Linear Decoders

Sparse Autoencoder Recap

In the sparse autoencoder, we had 3 layers of neurons: an input layer, a hidden layer and an output layer. In our previous description of autoencoders (and of neural networks), every neuron in the neural network used the same activation function. In these notes, we describe a modified version of the autoencoder in which some of the neurons use a different activation function. This will result in a model that is sometimes simpler to apply, and can also be more robust to variations in the parameters.

Recall that each neuron (in the output layer) computed the following:


\begin{align}
z^{(3)} &= W^{(2)} a^{(2)} + b^{(2)} \\
a^{(3)} &= f(z^{(3)})
\end{align}


where *a*(3) is the output. In the autoencoder, *a*(3) is our approximate reconstruction of the input *x* = *a*(1).

Because we used a sigmoid activation function for *f*(*z*(3)), we needed to constrain or scale the inputs to be in the range [0,1], since the sigmoid function outputs numbers in the range[0,1]. While some datasets like MNIST fit well with this scaling of the output, this can sometimes be awkward to satisfy. For example, if one uses PCA whitening, the input is no longer constrained to [0,1] and it's not clear what the best way is to scale the data to ensure it fits into the constrained range.

Linear Decoder

One easy fix for this problem is to set *a*(3) = *z*(3). Formally, this is achieved by having the output nodes use an activation function that's the identity function *f*(*z*) = *z*, so that *a*(3) =*f*(*z*(3)) = *z*(3). This particular activation function f(\cdot) is called the **linear activation function**(though perhaps "identity activation function" would have been a better name). Note however that in the *hidden* layer of the network, we still use a sigmoid (or tanh) activation function, so that the hidden unit activations are given by (say) \textstyle a^{(2)} = \sigma(W^{(1)}x + b^{(1)}), where \sigma(\cdot) is the sigmoid function, *x* is the input, and *W*(1) and *b*(1) are the weight and bias terms for the hidden units. It is only in the *output* layer that we use the linear activation function.

An autoencoder in this configuration--with a sigmoid (or tanh) hidden layer and a linear output layer--is called a **linear decoder**. In this model, we have \hat{x} = a^{(3)} = z^{(3)} = W^{(2)}a + b^{(2)}. Because the output \hat{x}  is a now linear function of the hidden unit activations, by varying *W*(2), each output unit *a*(3) can be made to produce values greater than 1 or less than 0 as well. This allows us to train the sparse autoencoder real-valued inputs without needing to pre-scale every example to a specific range.

Since we have changed the activation function of the output units, the gradients of the output units also change. Recall that for each output unit, we had set set the error terms as follows:


\begin{align}
\delta_i^{(3)}
= \frac{\partial}{\partial z_i} \;\;
        \frac{1}{2} \left\|y - \hat{x}\right\|^2 = - (y_i - \hat{x}_i) \cdot f'(z_i^{(3)})
\end{align}


where *y* = *x* is the desired output, \hat{x} is the output of our autoencoder, and f(\cdot) is our activation function. Because in the output layer we now have *f*(*z*) = *z*, that implies *f*'(*z*) = 1 and thus the above now simplifies to:


\begin{align}
\delta_i^{(3)} = - (y_i - \hat{x}_i)
\end{align}


Of course, when using backpropagation to compute the error terms for the *hidden* layer:


\begin{align}
\delta^{(2)} &= \left( (W^{(2)})^T\delta^{(3)}\right) \bullet f'(z^{(2)})
\end{align}


Because the hidden layer is using a sigmoid (or tanh) activation *f*, in the equation above f'(\cdot)should still be the derivative of the sigmoid (or tanh) function.

# Exercise:Learning color features with Sparse Autoencoders

|  |
| --- |
| Contents  [[hide](javascript:toggleToc())]   * [1 Learning color features with Sparse Autoencoders](http://deeplearning.stanford.edu/wiki/index.php/Exercise:Learning_color_features_with_Sparse_Autoencoders#Learning_color_features_with_Sparse_Autoencoders)   + [1.1 Dependencies](http://deeplearning.stanford.edu/wiki/index.php/Exercise:Learning_color_features_with_Sparse_Autoencoders#Dependencies)   + [1.2 Learning from color image patches](http://deeplearning.stanford.edu/wiki/index.php/Exercise:Learning_color_features_with_Sparse_Autoencoders#Learning_from_color_image_patches)   + [1.3 Step 0: Initialization](http://deeplearning.stanford.edu/wiki/index.php/Exercise:Learning_color_features_with_Sparse_Autoencoders#Step_0:_Initialization)   + [1.4 Step 1: Modify your sparse autoencoder to use a linear decoder](http://deeplearning.stanford.edu/wiki/index.php/Exercise:Learning_color_features_with_Sparse_Autoencoders#Step_1:_Modify_your_sparse_autoencoder_to_use_a_linear_decoder)   + [1.5 Step 2: Learn features on small patches](http://deeplearning.stanford.edu/wiki/index.php/Exercise:Learning_color_features_with_Sparse_Autoencoders#Step_2:_Learn_features_on_small_patches) |

## Learning color features with Sparse Autoencoders

In this exercise, you will implement a [linear decoder](http://deeplearning.stanford.edu/wiki/index.php/Linear_Decoders) (a sparse autoencoder whose output layer uses a linear activation function). You will then apply it to learn features on color images from the STL-10 dataset. These features will be used in an later [exercise on convolution and pooling](http://deeplearning.stanford.edu/wiki/index.php/Exercise:Convolution_and_Pooling) for classifying STL-10 images.

In the file [linear\_decoder\_exercise.zip](http://ufldl.stanford.edu/wiki/resources/linear_decoder_exercise.zip) we have provided some starter code. You should write your code at the places indicated "YOUR CODE HERE" in the files.

For this exercise, you will need to copy and modify **sparseAutoencoderCost.m** from the [sparse autoencoder exercise](http://deeplearning.stanford.edu/wiki/index.php/Exercise:Sparse_Autoencoder).

### Dependencies

You will need:

* sparseAutoencoderCost.m (and related functions) from [Exercise:Sparse Autoencoder](http://deeplearning.stanford.edu/wiki/index.php/Exercise:Sparse_Autoencoder" \o "Exercise:Sparse Autoencoder)

The following additional file is also required for this exercise:

* [Sampled 8x8 patches from the STL-10 dataset (stl10\_patches\_100k.zip)](http://ufldl.stanford.edu/wiki/resources/stl10_patches_100k.zip)

*If you have not completed the exercise listed above, we strongly suggest you complete it first.*

### Learning from color image patches

In all the exercises so far, you have been working only with grayscale images. In this exercise, you will get to work with RGB color images for the first time.

Conveniently, the fact that an image has three color channels (RGB), rather than a single gray channel, presents little difficulty for the sparse autoencoder. You can just combine the intensities from all the color channels for the pixels into one long vector, as if you were working with a grayscale image with 3x the number of pixels as the original image.

### Step 0: Initialization

In this step, we initialize some parameters used in the exercise (see starter code for details).

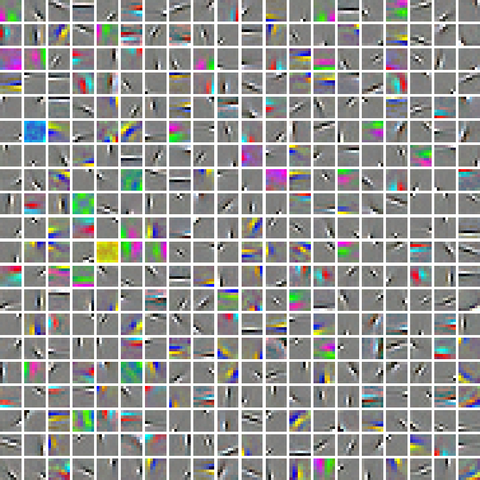
### Step 1: Modify your sparse autoencoder to use a linear decoder

Copy sparseAutoencoderCost.m to the directory for this exercise and rename it tosparseAutoencoderLinearCost.m. Rename the function sparseAutoencoderCost in the file tosparseAutoencoderLinearCost, and modify it to use a [linear decoder](http://deeplearning.stanford.edu/wiki/index.php/Linear_Decoders). In particular, you should change the cost and gradients returned to reflect the change from a sigmoid to a linear decoder. After making this change, check your gradients to ensure that they are correct.

### Step 2: Learn features on small patches

You will now use your sparse autoencoder to learn features on a set of 100,000 small 8x8 patches sampled from the larger 96x96 STL-10 images (The [STL-10 dataset](http://www.stanford.edu/~acoates/stl10/) comprises 5000 training and 8000 test examples, with each example being a 96x96 labelled color image belonging to one of ten classes: airplane, bird, car, cat, deer, dog, horse, monkey, ship, truck.)

The code provided in this step trains your sparse autoencoder for 400 iterations with the default parameters initialized in step 0. This should take around 45 minutes. Your sparse autoencoder should learn features which when visualized, look like edges and "opponent colors," as in the figure below.

[](http://deeplearning.stanford.edu/wiki/index.php/File:CNN_Features_Good.png)

If your parameters are improperly tuned (the default parameters should work), or if your implementation of the autoencoder is buggy, you might instead get images that look like one of the following:

|  |  |
| --- | --- |
| [Cnn Features Bad1.png](http://deeplearning.stanford.edu/wiki/index.php/File:Cnn_Features_Bad1.png) | [Cnn Features Bad2.png](http://deeplearning.stanford.edu/wiki/index.php/File:Cnn_Features_Bad2.png) |

The learned features will be saved to STL10Features.mat, which will be used in the later [exercise on convolution and pooling](http://deeplearning.stanford.edu/wiki/index.php/Exercise:Convolution_and_Pooling).