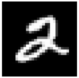
Convolutional Neural Networks For Image Classification

Alex Williams

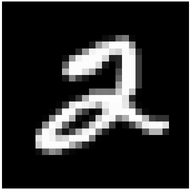
March 16, 2017

The Dataset: MNIST

- Consists of grayscale images of the digits 0 through 9.
- Images have dimension 28×28 pixels.
- Grayscale images are really just a two dimensional array of integers with entries that have values from 0-255.
- Actual size:



- Enlarged:



The Dataset: MNIST

```
[[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  13  25 100 122   7   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0  33 151 208 252 252 252 146   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0  40 152 244 252 253 224 211 252 232  40   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0  15 152 239 252 252 252 216  31  37 252 252  60   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0  96 252 252 252 252 217  29   0  37 252 252  60   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0 181 252 252 220 167  30   0   0  77 252 252  60   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0  26 128  58  22   0   0   0   0 100 252 252  60   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 157 252 252  60   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0 110 121 122 121 202 252 194   3   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0  10  53 179 253 253 255 253 253 228  35   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   5  54 227 252 243 228 170 242 252 252 231 117   6   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   6  78 252 252 125  59   0  18 208 252 252 252 252  87   7   0   0   0   0   0   0]
 [  0   0   0   0   0   0   5 135 252 252 180  16   0  21 203 253 247 129 173 252 252 184  66  49  49   0   0   0]
 [  0   0   0   0   0   3 136 252 241 106  17   0  53 200 252 216  65   0  14  72 163 241 252 252 223   0   0   0]
 [  0   0   0   0   0 105 252 242  88  18  73 170 244 252 126  29   0   0   0   0   0  89 180 180  37   0   0   0]
 [  0   0   0   0   0 231 252 245 205 216 252 252 252 124   3   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0 207 252 252 252 252 178 116  36   4   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0  13  93 143 121  23   6   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]]
```

The Task: Multiclass Classification

- We are given a training and a testing set of examples where each example example belongs to one of 10 classes.
- The class labels are $\{0, 1, 2, \ldots, 9\}$. Each image has a label that corresponds the digit that the image contains: An image that contains the digit 3 has the label 3.
- We want to produce a function that takes an image as input and predicts its class. We'll use two types of functions to do this: **feed forward neural networks** and **convolutional neural networks**. I'll describe these models using the following slides and then I'll show you code the code for them at the end.

Feed Forward Network

- Let's use a FFN with three hidden layers with 500 units each. The output layer will have 10 units, one for each class.
- To feed a 28×28 image into our network, we'll flatten the 2-D array into a column vector with 784 entries:
  - $\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{784} \end{bmatrix} \in \mathbb{R}^{784}$
- Our network is of the form $\hat{y}(\boldsymbol{x}) = f^{(4)}(f^{(3)}(f^{(2)}(f^{(1)}(\boldsymbol{x}))))$, where:
  - $f^{(4)}$ is the output layer, and $f^{(1)}$ is the first hidden layer.

Feed Forward Networks

- $f^{(1)}$ is of the form:

$$
\begin{aligned}
f^{(1)}(\boldsymbol{x}) &= g(\boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}) \\
&= g\left(\begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,784} \\ w_{2,1} & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ w_{500,1} & w_{500,2} & \cdots & w_{500,784} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{784} \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_{500} \end{bmatrix} \right) \\
&= \begin{bmatrix} g(\sum_{j=1}^{784} w_{1,j}x_j + b_1^{(1)}) \\ \vdots \\ g(\sum_{j=1}^{784} w_{500,j}x_j + b_{500}^{(1)}) \end{bmatrix}
\end{aligned}
$$

- g is a nonlinear function like the sigmoid function.
- The other hidden layers are defined in a similar way.

Feed Forward Networks

- $f^{(4)}$ is of the form:

$$
\begin{aligned}
f^{(4)}(\boldsymbol{z}^{(3)}) &= \mathrm{softmax}(\boldsymbol{W}^{(4)}\boldsymbol{z}^{(3)} + \boldsymbol{b}^{(4)}) \\
&= \mathrm{softmax}\left(\begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,500} \\ w_{2,1} & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ w_{10,1} & w_{10,2} & \cdots & w_{10,500} \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_{500} \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_{10} \end{bmatrix} \right) \\
&= \begin{bmatrix} \mathrm{softmax}_1(\sum_{j=1}^{500} w_{1,j}x_j + b_1^{(4)}) \\ \vdots \\ \mathrm{softmax}_{10}(\sum_{j=1}^{500} w_{10,j}x_j + b_{10}^{(4)}) \end{bmatrix} \\
&= \boldsymbol{z}^{(4)}
\end{aligned}
$$

- $\boldsymbol{z}^{(3)}$ is the vector containing the outputs form the 3rd hidden layer.

Feed Forward Networks

- Continuing from the last slide:

$$
\begin{aligned}
f^{(4)}(\boldsymbol{x}) &= \mathrm{softmax}(\boldsymbol{W}^{(4)}\boldsymbol{z}^{(3)} + \boldsymbol{b}^{(4)}) \\
&= \begin{bmatrix} \mathrm{softmax}(\sum_{j=1}^{500} w_{1,j}x_j + b_1^{(4)})_1 \\ \vdots \\ \mathrm{softmax}(\sum_{j=1}^{500} w_{10,j}x_j + b_{10}^{(4)})_{10} \end{bmatrix} \\
&= \boldsymbol{z}^{(4)}
\end{aligned}
$$

- Let $\boldsymbol{a}^{(4)} = \boldsymbol{W}^{(4)}\boldsymbol{z}^{(3)} + \boldsymbol{b}^{(4)}$.
- $\mathrm{softmax}(\boldsymbol{a}^{(4)})_i = \frac{e^{\boldsymbol{a}_i^{(4)}}}{\sum_{j=1}^{10} e^{\boldsymbol{a}_j^{(4)}}}$ for $i \in \{1, \ldots, 10\}$.
- The softmax function ensures that the entries of $\boldsymbol{z}^{(4)}$ are decimals between 0 and 1 that sum to 1.

Feed Forward Networks

- The ith entry of the vector that is produced by the softmax output layer is interpreted as the probability that the image $\boldsymbol{x}$ that is fed to the network is in class $i-1$. So, for example, $\boldsymbol{z}_1^{(4)} = softmax(\boldsymbol{a}^{(4)})_1$ is the probability that the the input image is a 0.
- $f^{(4)}(\boldsymbol{x}) = \boldsymbol{z}^{(4)} = \begin{bmatrix} \text{Probability that } \boldsymbol{x} \text{ is a zero} \\ \vdots \\ \text{Probability that } \boldsymbol{x} \text{ is a nine} \end{bmatrix}$
- We predict that the input image belongs to the class that has the highest probability in the output layer.

Convolutional Neural Networks

- We can also use a convolutional neural network for the same task.
- Convolutional neural networks are state of the art when it comes to machine learning tasks involving images and video.
- A convolutional neural network uses two special layers that aren't used in feed forward networks: **convolutional layers** and **pooling layers**.

Convolutional Layers

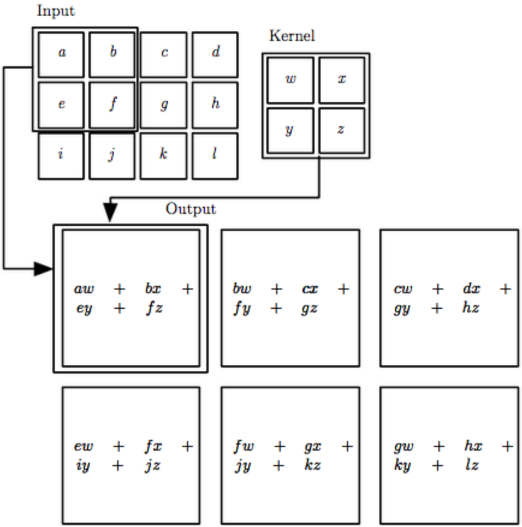
- I'll describe the first layer of a convolutional neural network similar to the one that I'll show you the code for later.

Convolutional Layers

- In the first layer of the CNN, we accept an MNIST image as a 28×28 array (We don't flatten it into a vector).
- We take a 5×5 grid of weights called a kernel and we shift it across the image array.
- At each location we take the elementwise product of the 5×5 kernel and the 5×5 subarray of the image that the kernel is located over. Then we sum the elements of elementwise product to get a scalar value. We'll call this scalar a.
- We'll then add a bias b to a and plug the sum into a nonlinear function (like the sigmoid function): $g(a+b)$.
- This convolutional layer has an output that is a 24×24 array. $g(a+b)$ is one value in this output array. Each element in the output array corresponds to one of the locations of the kernel.

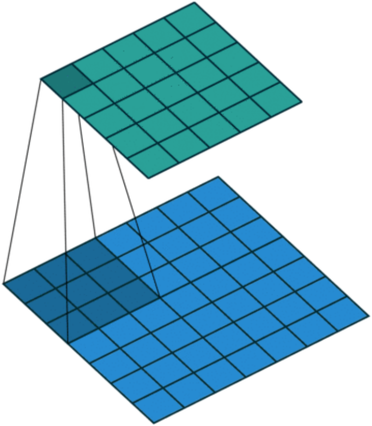
Convolutional Layers

- A simple example...



Convolutional Layers

- Here's an animation that helps illustrate what is occurring.



Convolutional Layers

- Now, why does it make sense to use this type of layer with image data? Notice that similar low level features can occur anywhere in an image. So, for instance, curves, edges, and small patches of a particular color can occur in all areas of an image. Using a convolutional layer allows us to use the same relatively small set of weights at all the different possible positions of the kernel to detect such features.

Pooling Layers

- Pooling layers are used in convolutional neural network to reduce the number of weights and computations that are needed.
- They are often applied after convolutional layers.
- I'll now describe a pooling layer applied after the convolutional layer that we just discussed.

Pooling Layers

- Consider the 24×24 array that we produced as output from the convolutional layer.
- We'll select a pool size of 2×2. This means that we'll divide up the 24×24 array into squares of dimension 2×2. Out of the four pixels that are in each 2×2 square, we select the maxmimum pixel value and include this in the pooling layer's 12×12 output array. Note that we have reduced the size of the input array by half.

Pooling Layers

- Consider the following 2-D array:

1	5	0	1	1	9
0	3	-1	0	0	1
7	-1	5	0	0	1
3	3	-1	10	10	1
1	5	0	2	2	10
5	3	4	3	5	2

- The next picture shows the output array after max pooling:

5	1	9
7	10	10
5	4	10

Convolutional Networks

- In a CNN, after several convolutional layers and pooling layers, it is common to flatten the output of these layers into a column vector and then use several fully connected layers (the same type of layers used in feed forward nets).
- For our MNIST task our convolutional net will use the same output layer as our feed forward net.

Loss Function and Training

- For both the feed forward network and the convolutional neural network we'll use a loss function called the **multiclass log loss**.
- Suppose we had a training set with m examples, where y_i is the actual label that the ith image has.
- Then $\hat{y}_{y_i}$ denotes the probability that the model gives for the ith training example having the actual label y_i.
- The multiclass log loss is: $L(\boldsymbol{y}_{train}|\boldsymbol{X}_{train}, \boldsymbol{W}, \boldsymbol{b}) = -\sum_{i=1}^{m}[\ln(\hat{y}_{y_i})]$
  - By minimizing the multiclass log loss, we maximize the likelihood of observing the data in our training set.
- For our training procedure we can use stochastic gradient descent or one of its variants. Gradients are computed via backpropagation.

Keras + TensorFlow

- TensorFlow
  - Open source, developed by Google
  - By far the most popular deep learning framework
    - 50,573 stars, 23,609 forks. 9th most forked, and 9th most starred repo on GitHub
  - Low level code is written in C++, has a Python front end
- Keras
  - A simplified Python front end for TensorFlow and Theano
  - Will soon be added to the core TensorFlow code

MNIST Code

- Code
- TensorBoard
- Accuracy Difference

Sources

- Convolution Animation
- Simple Convolution Example