Lee Beadle / Chad Etheredge

CS 470 Artificial Intelligence

Term Project – Image Recognition System

4/30/2018

**Image Recognition Through Machine Learning and Deep Learning**

        What is image recognition? For humans, it is often a glance at an object and the corresponding recognition. It happens in the blink of an eye and without much effort. For machines, the ability to view an image and identify that image is much more complex. Artificial intelligence in the form of machine learning, neural networks, and deep learning helps solve this problem. The following system is an example of an AI system for image recognition that uses two implementations to achieve its goal. First, a basic machine learning classifier is developed using a Softmax algorithm. The second, is a more complex two-layer neural network implementation for deep learning applications. Both implementations were built following the tutorial blog “How to Build a Simple Image Recognition System with TensorFlow” by Wolfgang Beyer. As an additional note, source files used in this project were pulled from Wolfgang Beyer’s public GitHub, under “Wolflib”, and were used to complete the tutorial and system. Links to these are provided under our references section.

**Our Approach**

        The first approach to this AI system is through a machine learning implementation. The second is through a two-layer fully connected neural network. Both systems seek to analyze a dataset of pictures and predict which images belong to a specific category. For this project, the Softmax Classification system is built first and then altered to become a fully connected two-layer neural network. Also, both of these programs make use of TensorFlow’s powerful graph data structure and the Numpy package for calculation. The data analyzed is the CIFAR-10 dataset. This dataset is a collection of 60,000 images that are classified into ten categories that contain both objects and animals. The categories are airplane, car, bird, cat, deer, dog, frog, horse, ship, and truck. Also, the CIFAR-10 dataset is a standardized collection in which each image is 32x32 pixels. It is important to note that the small size of the pictures can actually hinder human recognition of the objects in the images but eases the computational workload for the system.

**Softmax Implementation**

        To begin constructing the Softmax Classifier, a means for the computer to recognize images is needed. This is accomplished by building a mathematical model that encompasses two critical stages to function. The first stage, is the training phase. In this phase, 50,000 of the total images from the training subset are analyzed but system. Operations performed on the training data provide the knowledge base for making the predictions necessary for image recognition. This first portion of the model, utilizes supervised learning to train the system. With supervised learning, the system is provided both the raw input data and correct image classification. This ensures that the system can evaluate and scale how accurate its predictions are through each training step.

The testing phase, the second stage of the model, consists of 10,000 images and represents the pool of data in which the machine will attempt to accurately recognize objects or animals. Both phases of this model reflect the accuracy of recognition which ultimately represents the most important metric for this project.

        However, one important aspect has been overlooked: How *exactly* does the system recognize objects or in animals within each image? To begin, the images are first enumerated in terms the system can manage efficiently. Each image is broken into its base form of individual pixels. With a 32x32 image, there are 1024 individual pixels. However, each pixel can be represented by one of three possible color scales: Red, green, and blue. Therefore, each image can be numerically represented by 3072 total values. When the system observes each individual pixel, a probabilistic prediction, using the Softmax function, is made on whether that pixel’s color is more or less likely to belong to a specific class. This prediction’s likelihood is represented by a weight. The color channel value is then multiplied by the weight and all 3072 values are summed to single value for each of the category classes. Finally, the highest scoring class is taken as the prediction for the image.

        To begin building the model, the model is fed input images in the form of the numerical values. The model’s initial input uses simple starting values as its parameters. During training, the model accepts the image input and compares them with known image classifications. As the model runs, the system updates its parameters to match known classifiers. The key to the model updating is performing optimization of the weights and biases variables and model parameters. Therefore, the more optimized these values are at each iteration of training, the more accurate the training results will become.

        The next step is to define the main component of the computational effort. In this system, a TensorFlow graph is used to handle model and parameter optimization. Built-in, the library uses different methods of optimization to trigger updates to the parameters as necessary. Once such method is the auto-differentiation technique of TensorFlow. In summary, this technique allows TensorFlow to predict what parameters will result in higher or lower loss by using the gradient descent function. In this system, the gradient descent accepts a parameter that represents the learning rate. Ultimately, the learning rate allows the system to determine how significant updates to model parameters are after completion of a training step.

        Once the TensorFlow graph has been defined, it can be initialized to run. After being started, the graph begins the first phase of training via a loop construct. During this phase, the Numpy module is used to randomly select images from the training subset of data. After each step, the accuracy of the training phase is calculated. During this accuracy check, the TensorFlow graph provides the means to update the model’s parameters. After training, ideally, the model’s parameters are to updated with the best configuration for accurate image recognition. Now, the testing subset (the 10,000 remaining images) can be used as input for the model.

        The results of the Softmax Classifier vary with different configurations of the batch size, learning rate, and max step parameters. For the simplicity of this report, the following parameter values: 100, 0.005, and 1000, respectively, will be discussed (other test results may be viewed in the included results.xlsx). When discussing the results, it is also important to note that with ten possible classifications of images, there is a one in ten chance, the machine could randomly guess the correct category. Therefore, we want our system to perform better (ideally much greater) than ten percent on testing accuracy. On average, we achieved a testing accuracy of 26.01%. The maximum accuracy was 31.39% for the same three runs. From this, we can conclude that the Softmax Classifier, on average, at least performs better than random guessing by a factor of about 2.6. It should also be noted that this approach only uses a basic method of machine learning and the model parameters could be further optimized for enhanced performance. However, those adjustments go beyond the scope of this particular project.

**Two-Layer Fully Connected Neural Network Implementation**

        The second implementation of an image recognition system utilizes a two-layer fully connected neural network. This application represents a deep learning approach to the problem. We know from our initial Softmax Classifier results that a basic machine learning approach with a Softmax function offered at least double the accuracy of random guessing. To improve upon these results, we must alter the model of the Softmax Classifier. This is accomplished by creating a two-layer neural network. This network consists of a hidden layer and an output layer. For this model, the input received does not represent a true layer of the network as the CIFAR-10 data is only read in and not transformed by operations. Also, this implementation represents a neural network in which the input for the first layer neurons are the corresponding image pixel values. The model is full-connected because the output of the layer one neurons are the total inputs for the layer two neurons. It should also be noted that that the neurons of each layer are independent of each other, but can pass multiple outputs to the next layer neurons. Also, in this system, both layers are more similar than different to each other.

        The first layer of the network makes use of an operation known as a Rectified Linear Unit (ReLU). To alter the model from the Softmax Classifier to the neural network, the model parameters for the weighted sum values are altered. When calculating training scores for calculations, negative numbers are no longer generated as outputs. Instead, negative values for this variable are changed to zero while allowing positive values to pass through as input to the next neuron. This is achieved through a rectifier function of the TensorFlow library. This functionality allows for nonlinearity to be introduced into the layers so that each successive layers exhibits improved performance. The input values for this layer are the individual image pixel values. The output of each neuron then becomes a rectified score for training. These units are called “hidden units” and reflect the results of the more basic Softmax Classifier with nonlinearity.

        Next, these results are fed into the output layer. Ultimately, the output layer receives the rectified pixel score values as input and then outputs the final value of the training score which represents the classification. To accomplish this, the hidden units are first multiplied by the weights. Next, the bias is then added to the results and the highest score is chosen for each class. This score represents the overall output of layer two.

        Once the entire network structure is defined, the model for the classifier can be completed. As with the Softmax Classifier, an important step is to grade the accuracy of each training stage. With the neural network implementation, further evaluation is necessary to ensure improved results. This takes the form of improved loss calculation, introducing regularization, improved variable optimization, and enhanced performance evaluation. After each iteration of the model, these methods are performed to further evaluate the overall accuracy of the predictions. Also, like in Softmax, the classifier updates the model parameters as necessary to attempt to improve each successive stage of training and evaluation.

        The results of the two-layer model also vary with different configuration options. However, the performance of the two-layer neural network is visibly increased from the Softmax Classifier. With the neural network, the same configuration options of batch size, learning rate, and max steps were declared on the command line. However, the learning rate value was initially set to 0.001 for time constraints. This is due to the neural network generally taking longer to run than the Softmax Classifier. Also, an additional argument for the number of hidden layer neurons was used. This value was initialized at 120 and remained a constant for the neural network testing process. When testing with the values 100, 0.001, 1000, and 120, respectively, the average accuracy was 37.25%. The maximum accuracy was very close at 37.60%. Therefore, on average, the neural network performed better than the Softmax Classifier. However, it should also be noted that one particular run of the neural network had an average testing accuracy 49.39% and training accuracy of 81%. The configuration settings for this run were a batch size of 100, learning rate of 0.0005, max step size of 20,000, and hidden layer neurons count of 120. In this case, the performance increase could be due to the learning rate being halved but the training duration increased. In short, the training process took longer but had a greater amount of time to achieve optimization.

**Future Work**

The classifier implementations in this system provide two basic methods of image recognition through machine learning and deep learning. However, future work could enhance both these systems to create more complex implementations. One method would be to provide a means to introduce new images to the system after it has been trained. The idea with this expansion would be more for visual understanding. Allowing the user to pick a single image at a time for the system to classify. This could be accomplished by providing a command line argument with an image file name. The system would output a string identifier for the class of the image it predicts.

Another more complex approach would be to enhance the neural network by creating a convolutional neural network. With a fully-connected implementation, each neuron or pixel is connected to every other neuron. Thus, a given neuron is just as related to adjacent neurons as it is to a neuron associated with a pixel on the opposite side of the image. In a convolutional neural network, the system would treat individual pixels with higher degrees of significance based on some other parameter or relation. This could be accomplished by breaking images down to subsets that are a factor of image size. For example, you might subdivide a 32x32 pixel image into 8x8 or 4x4 regions. A justification for this is that objects may often not be found in the center of the frame or the object by be transformed in some way, such as a different camera perspective. Using regional subsets could allow the system to search for patterns found in those subsets in other areas of the frame and potentially provide greater testing accuracy.

**Conclusion**

The preceding tutorial, analysis, and experience has greatly increased our understanding of AI systems with machine learning, neural networks, and deep learning. Also, it has provided a great deal of exposure to libraries such as TensorFlow. Tensorflow is a powerful library, not only does it provide all of the tools for the development of neural networks, it also provides many tools for the analysis of those systems. Overall, the experience has provided us with the core understanding and tools necessary to begin developing both simple and more complex systems for different topics and problems in AI.

**References**

CIFAR-10

<https://www.cs.toronto.edu/~kriz/cifar.html>

“How to Build a Simple Image Recognition System with TensorFlow” by Wolfgang Beyer

<http://www.wolfib.com/Image-Recognition-Intro-Part-1/>

<http://www.wolfib.com/Image-Recognition-Intro-Part-2/>

TensorFlow Resources

<https://www.tensorflow.org/install/install_windows>

<https://www.tensorflow.org/programmers_guide/summaries_and_tensorboard>

Wolflib GitHub

<https://github.com/wolfib/image-classification-CIFAR10-tf>

Lee Beadle / Chad Etheredge

CS 470 Artificial Intelligence

Term Project – Image Recognition System

4/30/2018

**Image Recognition Through Machine Learning and Deep Learning**

        What is image recognition? For humans, it is often a glance at an object and the corresponding recognition. It happens in the blink of an eye and without much effort. For machines, the ability to view an image and identify that image is much more complex. Artificial intelligence in the form of machine learning, neural networks, and deep learning helps solve this problem. The following system is an example of an AI system for image recognition that uses two implementations to achieve its goal. First, a basic machine learning classifier is developed using a Softmax algorithm. The second, is a more complex two-layer neural network implementation for deep learning applications. Both implementations were built following the tutorial blog “How to Build a Simple Image Recognition System with TensorFlow” by Wolfgang Beyer. As an additional note, source files used in this project were pulled from Wolfgang Beyer’s public GitHub, under “Wolflib”, and were used to complete the tutorial and system. Links to these are provided under our references section.

**Our Approach**

        The first approach to this AI system is through a machine learning implementation. The second is through a two-layer fully connected neural network. Both systems seek to analyze a dataset of pictures and predict which images belong to a specific category. For this project, the Softmax Classification system is built first and then altered to become a fully connected two-layer neural network. Also, both of these programs make use of TensorFlow’s powerful graph data structure and the Numpy package for calculation. The data analyzed is the CIFAR-10 dataset. This dataset is a collection of 60,000 images that are classified into ten categories that contain both objects and animals. The categories are airplane, car, bird, cat, deer, dog, frog, horse, ship, and truck. Also, the CIFAR-10 dataset is a standardized collection in which each image is 32x32 pixels. It is important to note that the small size of the pictures can actually hinder human recognition of the objects in the images but eases the computational workload for the system.

**Softmax Implementation**

        To begin constructing the Softmax Classifier, a means for the computer to recognize images is needed. This is accomplished by building a mathematical model that encompasses two critical stages to function. The first stage, is the training phase. In this phase, 50,000 of the total images from the training subset are analyzed but system. Operations performed on the training data provide the knowledge base for making the predictions necessary for image recognition. This first portion of the model, utilizes supervised learning to train the system. With supervised learning, the system is provided both the raw input data and correct image classification. This ensures that the system can evaluate and scale how accurate its predictions are through each training step.

The testing phase, the second stage of the model, consists of 10,000 images and represents the pool of data in which the machine will attempt to accurately recognize objects or animals. Both phases of this model reflect the accuracy of recognition which ultimately represents the most important metric for this project.

        However, one important aspect has been overlooked: How *exactly* does the system recognize objects or in animals within each image? To begin, the images are first enumerated in terms the system can manage efficiently. Each image is broken into its base form of individual pixels. With a 32x32 image, there are 1024 individual pixels. However, each pixel can be represented by one of three possible color scales: Red, green, and blue. Therefore, each image can be numerically represented by 3072 total values. When the system observes each individual pixel, a probabilistic prediction, using the Softmax function, is made on whether that pixel’s color is more or less likely to belong to a specific class. This prediction’s likelihood is represented by a weight. The color channel value is then multiplied by the weight and all 3072 values are summed to single value for each of the category classes. Finally, the highest scoring class is taken as the prediction for the image.

        To begin building the model, the model is fed input images in the form of the numerical values. The model’s initial input uses simple starting values as its parameters. During training, the model accepts the image input and compares them with known image classifications. As the model runs, the system updates its parameters to match known classifiers. The key to the model updating is performing optimization of the weights and biases variables and model parameters. Therefore, the more optimized these values are at each iteration of training, the more accurate the training results will become.

        The next step is to define the main component of the computational effort. In this system, a TensorFlow graph is used to handle model and parameter optimization. Built-in, the library uses different methods of optimization to trigger updates to the parameters as necessary. Once such method is the auto-differentiation technique of TensorFlow. In summary, this technique allows TensorFlow to predict what parameters will result in higher or lower loss by using the gradient descent function. In this system, the gradient descent accepts a parameter that represents the learning rate. Ultimately, the learning rate allows the system to determine how significant updates to model parameters are after completion of a training step.

        Once the TensorFlow graph has been defined, it can be initialized to run. After being started, the graph begins the first phase of training via a loop construct. During this phase, the Numpy module is used to randomly select images from the training subset of data. After each step, the accuracy of the training phase is calculated. During this accuracy check, the TensorFlow graph provides the means to update the model’s parameters. After training, ideally, the model’s parameters are to updated with the best configuration for accurate image recognition. Now, the testing subset (the 10,000 remaining images) can be used as input for the model.

        The results of the Softmax Classifier vary with different configurations of the batch size, learning rate, and max step parameters. For the simplicity of this report, the following parameter values: 100, 0.005, and 1000, respectively, will be discussed (other test results may be viewed in the included results.xlsx). When discussing the results, it is also important to note that with ten possible classifications of images, there is a one in ten chance, the machine could randomly guess the correct category. Therefore, we want our system to perform better (ideally much greater) than ten percent on testing accuracy. On average, we achieved a testing accuracy of 26.01%. The maximum accuracy was 31.39% for the same three runs. From this, we can conclude that the Softmax Classifier, on average, at least performs better than random guessing by a factor of about 2.6. It should also be noted that this approach only uses a basic method of machine learning and the model parameters could be further optimized for enhanced performance. However, those adjustments go beyond the scope of this particular project.

**Two-Layer Fully Connected Neural Network Implementation**

        The second implementation of an image recognition system utilizes a two-layer fully connected neural network. This application represents a deep learning approach to the problem. We know from our initial Softmax Classifier results that a basic machine learning approach with a Softmax function offered at least double the accuracy of random guessing. To improve upon these results, we must alter the model of the Softmax Classifier. This is accomplished by creating a two-layer neural network. This network consists of a hidden layer and an output layer. For this model, the input received does not represent a true layer of the network as the CIFAR-10 data is only read in and not transformed by operations. Also, this implementation represents a neural network in which the input for the first layer neurons are the corresponding image pixel values. The model is full-connected because the output of the layer one neurons are the total inputs for the layer two neurons. It should also be noted that that the neurons of each layer are independent of each other, but can pass multiple outputs to the next layer neurons. Also, in this system, both layers are more similar than different to each other.

        The first layer of the network makes use of an operation known as a Rectified Linear Unit (ReLU). To alter the model from the Softmax Classifier to the neural network, the model parameters for the weighted sum values are altered. When calculating training scores for calculations, negative numbers are no longer generated as outputs. Instead, negative values for this variable are changed to zero while allowing positive values to pass through as input to the next neuron. This is achieved through a rectifier function of the TensorFlow library. This functionality allows for nonlinearity to be introduced into the layers so that each successive layers exhibits improved performance. The input values for this layer are the individual image pixel values. The output of each neuron then becomes a rectified score for training. These units are called “hidden units” and reflect the results of the more basic Softmax Classifier with nonlinearity.

        Next, these results are fed into the output layer. Ultimately, the output layer receives the rectified pixel score values as input and then outputs the final value of the training score which represents the classification. To accomplish this, the hidden units are first multiplied by the weights. Next, the bias is then added to the results and the highest score is chosen for each class. This score represents the overall output of layer two.

        Once the entire network structure is defined, the model for the classifier can be completed. As with the Softmax Classifier, an important step is to grade the accuracy of each training stage. With the neural network implementation, further evaluation is necessary to ensure improved results. This takes the form of improved loss calculation, introducing regularization, improved variable optimization, and enhanced performance evaluation. After each iteration of the model, these methods are performed to further evaluate the overall accuracy of the predictions. Also, like in Softmax, the classifier updates the model parameters as necessary to attempt to improve each successive stage of training and evaluation.

        The results of the two-layer model also vary with different configuration options. However, the performance of the two-layer neural network is visibly increased from the Softmax Classifier. With the neural network, the same configuration options of batch size, learning rate, and max steps were declared on the command line. However, the learning rate value was initially set to 0.001 for time constraints. This is due to the neural network generally taking longer to run than the Softmax Classifier. Also, an additional argument for the number of hidden layer neurons was used. This value was initialized at 120 and remained a constant for the neural network testing process. When testing with the values 100, 0.001, 1000, and 120, respectively, the average accuracy was 37.25%. The maximum accuracy was very close at 37.60%. Therefore, on average, the neural network performed better than the Softmax Classifier. However, it should also be noted that one particular run of the neural network had an average testing accuracy 49.39% and training accuracy of 81%. The configuration settings for this run were a batch size of 100, learning rate of 0.0005, max step size of 20,000, and hidden layer neurons count of 120. In this case, the performance increase could be due to the learning rate being halved but the training duration increased. In short, the training process took longer but had a greater amount of time to achieve optimization.

**Future Work**

The classifier implementations in this system provide two basic methods of image recognition through machine learning and deep learning. However, future work could enhance both these systems to create more complex implementations. One method would be to provide a means to introduce new images to the system after it has been trained. The idea with this expansion would be more for visual understanding. Allowing the user to pick a single image at a time for the system to classify. This could be accomplished by providing a command line argument with an image file name. The system would output a string identifier for the class of the image it predicts.

Another more complex approach would be to enhance the neural network by creating a convolutional neural network. With a fully-connected implementation, each neuron or pixel is connected to every other neuron. Thus, a given neuron is just as related to adjacent neurons as it is to a neuron associated with a pixel on the opposite side of the image. In a convolutional neural network, the system would treat individual pixels with higher degrees of significance based on some other parameter or relation. This could be accomplished by breaking images down to subsets that are a factor of image size. For example, you might subdivide a 32x32 pixel image into 8x8 or 4x4 regions. A justification for this is that objects may often not be found in the center of the frame or the object by be transformed in some way, such as a different camera perspective. Using regional subsets could allow the system to search for patterns found in those subsets in other areas of the frame and potentially provide greater testing accuracy.

**Conclusion**

The preceding tutorial, analysis, and experience has greatly increased our understanding of AI systems with machine learning, neural networks, and deep learning. Also, it has provided a great deal of exposure to libraries such as TensorFlow. Tensorflow is a powerful library, not only does it provide all of the tools for the development of neural networks, it also provides many tools for the analysis of those systems. Overall, the experience has provided us with the core understanding and tools necessary to begin developing both simple and more complex systems for different topics and problems in AI.

**References**

CIFAR-10

<https://www.cs.toronto.edu/~kriz/cifar.html>

“How to Build a Simple Image Recognition System with TensorFlow” by Wolfgang Beyer

<http://www.wolfib.com/Image-Recognition-Intro-Part-1/>

<http://www.wolfib.com/Image-Recognition-Intro-Part-2/>

TensorFlow Resources

<https://www.tensorflow.org/install/install_windows>

<https://www.tensorflow.org/programmers_guide/summaries_and_tensorboard>

Wolflib GitHub

<https://github.com/wolfib/image-classification-CIFAR10-tf>