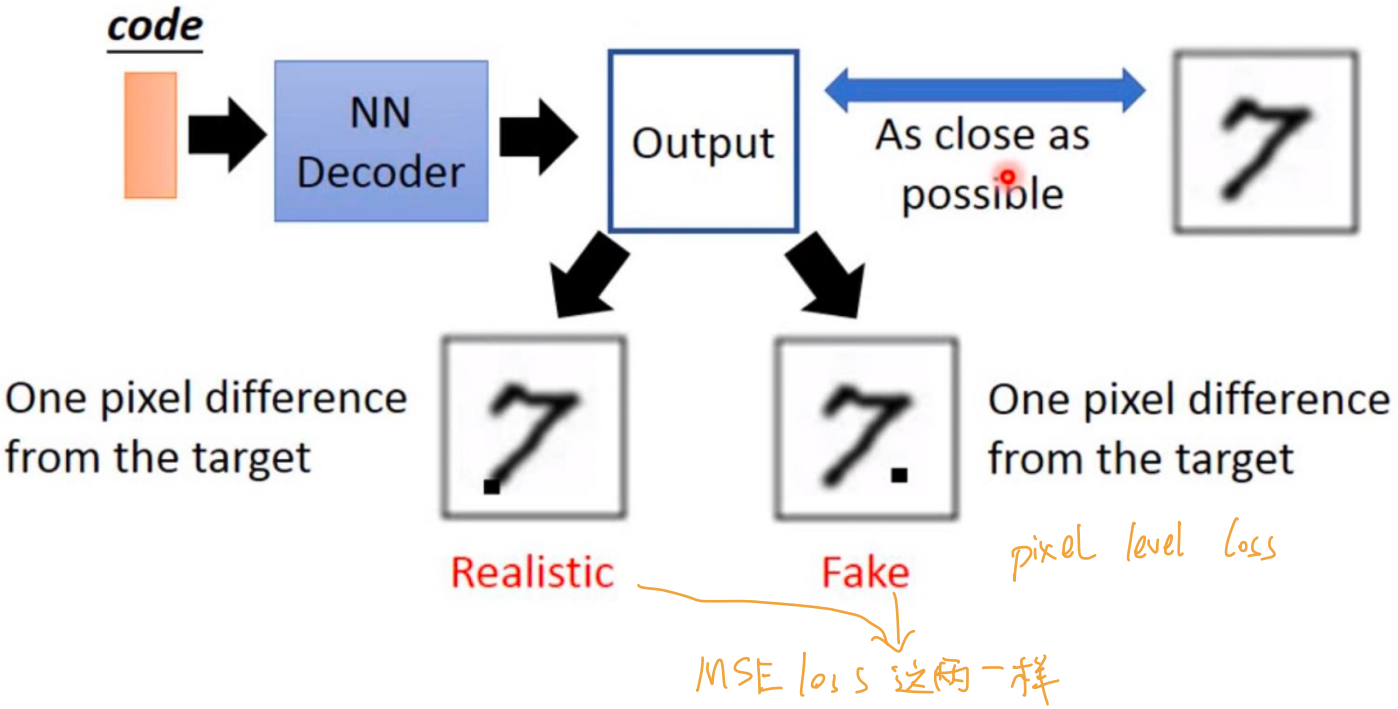




Problems of VAE

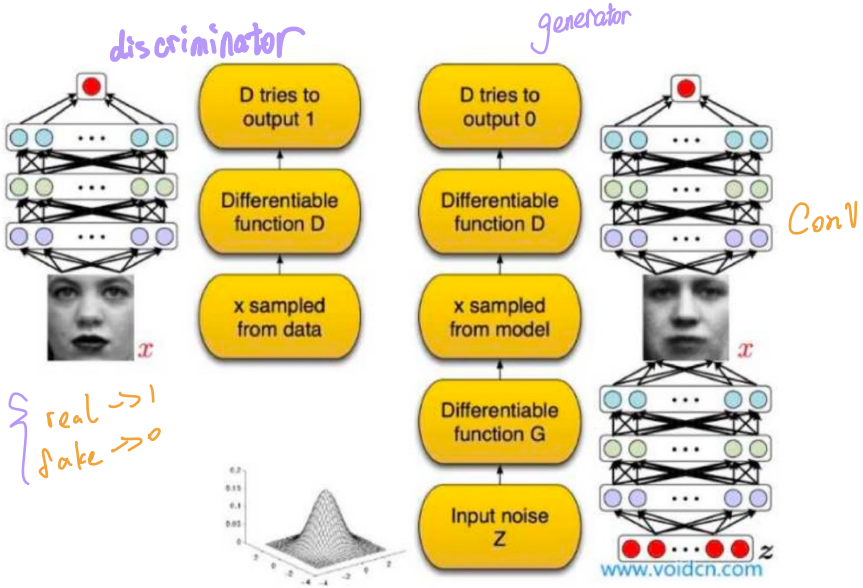
variational

- It does not really try to simulate real images



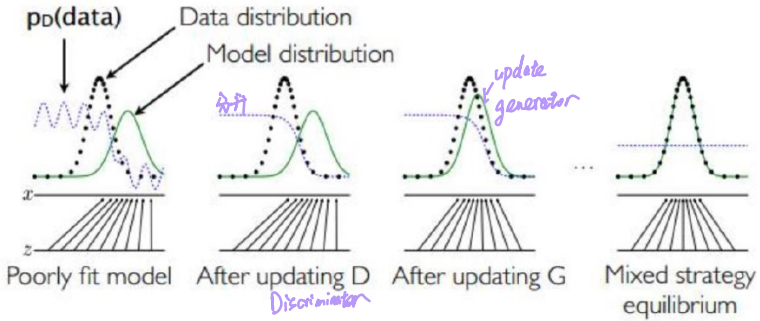
Generative Adversarial Nets

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_z(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))].$$



$$\min_G V(D,G) = E_{z\sim p_z(z)}\left[log(1 - D(G(z)))\right]$$

generator



$$p_g(x) = p_{data}(x)$$

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**
  for k steps **do**
  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
  - Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)\right].$$

  end for
  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Update the generator by descending its stochastic gradient:

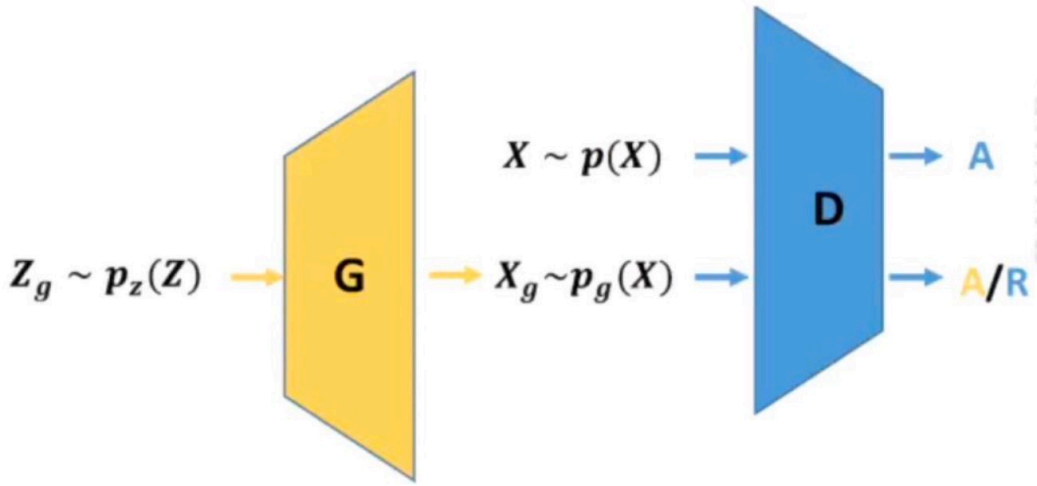
$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

GANs (Goodfellow et al. [2014])

- Given p(x), learn $p_g(x) \approx p(x)$
- A two-player game



The global optima is $p_g(x) = p(x)$.

The Two-player Formulation is Restricted

- *G* and *C* may not be optimal at the same time. Here is an example of good *C* with poor *G* (Salimans et al. [2016]).

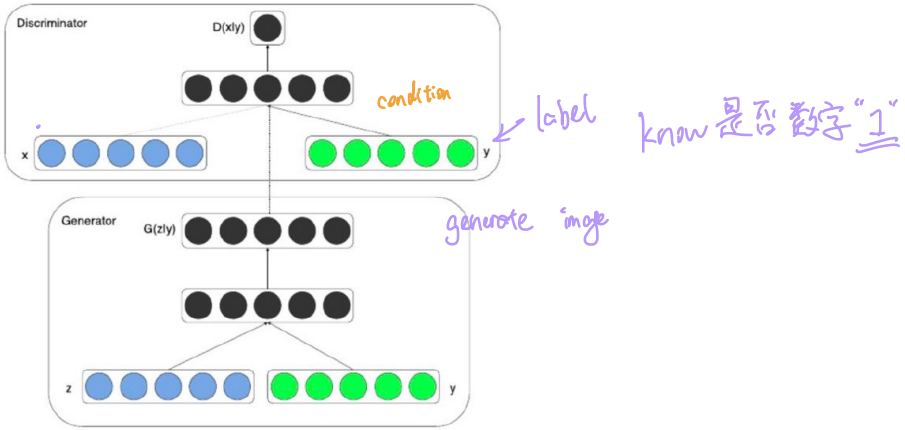
- *G* cannot control the semantics of the generated samples.

Generative Adversarial Nets

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_z(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))].$$

Figure 2: Generated MNIST digits, each row conditioned on one label

Conditional Generative Adversarial Nets



know 是否数字"1"

generate image

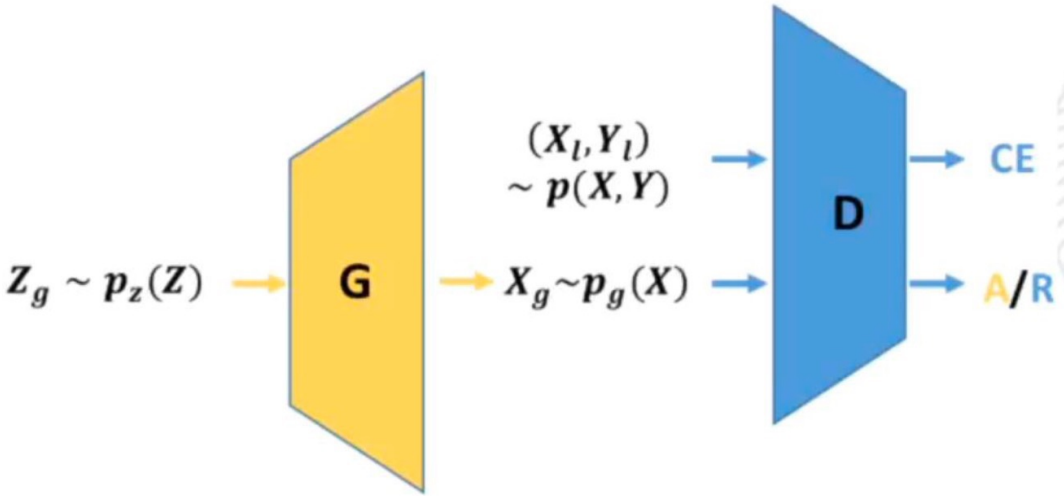
$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x}|\boldsymbol{y})] + \mathbb{E}_{\boldsymbol{z}\sim p_z(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z}|\boldsymbol{y})))].$$

	User tags + annotations	Generated tags
	montanha, trem, inverno, frio, people, male, plant life, tree, structures, transport, car	taxi, passenger, line, transportation, railway station, passengers, railways, signals, rail, rails
	food, raspberry, delicious, homemade	chicken, fattening, cooked, peanut, cream, cookie, house made, bread, biscuit, bakes
	water, river	creek, lake, along, near, river, rocky, treeline, valley, woods, waters
	people, portrait, female, baby, indoor	love, people, posing, girl, young, strangers, pretty, women, happy, life

Table 2: Samples of generated tags

SSL-GANs (Springenberg [2015], Salimans et al. [2016])

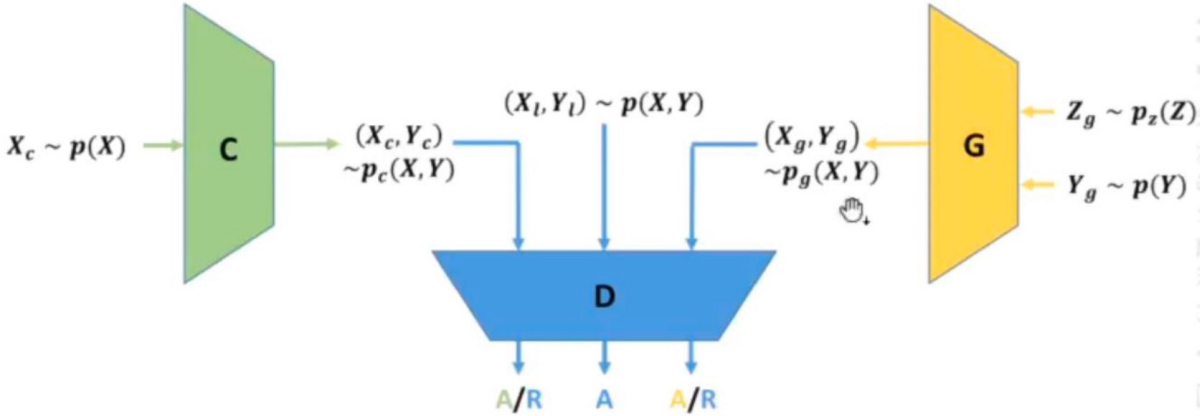
- Given partially labeled data, learn $p_c(y|x)$ and $p_g(x)$
- A two-player game with parameter-sharing



where $D(x)$ outputs units of dimension $|\mathcal{Y}|$ or $|\mathcal{Y}| + 1$.

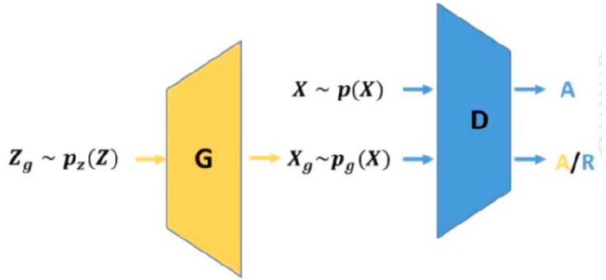
Three-player Game

- Given partially labeled data, learn $p_c(y|x)$ and $p_g(x|y)$

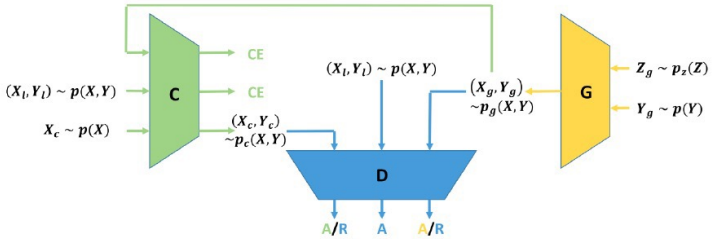


where $D(x)$ outputs a binary unit to justify (x, y) pairs. We call it Triple-GAN (three players and three distributions of interests).

Triple Generative Adversarial Nets, NIPS2017



$$\min_G \max_D U(D,G) = E_{x\sim p(x)}[\log(D(x))] + E_{z\sim p_z(z)}[\log(1-D(G(z)))],$$



$$\min_{C,G} \max_D U(C,G,D) = E_{(x,y)\sim p(x,y)}[\log D(x,y)] + \alpha E_{(x,y)\sim p_c(x,y)}[\log(1-D(x,y))] + (1-\alpha)E_{(x,y)\sim p_g(x,y)}[\log(1-D(G(y,z),y))],$$

Add supervised loss (cross-entropy loss) to C

$$\mathcal{R}_{\mathcal{L}} = E_{(x,y)\sim p(x,y)}[-\log p_c(y|x)]$$

$$\min_{C,G} \max_D \tilde{U}(C,G,D) = E_{(x,y)\sim p(x,y)}[\log D(x,y)] + \alpha E_{(x,y)\sim p_c(x,y)}[\log(1-D(x,y))] + (1-\alpha)E_{(x,y)\sim p_g(x,y)}[\log(1-D(G(y,z),y))] + \mathcal{R}_{\mathcal{L}}.$$

Good *C* Results in Good *G*

(a) Automobile

(b) Horse

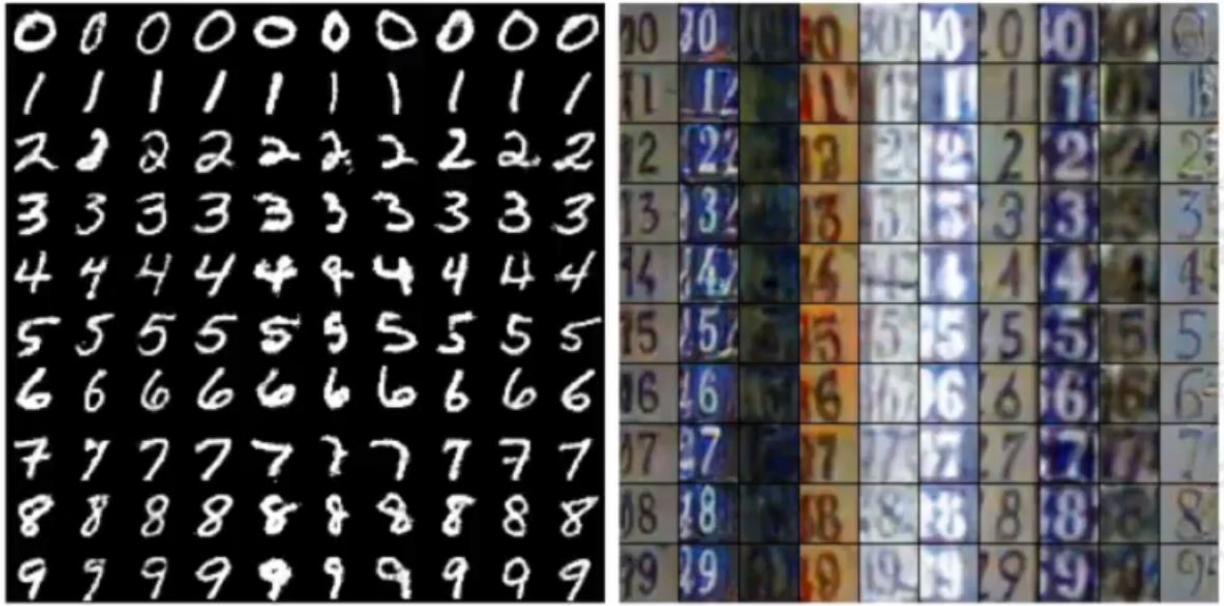


Figure: Same y for each row. Same z for each column.

style

Home work: python + numpy

https://cs231n.github.io/python-numpy-tutorial/