

Lecture 1 CNN for VR

Computer Vision

CS 131

CS 231a

Input Primal \rightarrow 2D \rightarrow 3D model
Image \rightarrow Sketch \rightarrow sketch representation

Image Segmentation

use the test data at last

Cross Validation (Idea 4)

data fields

train test train test 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

image $\xrightarrow{3 \times 3 \times 3}$ $f(x, w) \xrightarrow{\text{parameters/weights}}$ 10 numbers giving class scores

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

Linear Classifier is only learning one template for each class

Define a linear score function

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 $\xrightarrow{\text{train/test}}$

Choose hyperparameters that work best on test data

BAD: No idea how it will perform on new data

Idea 3: data $\xrightarrow{\text{train/validation/test}}$ Better!

choose one that work best on val and evaluate on test

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: $s_j = f(x_i, W)$

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

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

Hinge loss ~~$\frac{s_j - s_{y_i} + 1}{s_j + 1}$~~

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:

$$\begin{aligned} L_i &= \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1) \\ &= \max(0, 5.1 - 3.2 + 1) \\ &\quad + \max(0, -1.7 - 3.2 + 1) \\ &= \max(0, 2.9) + \max(0, -3.9) \\ &= 2.9 + 0 \\ &= 2.9 \end{aligned}$$

$L = \frac{1}{N} \sum_i L_i$ SVM: beginning set $\Rightarrow L \approx \text{class number} - 1$

$L(W) = \frac{1}{N} \sum_i L_i(f(x_i, W), y_i) \rightarrow$ Data loss: Model predictions should match training data
 $+ \lambda R(W)$

$L(W) = \frac{1}{N} \sum_i L_i(f(x_i, W), y_i) + \lambda R(W) \leftarrow$ Regularization: Model should be simple \rightarrow work on test data

λ : hyper-parameter

$R(W)$: L2: $R(W) = \sqrt{\sum w_i^2}$

$L_1: R(W) = \sum |w_i|$

Elastic net ($L_1 + L_2$): $R(W) = \sqrt{\sum (w_i^2 + \lambda |w_i|)}$

max norm, dropout, Batch normalization, stochastic depth

Softmax Classifier (Multinomial Logistic Regression)

maximize the log likelihood

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

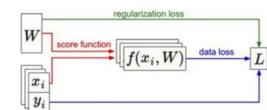
Recap

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

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

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

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



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 # perform 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 \rightarrow Feature Representation \rightarrow Class

Image Features: Motivation

\rightarrow feature transform

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

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 $\rightarrow X \leftarrow np.mean(X, axis=0)$

normalize $\rightarrow X \leftarrow np.std(X, axis=0)$

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

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"

$f(x) = \max(0, \alpha|x|, x)$

Parametric Rectifier (PReLU)

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

Batch normalization

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

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

Usually inserted after FC / conv, before nonlinearity

FC \rightarrow BN \rightarrow tanh \rightarrow FC \rightarrow BN \rightarrow tanh $\rightarrow \dots$

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 \rightarrow loss \nearrow (good)

sanity check

start with small regularization and

find learning rate \rightarrow loss \downarrow

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



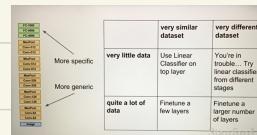
Data Augmentation

Drop Connect

Fractional Max Pooling

Stochastic Depth (dropout layers, crazy but may work)

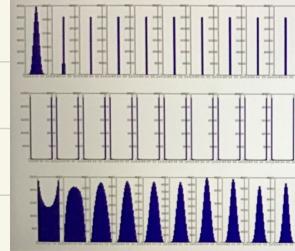
Transfer Learning



CNN pre-trained on ImageNet $\xrightarrow{\text{Object detection}}$
 $\xrightarrow{\text{Image caption}}$

Lecture 7 Training Neural Networks

Last time: Weight Initialization



Initialization too small:
Activations go to zero, gradients also zero,
No learning

Initialization too big:
Activations saturate (for tanh),
Gradients zero, no learning

Initialization just right:
Nice distribution of activations at all layers,
Learning proceeds nicely

Stanford

Optimization

Problems with SGD

local minima/saddle point

saddle points are much more common in high dimension

SGD + Momentum

AdaGrad grad-squared

RMSProp

Adam \rightarrow 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