CS231n Assignment 2

Questions & Answers

Contents

Fully Connected Nets: 2
- Inline Question 1: 2
- Inline Question 2: 2

Batch Normalization: 2
- Inline Question 1: 2
- Inline Question 2: 3
- Inline Question 3: 3
- Inline Question 4: 4

Dropout : 4
- Inline Question 1: 4
- Inline Question 2: 4

Pytorch: 5
- Describe what you did 5

Fully Connected Nets:

Inline Question 1:

Did you notice anything about the comparative difficulty of training the three-layer network vs. training the five-layer network? In particular, based on your experience, which network seemed more sensitive to the initialization scale? Why do you think that is the case?

5 ლეიერიანი ბევრად უფრო რთული გასაწრვთნელია და ბევრად უფრო მგრძნობიარეა ინიციალიციისადმი. რაც უფრო მეტი ლეიერი აქვს ნეირონულ ქსელს, მით უფრო მეტად პრობლემურია მისი გრადიენტები,შეიძლება vanishing gradient-ის პრობლემას წავაწყდეთ, რის შედეგადაც მოდელი ვერაფერს ისწავლის. რეალურად, 5 ლეიერიანში სიზუსტე არ იცვლება , რაც ალბათ სწორედ გრადიენტის პრობლემის ბრალია, 3 ლეიერიანი კი უფრო მარტივია, რადგან ნაკლებად მგრძნობიარეა.

Inline Question 2:

AdaGrad, like Adam, is a per-parameter optimization method that uses the following update rule:

```
cache += dw**2
w += - learning_rate * dw / (np.sqrt(cache) + eps)
```

John notices that when he was training a network with AdaGrad that the updates became very small, and that his network was learning slowly. Using your knowledge of the AdaGrad update rule, why do you think the updates would become very small? Would Adam have the same issue?

ჯერ განვიხილოთ როგორ მუშაობს AdaGrad, cache-ს ვუმატებთ და ვუტმატებთ გრადიენტის კვადრატს, რის შედეგადაც, ბუნებრივია იგი იზრდება , ძალიან მცირე update აქვს წონებს და ამიტომაც learning rate მცირდება(ან ჩერდება).მოკლედ, რაც უფრო მეტი დრო გადის, მით უფრო იზრდება cache , რაც ძალიან ამცირებს სიჩქარეს (წარსულს არ ივიწყებს ეს ალგორითმი).Adam-ს არ აქვს იგივე პრობლემა, რადგან moving average of gradinets-ს იყენებს, მაგრამ განსხვავება ისაა, რომ ძველი მნიშვნელობები ნელ-ნელა ქრება (ნაკლები მნიშვნელობა ენიჭება ძველ გრადიენტებს და cache არ იზრდება მუდმივად), შესაბამისად მემორი ბალანსირებულია.

Batch Normalization:

Inline Question 1:

Describe the results of this experiment. How does the weight initialization scale affect models with/without batch normalization differently, and why?

ექსპერიმენტი ნათლად აჩვენებს, რომ ბეჩ ნორმალიზაცია ბევრად უფრო მდგრადს ხდის ნეირონულ ქსელს წონებთან მიმართებაში. საბაზისო მოდელი მხოლოდ წონების ფრთხილად და სწორად შერჩევის შემთხვევაში მუშაობს კარგად, თუმცა ბეჩ ნორმალიზაციის შემდეგ მოდელი საკმაოდ კარგად მუშაობს ინიციალიზაციის მნიშვნელობების გაცილებით ფართო

დიაპაზონზე. მოკლედ, ბეჩ ნორმალიზაციის საშუალებით ბევრად უფრო ნაკლებად სენსიტიურია hyperparameters-ების მიმართ შენი მოდელი, რაც ნეირონული ქსელების თრეინინგს უფრო მარტივს ქმნის. ბეჩ ნორმალიზაციის გარეშე, small scales იწვევს vanishing gradients (ძალიან ცუდად სწავლობს ამ დროს), large scales - exploding gradients. ლეიერების ინფუთების სტანდარტიზაციას ახდენს აქტივიზაციის ნორმალიზირებით. შედეგად, ამით ერთგვარად ჩარჩოში აქცევს გრადიენტებს, წონების მიუხედავად.

Inline Question 2:

Describe the results of this experiment. What does this imply about the relationship between batch normalization and batch size? Why is this relationship observed?

უფრო დიდი batch size -> უკეთესი შედეგი. ექსპერიმენტი გვანახებს, რომ ბეჩ ნორმალიზაცია მით უფრო უკეთ მუშაობს, რაც უფრო დიდ ბეჩ საიზებთან გვაქვს საქმე. ეს ურთიერთობა კი გამომდინარეობს იქედან, რომ ბეჩ ნორმალიზაცია ითვლის საშუალოს და ვარიაციას მინი-ბეჩზე. როცა ბეჩის ზომა არის პატარა, სტატისტიკურ მონაცემებს აქვს ხმაური, არ არიან სანდონი, მოკლედ, არასტაბირული ნორმალიზაცია იქმნება ასე. რის შედეგადაც, თრეინინგიც საკმაოდ ცუდია და კონვერგენციაც ნელდება. უფრო დიდი ზომის ბეჩის შემთხევაში, სტატისტიკური მონაცემები ბევრად უფრო სანდოა, უფრო ახლოსაა რეალური დეითას განაწილებასთან და შედეგად, გვაქვს უკეთესი ნორმალიზაცია. ანუ, ამ შემთხვევაში გვაქვს უკეთესი, უფრო სტაბილური სტატისტიკური მონაცემები, სწრაფი კონვერგენცია, უკეთესი თრეინინგი.

Inline Question 3:

Which of these data preprocessing steps is analogous to batch normalization, and which is analogous to layer normalization?

1. Scaling each image in the dataset, so that the RGB channels for each row of pixels within an image sums up to 1.
2. Scaling each image in the dataset, so that the RGB channels for all pixels within an image sums up to 1.
3. Subtracting the mean image of the dataset from each image in the dataset.
4. Setting all RGB values to either 0 or 1 depending on a given threshold.

▼ ბეჩის ნორმალიზაცია - Subtracting the mean image of the dataset from each image in the datase (ნორმალიზაციას ახდენს იმ სტატისტიკურ მონაცემებზე დაყრდნობით, რომელიც ბევრ მაგალითზეა დათვლილი).

▼ ლეიერის ნორმალიზაცია - Scaling each image in the dataset, so that the RGB channels for all pixels within an image sums up to 1 (ნორმალიზაციას ახდენს იმ სტატისტიკურ მონაცემებზე დაყრდნობით, რომელიც ერთ მაგალითზე, თუმცა ამ მაგალითის ყველა feature-ზეა დათვლილი).

Inline Question 4:

When is layer normalization likely to not work well, and why?

1. Using it in a very deep network
2. Having a very small dimension of features
3. Having a high regularization term

Having a very small dimension of features - ამ შემთხვევაში დათვლილი ვარიაცია და საშუალო ნაკლებად სანდოა, შედეგად, გვაქვს ცუდი ნორმალიზაცია, რის გამოც აქტივიზაციების სწორად სტანდარტიზირება არ მოხდება. როცა ნორმალიზირება ხდება მცირე რაოდენობის feature-ზე, აუთლაიერების მნიშვნელობები სხვადასხვანაირ გავლენას იქონიებენ პარამეტრებზე და რეალურად, მთელი ფოინთი სტატისტიკური გაუმჯობესებისა, ქრება.

Dropout :

Inline Question 1:

What happens if we do not divide the values being passed through inverse dropout by p in the dropout layer? Why does that happen?

დროფაუთის დროს ნეირონების ნაწილი შემთხვევით არ აქტიურდება ლერნინგის დროს. მაგალითად, თუ p = 0.6, ეს ნიშნავს, რომ ყოველ ეტაპზე 60% ნეირონები აქტიურია, ხოლო 40% გათიშულია. თუ არ გავყოფთ ამ რიცხვზე,აქტივაციების საშუალო შემცირდება p-ჯერ, შედეგად, ლერნინგის დროს სიგნალებიც p-ჯერ მცირე იქნება. მოკლედ, თუ არ გავყოფთ, სწავლობს პატარა სიგნალებზე, ხოლო ტესტირება კი ხდება შედარებით დიდ სიგნალებზე,შედეგად, ექნება ძალიან ცუდი ფრედიქშენები.

Inline Question 2:

Compare the validation and training accuracies with and without dropout -- what do your results suggest about dropout as a regularizer?

თრეინის შემთხვევაში, დროფაუთის გარეშე უკეთესი მოდელი გვაქვს, თითქმის 100% accuracy-ს აღწევს, ხოლო დროფაუთიანი კი - 90%. ვალიდაციის შემთხვევასი კი პირიქით ხდება - დროფაუთიანი მოდელი ცალსახად უკეთ მუშაობს. გრაფიკი ასახავს დროფაუთის რეგულარიზაციის არსს - იგი ამცირებს მოდელის მიერ დამახსოვრებული თრეინინგ დეითას მოცულობას და ასე ასწავლის უფო ზოგად, robust ფიჩერებს. მოკლედ, თრეინინგ accuracy ეწირება, მაგრამ მოდელი კარგად მუშაობს ახალ მაგალითებზე.

Pytorch:

Describe what you did

In the cell below you should write an explanation of what you did, any additional features that you implemented, and/or any graphs that you made in the process of training and evaluating your network.

დამხმარე ფუნქცია konv_cr() - > Conv2d + BatchNorm + ReLU + pooling, ქმნის კონვოლუციურ ბლოკებს. ვიყენებ ადამის ოპტიმიზაციას, ლერნინგ რეითით 0.002 და რეგულარიზაცია 5e-5. ვიწყებ 96 ფილტრიდან და ვზრდი 384-მდე, მეტი ფილტრი = უფრო მრავალფეროვანი ფიჩერები. MaxPooling-ის გამოყენება პირობაშივეა რეკომენდირებული, იგი გვეხმარება high-dimensional გამოთვლების ბევრად გამარტივებაში, ამასთან გამოვიყენე global average pooling, რომელიც ამცირებს overfitting-ის შანსებს. გამოვიყენებ ასევე ბეჩ ნორმალიზაია, რომელიც გამოყენებულია ყოველი კონვოლუციური ფენის შემდგომ, ასევე დროფაუთი - რომელიც ნეირონულ ქსელს უფრო უმარტივებს სწავლებას.