STATISTICAL METHODS FOR MACHINE LEARNING

Instructor: Nicolò Cesa-Bianchi
Project: Neural network classification with TensorFlow

Dataset: SVHN (HYPERLINK "http://ufldl.stanford.edu/housenumbers/" \h http://ufldl.stanford.edu/housenumbers/ HYPERLINK "http://ufldl.stanford.edu/housenumbers/" \h)

Student: Venkata Sai Vamsi Peesapati (933987) - Data Science and Economics

1.Convolutional Neural Network (CNN):

Neural networks are parametric algorithms that train on a predetermined set of variables (unlike a k-NN or Tree Predictors). Neural Networks are linear classifiers that try to differentiate the different data points using Separating Hyperplanes (In linearly separable case).

Neural Networks train with help of a Convex loss function where the goal is to find the local minima and descent in the negative direction of the gradient proportional to the gradient (A commonly known technique called Gradient descent – which happens inside the Backpropagation algorithm).

The Perceptron algorithm finds a homogeneous separating hyperplane by running through the training examples one after the other. The current linear classifier is tested on each training example and, in case of misclassification, the associated hyperplane is adjusted.
If the algorithm terminates, then w is a separating hyperplane. Perceptron always terminates on linearly separable training sets.

The Perceptron algorithm accesses training data in a sequential fashion, processing each training example in linear time, making Perceptron very competent on large training data sets. It is also very good at dealing scenarios in which new training data are generated all the time.

Model update h(t) --> h(t+1) is typically local.

In Convolutional Neural Network, unlike Regular Neural Network connecting to all other neurons from the previous layer, while processing a part of an image it connects to the only required neurons which are in its surroundings or close to it. Also, CNN take advantage of the shared-weights architecture reducing the memory footprint of the network. CNN like any other Neural network learns through the process of Backpropagation.

Convoluting - An image is composed on number of channels mentioned with the filter being applied on it, we can choose the number of filters. At end, we apply activation function; can be a Max function or ReLU function. As output, a tensor is created for each filter applied.

Again, these tensors are allowed as inputs to the next layer in the convolution. Then, number of filters has to be chosen and applied on the channels. Later, on applying activation function, the same Max function tensors are produced as output, equal to the number of filters applied earlier.

Pooling - It aggregates some values into a single output value. It reduces the size of the output tensor from each input tensor. Depth remains the same after pooling. *Max Pooling* takes the maximum input of a particular convoluted feature. *Average Pooling* takes the average input of a particular convoluted feature.

Fully Connected - Fully connected layers connect every neuron in one layer to every neuron in another layer. It is in principle the same as the traditional Multi-Layer Perceptron Neural Network (MLP). The flattened matrix goes through a fully connected layer to classify the images.

Number of parameters in the network: Input * Output + Biases

2. Theoretical background of some methods/techniques used:

<u>Dropout:</u>

Dropout as a Regularization method, helps in training a complex model without additional computational expense of training and maintaining multiple models, by randomly dropping out nodes during training.

The Dropout layer randomly sets input units to 0 with a frequency of rate at each step during training time, which helps prevent overfitting. Inputs not set to 0 are scaled up by **1 / (1 - rate)** such that the sum over all inputs is unchanged.

Regularization:

Regularization adds a penalty on the different parameters of the model to reduce the freedom of the model. Hence, the model will be **less likely to fit the noise** of the training data and will improve the generalization abilities of the model.

L2 regularizer(also called Ridge), whose penalty is computed as:
Loss = l2 * reduce_sum(square(x))

i.e. L2 regularization adds a penalty equal to the sum of the squared value of the coefficients.
The L2 regularization will force **the parameters to be relatively small**, the bigger the penalization, the smaller (and the more robust) the coefficients are.

The L1 regularization (also called Lasso)
The L2 regularization (also called Ridge)
The L1/L2 regularization (also called Elastic net)

Regularizers allow us to apply penalties on layer parameters or layer activity during optimization. These penalties are summed into the loss function that the network optimizes.
Regularization penalties are applied on a per-layer basis. The exact API will depend on the layer, but many layers (e.g. Dense, Conv1D, Conv2D and Conv3D) have a unified API.

These layers expose 3 keyword arguments:

Kernel_regularizer: Regularizer to apply a penalty on the layer’s kernel
Bias_regularizer: Regularizer to apply a penalty on the layer’s bias
Activity_regularizer: Regularizer to apply a penalty on the layer’s output

The value returned by the activity_regularizer object gets divided by the input batch size so that the relative weighting between the weight regularizers and the activity regularizers does not change with the batch size.
We can access a layer’s regularization penalties by calling **layer.losses** after calling the layer on inputs.

Stochastic Gradient Descent (SGD):

Stochastic gradient descent (SGD) in contrast performs a parameter update for each training example x(i) and label y(i).

The use of SGD in the neural network setting is motivated by the high cost of running back propagation over the full training set. SGD can overcome this cost and still lead to fast convergence. SGD performs frequent updates with a high variance that cause the objective function to fluctuate heavily as in Image below.

Generally, each parameter update in SGD is computed w.r.t a few training examples or a **minibatch**. The reason for this is, first this reduces the variance in the parameter update and can lead to more stable convergence, second this allows the computation to take advantage of highly optimized matrix operations that should be used in a well vectorized computation of the cost and gradient.

Mini batch of n training $\theta=\theta-\eta\cdot\nabla\theta J(\theta;x(i:i+n);y(i:i+n))$.

A typical minibatch size is 256, although the optimal size of the minibatch can vary for different applications and architectures. In SGD the learning rate α is typically much smaller than a corresponding learning rate in batch gradient descent because there is much more variance in the update.

A better approach is to evaluate a held-out set after each epoch and anneal the learning rate when the change in objective between epochs is below a small threshold. This tends to give good convergence to a local optima. If the data is given in some meaningful order, this can bias the gradient and lead to poor convergence. Generally, a good method to avoid this is to randomly shuffle the data prior to each epoch of training

Stochastic gradient methods can generally be written

Stochastic momentum methods are a second family of techniques that have been used to accelerate training which choose a local distance measure constructed using the entire sequence of iterates $(w1,\cdots,wk)$.. These methods can generally be written as

SHAPE * MERGEFORMAT

Adaptive Gradient Descent:

Gradient descent is a first-order iterative optimization algorithm for finding a local minimum of a differentiable function. we use gradient descent to update the parameters of our model.

Ada delta optimization is a stochastic gradient descent method that is based on adaptive learning rate per dimension to address two drawbacks:

The continual decay of learning rates throughout training
The need for a manually selected global learning rate

Ada delta is a more robust extension of Adagrad that adapts learning rates based on a moving window of gradient updates, instead of accumulating all past gradients.

Adam optimization is a stochastic gradient descent method that is based on adaptive estimation of first-order and second-order moments, Adam combines the best properties of the AdaGrad and RMSProp algorithms to provide an optimization algorithm that can handle sparse gradients on noisy problems.

Batch Normalization:

A powerful technique that reshapes the distribution of activations (making them close to Normal Distribution), which improves the performance and stability of the network.

Batch Norm = (Activation – Mean Activation)/S.D

With Mean Activation close to 0 and Active Standard Deviation of the distribution close to 1.

3. The Street View House Numbers (SVHN) Dataset:

SVHN is a real-world image dataset for developing machine learning and object recognition algorithms with minimal requirement on data preprocessing and formatting. It can be seen as similar in flavor to HYPERLINK "http://yann.lecun.com/exdb/mnist/" \h MNIST HYPERLINK "http://yann.lecun.com/exdb/mnist/" \h (e.g., the images are of small cropped digits), but incorporates an order of magnitude more labeled data (over 600,000 digit images) and comes from a significantly harder, unsolved, real world problem (recognizing digits and numbers in natural scene images). SVHN is obtained from house numbers in Google Street View images.

Overview:

10 classes, 1 for each digit. Digit '1' has label 1, '9' has label 9 and '0' has label 10.
73257 digits for training, 26032 digits for testing, and 531131 additional, somewhat less difficult samples, to use as extra training data
Comes in two formats:

Original images with character level bounding boxes.
MNIST-like 32-by-32 images centered around a single character (many of the images do contain some distractors at the sides).

I used the Format 2 version of the dataset available at (HYPERLINK "http://ufldl.stanford.edu/housenumbers/" \h http://ufldl.stanford.edu/housenumbers/ HYPERLINK "http://ufldl.stanford.edu/housenumbers/" \h)

Format 2: Cropped Digits: HYPERLINK "http://ufldl.stanford.edu/housenumbers/train_32x32.mat" \h train_32x32.mat HYPERLINK "http://ufldl.stanford.edu/housenumbers/train_32x32.mat" \h , HYPERLINK "http://ufldl.stanford.edu/housenumbers/test_32x32.mat" \h test_32x32.mat HYPERLINK "http://ufldl.stanford.edu/housenumbers/test_32x32.mat" \h , HYPERLINK "http://ufldl.stanford.edu/housenumbers/extra_32x32.mat" \h extra_32x32.mat HYPERLINK "http://ufldl.stanford.edu/housenumbers/extra_32x32.mat" \h

Character level ground truth in an MNIST-like format. All digits have been resized to a fixed resolution of 32-by-32 pixels. The original character bounding boxes are extended in the appropriate dimension to become square windows, so that resizing them to 32-by-32 pixels does not introduce aspect ratio distortions. Nevertheless, this preprocessing introduces some **distracting** digits to the sides of the digit of interest. Loading the .mat files creates 2 variables: **X** which is a 4-D matrix containing the images, and **y** which is a vector of class labels. To access the images, X(:,:,:,i) gives the i-th 32-by-32 RGB image, with class label y(i).

4. Description of Experiments:

The goal is to correctly predict the label which is the center number in a given image. So, our final layer always contains 10 neurons corresponding to 10 labels respectively.
We have used the train_32x32.mat for training and test_32x32.mat as validation set. We have not used the extra_32 x32.mat as it contains much easier to train images.
Loss function used for all the experiments: **Sparse Categorical Cross Entropy**, which makes the labels mutually exclusive for each training data point making it suitable for our **Single-label Classification**
SoftMax activation function for Classification because it creates an exponential probability distribution in the range of (0,1) making it easy for classification based on the activation with highest probability.

Using **accuracy** as metric for the Classification task

Data preprocessing:

Pre-processing the 32-by-32 images from the SVHN dataset centered around a single digit. In this dataset all digits have been resized to a fixed resolution of 32-by-32 pixels. The original character bounding boxes are extended in the appropriate dimension to become square windows, so that resizing them to 32-by-32 pixels does not introduce aspect ratio distortions
Transposing the train and test data by converting it from:

(width, height, channels, size) -> (size, width, height, channels)
Normalized the pixel values to be between 0 and 1

Verifying the data: To verify that the dataset looks correct, defined a function to plot some images from the training set, test set and display the class name above each image

Converted the Label 10's to Label 0's

Network Architecture:
A **filter** size of (3,3) and a MaxPool of (2,2) is used across all models.
For all the models ReLu (for training) is used as the activation function for all the layers (Conv2D, Dense layers) except for the last output layer where Softmax activation (for Classification) is used.

ReLu:

Defined as the positive part of the argument where x is the input to a neuron, allows faster and effective training of deep neural architectures on large and complex datasets.
Trained all the models for 10 epochs (except the last model we tried to train till 40 epochs)
Using a basic CNN model (initial model in the notebook), tested the performance of optimizers Adam (Adaptive) and SGD (Stochastic Gradient Descent). Adam (89.6% accuracy on test_set using model) performed slightly better than SGD (88.1% accuracy on test_set using modelSGD)
Hence using Adam as optimizer, checked the performance of the network using Dropout as well as Ridge (L2 Regularizer) as regularization methods on 2 different models of same network architecture. Dropout (86.5% accuracy on test_set using modelDropout) is a better regularization approach over Ridge (81.4% accuracy on test_set using modelRidge) in this case.
Hence using Adam optimizer and Dropout as regularization method, used Batch Normalization over a much complex network architecture than all previous models. We found the accuracy on test_set improved to **91.7%**
When combined the above approach with Real time Data Augmentation, we noticed the training accuracy further improving. (Google Colab GPU RAM crashed around 32/40 epochs with an accuracy ~ 93%)
Based on the performance of our models, we see the hyperparameters are chosen correctly for the task.

5. Conclusion:

SVHN is a very large and extensive dataset that comes from a significantly more difficult problem where images contain a lot of clutter and noisy features.

CNN’s are a good approach for Image Classification Tasks
Dropout works as a better regularizer over Ridge
Stochastic gradient descent (SGD) performs a parameter update for training label x(i) and y(i) whereas Adaptive Moment Estimation (Adam) computes adaptive learning rates for each parameter in storing an exponentially decaying average of past gradients.

Adam has convergence problems where SGD can converge better with longer training time. However, adaptive methods train the network faster

Batch Normalization is a good technique to improve the overall performance of the network
Real time Data Augmentation can help the network achieve better performance. However, this will take longer training time and memory.