Weekly Report(Oct.1,2018-Oct.13,2018)ZhangYuandi

Zhang Yuandi
15826138027@163.com

Abstract

In the last weeks, I have learned Lec6 to Lec10 of **CS231n** and Week2 to Week3 of **Neural Networks for Machine Learning**.

1 CS231n

Generally, we need to focus on the following three parts when training our networks:

- One time setup
- Training dynamics
- Evaluation

1.1 Activation Functions

Let's take a look at the functions we usually use, and pay attention to their pros and cons.

- **sigmoid**
  The main problems of it are:
  1. Saturated neurons kill the gradients
  2. Sigmoid outputs are not zero-centered
  3. exp() is a bit compute expensive
- **tanh(x)**
  It fixes the zero-centered problem but still kiss gradients when saturated.
- **ReLU**
  It doesn't saturate and compute efficiently. But it doesn't have zero-centered output. And it would never update if initiated badly.
- **Leaky ReLU and ELU**
  They work much better than sigmoid or tanh. And negative saturation regime of ELU compared with Leaky ReLU adds some robustness but its computation requires exp().
- **Maxout "Neuron"**
  $$\max(\omega_1^T x + b_1, \omega_2^T x + b_2)$$
  It doesn't have the basic form of dot product. It generalizes ReLU and Leaky ReLU and has linear regime and it does not saturate and die. But the problem is the double number of parameters.

1.2 Data Preprocessing

Before training, we need to preprocess data like make them zero-centered or normalized. Otherwise, we might come across problems.

1.3 Weight Initialization

For small numbers, we can use W = 0.01 $\times$ np.random.randn(D, H). And there are also other ways to do weight initialization. But I doesn't quite understand how these works. These functions confuse me.

1.4 Batch Normalization

Input: Valuse of x over a mini-batch:$\mathscr{B} = \{x_1...m\}$; Parameters to be learned: γ, β **Output:** $y_i = BN_{\gamma,\beta}(x_i)$

$\mu_{\mathscr{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$

$\sigma^2_{\mathscr{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu^2_{\mathscr{B}}$

$\hat{x_i} \leftarrow \frac{x_i - \mu_{\mathscr{B}}}{\sqrt{\sigma_{\mathscr{B}}{}^2 + \epsilon}}$

$y_i \leftarrow \gamma\hat{x_i} + \beta \equiv BN_{\gamma,\beta}(x_i)$

And the following steps are babysitting learning and hyperparameter search. Because I don't have much practice, I didn't write details about them down.

1.5 Fancier optimization

Considering some pros of SGD, we can add Momentum to it.

$$v_{t+1} = \rho v_t + \nabla f(x_t)$$

$$x_{t+1} = x_t - \alpha v_{t+1}$$

Usually we want to update in terms of $x_t, \nabla f(x_t)$, we can cange the variable $\tilde{x_t} = x_t + \rho v_t$ and rearrange:

$$v_{t+1} = \rho v_t - \alpha \nabla f(\tilde{x_t})$$

$$\tilde{x_{t+1}} = \tilde{x_t} - \rho v_t + (1+\rho)v_{t+1}$$

AdaGrad adds element-wise scaling of the gradient based on the historical sum of squares in each dimension. They all have learning rate as a hyperparameyer. The learning rate should decay over time. The exponential decay:

$$\alpha = \alpha_0 e^{-kt}$$

The 1/t decay:

$$\alpha = \alpha_0/(1+kt)$$

1.6 Regularization

In each forward pass, we can randomly set some neurons to zero. In this way, we can force the network to have a redundant representation and prevent co-adaptation of features. When testing, use fixed state to normalize.$y = f(x) = E_z[f(x,z)] = \int p(z)f(x,z)dz$. Then we can do transfer learning use what we learned.
CNN and RNN I have learned and written before so I didn't mention them in this report. And where can I find the assignments of CS231n? I can only get the slides for each lecture.

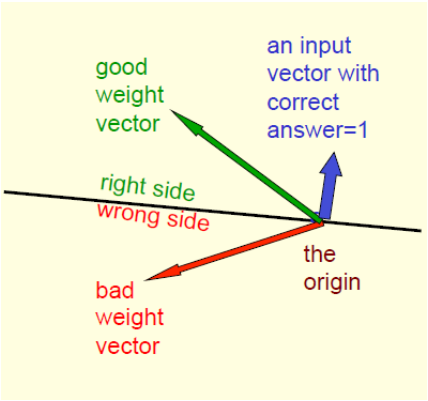
2 Neural Networks for Machine Learning

An overview of main types of neural networks architecture:

- Feed-forward neural networks
- Recurrent networks
- Recurrent neural networks
- Symmetrically connected networks

2.1 Perceptrons

Weight space is important. This space has one dimension per weight. A point un the space represents a particular setting of all the weights. The image below shows some symbolic vectors:



If we have enough features and can choose them by hand, we can do almost anything. But once the hand-coded features have been determined, there are very strong limitations on what a perceptron can learn.

2.2 Learning the weights of a linear neuron

$$y = \sum_i w_i x_i = w^T x$$

But sometimes the weight will get worse. We can define the error as the squared residuals summed over all training cases. Then differantiate to get error derivatives for weights.

2.3 Learning the weights of a logistic output neuron

$$z = b + \sum_i x_i w_i$$

$$y = \frac{1}{1 + e^{-z}}$$

We can use the chain rule to get the derivatives needed for learning the weights of a logistic unit.

$$\frac{\partial E}{\partial w_i} = \sum_n \frac{\partial y^n}{\partial w_i} \frac{\partial E}{\partial y^n} = -\sum_n x_i^n y^n (1 - y^n)(t^n - y^n)$$

3 Plans for Next Week

1. Learn lecture11 to lecture13 of **CS231n**.
2. Learn week4 and week5 of **Neural Networks for Machine Learning**.