Detecting Speech Activity Using Convolutional Neural Network (CNN) Classifier

**Srinivas Kulkarni**

EESC 6364 Pattern Recognition - Fall 2016

**Detecting Speech Activity Using Convolutional Neural Network (CNN)**

**Classifier**

***Introduction***

Convolutional Neural Network (CNN) is a branch of supervised machine learning, used for both classification and regression problem. CNN is widely used in image recognition and voice recognition problems. It has neurons which contain weights and biases. Input is multiplied with weights followed by bias addition to get output. These output are then followed by non-linearity function. Each neuron here are called nodes, each layer will have multiple nodes. Last layer will give us the score, which is input to softmax layer. Softmax layer converts score to probability. At output layer we can use cross entropy technique to find the label.

In this project CNN classifies image inputs as signal+noise or noise. Input to architecture is row major image and output is the label (1 for signal+noise, 2 for noise). Architecture of CNN contains many layers called hidden layers, between input and the output. Each hidden layer is either convolution layer, pooling layer or activation function (discussed in detail later).

***Overview***

Following paragraph talks about the general overview of ConvNet architecture. Each layer transforms three dimensional input to three dimensional output, one after the other at end giving single number as probability of each class. In general CNN contains following layers

* Input layer: Pixel values of the input image, our case 40\*40\*1.
* Conv Layer: This layer contains many small filters, containing weights (randomly initialized first, later updated). These filters are convolved over the image to produce output. If we use 20 filters, Output will be 40\*40\*20.
* Activation Function: We use this is introduce non linearity in the architecture.
* Pooling layer: Down sample the input, normally height and width are reduced to half. Our case: 20\*20\*20
* Fully Connected Network: Outputs each class score (our case 1\*2), which is used to classify image.

Power of ConvNet lies in the filter (or kernels), which are learned by model. **Back-propagation** **algorithm** updates weights and biases. **Gradient descent algorithm** helps us to find the minimum error between the predicted and ground truth values.

***Terms used in CNN***

1. Padding: Conv layers are padded with zero. If the CONV layers were to not zero-pad the inputs and only perform valid convolutions, then the size of the volumes would reduce by a small amount after each CONV, and the information at the borders would be “washed away” too quickly.
2. Stride: Provides information about the number of pixels filter moves along the given axis, and performs convolution
3. Learning Rate: In gradient descent technique, gradient provides the information about the direction, learning rate determines the magnitude of the movement along the gradient
4. Batch Size: We can’t use all the hundred thousand of images at once to train, because 1. Memory cannot handle such high data. 2. Convergence will be faster if we use stochastic gradient descent (Batch size is used in SGD).
5. Objective: Objective function provide us the information about the loss for the updated weights. Our aim is to reduce objective function as low as possible.
6. Top1error: We have two error in our case. 1. Predicted = Signal, Actual = Noise, 2. Predicted = Noise, Actual = Signal. Top1error informs value of least among them.

***Architecture Used For The Project***

We had three different types of images; signal to noise ratio (SNR) 0, 5 and 10. In each SNR there were 5 different batches. I used cross validation technique to train on different batches. Accuracy is averaged over all the five batches, so is confusion matrix.

1. Input layer: Input image is height=40pixel, width=40pixel, channel = 1.
2. Block 1:

* Conv layer 1 : Filter Size = 5x5, Number of Filter = 40, Stride = 1, Pad=2
* Pooling: Pool size = ‘MAX’, 3x3, Stride = 2, Pad = [0 1 0 1]
* Activation layer: ReLu

1. Block 2:

* Conv layer 2: Filter Size = 5x5, Number of Filter = 40, Stride = 1, Pad=2
* Activation layer: ReLu

1. Block 3:

* Conv layer 3: Filter Size = 5x5, Number of Filter = 40, Stride = 1, Pad=2
* Pooling: Pool size = ‘AVG’, 3x3, Stride = 2, Pad = [0 1 0 1]
* Activation layer: ReLu

1. Block 4:

* Conv layer 4: Filter Size = 5x5, Number of Filter = 40, Stride = 1, Pad=2
* Pooling: ‘AVG’, 3x3, Stride = 2, Pad = [0 1 0 1]
* Activation layer: ReLu

1. Block 5:

* Fully Connected Network: Outputs scores, number of scores equal to number of classes.

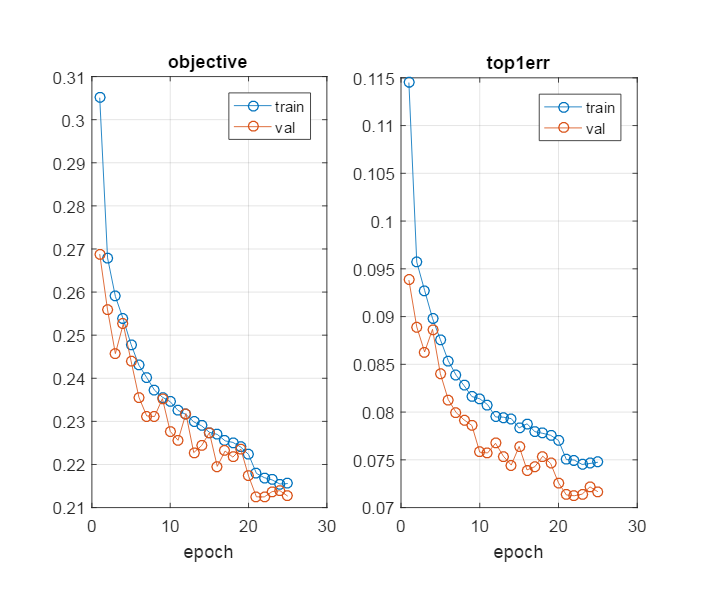
1. Output layer:

* Softmax Function: Use Softmax function to determine, Probability.

***Analysis and experiments to improve accuracy and computational efficiency***

Major part of the project was spent to improve the efficiency of training model, and increasing the accuracy. I read through the reference given and browsed INTERNET for various tricks and techniques. Below I made the list of various methods I tried, to resolve the problem.

1. **Filter Shape**: Filter shapes, usually based on the dataset. For images sized (40x40) filter size of 3x3 or 5x5 range are found better, also larger filters are computationally expensive. One more advantage of small filters are they tend to capture the features of the image in more detail. The trick is thus to find the right filter shapes in order to create abstractions at the proper scale, given a particular dataset.
2. **Number of Filters**: Number of filters used, determine the depth of each layer. Each filter (usually all are of same size) when activated enhances the particular area of the image and store it as a feature. This feature stored will grow deeper (adds more details) as we move forward in architecture.
3. **Pooling Shape (Down sampling)**: Pooling is mainly used for down sampling, with pooling we lose spatial information in the image, but we store the feature that has maximum impact on the final output. Pooling size normally used are 2x2 or 3x3, which down sample the image to half the original size. Unlike in convolution layer, here there are no complex calculations, in pooling its either max pixel in the filter or average of pixels in the filter will be output.
4. **Decay the learning rate after certain epoch**: Learning rate determines the step size along the direction of gradient. In general for all machine learning problems, error initially, with random weights and biases will be high. Eventually with updated weights, error decreases and flattens. Hence we are allowed to have higher learning rate (bigger jumps) initially, then decrease after certain epochs. This helps is faster convergence.
5. **Number of Hidden Layers**: This is important factor in ConvNet. At each layer some feature of image is captured. For example in face recognition problem, first layer looks for outlines in the image, shape of the object. Consecutive layers enhances detail feature in the face (eye shape, eye color). Hence Number of hidden layer is application dependent, which usually determined by experiment (and experience). For small scale project (our case) 5-8 layers are optimal.
6. **Data Augmentation**: This can be used for two reasons. 1. If we have less image data, we can generate new images with added noise, flipped, cropped. 2. This is done to avoid over-fitting for the training data.
7. **Regularization**: Magic of machine learning model, lies in loss optimization and Regularization. Without regularization, all models would be over-fit for given data, and never generalized. Regularization is like penalty model can bear, higher the penalty, model tend to not update for every training data, hence generalization. In ConvNet, we can use L1, L2, droputs.
8. **Initializations**: We start with random weights while training which converges to get minimum error. Weights initialization is a active area of research, helping to determine fast converging initial weights. There are various methods, 1. All Zero Initialization: initialize with zero, would essentially make all filters learn the same feature, not a good method, 2. Initialization with Small Random Numbers: Gaussian random number with small deviation are found to be better. If deviation is too small, during back propagation all weights become zero. With higher deviation, weights never converge.
9. **Convolution – multiplication**: Convolution is time domain is multiplication in frequency domain. We can essentially convert to frequency domain, multiply inverse back to time. This is better option when total operations are more than 64 bit, otherwise FFT overhead will be costlier.
10. **Double – Single Precision**: In default MATLAB does double precision, which is computationally expensive, hence convert all the data to single for faster computation and convergence.
11. **Better optimizer**: Gradient descent is naive methods, which moves along gradient with some step size. But with higher dimension and complex optimization problem, gradient descent is not ideal choice. Better optimization choice would be 1. [Stochastic Gradient Descent](https://en.wikipedia.org/wiki/Stochastic_gradient_descent), 2. Adagrad, 3. RMS Prop, 4. Adam.

***Interpretation Of Results Obtained***

**Validation and Training**

In machine learning problem, data given will be split to train-validation-test. We use training dataset to train, validation to determine with what parameter model is giving good result (not done on training because model have already seen training datasets). Finally test datasets to see how model performs to unknown datasets. Above figure shows the loss function (objective), which reduces as epochs increase. Our aim is to get zero loss (minimum loss). Top1error signifies the minimum error of two. We can see training error is less than validation, as model knows training datasets, validation is on unseen datasets.

**Accuracy and Memory**

Various methods are used to determine how good model is, to list some,

1. f score

2. Accuracy

3. ROC curve

4. Memory used by model

5. Computation Time

6. Confusion Matrix

Following are parameters we used in this project

1. **Accuracy**: Accuracy is determined by calculating the percentage of correctly classified samples. (We know the ground truth and predicted values)

Overall accuracy in our case: **91.32%**

2. **Memory Consumption**: All algorithms are finally implemented on hardware chips, hence in embedded applications memory plays a huge role. More filter, more weights constitute to more memory utilization. Also Graph of architecture adds to memory of the model. In summary

**Total memory = Architecture graph + memory of weights + biases**

Memory consumption in our case: **481kB (1.2e+05 parameters)**

3. **Computation Time**: Time taken to correctly classify single image. This is critical in real time signal processing applications. Clearly this must be as low as possible.

Computation Time in our case: **5ms**

4. **Confusion matrix**: it’s a 2x2 (in our case) matrix with columns as ground truth values and rows as predicted values. 4 elements of matrix are

1. True Positive (Actual = True, Predicted = True)

2. True Negative (Actual = True, Predicted = False)

3. False Positive (Actual = False, Predicted = False)

4. False Negative (Actual = False, Predicted = True)

Confusion Matrix in our case:

PREDICTED VALUES

Speech + Noise Noise

|  |  |  |  |
| --- | --- | --- | --- |
| ACTUAL VALUES | Speech+Noise | 6171  (98.22 %) | 111  (1.7%) |
| Noise | 978  (15.5%) | 5303  (84.4%) |

Best Model will have higher number in diagonal elements, and as low as possible in non-diagonal elements.

***References***

1. Yoshua Bengio Submitted on 24 Jun 2012 ([v1](https://arxiv.org/abs/1206.5533v1)), last revised 16 Sep 2012 (this version, v2) [Practical recommendations for gradient-based training of deep architectures](http://arxiv.org/abs/1206.5533)

2. Convolutional Neural Networks (LeNet) DeepLearning 0.01 documentation

[<http://deeplearning.net/tutorial/lenet.html>]

3. [Andrej Karpathy](http://cs231n.github.io/cs.stanford.edu/people/karpathy/) [CS231n: Convolutional Neural Networks for Visual Recognition](http://cs231n.stanford.edu/).

[http://cs231n.github.io]

4. MATLAB toolbox implementing *Convolutional Neural Networks* (CNNs) [http://www.vlfeat.org/matconvnet/]