CNN for Visual Recognition

Lecture 1 CNN for VR

Computer Vision

CS 131
CS 231a

Input Image → Primal Sketch → 2 ½-D sketch → 3D model representation

Image Segmentation

Lecture 2 Image Classification pipeline

Python Numpy Piazza
Google Cloud
Assignment 1
Image Classification

```
def classifier(image):
    # magic
    return class
```

Challenges
Data-Driven Approach
1. collect dataset of images and labels
2. ML → classifier
3. Evaluate (new images)

```
def train(images, labels):
    # ML
    return model
```
← memorize data/label

```
def predict(model, test_images):
    # model predict
    return test_labels
```
← predict label of the most similar training image

CIFAR10

Distance Metric to compare images
L1 distance: $d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$
test image - training image = pixel-wise absolute value
Manhattan distance
L2 distance: $d_2(I_1, I_2) = \sqrt{\sum_p (I_1^p - I_2^p)^2}$
Euclidean distance
k-nearest neighbor (kNN)
L1 distance: boundary tend to follow coordinate axes
Hyperparameters: choices about the algorithm that we set rather than learn
Very problem-dependent

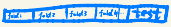
Setting Hyperparameters

Idea 1: Choose one that works best on the data
BAD: k=1 always works perfectly on training data

Idea 2: data → train / test
Choose hyperparameters that work best on test data
BAD: No idea how it will perform on new data

Idea 3: data → train / validation / test
Better!
choose one that work best on val and evaluate on test
use the test data at last

Cross Validation (Idea 4)
data → folds + test



try each fold as validation and average the results
Usually for small datasets, but not used too frequently in deep learning
kNN on images is never used
1. very slow
2. Distance metrics on pixels are not informative
Curse of dimensionality

Image Classification

training set of images and labels
predict labels on the test set
kNN: predict based on nearest training examples

Linear Classification (simple but important)

CIFAR10

Parametric Approach

$f(x, W)$ → 10 numbers giving class scores
parameters/weights

$f(x, W) = Wx + b$

Linear Classifier is only learning one template for each class

Define a (linear) score function

Lecture 3

Check out Project Ideas on Piazza

Challenges of recognition

Loss function: tells how good(bad) the classifier is

A dataset of examples: $\{(x_i, y_i)\}_{i=1}^N$

x_i: image y_i: label

Loss over the dataset: $L = \frac{1}{N}\sum_i L_i(f(x_i, W), y_i)$

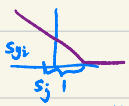
Multi-class SVM loss:

score vector: $s = f(x_i, W)$

$L_i = \sum_{j \neq y_i} \begin{cases} 0 & s_{y_i} \geq s_j + 1 \\ s_j - s_{y_i} + 1 & \text{其他} \end{cases}$

$= \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$

Hinge loss



s_{y_i}: 第 y_i 类的 s, s_j: 第 j 类的 s

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:

cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**
Losses:	2.9		

Multiclass SVM loss:

Given an example (x_i, y_i) where x_i is the image and where y_i is the (integer) label,

and using the shorthand for the scores vector: $s = f(x_i, W)$

the SVM loss has the form:

$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$

= max(0, 5.1 - 3.2 + 1)
+max(0, -1.7 - 3.2 + 1)
= max(0, 2.9) + max(0, -3.9)
= 2.9 + 0
= 2.9

$L = \frac{1}{N}\sum_{i=1}^N L_i$ SVM: beginning $s \approx 0 \Rightarrow L \approx$ class number -1

$L(W) = \frac{1}{N}\sum_{i=1}^N L_i(f(x_i, W), y_i)$ → Data loss: Model predictions should match training data

$+\lambda R(W)$

$L(W) = \frac{1}{N}\sum_{i=1}^N L_i(f(x_i, W), y_i) + \lambda R(W)$ ← Regularization: Model should be simple → work on test data

λ: hyper-parameter

$R(W)$: L2: $R(W) = \sum\sum W^2$

L1: $R(W) = \sum\sum |W|$

Elastic net (L1 + L2): $R(W) = \sum\sum (\beta W^2 + |W|)$

max norm, dropout, Batch normalization, stochastic depth

Softmax Classifier (Multinomial Logistic Regression)

maximize the log likelihood

$L_i = -\log P(Y = y_i | X = x_i)$

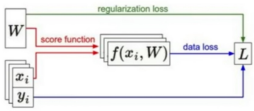
Recap

- We have some dataset of (x,y)
- We have a **score function:** $s = f(x; W) \overset{\text{e.g.}}{=} Wx$
- We have a **loss function:**

$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$ Softmax

$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$ SVM

$L = \frac{1}{N}\sum_{i=1}^N L_i + R(W)$ Full loss



Optimizing

for debugging

Numerical gradient: approximate, slow, easy to write

Analytic gradient: exact, fast, error-prone

#Vanilla gradient Descent

```
while True:
    weights_grad = evaluate_gradient(loss_fun, data, weights)
    weights += - step_size * weight_grad #perfor parameter update
```

Stochastic Gradient Descent (SGD)

Full sum expensive when N is large!

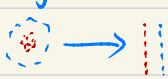
Approximate sum using a minibatch of examples

32/64/128 common

Aside: Image Features

Image → Feature Representation → Class

Image Features: Motivation



Histogram of Oriented Gradients (HoG)

Bag of Words

Lecture 4 Backpropagation & NN

Computational graphs

Backpropagation

Neural Turing Machine

local gradient

add gate: gradient distributor

max gate: gradient router

mul gate: gradient switcher

Gradients for vectorized code

Neural Networks

non-linear functions

Lecture 5 Convolutional Neural Networks

FC: input → Wx → activation

Conv: image → filter: convolve the filter with the image

a chunk of the image

Activation map

"The convolution of two signals" → CNN

Conv → RELU → Conv → RELU → ... → pool → ... → FC

Output size: $\frac{N-F}{stride}+1$

zero padding → maintain the same size

↳ convolve shrinks volumes spatially (not good)

pooling (Max Pooling)

Lecture 6 Train Neural Networks

Training:

1. one time setup
2. training dynamics
3. evaluation

Activation Function

sigmoid

problems: 1. saturated neurons "kill" the gradients
2. outputs are not zero-centered

tanh: 1. zero-centered
2. kills gradients when saturated

ReLU: 1. Does not saturate
2. computationally efficient
3. Converges faster
4. more biologically plausible than sigmoid

Not zero-centered

Leaky ReLU: "Will not die"

$f(x) = \max(0.01x, x)$

Paramic Rectifier (PReLU)

$f(x) = \max(\alpha x, x)$

Exponential Linear Units (ELU)

Maxout "Neuron" $\max(w_1^T x + b_1, w_2^T x + b_2)$

In practice:

- Use ReLU. Be careful with learning rate
- Try out Leaky ReLU / Maxout / ELU
- Try out tanh (don't expect much)
- Don't use sigmoid

Data Processing

Preprocess the data

center → X -= np.mean(X, axis=0)

normalize → X /= np.std(X, axis=0)

In practice: center only

zero-mean preprocessing

Weight Initialization

Small random numbers

↳ works for small networks, problems with deeper...

"Xavier initialization" ✓

Resonable initialization

W = np.random.randn(fan_in, fan_out)/np.sqrt(fan_in)

↑ layer initialization

ReLU breaks nonlinearity

proper initialization

Batch normalization

1. compute the empirical mean and variance independently for each dimension

2. Normalize $\hat{x}^{(k)} = \frac{x^{(k)} - E[x^{(k)}]}{\sqrt{Var[x^{(k)}]}}$

Usually inserted after FC / Conv, before nonlinearity

FC → BN → tanh → FC → BN → tanh → ...

Improves gradient flow through the network

Forcing the input data to be unit gaussian

Babysitting the learning process

Choose the architecture

Check the loss is reasonable

+ regularization → loss ↑ (good)

sanity check

start with small regularization and find learning rate → loss ↓

loss not going down: lr too low

loss exploding: lr too high

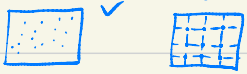
Hyperparameter Optimization

cross-validation strategy

coarse search

it's best to optimize in log space

Random Search ✓ Grid Search



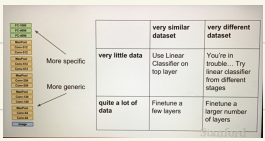
Data Augmentation

DropConnect

Fractional Max Pooling

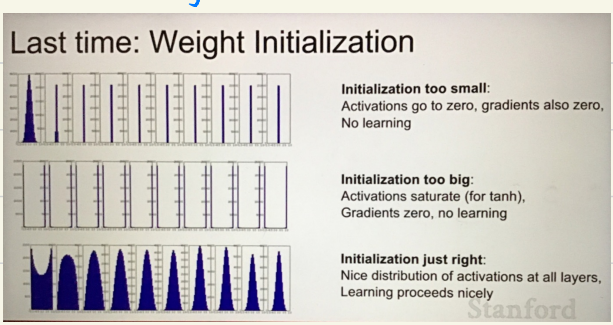
Stochastic Depth (dropout layers, crazy but may work)

Transfer Learning



CNN pre-trained on ImageNet → Object detection, Image caption

Lecture 7 Training Neural Networks



Optimization

Problems with SGD

local minima/saddle point

saddle points are much more common in high dimension

SGD + Momentum

AdaGrad grad-squared

RMSProp

Adam → momentum + Bias correction + AdaGrad/RMSProp

In practice:

Adam is a good default in most cases

If you can afford to do full batch updates, try out L-BFGS (remember to disable all sources of noise)

Model Ensembles

Regularization

Dropout

Lecture 9 CNN Architectures

LeNet-5 conv-pool-conv-pool-FC-FC

AlexNet

VGGNet

GoogLeNet

ResNet

others:

- Network in Network (NiN)
- Wide Residual Networks
- Deep Networks with Stochastic Depth
- Densely Connected Convolutional Networks

Lecture 10 Recurrent Neural Networks

RNN: Process Sequence

$h_t = f_W(h_{t-1}, x_t)$

new state ↓ old state, input vector at some time step

$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$

$y_t = W_{hy}h_t$

Re-use the same weight matrix at every time step

Sequence to Sequence: Many to one + one to many

Truncated Backpropagation through time

Run forward and backward through chunks of the sequence instead of whole sequence

Image Captioning: CNN+RNN

Visual Question Answering

$$LSTM\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$c_t = f \odot c_{t-1} + i \odot g$, $h_t = o \odot \tanh(c_t)$

Lecture 11 Detection & Segmentation

Semantic Segmentation

Label each pixel in the image with a category label

Idea1: sliding window → very inefficient

Idea2: Fully Convolutional

Input $3 \times H \times W$ → Convolutions $D \times H \times W$ → scores $C \times H \times W$ → predictions $H \times W$

Design networks as a bunch of convolutional layers, with downsampling and upsampling inside the network

Input $3 \times H \times W$ → High-res $D_1 \times H/2 \times W/2$ → mid-res $D_2 \times H/4 \times W/4$ → low-res $D_3 \times H/4 \times W/4$ → mid-res ... → pred $H \times W$

"unpooling" "Bag of nails" Transpose Convolution

Classification+Localization

Object detection (different)

Image → CNN → vector → FC → classification

vector → FC → Box coordinate (x, y, w, h)

Treat localization as regression

Aside: Human Pose Estimation

Object detection

Each image needs a different number of outputs

Sliding window: Apply a CNN to different crops of the image, CNN classifies each crop as object or background

Region Proposals: Find "blobby" image regions that are likely to contain objects, relatively fast to run

R-CNN regions of interest (ROI)

Fast R-CNN RoIs from a proposal method

Faster R-CNN → make CNN do proposals

image → CNN → feature map → region proposal network (RPN) → ...

Aside: object detection + captioning = Dense Captioning

Instance Segmentation

Mask R-CNN

Lecture 12 Visualizing & Understanding

First Layer: Visualize Filters

Last Layer: Nearest Neighbors

Dimensionality Reduction

Saliency maps: Segmentation without supervision

Visualizing CNN features: Gradient Ascent

Fooling Images / Adversarial Examples

1. start from a arbitrary Image
2. Pick an arbitrary class
3. Modify the image to maximize the class
4. Repeat until network is fooled

Deep Dream: Amplify existing features

Feature Inversion

Given a CNN feature vector for a Image, find a new image that:
- Matches the given feature vector
- "looks natural" (image prior regularization)

Neural Texture Synthesis: Gram Matrix

Neural style Transfer

problem: Style transfer requires many forward/backward passes through VGG, very slow

Solution: Tran another neural network to perform style transfer

Summary

Many methods for understanding CNN representations

Activations: Nearest neighbors, Dimensionality reduction, maximal patches, occlusion
Gradients: Saliency maps, class visualization, fooling images, feature inversion
Fun: DeepDream, Style Transfer.

Lecture 13 Generative Models

Supervised vs Unsupervised Learning

Unsupervised: no labeled data

Goal: Learn some underlying hidden structure of the data

Supervised vs Unsupervised Learning

Supervised Learning	Unsupervised Learning
Data: (x, y) x is data, y is label	Training data is cheap **Data**: x Just data, no labels!
Goal: Learn a *function* to map x -> y	**Goal**: Learn some underlying hidden *structure* of the data
Examples: Classification, regression, object detection, semantic segmentation, image captioning, etc.	**Examples**: Clustering, dimensionality reduction, feature learning, density estimation etc.

Holy grail: Solve unsupervised learning => understand structure of visual world

Generative models

Given training data, generate new samples from same distribution

PixelRNN and PixelCNN

PixelRNN: Generate image pixels starting from cornor

Dependency on previous pixels using RNN (LSTM)

Drawback: sequential generation is slow

Pixel CNN: still generate image pixels starting from corner

Dependency on previous pixel now modeled using a CNN over context region

Variational Autoencoders (VAE)

Autoencoder: unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data

Input data → Encoder → Features → Decoder → Reconstructed input data

Generative Adverarial Networks (GAN)

Input: random noise → Generator Network → Output: Sample from training distribution

Generator network: try to fool the discriminator by generating real-looking images

Discriminator network: try to distinguish between real and fake images

Training GANs: Two-player game

Lecture 14 Deep Reinforcement Learning

Problems involving an agent interacting with an environment which provides numeric reward signal

Goal: Learn how to take actions in order to maximize reward

state → Agent → action → Environment → state

Markov Decision Process (MDP)

Q-Learning → can be very complicated

Policy Gradients

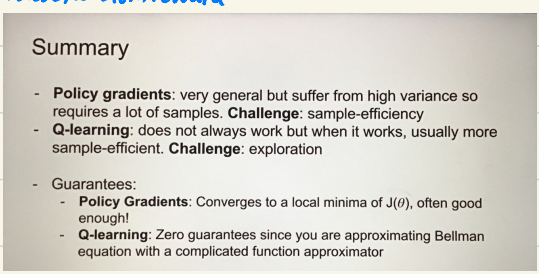
reinforcement algorithm

Variance reduction

Actor-critic Algorithm

Recurrent Attention Model (RAM)

state, action, reward



Lecture 15 Efficient Methods & Hardware for DL

large model → more memory → more energy

Hardware → General Purpose
Hardware → Specialized HW

Number Representation

Prunning Neural Network

weight sharing

Quantization

Low Rand Approximation

Binary / Ternary Net

Winograd Transformation