COMS W4701: Artificial Intelligence
Written Homework 4

Huibo Zhao (hz2480)

December 5, 2017

Problem 1

Why does AlexNet use RelUs instead of sigmoid activation functions?

Generally speaking, ReLUs train faster and we indeed need such a fast learning network because of the large datasets we have. As stated in article, the standard way to model a neuron's output is tanh or sigmoid function and they are slower than neurons with ReLUs. When reaching the same training error, the neurons with ReLUs take fewer iterations.

What is drop-out? What problem is addressed by this technique and why does it help system performance?

According to the article, dropout refers to a technique "consisting of setting to zero the output of each hidden neuron with probability 0.5". The neurons that are dropped out will not be engaged in forward pass and back propagation anymore.

It reduces overfitting.

It helps system performance. Normally, we could combine the predictions of many different models for reducing test errors but that is an expensive method for big neural networks. Instead, dropout will only cost approximately a factor of two for training.

What is data augmentation? What problem is addressed by this technique and why does it help system performance?

Data augmentation refers to "artificially enlarging the dataset using label-preserving transformations". In this article, two forms of data augmentation are introduced. The first form involves extracting random patches and training network on these extracted patches and increase the size of our training set. During the test time, the network makes prediction by extracting ten patches (five patches plus their horizontal reflections) and averaging the predictions on the ten patches. The second form involves altering RGB intensities on training data. It adds multiples of the found principal components to each training image.

It also reduces overfitting.

As stated in article, the second form reduces the top-1 error rate by over 1 percent. Additionally, both the two forms produce transformed with little computation and we don't need to store those transformed images on disk. We consider it to be very little computation because the transformed images are generated on CPU while the training process is done on GPU with previous batch of images.

Part 2

Level 1:

- choose X_1:	- choose X_2:	choose X_3:
$H(<\frac{3}{5}, \frac{2}{5}>) = 0.97$		
$H(<\frac{2}{4}, \frac{2}{4}>) = 1$	$H(<\frac{2}{2}, \frac{0}{2}>) = 0$	$H(<\frac{2}{3}, \frac{1}{3}>) = 0.92$
$H(<\frac{1}{1}, \frac{0}{1}>) = 0$	$H(<\frac{2}{3}, \frac{1}{3}>) = 0.92$	$H(<\frac{1}{2}, \frac{1}{2}>) = 1$
$Gain = 0.97 - \frac{4}{5} \times 1 = 0.17$	$Gain = 0.97 - \frac{3}{5} \times 0.92 = 0.42$	$Gain = 0.97 - \frac{3}{5} \times 0.92 - \frac{2}{5} \times 1 = 0.02$

We chose X_2 as our root node.

On yes side of X_2 ($X_2 = 1$)

$H(<\frac{1}{3}, \frac{2}{3}>) = 0.92$

- choose X_1:	- choose X_3:
$H(<\frac{2}{2}, \frac{0}{2}>) = 0$	$H(<\frac{1}{1}, \frac{0}{1}>) = 0$
$H(<\frac{1}{1}, \frac{0}{1}>) = 0$	$H(<\frac{1}{2}, \frac{1}{2}>) = 1$
$Gain = 0.92 - 0 = 0.92$	$Gain = 0.92 - \frac{2}{3} \times 1 = 0.25$

We chose X_1

On no side of X_2 ($X_2 = 0$)

$H(<\frac{2}{2}, \frac{0}{2}>) = 0$, no need to build further decision tree

The final decision tree looks like this

$$\begin{tikzcd}
 & X_2 \arrow[dl, "1"'] \arrow[dr, "0"] & \\
X_1 \arrow[d, "1"'] \arrow[dr, "0"] & & \boxed{0} \\
\boxed{1} & \boxed{0} &
\end{tikzcd}$$

Part 3

X_1	X_2	W_0	W_1	W_2	Sum	output	target y
0	1	0	0	0	0	0	1
1	1	1	0	1	2	1	0
0	0	0	-1	0	0	0	1
1	0	1	-1	0	0	0	1
0	1	2	0	0	2	1	1
1	1	2	0	0	2	1	0
0	0	1	-1	-1	1	1	1
1	0	1	-1	-1	0	0	1
0	1	2	0	-1	1	1	1
1	1	2	0	-1	1	1	0
0	0	1	-1	-2	1	1	1
1	0	1	-1	-2	0	0	1
0	1	2	0	-2	0	0	1
1	1	3	0	-1	2	1	0
0	0	2	-1	-2	2	1	1
1	0	2	-1	-2	1	1	1
0	1	2	-1	-2	0	0	1
1	1	3	-1	-1	1	1	0
0	0	2	-2	-2	2	1	1
1	0	2	-2	-2	0	0	1
0	1	3	-1	-2	1	1	1
1	1	3	-1	-2	0	0	0
0	0	3	-1	-2	3	1	1
1	0	3	-1	-2	2	1	1

✓

final weight = $W_0 : 3$ $W_1 : -1$ $W_2 : -2$

Problem 4

(a)

Suppose the output of the hidden layer with two neurons is g(x) = $w_1x+d_1+w_2x+d_2$, then the output for output layer is $w_3g(x)+d_3$ when can be simplified as $(w_3w_1+w_3w_2)x+w_3d_1+w_3d_2+d_3$. If we let w = $w_3w_1+w_3w_2$ and d = $w_3d_1+w_3d_2+d_3$, this is the network with no hidden units but computes the same function.

(b)

When we have more than n hidden layers, it is the same thing because of the linear activation function. Basically, when we applied a linear activation function to a linear function, the output is still a linear form. The weight factor w will "cumulate in front of a" and the rest of terms will be added up together which can be regarded as a linear form

(c)

Before transformation, on the hidden layer, each neuron has n weights corresponding to the n inputs, and the hidden layer has total h neurons, so it has hn number of weights. On the output layer, each node has h weights corresponding to the h neurons, and the output layer has total n nodes, so it has hn number of weights. In total , it has 2hn number of weights.

After transformation, we have no hidden layer. For output layer, each node has n weights corresponding to the n inputs, and we have n nodes. So in total we have n^2 number of weights

When h << n, we can see that the network after transformation has much more number of weights than the original network. This result means using linear activation functions for network is generally not a good idea.