experience on mp1&mp2

Yihui He

I’m international exchange student
CS 2nd year undergrad
Xi’an Jiaotong University, China

yihuihe@foxmail.com

May 18, 2016
Overview

1. **mp1**
   - tricks
   - new model

2. **mp2**
   - tricks
   - choosing from different models
   - delving into one model
Goal

- Input: CIFAR 10 image
- Architecture: two-layer neural network
- Output: prediction among 10 classes
tuning hyperparameters

- determine relation\(^1\) between parameter and backpropagation error: linear, $\theta \propto \delta$ or exponential, $\log(\theta) \propto \delta$
- run a grid search (or random search) on a small part of our big dataset

```python
for hidden_neurons in range(150, 600, 50):
    for learning_rate in [1e-3*10**i for i in range(-2, 3)]:
        for norm in [0.5*10**i for i in range(-3, 3)]:
            [loss_history, accuracy] = \%
            train(small_dataset, \%
                  hidden_neurons, learning_rate, norm)
            # dump loss, accuracy history for each setting
            # append highest accuracy of each setting to a .csv
```

\(^1\) stanford cs231n
### Choosing number of hidden neurons

#### Table: top accuracy

<table>
<thead>
<tr>
<th>hidden neurons</th>
<th>learning rate</th>
<th>regularization strength</th>
<th>validation accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>350</td>
<td>0.001</td>
<td>0.05</td>
<td>0.516</td>
</tr>
<tr>
<td>400</td>
<td>0.001</td>
<td>0.005</td>
<td>0.509</td>
</tr>
<tr>
<td>250</td>
<td>0.001</td>
<td>0.0005</td>
<td>0.505</td>
</tr>
<tr>
<td>250</td>
<td>0.001</td>
<td>0.05</td>
<td>0.501</td>
</tr>
<tr>
<td>150</td>
<td>0.001</td>
<td>0.005</td>
<td>0.5</td>
</tr>
<tr>
<td>500</td>
<td>0.001</td>
<td>0.05</td>
<td>0.5</td>
</tr>
</tbody>
</table>
Update methods affect converge rate

1000 iterations, batch size 100

Table: Differences between update methods

<table>
<thead>
<tr>
<th>Accuracy</th>
<th>Train</th>
<th>Validation</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>SGD</td>
<td>.27</td>
<td>.28</td>
<td>.28</td>
</tr>
<tr>
<td>Momentum</td>
<td>.49</td>
<td>.472</td>
<td>.458</td>
</tr>
<tr>
<td>Nesterov</td>
<td>.471</td>
<td>.452</td>
<td>.461</td>
</tr>
<tr>
<td>RMSprop</td>
<td>.477</td>
<td>.458</td>
<td>.475</td>
</tr>
</tbody>
</table>

These update methods can’t make final accuracy higher (sometimes even lower than fine-tuned SGD), but make training much faster.
Accuracy improves about 3%. Only need to change one line in code:

```python
a2 = np.maximum(X.dot(W1)+b1, 0)  
a2 *= (np.random.randn(*a2.shape) < p) / p  # add this line
scores = a2.dot(W2)+b2
```

p : dropout rate (usually chosen from 0.3 0.5 0.7)  
a2 : activation in the second layer.
initialization methods

Three comment initialization for fully connected layer:

- $N(0, 1) \sqrt{1/n}$
- $N(0, 1) \sqrt{2/(n_{in} + n_{out})}$
- $N(0, 1) \sqrt{2/n}$

Significance can’t be seen from our two layers shallow neural net. However, initialization is super important in mp2(deep neural net).
questions about these tricks?
After using tricks we mentioned, accuracy is around 55%, neural network architecture is already fixed.

how do we improve accuracy?
At the very bottom of leaderboard (State-of-the-art is 96%):

<table>
<thead>
<tr>
<th>Score</th>
<th>Title of Paper</th>
<th>Conference/Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>83.96%</td>
<td>Discriminative Learning of Sum-Product Networks</td>
<td>NIPS 2012</td>
</tr>
<tr>
<td>82.9%</td>
<td>Stable and Efficient Representation Learning with Nonnegativity Constraints</td>
<td>ICML 2014</td>
</tr>
<tr>
<td>82.2%</td>
<td>Learning Invariant Representations with Local Transformations</td>
<td>ICML 2012</td>
</tr>
<tr>
<td>82.18%</td>
<td>Convolutional Kernel Networks</td>
<td>arXiv 2014</td>
</tr>
<tr>
<td>82%</td>
<td>Discriminative Unsupervised Feature Learning with Convolutional Neural Networks</td>
<td>NIPS 2014</td>
</tr>
<tr>
<td>80.02%</td>
<td>Learning Smooth Pooling Regions for Visual Recognition</td>
<td>BMVC 2013</td>
</tr>
<tr>
<td>80%</td>
<td>Object Recognition with Hierarchical Kernel Descriptors</td>
<td>CVPR 2011</td>
</tr>
<tr>
<td>79.7%</td>
<td>Learning with Recursive Perceptual Representations</td>
<td>NIPS 2012</td>
</tr>
<tr>
<td>79.6%</td>
<td>An Analysis of Single-Layer Networks in Unsupervised Feature Learning</td>
<td>AISTATS 2011</td>
</tr>
</tbody>
</table>

^{2}\text{rodrigob.github.io}
The new model I used benefit from two preprocessing techniques:

1. PCA whitening
2. Kmeans
3. plug in our two-layer neural network (the original paper use SVM at the end)

---

Learn a feature representation:

1. Extract random patches from unlabeled training images.
2. Apply a pre-processing stage to the patches.
3. Learn a feature-mapping using an unsupervised learning algorithm.

Given the learned feature mapping, we can then perform feature extraction:

1. Break an image into patches.
2. Cluster these patches.
3. Concatenate cluster result of each patch \( \{0,0,...,1,...,0\} \), as new representation of this image.
steps

Input Image

\[ x \]

\[ \text{d channels} \]

\[ f(x) \]

\[ \begin{bmatrix}
    f_1(x) \\
    f_2(x) \\
    \vdots \\
    f_k(x)
\end{bmatrix} \]

Image Representation

\[ \text{K channels} \]

\[ y^{(1)} \]

\[ \frac{(n - w)}{s + 1} \]

Classifier Features

\[ \sum \]

\[ \varphi_1 \]

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]

\[ \varphi_{4K} \]
Use PCA whitening without dimension reduction.
Kmeans visualize

Select 1600 clusters
PCA whitening effect on K-means

Some cluster centroids

(a) K-means (with and without whitening)
(b) GMM (with and without whitening)
(c) Sparse Autoencoder (with and without whitening)
(d) Sparse RBM (with and without whitening)
When should we stop training?

Classification accuracy history

Epoch

Yihui He  mp1&mp2 experience share
<table>
<thead>
<tr>
<th></th>
<th>Naive</th>
<th>Dropout</th>
<th>Preprocessed</th>
</tr>
</thead>
<tbody>
<tr>
<td>hidden nodes</td>
<td>350</td>
<td>500</td>
<td>200</td>
</tr>
<tr>
<td>learning rate</td>
<td>$1 \times 10^{-3}$</td>
<td>$1 \times 10^{-4}$</td>
<td>$5 \times 10^{-4}$</td>
</tr>
<tr>
<td>learning rate Decay</td>
<td>.95</td>
<td>.95</td>
<td>.99</td>
</tr>
<tr>
<td>regularization</td>
<td>L2,0.05</td>
<td>Dropout,.5</td>
<td>Dropout,.3</td>
</tr>
<tr>
<td>Activation</td>
<td>ReLU</td>
<td>Leaky ReLU</td>
<td>ReLU</td>
</tr>
<tr>
<td>Update method</td>
<td>SGD</td>
<td>Momentum,0.9</td>
<td>Momentum,0.95</td>
</tr>
<tr>
<td>Iterations</td>
<td>$1 \times 10^4$</td>
<td>$1 \times 10^4$</td>
<td>$7 \times 10^4$</td>
</tr>
<tr>
<td>Batch size</td>
<td>100</td>
<td>100</td>
<td>128</td>
</tr>
<tr>
<td>Time(min)</td>
<td>15</td>
<td>80</td>
<td>110</td>
</tr>
<tr>
<td>Train accuracy</td>
<td>60%</td>
<td>65%</td>
<td>80%</td>
</tr>
<tr>
<td>Validation</td>
<td>55%</td>
<td>62%</td>
<td>75%</td>
</tr>
<tr>
<td>Test</td>
<td>52%</td>
<td>55%</td>
<td>74%</td>
</tr>
</tbody>
</table>
The result I got is 75%, the original paper get 79%. It’s because I forgot to subtract mean before doing PCA whitening. After fix this bug, accuracy increases to 77%. Much closer.

Huge difference! Mean image subtraction is important.
questions on PCA whitening and Kmeans?
1. mp1
   - tricks
   - new model

2. mp2
   - tricks
   - choosing from different models
   - delving into one model
Goal

- Input: CIFAR 100 image
- Architecture: Not determined
- Output: prediction among 20 classes
tricks that show little difference in my experiments

- Dropout
- Update methods
- PCA whitening and Kmeans
Initialization methods

Becomes more and more important when network goes deep. Recall that we have two problems: gradient vanishing \((\beta_w\beta_\alpha)^p \ll 1\) and gradient exploding \((\beta_w\beta_\alpha)^p \gg 1\):

- Orthogonal initialization
- LUSV initialization
- Xavier initialization
- Kaiming He\(^4\) initialization method (*works best*)

Kaiming He’s initialization method

The idea is scale backward pass signal to 1 at each layer. Implementation is very simple.

\[ std = \sqrt{\frac{2}{Depth_{in} / receptionFieldSize}}. \]

\(Depth_{in}\): number of filters of previous layer comes in.
\(receptionFieldSize\): eg. 3x3
could to make 30 layers deep net converge

\[
\frac{1}{2} \hat{n}_l Var[w_l] = 1 \quad \text{ours}
\]

\[
\hat{n}_l Var[w_l] = 1 \quad \text{Xavier}
\]
More hidden neurons may not show any superior, only increasing time cost.

Adding hidden layers sometimes make things worse. Kaiming He⁵ found that about 30% redundant computation comes from the fully connected layers. Fully connected layer is less efficient than conv layer.

One solution: replace the fully connected layer between the last conv layer and hidden layer with global average pooling.

New model

How do we improve it?
To my knowledge, I found these possible way to improve accuracy:

- XNOR net\textsuperscript{6}
- mimic learning\textsuperscript{7} (model compression)
- switch to faster framework(mxnet\textsuperscript{8}), rather than tensorflow :)
- residual neural network\textsuperscript{9}


\textsuperscript{7}Jimmy Ba and Rich Caruana. “Do deep nets really need to be deep?” In: \textit{Advances in neural information processing systems.} 2014, pp. 2654–2662.


\textsuperscript{9}He et al., “Deep Residual Learning for Image Recognition”.

what is XNOR net?

\[ x^b = \begin{cases} 
+1 & \text{with probability } p = \sigma(x), \\
-1 & \text{with probability } 1 - p,
\end{cases} \]

\( \sigma(x) \), activation function
# XNOR net speed

<table>
<thead>
<tr>
<th>Network Variations</th>
<th>Operations used in Convolution</th>
<th>Memory Saving (Inference)</th>
<th>Time Saving on CPU (Inference)</th>
<th>Accuracy on ImageNet (AlexNet)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Standard Convolution</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Real-Value Inputs</td>
<td>+, −, ×</td>
<td>1x</td>
<td>1x</td>
<td>%56.7</td>
</tr>
<tr>
<td>Real-Value Weights</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Binary Weight</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Real-Value Inputs</td>
<td>+, −</td>
<td>~32x</td>
<td>~2x</td>
<td>%53.8</td>
</tr>
<tr>
<td>Binary Weights</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>BinaryWeight Binary Input (XNOR-Net)</strong></td>
<td>XNOR, bitcount</td>
<td>~32x</td>
<td>~58x</td>
<td>%44.2</td>
</tr>
</tbody>
</table>
what is mimic learning, basic idea

With a high accuracy teacher model, we not only tell the student neural network which label is true or wrong (0,1), also tell the student neural network some classes are close to each other and some are not.

Example

In CIFAR10, truck and car are in different classes, however, they share some common features. So when there's a car in the image, truck's probability is also high. Teacher model helps student model jointly learn these two concepts.
what is mimic learning, details

High level overview:

1. train a state-of-the-art neural network
2. get the $log(p_{deep}(y|X))$ for training set
3. replace the softmax layer of shallow neural network with a linear regressor
4. minimize log probability error:
   $$J(\theta) = \sum_{y \in labels} (log(p(y|X)) - log(p_{deep}(y|X)))^2$$
5. put back softmax layer
6. fine tuning
result from paper

![Graph showing the accuracy on TIMIT Test Set against the number of parameters (millions) for different models: ShallowNet, DeepNet, ShallowMimicNet, Convolutional Net, and Ensemble of CNNs.](image)
residual neural network

Basic idea:
Learn $f(x) - x$ instead of $f(x)$.
residual neural network

The only two differences between residual neural network and ConvNet:

1. no hidden layers
2. use shortcut module, which allows a layer skip the layer on top of it, and pass its value to the next layer.
Yihui He  

Figure 2. Residual learning: a building block.
traditional convolutional neural network
parameters on each layers

A commonly used VGGnet:

<table>
<thead>
<tr>
<th>Layer</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>conv3-64 x 2</td>
<td>38,720</td>
</tr>
<tr>
<td>conv3-128 x 2</td>
<td>221,440</td>
</tr>
<tr>
<td>conv3-256 x 3</td>
<td>1,475,328</td>
</tr>
<tr>
<td>conv3-512 x 3</td>
<td>5,899,776</td>
</tr>
<tr>
<td>conv3-512 x 3</td>
<td>7,079,424</td>
</tr>
<tr>
<td>fc1</td>
<td>102,764,544</td>
</tr>
<tr>
<td>fc2</td>
<td>16,781,312</td>
</tr>
<tr>
<td>fc3</td>
<td>4,097,000</td>
</tr>
<tr>
<td>TOTAL</td>
<td>138,357,544</td>
</tr>
</tbody>
</table>

Notice that 74% parameters are from fc1, however, actual accuracy improvement is from conv layers. Residual neural network, instead, uses all convolution layers and a global average pooling layer at the end.

## Architecture Comparison

**Table:** Differences between three architectures

<table>
<thead>
<tr>
<th></th>
<th>AlexNet</th>
<th>Kmeans</th>
<th>ResNet</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameters</td>
<td>1M</td>
<td>.4M</td>
<td>.13M</td>
</tr>
<tr>
<td>Layers</td>
<td>7</td>
<td>3</td>
<td>14</td>
</tr>
<tr>
<td>Learning rate</td>
<td>.1</td>
<td>$5 \times 10^{-4}$</td>
<td>.1</td>
</tr>
<tr>
<td>Regularization</td>
<td>L2</td>
<td>Dropout,.3</td>
<td>None</td>
</tr>
<tr>
<td>Epoch</td>
<td>10</td>
<td>140</td>
<td>18</td>
</tr>
<tr>
<td>Batch size</td>
<td>128</td>
<td>128</td>
<td>256</td>
</tr>
<tr>
<td>Time(min)</td>
<td>180</td>
<td>80</td>
<td>180</td>
</tr>
<tr>
<td>CIFAR10 Acc</td>
<td>82%</td>
<td>75%</td>
<td>84%</td>
</tr>
<tr>
<td>Train accuracy</td>
<td>90%</td>
<td>80%</td>
<td>86%</td>
</tr>
<tr>
<td>Test</td>
<td>56%</td>
<td>56%</td>
<td>63%</td>
</tr>
</tbody>
</table>
why residual neural network more efficient?

1. Less trainable parameters than neural networks that have the same depth.
2. Lower layer response.
3. Shortcut module allows error $\delta$ directly pass to previous layers, instead of going through each layer. It implicitly makes a deeper network shallower, so it won’t suffer much from gradient vanishing and exploding. It makes training faster.
Code, report and papers can be access via github:

mp1

mp2
https://github.com/yihui-he/Residual-neural-network
questions?