

Introduction to Deep Learning for Scientists and Engineers Part-II

Abhijat Vatsyayan ¹

October 13, 2020

Summary

- 1 Introduction and plan
- 2 Image processing
- 3 Representation in latent space
- 4 Autoencoders
- 5 Recurrent networks and sequences
- 6 Miscellaneous topics
- 7 References

Need for different kind of functions

Discussed previously

- Simple, linear layers can be connected together to form deep networks.
- Linear layers should be separated using non-linear functions (layers) - also referred to as activations, e.g., $RelU(x)$, $\sigma(x)$.
- Mathematically, learning is possible. In reality, people struggled to make deep networks learn.
 - Vanishing gradients
 - Compute capacity
 - Availability of data

New developments

- Activation functions
- Regularization techniques (drop off, batch normalization)
- Data (Google, Facebook, ...), standard datasets and competitions
 - Data collected by internet and social media companies, digital consumer products like Cameras and Phones.
 - Dataset and benchmarks created by research labs and universities ¹
 - Competitions and conferences organized around some of the datasets and benchmarks
- CPUs, GPUs, nVidia
- *"New" functions*

¹See Russakovsky et al. 2015 for an example

Datasets

- Modified National Institute of Standards and Technology - MNIST (60k/10k)
- Canadian Institute For Advanced Research - CIFAR-10 (50k/10k) and CIFAR-100 (2 level, 500/100)
- Pascal Visual Object Classes (VOC) - 22k images, 20 classes
- ...
- ImageNet

Using linear layer

Image as 2D Tensor(Matrix)

1	2	3	4	5
6	7	8	9	10
5	4	3	2	1
10	9	8	7	6

$$2D \rightarrow 1D$$

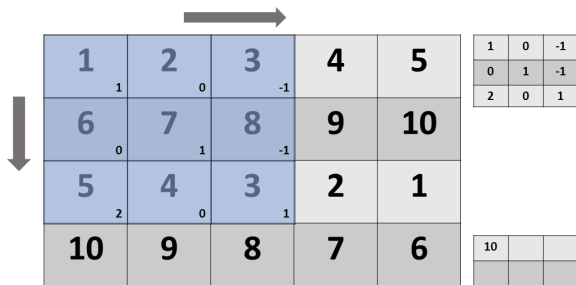
$$\mathbb{W}_{m \times 20} \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ \vdots \\ 8 \\ 7 \\ 6 \end{bmatrix} + \mathbb{B}_{m \times 1}$$

2D Convolution Example

1	2	3	4	5
6	7	8	9	10
5	4	3	2	1
10	9	8	7	6

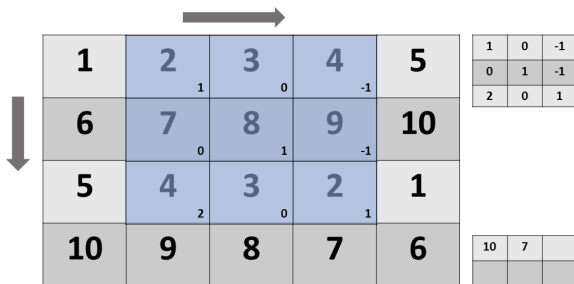
1	0	-1
0	1	-1
2	0	1

2D Convolution Example

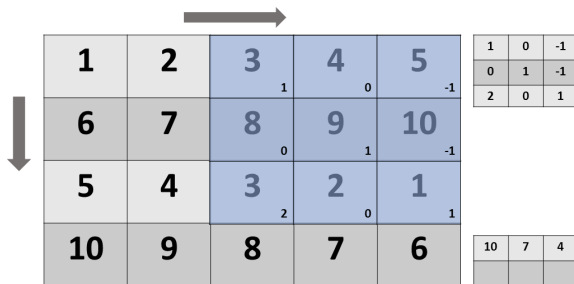


$$\begin{aligned}
 &(1 \times 1) + (2 \times 0) + (3 \times -1) + \\
 &(6 \times 0) + (7 \times 1) + (8 \times -1) + \\
 &(5 \times 2) + (4 \times 0) + (3 \times 1)
 \end{aligned}$$

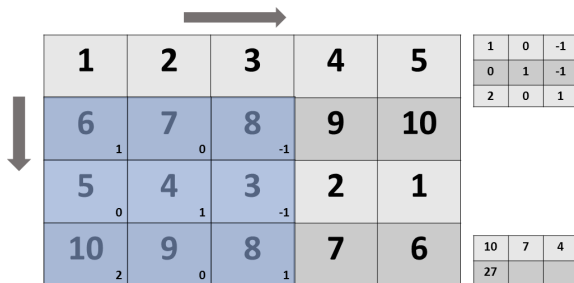
2D Convolution Example



2D Convolution Example



2D Convolution Example

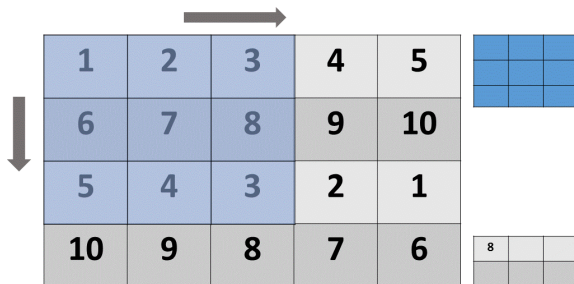


2D Convolution

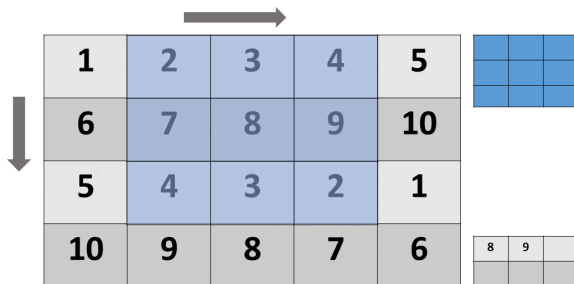
- Bias $wx + b$
- Stride
- Padding
- Layers or channels

For a 5×5 filter with bias, you need 26 parameters for gray scale images. If you have 3 channels (rgb), you need $5 \times 5 \times 3 + 1 = 76$ parameters.

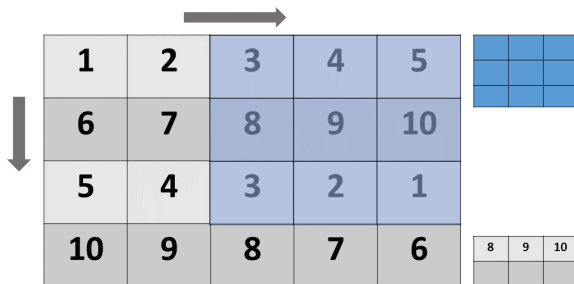
Max pooling



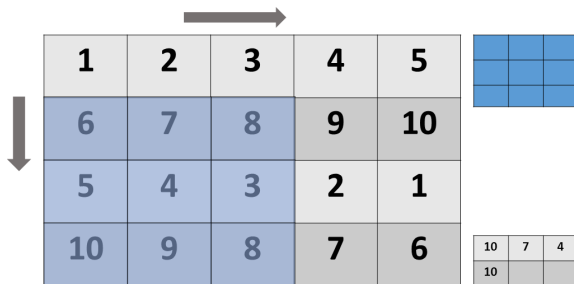
Max pooling



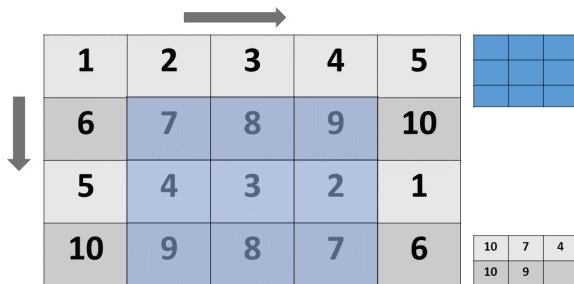
Max pooling



Max pooling



Max pooling



Max pooling

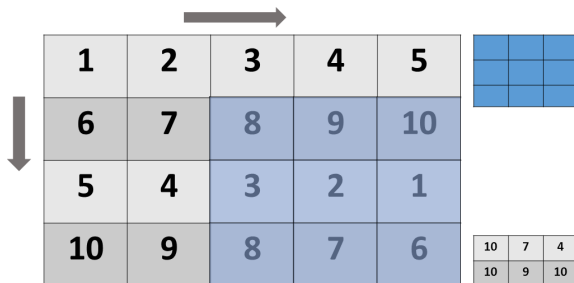
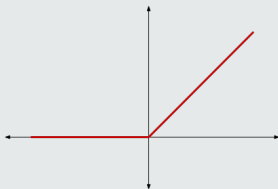


Image processing

Other functions

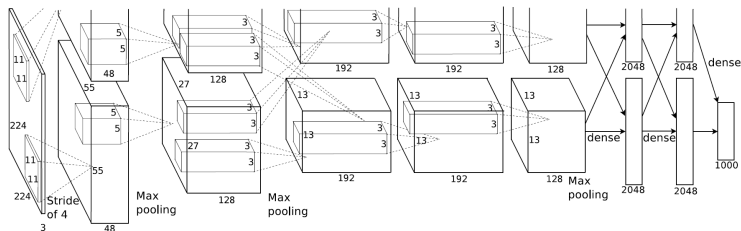
- ReLU activation $\max(x, 0)$
- Dropout layer



AlexNet 2012

- 8 Layers, 5 Convolutional, 3 fully connected
- Used ReLU and max-pooling
- 61M parameters

AlexNet 2012



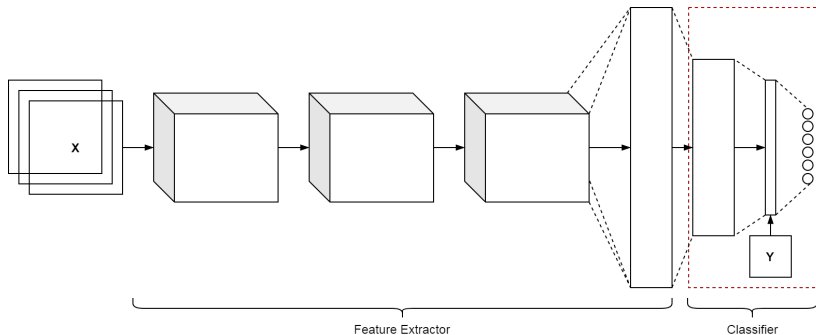
Alex Krizhevsky, Sutskever, Ilya and Hinton, Geoffrey E., "ImageNet Classification with Deep Convolutional Neural Networks", 2012

Typical convolution net, pytorch

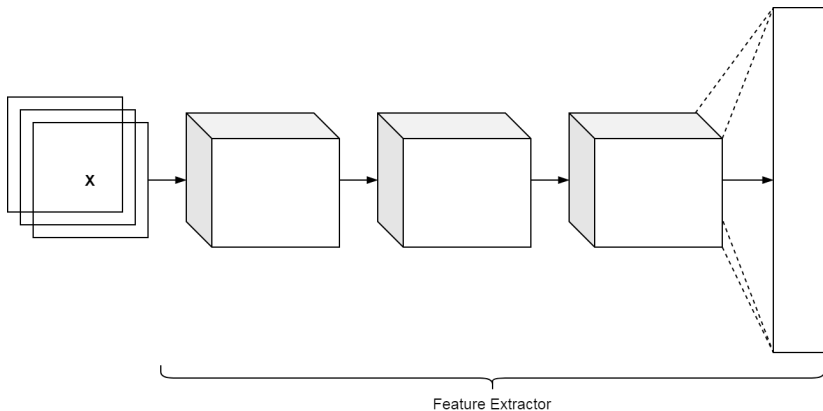
Listing 1: Typical (simple) CNN in pytorch

```
1 class ConvNet(nn.Module):
2     def __init__(self):
3         super(ConvNet, self).__init__()
4         self.conv1 = nn.Conv2d(3, 16, 3, padding=1)
5         self.lrelu1 = nn.LeakyReLU(.1)
6         self.conv2 = nn.Conv2d(16, 32, kernel_size=3, padding=1)
7         self.lrelu2 = nn.LeakyReLU(.1)
8         self.maxpool1 = nn.MaxPool2d(kernel_size=3, padding=1)
9         self.dropout1 = nn.Dropout(p=.25)
10        self.conv3 = nn.Conv2d(in_channels=32, out_channels=32, padding=1, kernel_size=3)
11        self.lrelu3 = nn.LeakyReLU(.1)
12        self.conv4 = nn.Conv2d(in_channels=32, out_channels=64, padding=1, kernel_size=3)
13        self.lrelu4 = nn.LeakyReLU(.1)
14        self.maxpool2 = nn.MaxPool2d(kernel_size=3, padding=1)
15        self.dropout2 = nn.Dropout(p=.25)
16
17        self.conv_layers = [self.conv1, self.lrelu1, self.conv2, self.lrelu2, self.maxpool1,
18                             self.conv3, self.lrelu3, self.conv4, self.lrelu4, self.maxpool2, self.dropout2]
19
20        self.fc1 = nn.Linear(in_features=64*4*4, out_features=256)
21        self.lrelu5 = nn.LeakyReLU(.1)
22        self.dropout3 = nn.Dropout(.25)
23        self.fc2 = nn.Linear(in_features=256, out_features=10)
24        self.softmax = nn.Softmax(dim=1)
```

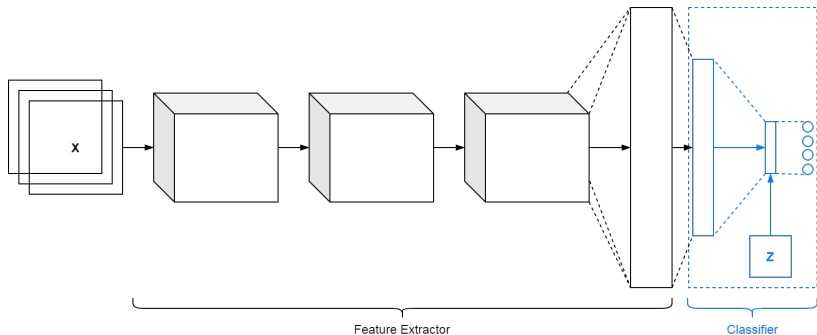
Typical (simplified) CNN



Typical (simplified) CNN - remove last layer



Typical (simplified) CNN - replace with new layer



Transfer learning - pytorch

```
2 import torch.nn as nn
  from torchvision import datasets, models, transforms
4
  model = models.resnet18(pretrained=True)
6  num_fts = model_ft.fc.in_features
8
  # Here the size of each output sample is set to 2.
  model.fc = nn.Linear(num_fts, 2)
```

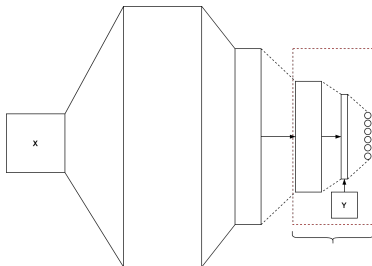
Listing 2: Using pretrained ResNet18

```
2 for param in model.parameters():
    param.requires_grad = False
4
  num_fts = model_ft.fc.in_features
  model.fc = nn.Linear(num_fts, 2)
```

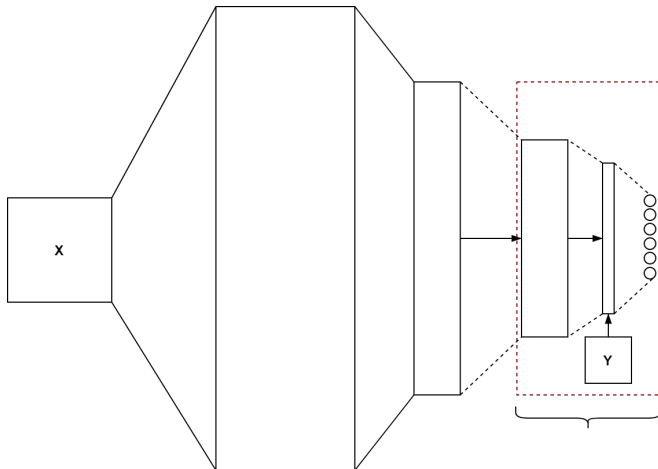
Listing 3: Freezing model parameters

Approximate identity function

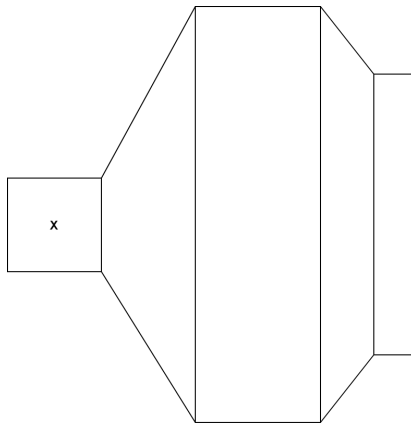
- Let us consider $y = f(x)$ as the network function.
- Let us consider another network function $y = g(x)$
- Let us consider a loss function, $L(x, g(f(x)))$. This could be
 - Mean squared distance
 - Absolute difference
 - "Some" other distance metric between vectors in "some" latent space



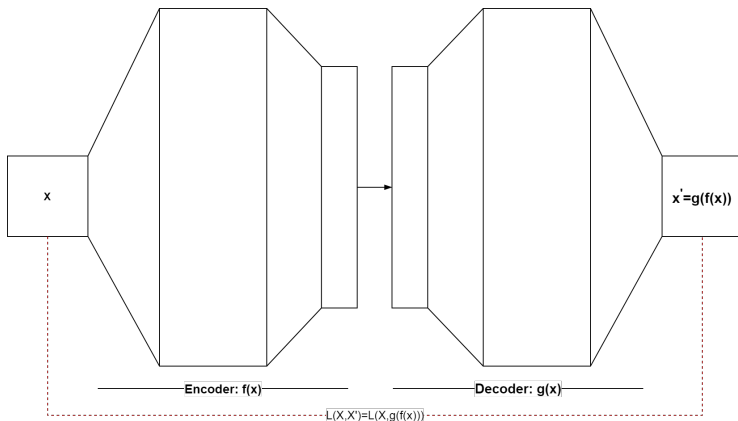
Autoencoders and identity function



Autoencoders and identity function



Autoencoders and identity function



Sequences

- Natural language tasks
- Event processing
- Statefull systems in general

Types

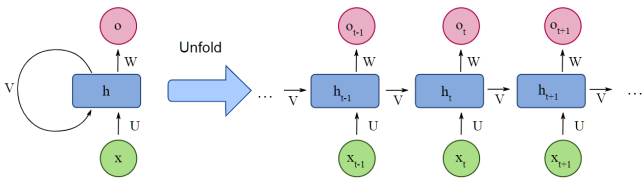
- Variable length sequences, all elements known ahead of time.
- Constant length sequences, all elements known ahead of time.
- Constant length sequences, revealed one element at a time.
- Variable length sequences, revealed one element at a time.

Functions of form $y_t = f(y_{t-1}, x_t; \theta)$

$$I_t = UX + Vh_{t-1} + b \tag{1}$$

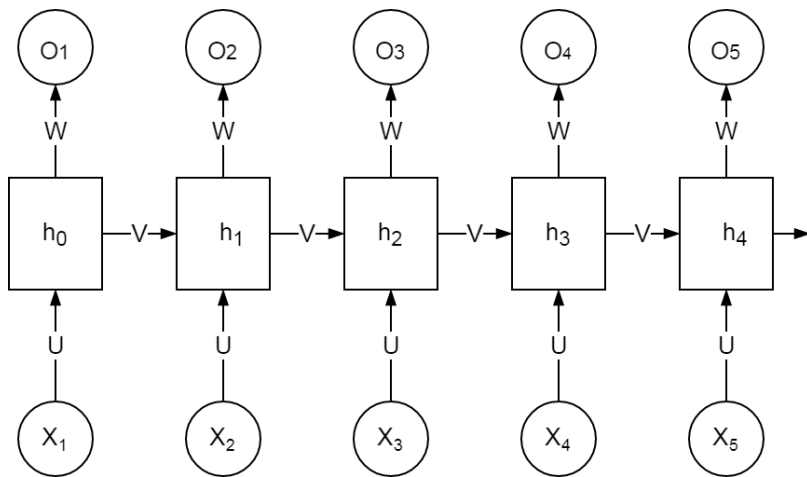
$$h_t = \tanh(I_t) \tag{2}$$

$$O_t = Wh_t + c \tag{3}$$



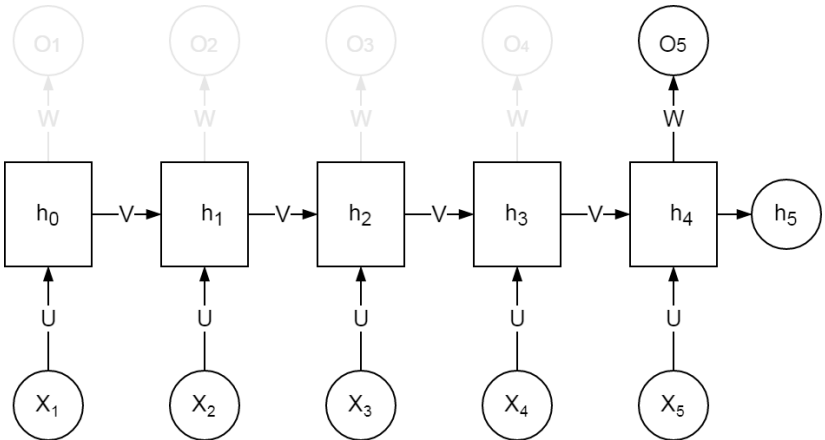
"Recurrent neural network" (2020) Wikipedia. Available at:
https://en.wikipedia.org/wiki/Recurrent_neural_network

Recurrent network architecture styles



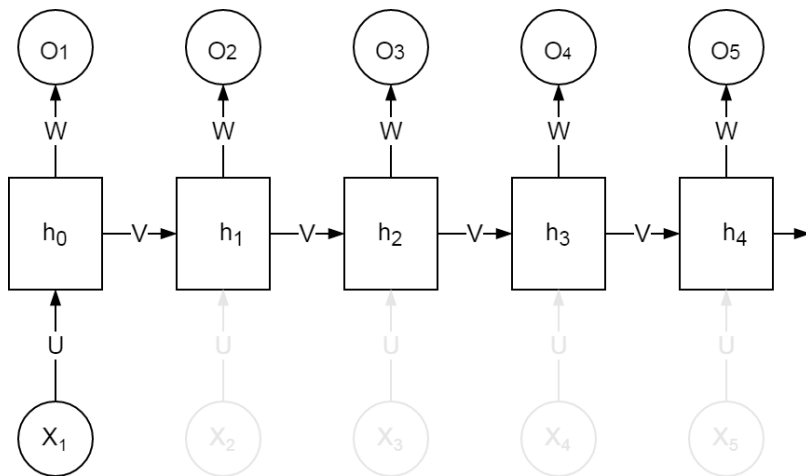
Sequence to sequence network

Recurrent network architecture styles



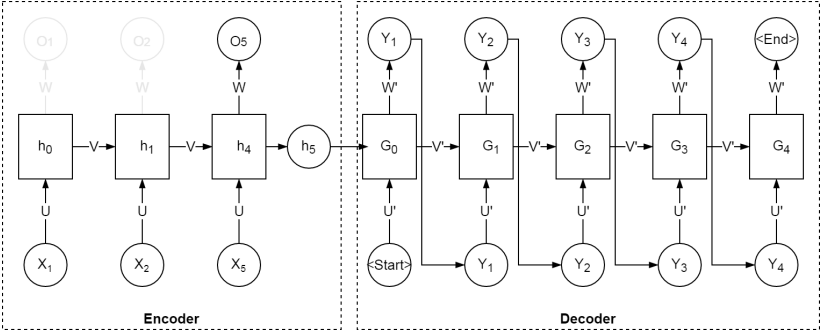
Sequence to vector network

Recurrent network architecture styles



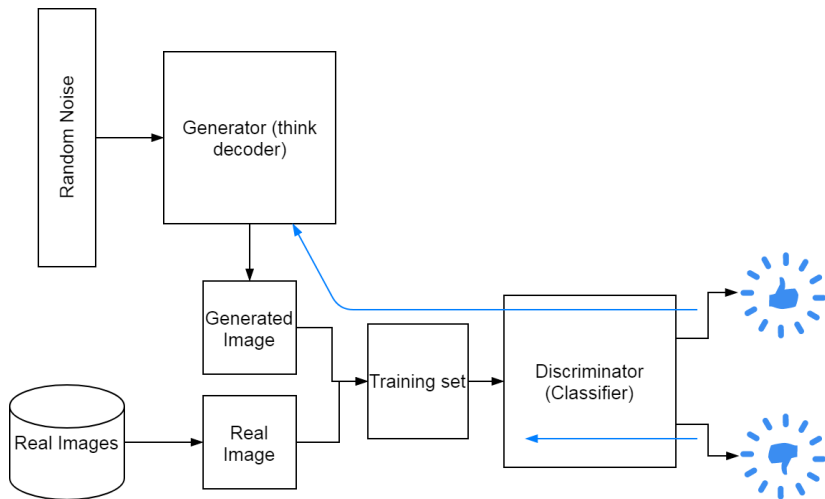
Vector to sequence network

Recurrent network architecture styles

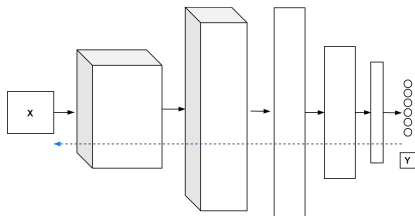


Sequence to sequence encoder-decoder network

Generative adversarial networks - intuition



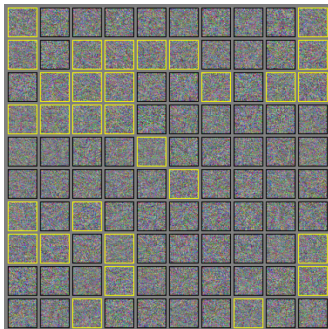
Adversarial attacks



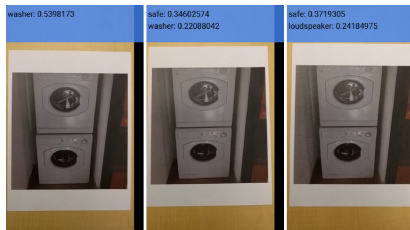
Let \hat{y} be desired, y be the predicted label, and \hat{x} be the desired image

- 1 Simple loss function,
 $L = D(\hat{y}, y)$
- 2 Another loss function,
 $L = D_1(\hat{y}, y) + \lambda D_2(\hat{x}, x)$

Adversarial examples



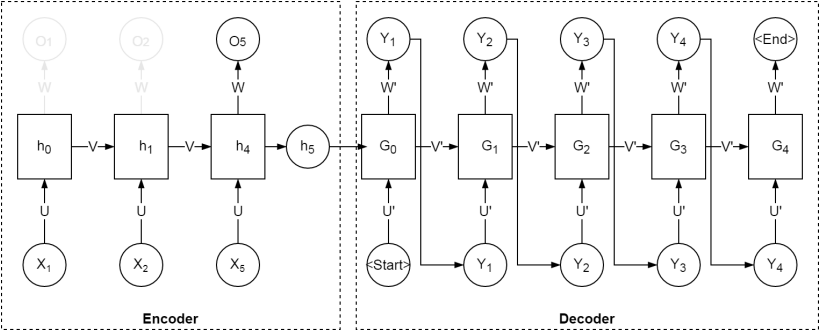
I. Goodfellow, J. Shlens, and C. Szegedy, Explaining and harnessing adversarial examples," in International Conference on Learning Representations, 2015.



A. Kurakin, I. Goodfellow, and S. Bengio, Adversarial examples in the physical world," ICLR Workshop, 2017.


What kinds of attacks can you think of?


Attention - intuition





Sequence to sequence encoder-decoder network

References


 V. Dumoulin and F. Visin, “A guide to convolution arithmetic for deep learning,” 2018.

 Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, pp. 2278–2324, Nov 1998.

 I. Goodfellow, J. Shlens, and C. Szegedy, “Explaining and harnessing adversarial examples,” in *International Conference on Learning Representations*, 2015.

 A. Kurakin, I. Goodfellow, and S. Bengio, “Adversarial examples in the physical world,” *ICLR Workshop*, 2017.

 I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. The MIT Press, 2016.

 I. Sutskever, O. Vinyals, and Q. V. Le, “Sequence to sequence learning with neural networks,” in *Advances in Neural Information Processing Systems 27* (Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, eds.), pp. 3104–3112, Curran Associates, Inc., 2014.