# **Convolutional Neural Networks**



## **Neural Network Architecture**

Objects are often **localized in space** so to find the faces in an image, not every pixel is important for classification—makes sense to drag a window across an image.



Similarly, to identify edges or other local structure, it makes sense to only look at **local information** 



VS.



#### 2d Convolution Layer

#### Example: 200x200 image

- Fully-connected, 400,000 hidden units = 16 billion parameters
- Locally-connected, 400,000 hidden units 10x10 fields = 40 million params
- Local connections capture local dependencies



## Convolution of images (2d convolution)

$$(I * K)(i, j) = \sum_{m} \sum_{n} I(i + m, j + n) K(m, n)$$



Image I



Image



Convolved Feature I \* K

## Stacking convolved images



**Repeat with d filters!** 

# Pooling

Pooling reduces the dimension and can be interpreted as "This filter had a high response in this general region"

## Single depth slice1124

max pool with 2x2 filters and stride 2

6	8
3	4





X





## **Pooling Convolution layer**



## Flattening



Flatten into a single vector of size 14\*14\*64=12544

### **Training Convolutional Networks**



**Train with SGD!** 



#### **Training Convolutional Networks**



#### Real example network: LeNet





#### Real example network: LeNet



# **Famous CNNs**



#### **ImageNet Dataset**

~14 million images, 20k classes



Deng et al. "Imagenet: a large scale hierarchical image database" '09



#### Breakthrough on ImageNet: ~the beginning of deep learning era



Krizhevsky, Sutskever, Hinton "ImageNet Claasification with Deep Convolutional Neural Networks", NIPS 2012.

#### **AlexNet**

8 layers, ~60M parameters

Top5 error: 18.2%

Techniques used: ReLU activation, overlapping pooling, dropout, ensemble (create 10 patches by cropping and average the predictions), data-augmentation (intensity of RGB channels)





Remove top fully-connected layer 7

Drop ~16 million parameters

1.1% drop in performance







# Remove both fully connected layers 6 and 7

Drop ~50 million parameters

5.7% drop in performance



[From Rob Fergus' CIFAR 2016 tutorial]

#### **AlexNet**

Remove upper convolutio / feature extractor layers (layer 3 and 4)

Drop ~1 million parameters

3% drop in performance



[From Rob Fergus' CIFAR 2016 tutorial]



Remove top fully connected layer 6,7 and upper convolution layers 3,4.

33.5% drop in performance.

Depth of the network is the key.

[From Rob Fergus' CIFAR 2016 tutorial]





#### Motivation: multiscale nature of images



Large kernel for global features, and smaller kernel for local features.

Idea: have multiple different-size kernels at any layer.

[Going Deep with Convolutions, Szegedy et al. '14]





Large kernel for global features, and smaller kernel for local features.

Idea: have multiple different-size kernels at any layer.

[Going Deep with Convolutions, Szegedy et al. '14]

#### **Inception Module**



Multiple filter scales at each layer

Dimensionality reduction to keep computational requirements down

[Going Deep with Convolutions, Szegedy et al. '14]

#### **Residual Networks**

Motivation: extremely deep nets are hard to train (gradient explosion/ vanishing)



Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

#### **Residual Networks**

Idea: identity shortcut, skip one or more layers.

**Justification:** network can easily simulate shallow network ( $F \approx 0$ ), so performance should not degrade by going deeper.



### **Residual Networks**

- 3.57% top-5 error on ImageNet
- First deep network with > 100 layers.
- Widely used in many domains (AlphaGo)





### **Densely Connected Network**

**Idea:** explicit forward output of layer to all future layers (by concatenation)

- Intuition: helps vanishing gradients, encourage reuse features (reduce parameter count)
- **Issues:** network maybe too wide, need to be careful about memory consumption





### **Neural Architecture / Hyper-Parameter Search**

Many design choices:

- Number of layers, width, kernel size, pooling, connections, etc.
- Normalization, learning rate, batch size, etc.

Strategies:

- Grid search
- Random search [Bergestra & Bengio '12]
- Bandit-based [Li et al. '16]
- Gradient-based (DARTS) [Liu et al. '19]
- Neural tangent kernel [Xu et al. '21]

# **Recurrent Neural Networks**



#### **Sequence Data**





检测语言 英语 中文 德语 ✓	← 中文(简体) 英	吾 日语 🗸
Deep learning is a popular area in Al.	G × 深度学习是Alf	内热门领域。    ☆
	Shēndù xuéxí shì Al de	rèmén lǐngyù.
. <b>↓</b>	38 / 5000	

#### **State-Space Model**

- $h_t$ : hidden state
- $X_t$ : input
- *Y<sub>t</sub>*: output
- $Y_t, h_t = f(h_{t-1}, X_t; \theta)$
- $h_{-1}$ : initial state



### **Recurrent Neural Network**

- $h_t$ : hidden state
- $X_t$ : input
- *Y<sub>t</sub>*: output
- $Y_t, h_t = f(h_{t-1}, X_t; \theta)$
- $h_{-1}$ : initial state



Fully-connect NN vs. RNN

- $h_t$ : a vector summarizes all past inputs (a.k.a. "memory")
- $h_{-1}$  affects the entire dynamics (typically set to zero)
- $X_t$  affects all the outputs and states after t
- $Y_t$  depends on  $X_0, \ldots, X_t$



### **Recurrent Neural Network**

- $h_t$ : hidden state
- $X_t$ : input
- *Y<sub>t</sub>*: output
- $Y_t, h_t = f(h_{t-1}, X_t; \theta)$
- $h_{-1}$ : initial state



Fully-connect NN vs. RNN

- RNN can be viewed as repeated applying fully-connected NNs
- $h_t = \sigma_1(W^{(1)}X_t + W^{(11)}h_{t-1} + b^{(1)})$
- $Y_t = \sigma_2(W^{(2)}h_t + b^{(2)})$
- $\sigma_1, \sigma_2$  are activation functions (sigmoid, ReLU, tanh, etc)



#### **Recurrent Neural Network**



#### Stack K layers of fully-connected NN

- $h_t^{(k)}$ : hidden state
- $X_t$ : input
- *Y<sub>t</sub>*: output
- $h_t^{(1)} = f_1^{(1)}(h_{t-1}^{(1)}, X_t; \theta)$
- $h_t^{(k)} = f_1^{(k)}(h_{t-1}^{(k)}, h_t^{(k-1)}; \theta)$
- $Y_t = f_2(h_t^{(K)}; \theta)$
- $h_{-1}^{(k)}$ : initial states



### **Training Recurrent Neural Network**

- $h_t$ : hidden state
- $X_t$ : input
- *Y<sub>t</sub>*: output
- $Y_t, h_t = f(h_{t-1}, X_t; \theta)$
- $h_{-1}$ : initial state



- Data:  $\{(X_t, D_t)\}_{t=1}^T$  (RNN can handle more general data format) • Loss  $L(\theta) = \sum_{t=1}^T \ell(Y_t, D_t)$
- Goal: learn  $\theta$  by gradient-based method
  - Back propagation

### **Back Propagation Through Time**

- $h_t = \sigma_1(W^{(1)}X_t + W^{(11)}h_{t-1} + b^{(1)})$
- $Y_t = \sigma_2(W^{(2)}h_t + b^{(2)})$
- $Z_t^{(1)}$ : pre-activation of hidden state
  - $(h_t = \sigma_1(Z_t^{(1)}))$
- $Z_t^{(2)}$  : pre-activation of output ( $Y_t = \sigma_2(Z_t^{(2)})$ )



#### **Back Propagation Through Time**

#### **Back Propagation Through Time**

#### Extensions

What if  $Y_t$  depends on the entire inputs?

- Biredictional RNN:
  - AN RNN for forward dependencies: t= 0,...,T
  - An RNN for backward dependencies: t= T,...0
  - $Y_t = f_2(h_t^f, h_t^b; \theta)$





RNN for sequence classification (sentiment analysis)

- $Y = \max_{t} Y_t$
- Cross-entropy loss

