CS231n

Convolutional Neural Networks for Visual Recognition

Sungchan Yi

September 10, 2019

Contents

1 Introduction

1.1 History of Computer Vision

- ...
- ImageNet: Image classification challenge
- Huge improvement on 2012 - Convolutional Neural Networks (**CNN**)

1.2 CS231n Overview

- Focus on **image classification**
- Object detection, action classification, image captioning ...
- **CNN** !

2 Image Classification

2.1 Image Classification

- Map: Input Image $\rightarrow$ Predetermined Category
- Semantic Gap - computers only see the pixel values of the image
- Many problems that cause the pixel values to change
  - Viewpoints
  - Illumination
  - Deformation (poses)
  - Occlusion (part of the object)
  - Background Clutter (object looks similar to the background)
  - Interclass Variation (objects come in different sizes and shapes)
- Image classification algorithms must be able to handle all these problems

2.2 Attempts to Classify Images

- Finding corners or edges to determine the object
  - Brittle, doesn't work very well

- **Not scalable** to work with other objects

- **Data-Driven Approach**
  1. Collect a dataset of images and labels
  2. Use ML algorithms to train a classifier
  3. Evaluate the classifier on new images

- Data-driven approach is much more general

2.3 Nearest Neighbor

1. Memorize all data and labels
2. Predict the label of the **most similar** training image

- How to compare images? **Distance Metric** !
  - L_1 distance (Manhattan Distance)[1]

$$d_1(I_1, I_2) = \sum_{\text{pixel } p} |I_1^p - I_2^p|$$

- With N examples, training takes $\mathcal{O}(1)$, prediction takes $\mathcal{O}(N)$.
- Generally, we want prediction to be fast, but slow training time is OK
- Only looking at a **single** neighbor may cause problems (outliers, noisy data, etc.)
- Motivation for **kNN**s

2.4 k-Nearest Neighbors

- Take the **majority vote** from k closest points
- L_2 distance (Euclidean Distance)

$$d_2(I_1, I_2) = \sqrt{\sum_{\text{pixel } p} (I_1^p - I_2^p)^2}$$

- L_1 distance depends on the coordinate system L_2 doesn't matter
- Generalizing the distance metric can lead to classifying other objects such as texts

[1] Dumb way to compare, but does reasonable things *sometimes* ...

2.5 Hyperparameters

- **Hyperparameters** are choices about the algorithm that we set, rather than by learning the parameter from data
- In kNNs, the value of k and the distance metric are hyperparameters
- **Problem dependent**. Must go through trial and error to choose the best hyperparameter
- Setting Hyperparameters
  - Split the data into 3 sets. Training set, validation set, and test set
  - Train the algorithm with many different hyperparameters on the training set
  - Evaluate with validation set, choose the best hyperparameter from the results
  - Run it once on the test set, and this result goes on the paper
- Never used on image data ...
  - Slow on prediction
  - Distance metrics on pixels are not informative (Distance metric may not capture the difference between images)
  - Curse of dimensionality - data points increase exponentially as dimension increases, but there may not be enough data to cover the area densely (Need to densely cover the regions of the n-dimensional space, for the kNNs to work well)

2.6 Linear Classification

- Lego blocks - different components, as building blocks of neural networks
- **Parametric Approach**
  - Image x, weight parameters W for each pixel in the image
  - A function $f : (x, W) \to$ numbers giving class scores for each class
  - If input x is an $n \times 1$ vector and there were c classes, W must be $c \times n$ matrix
- Classifier summarizes our knowledge on the data and store it inside the parameter W
- At test time, we use the parameter W to predict
- How to come up with the right structure for f ?
- **Linear** Classifier : $f(x, W) = Wx \ (+b)$
- There may be a bias term b to show preference for some class label

- (Idea) Each row of W will work as a template for matching the input image, and the dot product of each row and the input image vector will give the **similarity** of the input data to the class
- Problems with linear classifier
  - Learning single template for each class
  - If the class has variations on how the class might appear, the classifier averages out all the variations and tries to recognize the object by a single template
  - Using deeper models to remove this restriction will lead to better accuracy
- Linear classifiers draws hyperplanes on the n-dimensional space to classify images
- If hyperplanes cannot be drawn on the space, the linear classifier may struggle (parity problem, multi-modal data)

3 Loss Functions and Optimization

3.1 Motivation

- We saw that W can act as a template for each class
- How do we choose such W ?
- To choose the best W, we should be able to **quantify** the goodness of prediction across the training data

3.2 Loss Functions

- Dataset of examples: $\{(x_i, y_i)\}_{i=1}^{N}$, where x_i is an input data, and $y_i \in \mathbb{Z}$ is a correct label for the data
- Suppose we have a prediction function f, then the **loss over the dataset** is a sum of loss over examples
$$L = \frac{1}{N} \sum_i L_i\big(f(x_i, W), y_i\big)$$
- Now we want to choose a W that **minimizes the loss function**
- **Multiclass SVM loss**
  - Scores vector $s = f(x_i, W)$
  - SVM loss for each data (Hinge loss)
$$L_i = \sum_{j \neq y_i} \max\left(0, s_j - s_{y_i} + 1\right)$$

- As the score for the true category (s_{y_i}) increases, the loss goes down linearly, until it gets above a safety margin because now the example is correctly classified
- Simply put - We are happy[1] if *the score for the correct label is much higher than all the other scores* by some margin[2]
- Minimum loss is 0, maximum loss is ∞
- Quadratic hinge loss - May be used to put more loss on scores that are totally off (Depends on how we want to weigh off different mistakes that the classifier makes)

- Suppose we found a W that makes $L = 0$. But this W is not unique[3], so how do we choose such W ?
- We have only written down loss **in terms of the data**. We only told the classifier to find the W that fits the data
- But in practice, we only care about the performance on the test data

3.3 Regularization

- We add an additional **regularization** term to the loss function, which **encourages** the model to somehow pick a simpler W
- We use a regularization penalty(loss) $R(W)$, then

$$L = \frac{1}{N}\sum_i L_i\big(f(x_i, W), y_i\big) + \lambda R(W)$$

  and the λ is a hyperparameter that trades off between data loss and regularization loss

- Common regularization functions
  - L_2 regularization

$$R(W) = \sum_i \sum_j W_{ij}^2$$

  - L_1 regularization

$$R(W) = \sum_i \sum_j |W_{ij}|$$

- Anything that you do to the model that *penalizes the complexity of the model*
- L_1 regularization thinks less non-zero entries are less complex, L_2 thinks spreading numbers all across entries of W is less complex

[1] We are happy when the loss is small
[2] The constant 1 in the equation can actually be generalized
[3] $2W$ will also give 0 loss

3.4 Softmax Loss

- Multinomial Logistic Regression
- Multiclass SVM - no interpretation for each score
- Consider the scores as *unnormalized* $\log$ *probabilities of the classes*
- $s = f(x_i, W)$

$$\mathrm{P}\left(Y = k \,|\, X = x_i\right) = \frac{e^{s_k}}{\sum_j e^{s_j}} \quad \text{(softmax function)}$$

- Now we have a probability distribution, and we want this to match the distribution that put all it's weight on the correct label
- Loss is the $-\log$ of the probability of the true class[4] (**Cross-Entropy**)

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

- Minimum loss is 0, maximum loss is ∞

3.5 Optimization

- Minimize the loss function!
- Strategy $\#1$: Random Search (...)
- Strategy $\#2$: **Follow the slope** - Use the geometry of the space to find *which direction from here will decrease the loss function*
- **Gradients** ! - Directional derivatives to find out which direction we should take a step, to minimize the loss function
- Use calculus to compute analytic gradients and use them in code

3.6 Gradient Descent

- The direction of the negative gradient is the fastest decrease
- Key Idea: Starting from the initial guess x_0, iteratively compute x_n by the following

$$x_{n+1} = x_n - \gamma \cdot \nabla F(x_n)$$

  and hope to converge to some x

[4] You can view this as the KL divergence between the target and computed distribution, maximum likelihood estimate

- Constant γ is the **step size**, sometimes called the **learning rate**, which is an important hyperparameter
- This is **slow** ... Computing gradients for each iteration is too costly
- **Stochastic Gradient Descent** uses a **minibatch** (subset of training data) to estimate the true gradient

3.7 Image Features

- Feeding raw pixel values don't work well
- Compute various **feature representations** of the image, concatenate them and feed it to the classifier
- What is the best feature transform? [5]
- Example: Color Histogram, Histogram of Oriented Gradients (Local orientation of edges)

[5] For instance, conversion from polar to Cartesian coordinates