# Introduction to Deep Learning for Scientists and Engineers Part-II

Abhijat Vatsyayan [1]

November 19, 2020

# Summary

- Part I
    - Problem definition (rely on supervised learning)
    - Compute graph and gradients
    - A little about deep learning libraries.
- Part II
    - Need for different architectures
    - Convolution networks
    - Transfer learning, autoencoders
    - Recurrent networks
    - Miscellaneous topics
        - Adversarial attacks
        - GAN
        - Attention
- Not covered

# 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*

[1] See Russakovsky et al. 2015 for an example

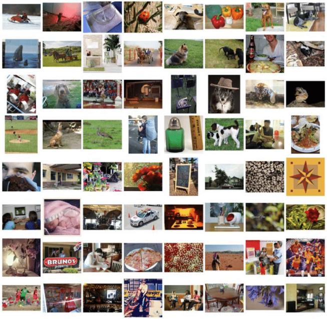
## Datasets

- 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

# ImageNet Large Scale Visual Recognition Challenge

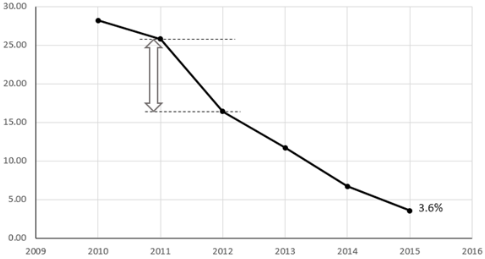
MNIST Dataset (60k, 10k)

ImageNet (14M+, 22k)



# ImageNet Large Scale Visual Recognition Challenge

- Publicly available dataset - ImageNet (14M+, 22k categories)
- Annual competition
  - Image classification
  - Object detection and localization
- Increasing depth
  - 8 layer AlexNet
  - 19 layer GoogLeNet
  - 152 layer ResNet



Top 5 classification error rate

# Using linear layer

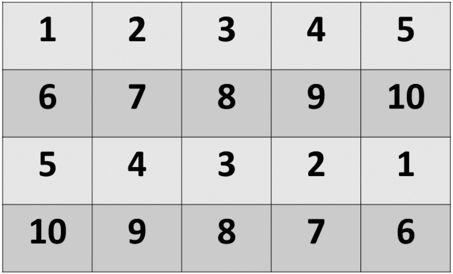
## Image as 2D Tensor(Matrix)

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

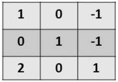
## 2D → 1D

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

# 2D Convolution Example

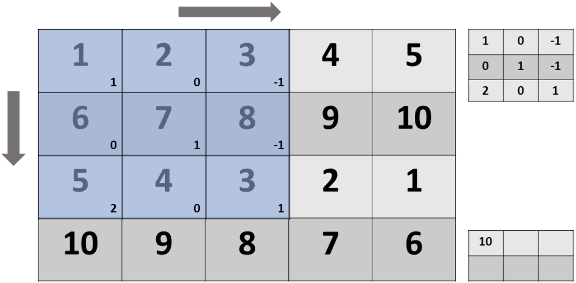


| | | | | |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 |
| 6 | 7 | 8 | 9 | 10 |
| 5 | 4 | 3 | 2 | 1 |
| 10 | 9 | 8 | 7 | 6 |



| | | |
|---|---|---|
| 1 | 0 | -1 |
| 0 | 1 | -1 |
| 2 | 0 | 1 |

# 2D Convolution Example



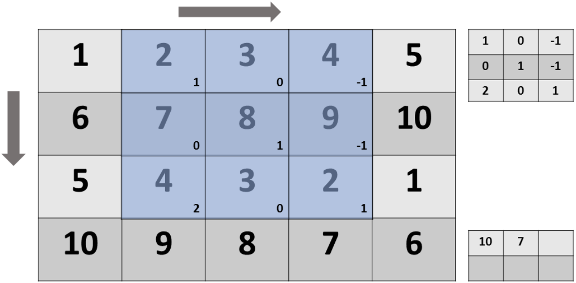
| | | | | |
|---|---|---|---|---|
| 1<br>1 | 2<br>0 | 3<br>-1 | 4 | 5 |
| 6<br>0 | 7<br>1 | 8<br>-1 | 9 | 10 |
| 5<br>2 | 4<br>0 | 3<br>1 | 2 | 1 |
| 10 | 9 | 8 | 7 | 6 |

| | | |
|---|---|---|
| 1 | 0 | -1 |
| 0 | 1 | -1 |
| 2 | 0 | 1 |

| | | |
|---|---|---|
| 10 | | |
| | | |

$$(1 \times 1) + (2 \times 0) + (3 \times -1) + \\ (6 \times 0) + (7 \times 1) + (8 \times -1) + \\ (5 \times 2) + (4 \times 0) + (3 \times 1)$$

# 2D Convolution Example

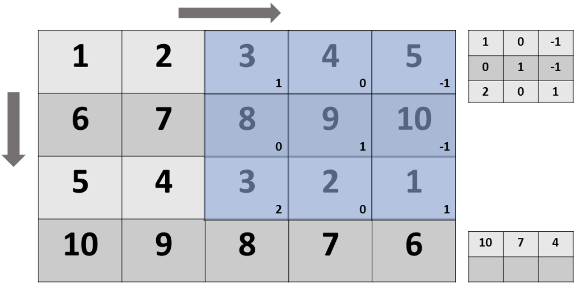


|  |  |  |  |  |
|---|---|---|---|---|
| 1 | 2<br>1 | 3<br>0 | 4<br>-1 | 5 |
| 6 | 7<br>0 | 8<br>1 | 9<br>-1 | 10 |
| 5 | 4<br>2 | 3<br>0 | 2<br>1 | 1 |
| 10 | 9 | 8 | 7 | 6 |

|  |  |  |
|---|---|---|
| 1 | 0 | -1 |
| 0 | 1 | -1 |
| 2 | 0 | 1 |

|  |  |  |
|---|---|---|
| 10 | 7 |  |
|  |  |  |

# 2D Convolution Example

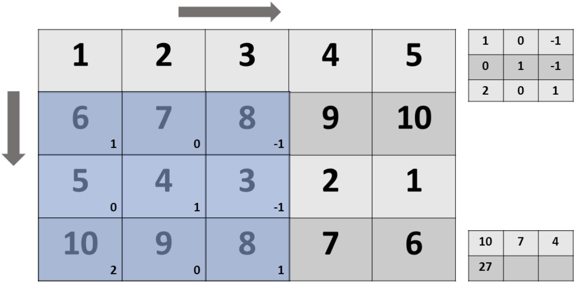


| | | | | |
|---|---|---|---|---|
| 1 | 2 | $3_{1}$ | $4_{0}$ | $5_{-1}$ |
| 6 | 7 | $8_{0}$ | $9_{1}$ | $10_{-1}$ |
| 5 | 4 | $3_{2}$ | $2_{0}$ | $1_{1}$ |
| 10 | 9 | 8 | 7 | 6 |

| | | |
|---|---|---|
| 1 | 0 | -1 |
| 0 | 1 | -1 |
| 2 | 0 | 1 |

| | | |
|---|---|---|
| 10 | 7 | 4 |
| | | |

# 2D Convolution Example



| | | | | |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 |
| 6<br>1 | 7<br>0 | 8<br>-1 | 9 | 10 |
| 5<br>0 | 4<br>1 | 3<br>-1 | 2 | 1 |
| 10<br>2 | 9<br>0 | 8<br>1 | 7 | 6 |

| | | |
|---|---|---|
| 1 | 0 | -1 |
| 0 | 1 | -1 |
| 2 | 0 | 1 |

| | | |
|---|---|---|
| 10 | 7 | 4 |
| 27 | | |

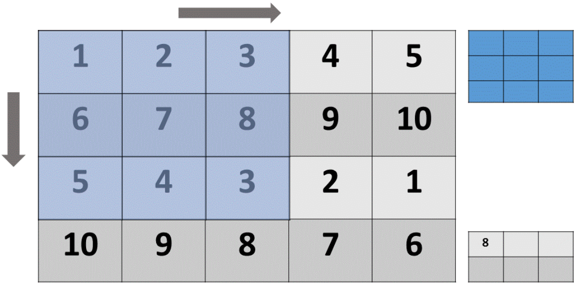
# 2D Convolution

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

For a $5 \times 5$ 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.

Layers typically have multiple filters, each filter resulting in a single output channel. Hence, a layer with 200 $5 \times 5$ filters (with bias) for 3 channel inputs will have $76 \times 200 = 15,200$ parameters. Corresponding output will contain 200 channels.

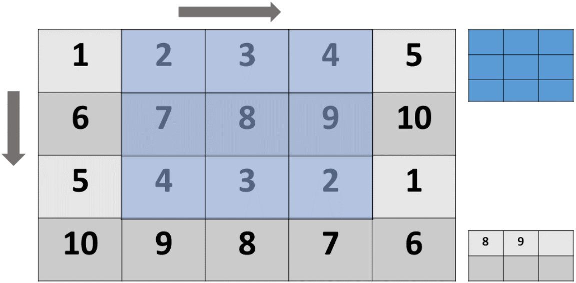
# Max pooling



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

| | | |
|---|---|---|
| 8 | | |
| | | |

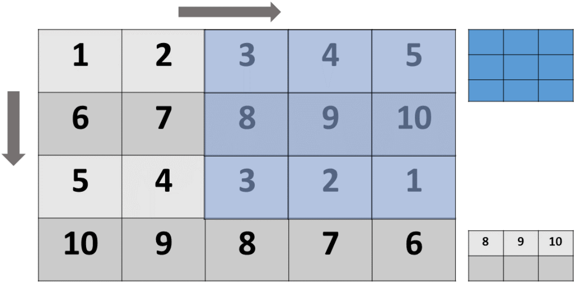
# Max pooling



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

| | | |
|---|---|---|
| 8 | 9 | |
| | | |

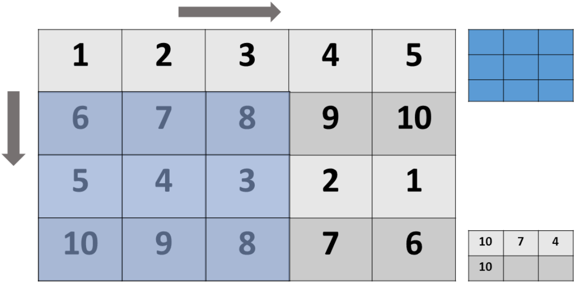
# Max pooling



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

|  |  |  |
|---|---|---|
| 8 | 9 | 10 |
|  |  |  |

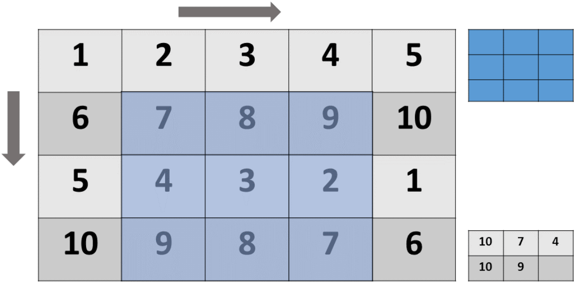
# Max pooling



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

|  |  |  |
|---|---|---|
| 10 | 7 | 4 |
| 10 |  |  |

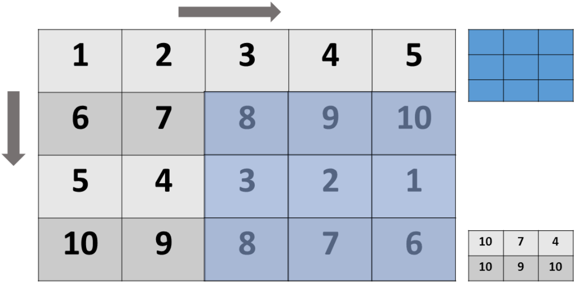
# Max pooling



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

| | | |
|---|---|---|
| 10 | 7 | 4 |
| 10 | 9 | |

# Max pooling



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

| | | |
|---|---|---|
| 10 | 7 | 4 |
| 10 | 9 | 10 |

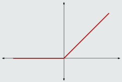
# Image processing

## Other functions

- RelU activation: $max(x, 0)$
- Leaky RelU

$$f(x; .1) = \begin{cases} x, & \text{if } x \geq 0 \\ .1x, & \text{otherwise} \end{cases}$$

- Dropout layer

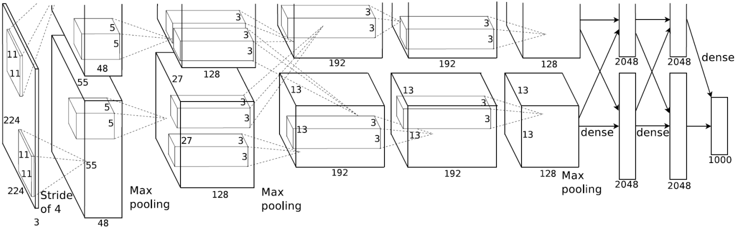


RelU

## 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

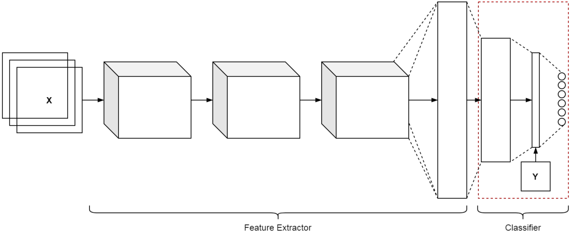
# Typical convolution net, pytorch

Listing 1: Typical (simple) CNN in pytorch

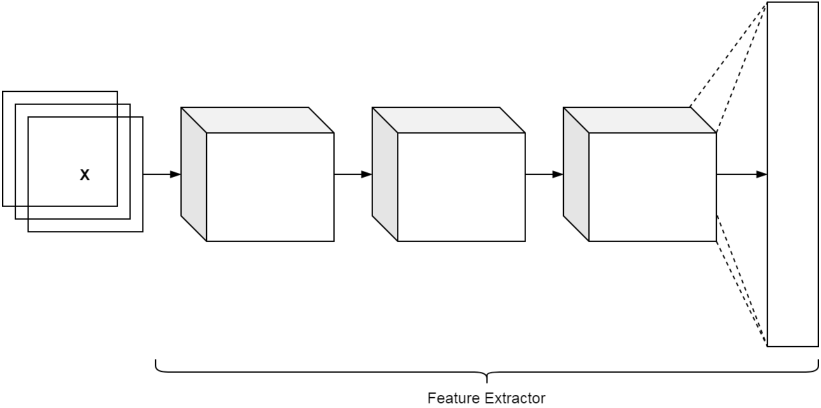
```
class ConvNet(nn.Module):
  def __init__(self):
    super(ConvNet, self).__init__()
    self.conv1 = nn.Conv2d(3, 16, 3, padding=1)
    self.lrelu1 = nn.LeakyReLU(.1)
    self.conv2 = nn.Conv2d(16, 32, kernel_size=3, padding=1)
    self.lrelu2 = nn.LeakyReLU(.1)
    self.maxpool1 = nn.MaxPool2d(kernel_size=3, padding=1)
    self.dropout1 = nn.Dropout(p=.25)
    self.conv3 = nn.Conv2d(in_channels=32, out_channels=32, padding=1, kernel_size=3)
    self.lrelu2 = nn.LeakyReLU(0.1)
    self.conv4 = nn.Conv2d(in_channels=32, out_channels=64, padding=1, kernel_size=3)
    self.lrelu3 = nn.LeakyReLU(0.1)
    self.maxpool2 = nn.MaxPool2d(kernel_size=3, padding=1)
    self.dropout2 = nn.Dropout(p=.25)

    self.conv_layers = [self.conv1, self.lrelu1, self.conv2, self.lrelu2, self.maxpool1,
    self.conv3, self.lrelu2, self.conv4, self.lrelu3, self.maxpool2, self.dropout2]

    self.fc1 = nn.Linear(in_features=64*4*4, out_features=256)
    self.lrelu4 = nn.LeakyReLU(.1)
    self.dropout3 = nn.Dropout(.25)
    self.fc2 = nn.Linear(in_features=256, out_features=10)
    self.softmax = nn.Softmax(dim=1)

    self.flat_layers = [self.fc1, self.lrelu4, self.dropout3, self.fc2, self.softmax]
```

# Transfer learning - image classification with CNN

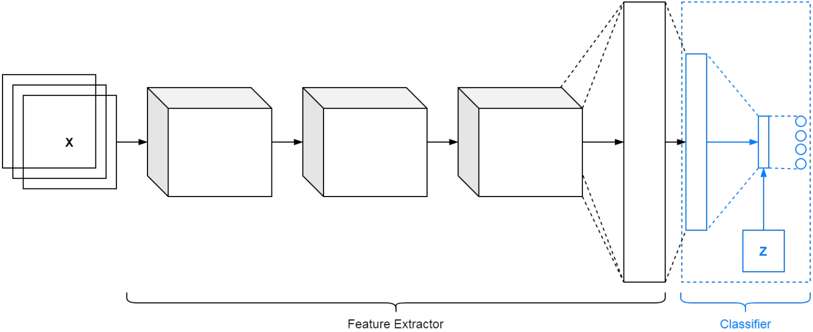
- Earlier layers in convolution networks extract generic features (edges etc.)
- Layers get more specialized towards the end
- Training can take weeks. A trained model represents significant effort from experts.



# Transfer learning



# Transfer learning



# Transfer learning - pytorch

```

import torch.nn as nn
from torchvision import datasets, models, transforms

model = models.resnet18(pretrained=True)
in_features = model.fc.in_features

# Here the size of each output sample is set to 2.
model.fc = nn.Linear(in_features, 2)
```

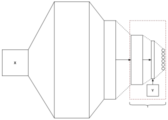
Listing 2: Using pretrained ResNet18

```
for param in model.parameters():
  param.requires_grad = False

in_features = model.fc.in_features
model.fc = nn.Linear(in_features, 2)
```

Listing 3: Freezing model parameters

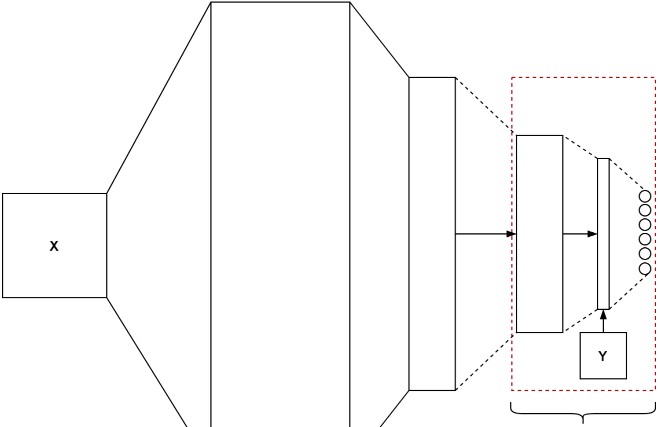
# Auto-encoders and approximate identity function

- Let us consider $y = f(x)$ as the network function.
- Let us consider another network function $z = g(y)$ and $z$ has same shape as $x$.
- Let us consider a loss function, $L(x, g(f(x)))$. This could be
  - Squared distance $\frac{1}{n}\sum_i (x_i - z_i)^2$
  - Absolute difference $\frac{1}{n}\sum_i |x_i - z_i|$
  - "Some" other distance metric between vectors in "some" latent space

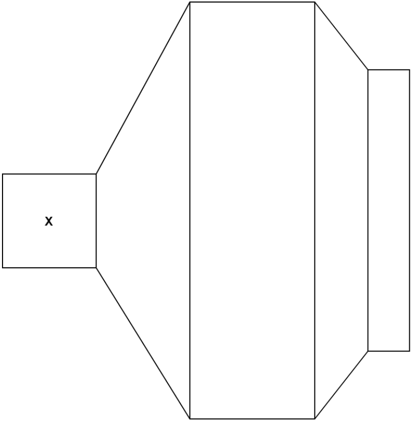


$y = f(x)$

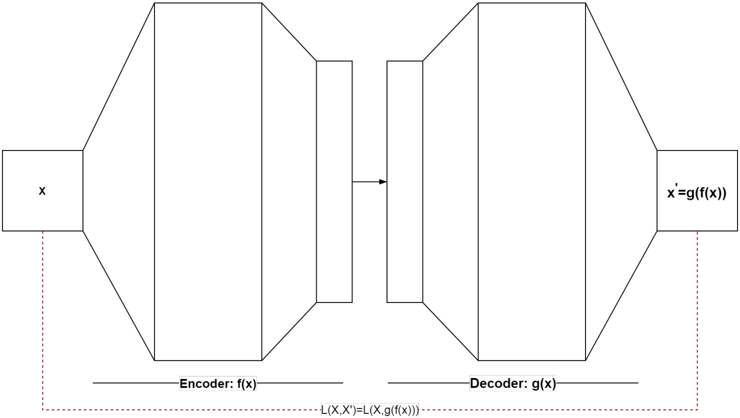
# Autoencoders and identity function



# Autoencoders and identity function



# Autoencoders and identity function



# Sequences

- Natural language tasks
- Event processing
- Stateful 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.

# Note about working with sentences

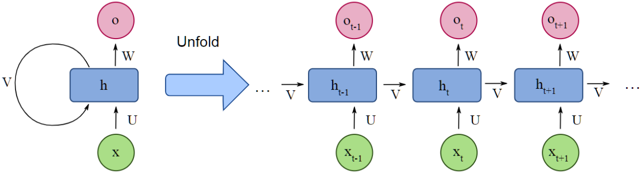
- One-hot encodings
  - Very long vectors
  - Vectors do not represent relationship between words.
- Word embeddings, e.g., word2vec
  - Uses a sliding window to predict a word given surrounding context
  - Captures some key relationship between words.
  - Given enough data, models often learn embeddings.
- Sub-word models
  - Works better with social media and informal speech
  - Languages that may ”create” new words on the fly (Sanskrit).

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

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

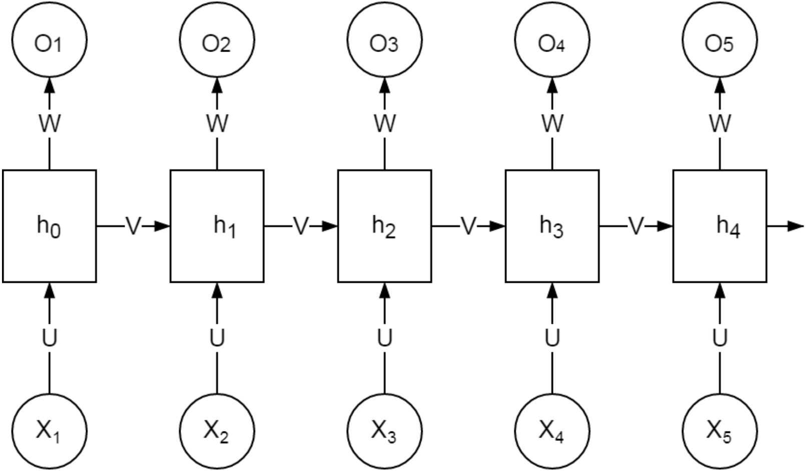
$$h_t = tanh(I_t) \tag{2}$$

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



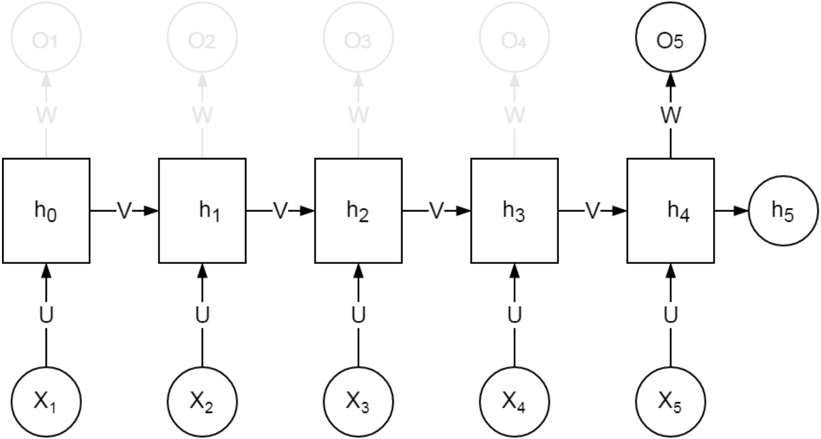
”Recurrent neural network” (2020) Wikipedia. Available at: https://en.wikipedia.org/wiki/Recurrent_neural_network

# Recurrent network architecture styles



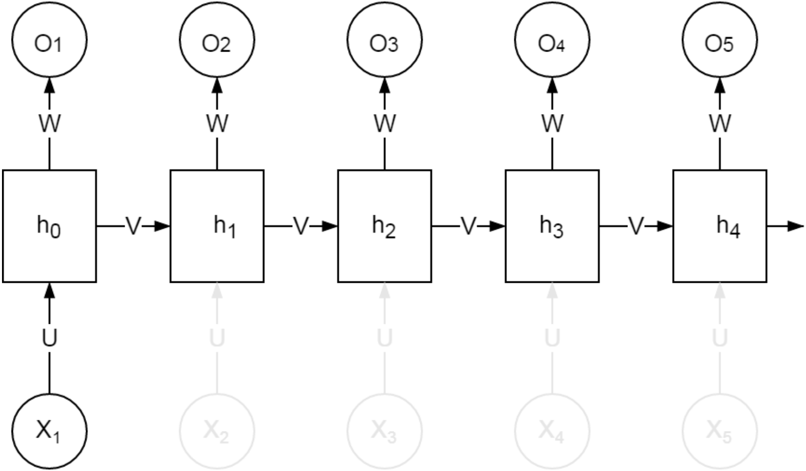
Sequence to sequence network

# Recurrent network architecture styles



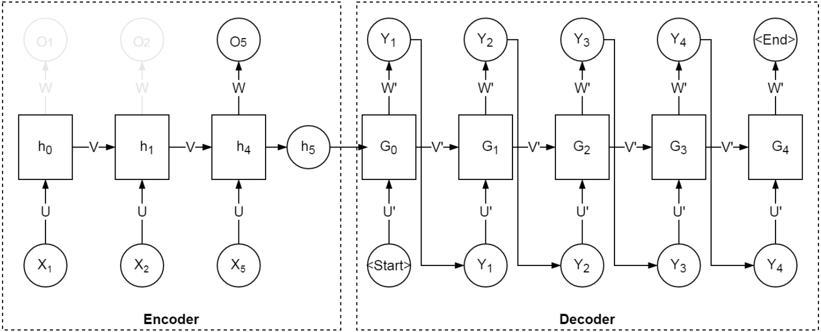
Sequence to vector network

# Recurrent network architecture styles



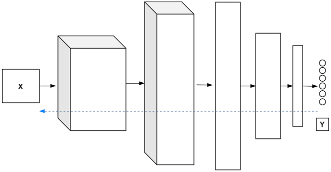
Vector to sequence network

# Recurrent network architecture styles



Sequence to sequence encoder-decoder network

# Adversarial attacks

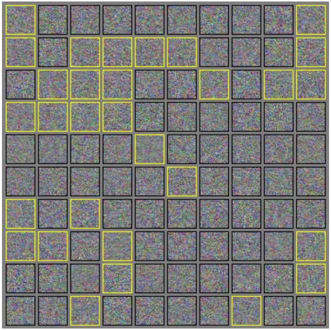


Classification network

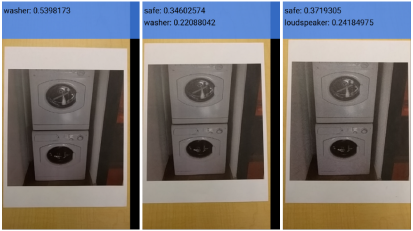
Let $\hat{y}$ be the desired label, $y$ be the predicted label.

- Consider the loss function $L = D_1(\hat{y}, y)$
- If $\hat{x}$ is the desired image, consider another loss function $L = D_1(\hat{y}, y) + \lambda D_2(\hat{x}, x)$

# Adversarial examples



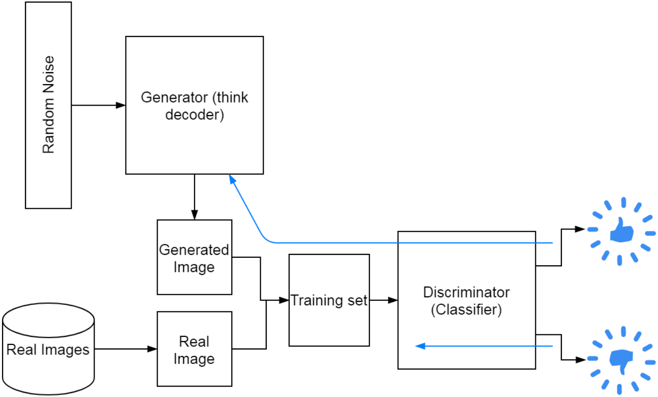
I. Goodfellow, J. Shlens, and C. Szegedy, Explaining and harnessing adversarial examples," in International Conference on Learning Representations, 2015.



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

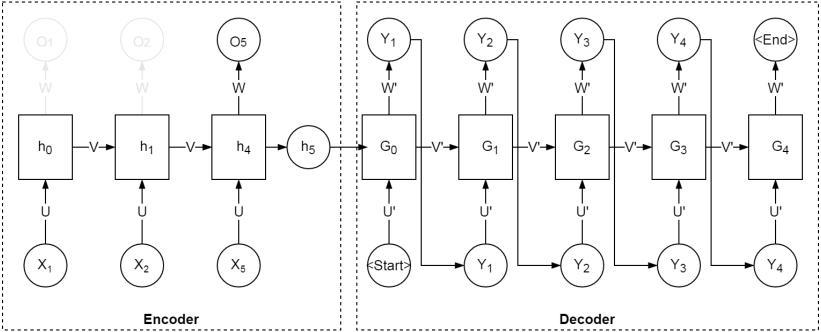
What kinds of attacks can you think of?

# Generative adversarial networks - intuition



Learning data distribution

# 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.