DEEP LEARNING

Report Homework 3

Enrico Cecchetti

253823

Introduction

Deep learning (also known as deep structured learning or hierarchical learning) is part of a broader family of machine learning methods based on learning data representations, as opposed to task-specific algorithms. Learning can be supervised, semi-supervised or unsupervised.

Deep learning models are inspired by information processing and communication pattern in Neuron System, specifically in Neuron Cells behavior.

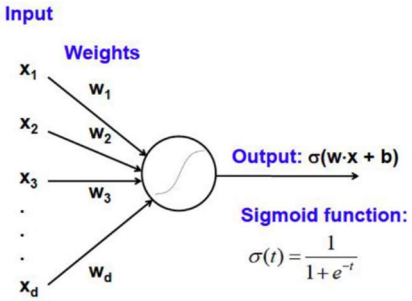
Model's Differences

In the previous model studied we tends to classify features without learning the model, only training it (e.g. SVM). Deep learning approach instead permits to learn all the feature hierarchy, from pixels to the classifier. Infact, every layer extract some feature from the output of the previous one, that permits to train all the layers jointly.

In this report we will focus on the 2 models used in the Homework, so the: Neural Network and the Convolutional Neural Network.

Neural Network

A Neural Network(NN) is either a biological neural network, made up of real biological neurons, or an artificial neural network, for solving artificial intelligence (AI) problems.
The connections of the biological neuron are modeled as weights. A positive weight reflects an excitatory connection, while negative values mean inhibitory connections. All inputs are modified by a weight and summed. This activity is referred as a linear combination.
Finally, an activation function controls the amplitude of the output. For example, an acceptable range of output is usually between 0 and 1, or it could be −1 and 1.



Neural Network: Old_NN()

In the old NN procedure is presented a model of NN just implemented:

```
# function to define an old style fully connected network (multilayer perceptrons)
class old_nn(nn.Module):
    def __init__(self):
        super(old_nn, self).__init__()
        self.fc1 = nn.Linear(32 * 32 * 3, 4096)
        self.fc2 = nn.Linear(4096, 4096)
        self.fc3 = nn.Linear(4096, n_classes)  # last FC for classification

    def forward(self, x):
        x = x.view(x.shape[0], -1)
        x = F.sigmoid(self.fc1(x))
        x = F.sigmoid(self.fc2(x))
        x = self.fc3(x)
        return x
```

Considering the step of loading the weight and biases just done, the next step is training the network.
Training consisting in minimize the loss, and it appens through 2 sequential operations:

- Forward propagation
- Backward propagation

The **forward pass** refers to calculation process, values of the output layers from the inputs data. Its traversing through all neurons from first to last layer.
A loss function is calculated from the output values.
And then **backward pass** refers to process of counting changes in weights (the learning phase), using gradient descent algorithm or similar. Computation is made from last layer, backward to the first layer.
Backward and forward pass makes together one **iteration.**
In the example given the 3 different steps disposed as:

```
# forward + backward + optimize
outputs = net(inputs)
loss = criterion(outputs, labels)
loss.backward()
optimizer.step()
```

Both the time spent and the accuracy of this NN are worst with the respect to the CNN, this is due to the fact that are not applied key concept like the Convolution Layer, and the pooling that we will see in detail in the next paragraph.

```
Accuracy of the network on the test set: 22 %
Finished Training
<matplotlib.legend.Legend at 0x7f5552f6dac8>
```

Convolutional Neural Network

The conventional-neural-networks is a subclass of neural-networks which have at least one convolution layer. They are great for capturing local information (eg neighbor pixels in an image or surrounding words in a text) as well as reducing the complexity of the model (faster training, needs fewer samples, reduces the chance of overfitting). The convolution units (as well as pooling units) are especially beneficial as:

- They reduce the number of units in the network (since they are *many-to-one mappings*). This means, there are fewer parameters to learn which reduces the chance of overfitting as the model would be less complex than a fully connected network.
- They consider the context/shared information in the small neighborhoods. This future is very important in many applications such as image, video, text, and speech processing/mining as the neighboring inputs (eg pixels, frames, words, etc) usually carry related information.

Convolutional Neural Network:CNN()

The code used for this implementation is:

```
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        #conv2d first parameter is the number of kernels at input (you get it from the
output value of the previous layer)
        #conv2d second parameter is the number of kernels you wanna have in your
convolution, so it will be the n. of kernels at output.
        #conv2d third, fourth and fifth parameters are, as you can read, kernel_size,
stride and zero padding :)
        self.conv1 = nn.Conv2d(3, 128, kernel_size=5, stride=2, padding=0)
        self.conv2 = nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=0)
        self.conv3 = nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=0)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2, padding=0)
        self.conv_final = nn.Conv2d(128, 256, kernel_size=3, stride=1, padding=0)
        self.fc1 = nn.Linear(256 * 4 * 4, 4096)
        self.fc2 = nn.Linear(4096, n_classes) #last FC for classification

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = F.relu(self.conv2(x))
        x = F.relu(self.conv3(x))
        x = F.relu(self.pool(self.conv_final(x)))
        x = x.view(x.shape[0], -1)
        x = F.relu(self.fc1(x))
        #hint: dropout goes here!
        x = self.fc2(x)
        return x
```

We use three main types of layers to build ConvNet architectures: **Convolutional Layer**, **Pooling Layer**, and **Fully-Connected Layer.** Optionally is used also a **Batch Normalization** function.

Convolutional layer

As previously introduced the Convolutional Neural Network is defined by at least one Convolutional layer. It is the core building block of a Convolutional Network that does most of the computational heavy lifting.

The idea is that when dealing with high-dimensional inputs such as images, as we saw above it is impractical to connect neurons to all neurons in the previous volume. Instead, we will connect each neuron to only a local region of the input volume.
Those neuron will share same parameters across different locations so it will bring better performances and an improvement on the analysis.

The problem is that every one of this layer requires a time to compute and produce as output an increasing in the number of new channels (we pass from 3 channel as input to 128 in output).
So, the more the output channel increase the more the computation will be heavy, but also the deepest is the learning and better the performances.
The decision should be a tradeoff between those input and output number.

In the examples 1/6,2/6 and 3/6 we can see clearly that the more are the channel and more is the time to compute (32/128/256= fast training – 512/1024= slow training), instead the more are the out and the more the precision and accuracy(from 30% to 40%) but also higher is the risk of overfitting.
Other parameter of the convolution are also the kernel size and the stride, but they remain the same for every iteration in the example so we don't consider that.

Pooling layer

Its function is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control overfitting. The Pooling Layer operates independently on every depth slice of the input and resizes it spatially, using the MAX operation.
There are also other kind of pooling but max is the most widely used.

The typical pooling size is 2x2, the same of our example.

Fully-Connected layer

The output from the convolutional layers represents high-level features in the data. While that output could be flattened and connected to the output layer, adding a fully-connected layer is a (usually) cheap way of learning non-linear combinations of these features.
Essentially the convolutional layers are the fully-connected layer is learning a (possibly non-linear) function in that space.

Batch Normalization

But the problem appears in the intermediate layers because the distribution of the activations is constantly changing during training. This slows down the training process because each layer must learn to adapt themselves to a new distribution in every training step. This problem is known as **internal covariate shift**.

So… what happens if we *force* the input of every layer to have approximately the same distribution in every training step? That's the main idea of Batch Normalization

However, these layers have since fallen out of favor because in practice their contribution has been shown to be minimal, if any.

Those are the result of the points 4a, b and c of the Homework

```
Files already downloaded and verified
[1,   195] loss: 3.627
Accuracy of the network on the test set: 23 %
[2,   195] loss: 2.871
Accuracy of the network on the test set: 31 %
[3,   195] loss: 2.501
Accuracy of the network on the test set: 35 %
[4,   195] loss: 2.241
Accuracy of the network on the test set: 37 %
[5,   195] loss: 2.028
Accuracy of the network on the test set: 40 %
[6,   195] loss: 1.841
Accuracy of the network on the test set: 41 %
[7,   195] loss: 1.655
Accuracy of the network on the test set: 41 %
[8,   195] loss: 1.474
Accuracy of the network on the test set: 42 %
[9,   195] loss: 1.300
Accuracy of the network on the test set: 42 %
[10,   195] loss: 1.135
Accuracy of the network on the test set: 43 %
[11,   195] loss: 0.970
Accuracy of the network on the test set: 42 %
[12,   195] loss: 0.813
Accuracy of the network on the test set: 43 %
[13,   195] loss: 0.664
Accuracy of the network on the test set: 42 %
[14,   195] loss: 0.534
Accuracy of the network on the test set: 42 %
[15,   195] loss: 0.417
Accuracy of the network on the test set: 42 %
[16,   195] loss: 0.320
Accuracy of the network on the test set: 43 %
[17,   195] loss: 0.236
Accuracy of the network on the test set: 42 %
[18,   195] loss: 0.173
Accuracy of the network on the test set: 43 %
[19,   195] loss: 0.129
Accuracy of the network on the test set: 42 %
[20,   195] loss: 0.094
Accuracy of the network on the test set: 42 %
Finished Training
```

```
Files already downloaded and verified
[1,   195] loss: 3.582
Accuracy of the network on the test set: 25 %
[2,   195] loss: 2.823
Accuracy of the network on the test set: 31 %
[3,   195] loss: 2.443
Accuracy of the network on the test set: 36 %
[4,   195] loss: 2.175
Accuracy of the network on the test set: 38 %
[5,   195] loss: 1.947
Accuracy of the network on the test set: 40 %
[6,   195] loss: 1.738
Accuracy of the network on the test set: 41 %
[7,   195] loss: 1.532
Accuracy of the network on the test set: 42 %
[8,   195] loss: 1.339
Accuracy of the network on the test set: 42 %
[9,   195] loss: 1.137
Accuracy of the network on the test set: 42 %
[10,   195] loss: 0.954
Accuracy of the network on the test set: 43 %
[11,   195] loss: 0.774
Accuracy of the network on the test set: 43 %
[12,   195] loss: 0.607
Accuracy of the network on the test set: 43 %
[13,   195] loss: 0.471
Accuracy of the network on the test set: 43 %
[14,   195] loss: 0.341
Accuracy of the network on the test set: 43 %
[15,   195] loss: 0.245
Accuracy of the network on the test set: 43 %
[16,   195] loss: 0.170
Accuracy of the network on the test set: 43 %
[17,   195] loss: 0.117
Accuracy of the network on the test set: 43 %
[18,   195] loss: 0.084
Accuracy of the network on the test set: 43 %
[19,   195] loss: 0.062
Accuracy of the network on the test set: 43 %
[20,   195] loss: 0.047
Accuracy of the network on the test set: 43 %
Finished Training
```

```
Files already downloaded and verified
[1,   195] loss: 3.638
Accuracy of the network on the test set: 23 %
[2,   195] loss: 2.894
Accuracy of the network on the test set: 30 %
[3,   195] loss: 2.520
Accuracy of the network on the test set: 34 %
[4,   195] loss: 2.259
Accuracy of the network on the test set: 37 %
[5,   195] loss: 2.048
Accuracy of the network on the test set: 38 %
[6,   195] loss: 1.859
Accuracy of the network on the test set: 40 %
[7,   195] loss: 1.671
Accuracy of the network on the test set: 42 %
[8,   195] loss: 1.506
Accuracy of the network on the test set: 43 %
[9,   195] loss: 1.328
Accuracy of the network on the test set: 42 %
[10,   195] loss: 1.164
Accuracy of the network on the test set: 42 %
[11,   195] loss: 1.011
Accuracy of the network on the test set: 42 %
[12,   195] loss: 0.852
Accuracy of the network on the test set: 42 %
[13,   195] loss: 0.702
Accuracy of the network on the test set: 42 %
[14,   195] loss: 0.569
Accuracy of the network on the test set: 42 %
[15,   195] loss: 0.453
Accuracy of the network on the test set: 42 %
[16,   195] loss: 0.352
Accuracy of the network on the test set: 42 %
[17,   195] loss: 0.264
Accuracy of the network on the test set: 42 %
[18,   195] loss: 0.195
Accuracy of the network on the test set: 42 %
[19,   195] loss: 0.147
Accuracy of the network on the test set: 42 %
[20,   195] loss: 0.111
Accuracy of the network on the test set: 42 %
Finished Training
```

a] The Batch normalization as expected doesn't affect very much the result.

b] Instead the number of Neurons in the network improve slightly the result, permitting a step forward from 42% to 43% but increase also the time consumed for the computation. Another problem of the application of too much Neurons is the Overfitting, in the second image the loss ratio is very low 0.047 but the accuracy remains always very high. This means Overfitting!

Before the 3rd point its important to know what is the dropout.

Simply put, dropout refers to ignoring units (i.e. neurons) during the training phase of certain set of neurons which is chosen at random. By "ignoring", I mean these units are not considered during a particular forward or backward pass. It's used prevent over-fitting

c] As result the result will remain the same but we can see that the loss ratio decrease with the accuracy that remain stationary to the same value. That poits to a better behaviour of our network, and a decrement of the probabilities of over-fitting.

Methods against overfitting

In addition to the other methods just explained in precedence, there is another method to prevent overfitting direct interacting with the initial dataset. This is referred to as **data augmentation**.
By applying transformations to the training data, you're adding synthetic data points. This exposes the model to additional variations without the cost of collecting and annotating more data. This can have the effect of reducing overfitting and improving the model's ability to generalize.

Example are Horizontal **flipping** or random **cropping**.

The intuition behind flipping an image is that an object should be equally recognizable as its mirror image. Note that horizontal flipping is the type of flipping often used. Vertical flipping doesn't always make sense, but this depends on the data.

The idea behind cropping is that to reduce the contribution of the background in the CNNs decision.

```
[1,   195] loss: 4.054
Accuracy of the network on the test set: 13 %
[2,   195] loss: 3.535
Accuracy of the network on the test set: 19 %
[3,   195] loss: 3.238
Accuracy of the network on the test set: 23 %
[4,   195] loss: 3.028
Accuracy of the network on the test set: 25 %
[5,   195] loss: 2.853
Accuracy of the network on the test set: 27 %
[6,   195] loss: 2.687
Accuracy of the network on the test set: 28 %
[7,   195] loss: 2.523
Accuracy of the network on the test set: 31 %
[8,   195] loss: 2.362
Accuracy of the network on the test set: 30 %
[9,   195] loss: 2.185
Accuracy of the network on the test set: 32 %
[10,   195] loss: 2.002
Accuracy of the network on the test set: 32 %
[11,   195] loss: 1.789
Accuracy of the network on the test set: 32 %
[12,   195] loss: 1.561
Accuracy of the network on the test set: 33 %
[13,   195] loss: 1.325
Accuracy of the network on the test set: 32 %
[14,   195] loss: 1.078
Accuracy of the network on the test set: 33 %
[15,   195] loss: 0.835
Accuracy of the network on the test set: 32 %
[16,   195] loss: 0.605
Accuracy of the network on the test set: 32 %
[17,   195] loss: 0.427
Accuracy of the network on the test set: 31 %
[18,   195] loss: 0.304
Accuracy of the network on the test set: 31 %
[19,   195] loss: 0.212
Accuracy of the network on the test set: 31 %
[20,   195] loss: 0.160
Accuracy of the network on the test set: 31 %
Finished Training
```

```
[1,   195] loss: 4.072
Accuracy of the network on the test set: 13 %
[2,   195] loss: 3.572
Accuracy of the network on the test set: 18 %
[3,   195] loss: 3.280
Accuracy of the network on the test set: 22 %
[4,   195] loss: 3.081
Accuracy of the network on the test set: 25 %
[5,   195] loss: 2.933
Accuracy of the network on the test set: 27 %
[6,   195] loss: 2.785
Accuracy of the network on the test set: 29 %
[7,   195] loss: 2.666
Accuracy of the network on the test set: 30 %
[8,   195] loss: 2.542
Accuracy of the network on the test set: 31 %
[9,   195] loss: 2.430
Accuracy of the network on the test set: 33 %
[10,   195] loss: 2.310
Accuracy of the network on the test set: 33 %
[11,   195] loss: 2.196
Accuracy of the network on the test set: 34 %
[12,   195] loss: 2.072
Accuracy of the network on the test set: 35 %
[13,   195] loss: 1.955
Accuracy of the network on the test set: 36 %
[14,   195] loss: 1.820
Accuracy of the network on the test set: 36 %
[15,   195] loss: 1.685
Accuracy of the network on the test set: 36 %
[16,   195] loss: 1.546
Accuracy of the network on the test set: 37 %
[17,   195] loss: 1.406
Accuracy of the network on the test set: 37 %
[18,   195] loss: 1.256
Accuracy of the network on the test set: 36 %
[19,   195] loss: 1.115
Accuracy of the network on the test set: 36 %
[20,   195] loss: 0.981
Accuracy of the network on the test set: 36 %
Finished Training
```

a] In the 5/6a we can clearly see that there is a major Improvement due to this technique.
The value of accuracy increase of 5% (from 31% to 36%) and the loss rate remain still high. That's good because it means that our network is increasing its performances without learning the model, so there is not overfitting. That's due to the initial modification inserted with the flipping.

b] with crop instead of flipping we reach a better improvement of 39%(8% from the normal one) with an higher loss rate in the training set. So, the conclusion is that those two technique works very well for prevents overfitting.

ResNet18

ResNet-18 is a convolutional neural network that is trained on more than a million images from the ImageNet database. The network is 18 layers deep and can classify images into 1000 object categories, such as keyboard, mouse, pencil, and many animals. As a result, the network has learned rich feature representations for a wide range of images.

Using ResNet with the additional Crops and Flips showed before is possible to train the network and reach a great accuracy ratio of 74% in our test set.
This process requires a very slow training comparable to our one with the convolutional with 1024 as output.
But, it reach a better accuracy because of the vast number

```
Files already downloaded and verified
[1,   390] loss: 1.899
Accuracy of the network on the test set: 71 %
[2,   390] loss: 0.750
Accuracy of the network on the test set: 77 %
[3,   390] loss: 0.398
Accuracy of the network on the test set: 78 %
[4,   390] loss: 0.183
Accuracy of the network on the test set: 79 %
[5,   390] loss: 0.076
Accuracy of the network on the test set: 79 %
[6,   390] loss: 0.032
Accuracy of the network on the test set: 79 %
[7,   390] loss: 0.018
Accuracy of the network on the test set: 79 %
[8,   390] loss: 0.022
Accuracy of the network on the test set: 78 %
[9,   390] loss: 0.021
Accuracy of the network on the test set: 76 %
[10,   390] loss: 0.064
Accuracy of the network on the test set: 74 %
Finished Training
```

of layer inside the structure, that is completely different to the example cited before that has only 2 convolutional layers, a pooling and a FC layer.
This network uses a residual network architecture and it solves the convergence and degradation of performance due to a strong gradient vanish problem for deeper networks.