SHARPNESS-AWARE MINIMIZATION

COURSE: MIT-5432

PROFESSOR: DR. ZHAI

AUTHORS: NICK PRANSKE, ANDREW SOLTIS, REID RORICK

Abstract

Sharpness-Aware Minimization (also known as SAM) is an optimizer that attempts to help models that struggle to generalize. Many models struggle to generalize, meaning while they are very much optimized to the data they were trained on, they are very poor when being applied to others. SAM seeks to minimize both the loss value and the sharpness simultaneously instead of focusing strictly on the loss value. SAM's usefulness comes in its ability to help models generalize over to more datasets. Studies have shown that there is a lot of potential, as well as proven, for improved results when compared to models without SAM. The SAM optimizer essentially brings the model through a loss landscape with much flatter minimums. Traditional optimizers and models are very sharp, essentially meaning that any slight deviation could cause the results to increase in error while SAM is more apt to handle the deviations.

Domain introduction

The domain chosen for this project was Computer Vision (CV). In a general sense, Computer Vision is the capability of a computer to mimic human vision and visual processing. By analyzing photo or video input, CV can perform detection, classification, or segmentation. In this use-case, the CV is performing classification – it is identifying what object is shown in the picture

Task introduction

The task of this project is to optimize the ResNet model architecture by replacing the current Adam optimizer with the Sharpness-Aware Minimization (SAM) optimizer. This model, in the chosen example, is applied to image classification datasets such as CIFAR-10 and SVHN (Street View House Numbers). As is exists, the Adam optimizer seeks to adjust the hyperparameters of the ResNet model such that the Loss Function is minimized. While this create a theoretical local and global minima for the model to reach using Stochastic Gradient Descent, the high Sharpness that usually accompanies a low Loss minimum makes it hard for the function to reach the optimal Loss value in practice. To solve this, SAM seeks to adjust the hyperparameters to account for both Sharpness and Loss, so that the Loss minimum is attainable for the model to achieve.

Dataset introduction

The dataset used is CIFAR-10. This dataset is commonly used in image recognition and classification tasks. CIFAR-10 consists of 60,000 images total. This total is split up between training and testing/validation images. With about 85% (50,000) of the images being a part of the training and about 15% (10,000) being validation images. Each image is 32 pixels by 32 pixels and has color. Within the dataset, there are 10 classes (each housing 6,000 images). The 10 classes include: airplanes, automobiles, birds, cats, deer, dogs, frogs, horses, ships, and trucks. The diverse nature of the images within the dataset and being in color help this dataset be more well-rounded for machine learning. The 60,000 images are also separated into 1 testing and 5 training batches each with a batch size of 10,000. The dataset needs to be divided into batches because it helps with computation. Instead of having your epochs go through all your data in every pass, they go through the different batches, allowing for better computational efficiency.

Problem formulation

The problem is that current, unoptimized models have a problem with overfitting and generalization. The reason that overfitting is bad and causes a lower accuracy and efficiency with generalization is because overfitting is where the model can give accurate predictions based on the training set, however, when new / outside data is introduced, it has poor performance. The SAM optimizer attempts to limit this problem. As described, SAM stands for Sharpness-Aware Minimization which at its core is used to limit model loss values and sharpness. SAM is added to a model to help it reduce these values essentially by creating flatter minimums so that small deviations do not result in a huge change in results. Overall, using the unoptimized model on more generalized data outside the training data is less accurate. By generalizing the model, it can perform better on a wider variety of data, which is something that SAM aims to improve.

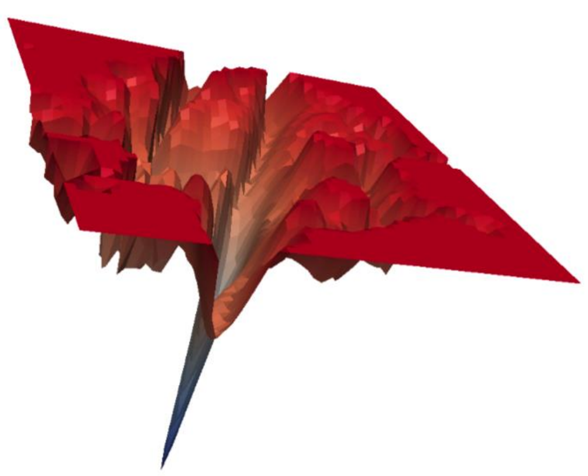
Related work

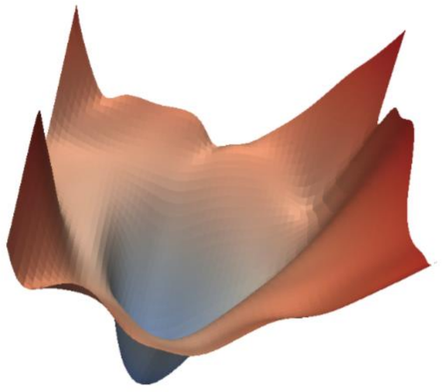
Some of the work that has been done for similar tasks are SWA (Stochastic Weight Averaging) and Weight Perturbation Methods. SWA focuses on using the concept of flatness and sharpness in the loss landscape to ensure better generalization. Weight Perturbation Methods also successfully improve standard generalization on multiple deep learning benchmarks. There is a lot of related work out there because generalization and overparametrized Deep Neural Networks are common in machine learning. The paper theoretically analyzes SAM's implicit bias for diagonal linear networks and shows that SAM always chooses a solution that enjoys better generalization properties than standard gradient descent for a certain class of problems.

Model/Optimizer details

The ResNet model architecture, which is short for Residual Network, is a deep neural network aimed to address the "vanishing gradient problem". Essentially, with other model architectures it becomes exponentially harder to train the model as the network gets deeper. This occurs when the gradient of the loss function becomes smaller and smaller, so it is difficult to propagate back through the network and adjust the weights accordingly. The ResNet solves this by introducing a "skip connection", which allows the gradients to flow more easily during training and backpropagation. Instead of mapping straight from the input to the output, it takes the difference (residual) between the two and maps based on that. Adding the residual to the convolutional layers allows for more effective propagation of the gradients. ResNet-20 is a variant of the ResNet architecture published in "Deep Residual Learning for Image Recognition" (2015) by He, Zhang, Ren, and Sun. The "20" comes from the 20 convolutional layers in the model. It utilizes the ReLU activation function and Adam optimization.

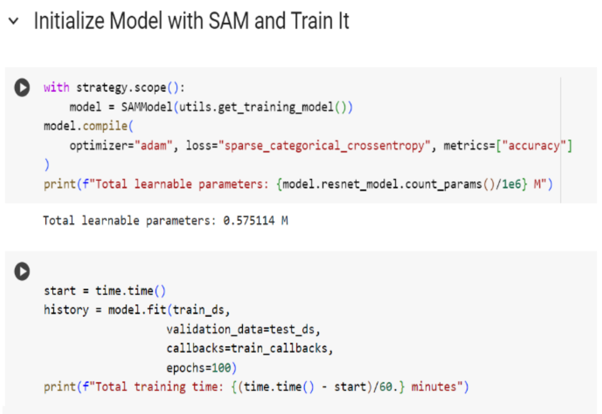
As stated in our report, SAM aims to minimize loss values and sharpness. This is not something that is just made for computer vision models. It can be used with many kinds of models because at its core it is made to help models improve generalization through flatter minimums. SAM is great in that it can be personally tailored to each model it will be used on. A few examples of models that SAM can be applied to are as follows: WRN-28-10, Shake-Shake, and PyramidNet. An optimizer is essentially just a function or algorithm that aids a model in increasing accuracy and reducing loss by adjusting attributes of the model and processes. Because SAM specifically was created for image recognition, you would need to adjust the weights for sharpness depending on the kind of model that you are using. SAM as a framework is easily able to be used on different models because of its compatibility with already existing gradient-based models/optimizers and with only needing to make relatively minor adjustments overall.





Implementation code snippets

We used Google Collab to run our code. In the first code snippet, we see the model being trained with the optimizer SAM. Here is where the SAM optimizer goes into effect to increase the model's accuracy and reduce loss. When you run the first code snippet it will begin to train the model with SAM. In the second code snippet, it is the results of a model that does not use SAM. This code allows us to compare the different results of a model that does not use same and one that does use it to be able to see if SAM does help to improve the accuracy through reducing loss values. We have also included a code snippet of importing and creating the SAM optimizer inside of the project.

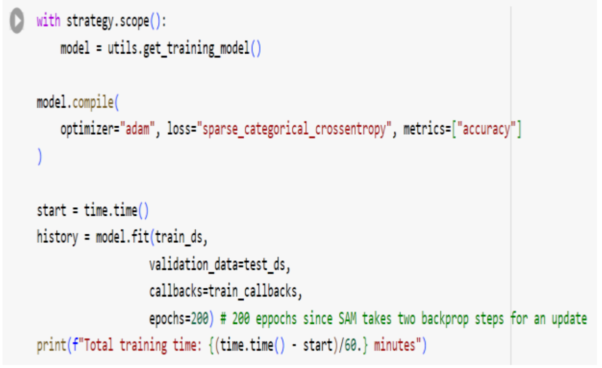


Initialize Model with SAM and Train It

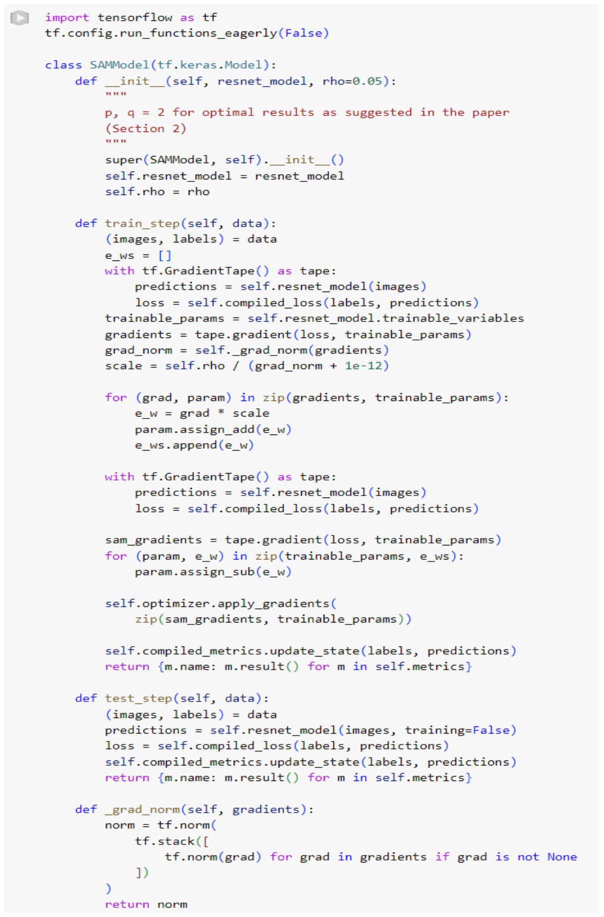
```
with strategy.scope():
    model = SAMModel(utils.get_training_model())
model.compile(
    optimizer="adam", loss="sparse_categorical_crossentropy", metrics=["accuracy"]
)
print(f"Total learnable parameters: {model.resnet_model.count_params()/1e6} M")
```

```
Total learnable parameters: 0.575114 M
```

```
start = time.time()
history = model.fit(train_ds,
                    validation_data=test_ds,
                    callbacks=train_callbacks,
                    epochs=100)
print(f"Total training time: {(time.time() - start)/60.} minutes")
```

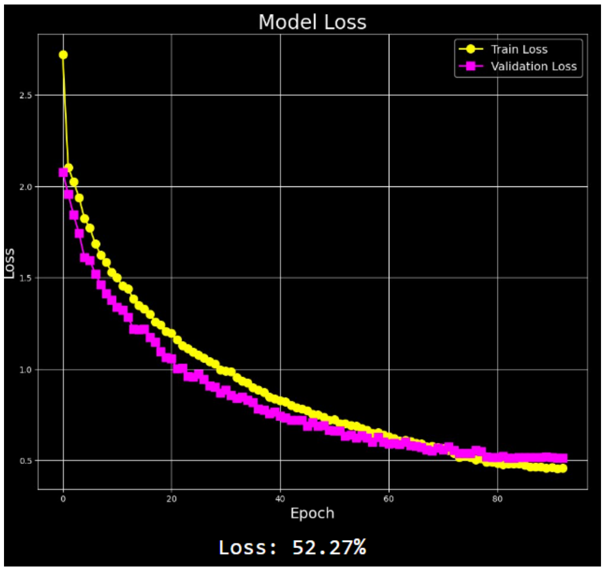


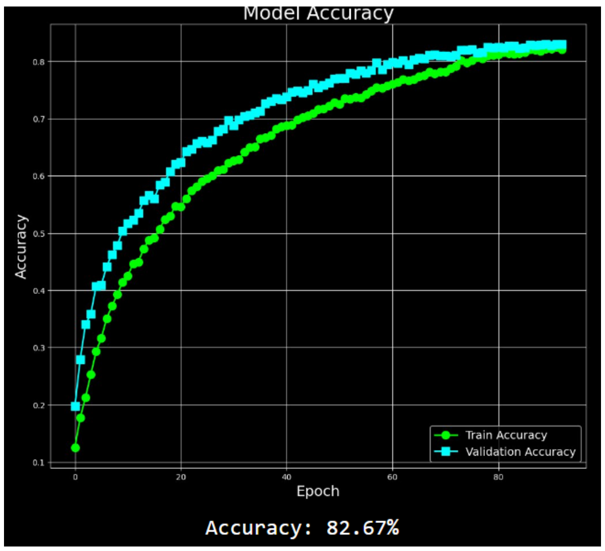
```
with strategy.scope():
    model = utils.get_training_model()

model.compile(
    optimizer="adam", loss="sparse_categorical_crossentropy", metrics=["accuracy"]
)

start = time.time()
history = model.fit(train_ds,
                    validation_data=test_ds,
                    callbacks=train_callbacks,
                    epochs=200) # 200 eppochs since SAM takes two backprop steps for an update
print(f"Total training time: {(time.time() - start)/60.} minutes")
```

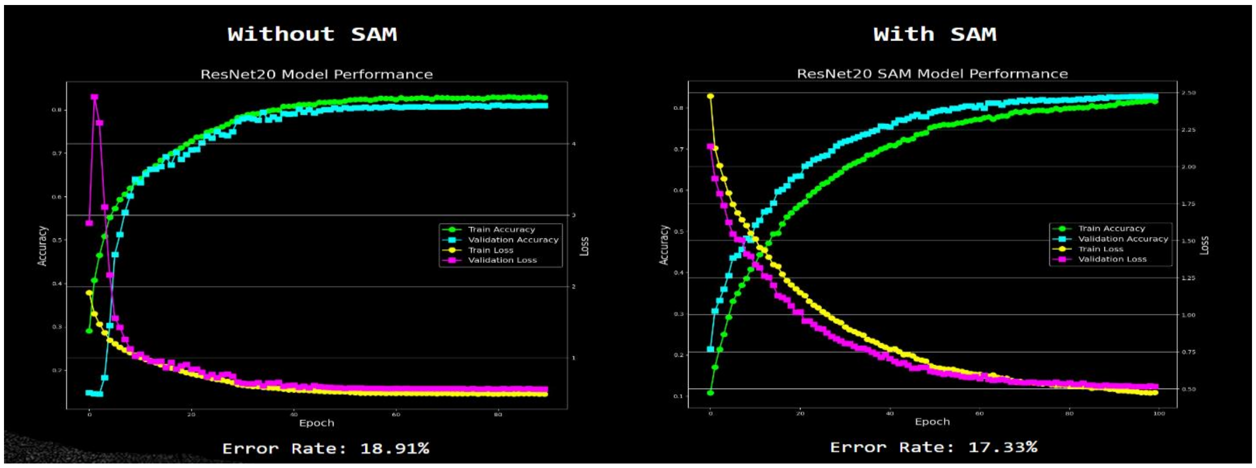


```
import tensorflow as tf
tf.config.run_functions_eagerly(False)

class SAMModel(tf.keras.Model):
    def __init__(self, resnet_model, rho=0.05):
        """
        p, q = 2 for optimal results as suggested in the paper
        (Section 2)
        """
        super(SAMModel, self).__init__()
        self.resnet_model = resnet_model
        self.rho = rho

    def train_step(self, data):
        (images, labels) = data
        e_ws = []
        with tf.GradientTape() as tape:
            predictions = self.resnet_model(images)
            loss = self.compiled_loss(labels, predictions)
        trainable_params = self.resnet_model.trainable_variables
        gradients = tape.gradient(loss, trainable_params)
        grad_norm = self._grad_norm(gradients)
        scale = self.rho / (grad_norm + 1e-12)

        for (grad, param) in zip(gradients, trainable_params):
            e_w = grad * scale
            param.assign_add(e_w)
            e_ws.append(e_w)

        with tf.GradientTape() as tape:
            predictions = self.resnet_model(images)
            loss = self.compiled_loss(labels, predictions)

        sam_gradients = tape.gradient(loss, trainable_params)
        for (param, e_w) in zip(trainable_params, e_ws):
            param.assign_sub(e_w)

        self.optimizer.apply_gradients(
            zip(sam_gradients, trainable_params))

        self.compiled_metrics.update_state(labels, predictions)
        return {m.name: m.result() for m in self.metrics}

    def test_step(self, data):
        (images, labels) = data
        predictions = self.resnet_model(images, training=False)
        loss = self.compiled_loss(labels, predictions)
        self.compiled_metrics.update_state(labels, predictions)
        return {m.name: m.result() for m in self.metrics}

    def _grad_norm(self, gradients):
        norm = tf.norm(
            tf.stack([
                tf.norm(grad) for grad in gradients if grad is not None
            ])
        )
        return norm
```

Testing results from your code







These are the results from our code. As seen from the graphs we provided, accuracy has a positive, upward trend with training and validation (depicted on the top right). The other graph (depicted on the top left) shows us the trend of loss between training and validation. In the graph, you see a downward trend for both. These results alone do not tell us everything that we want to know, so we need to compare them with and without SAM (shown in the bottom image). Once you compare the two, you can immediately see a difference in the results. You see that with SAM the error rate decreases. This is an obvious improvement, however, if you take a closer look at these results as well as read through the paper and documentation, you see that SAM is also more consistent.

Comparison

Within the paper's analysis, they used CIFAR-10 as well as multiple other datasets to test SAM's performance, like CIFAR-100 and ImageNet. Along with that, we used ResNet-20, however, the paper did not use ResNet-20 as a model, it uses multiple other ones, just like how our datasets are compared. This is because overall the people who created the paper had overall more knowledge on the topic as well as more time and assets to be able to test it against all the different criteria / models. Even though different models and datasets were used, we still arrived at a similar conclusion in the end. Both results showed us that SAM improves performance by decreasing loss, increasing accuracy, and increasing the consistency of the results. If there is anything to take away from both our results and the results of the original paper, SAM is essential when attempting to improve generalization and reduce overfitting because of how consistent it performs.

Discussion

The concept of classifying images using a CNN is interesting in and of itself. Additionally, though, and more relevant to the original paper and to this project, being able to better optimize and generalize those models is ultimately what makes technology like this applicable to real-world use cases. It is widely known that one of the most prevalent issues in data is its cleanliness and consistency. By using the concept of SAM, deep learning models will be able to be more accurately applied to less clean and consistent data, thus making them more usable in the real world.

One thing that has become clear throughout this class is that accuracy of models is incredibly important to their uses. The example that comes to mind is the cancer detection model mentioned by guest speaker Shane Jason Mock. In that example, each percentage of accuracy correlates to real lives. Therefore, having a good optimizer that improves model performance – no matter how small the improvement – is essential to building a good model.

Conclusion

The main conclusion is that models with SAM perform better than models without SAM. SAM increases performance and performs better, more consistently. SAM offers better performance consistently to a wider range of models. A model with SAM, in comparison to a model without, is a lot more generalized and works better against overfitting. This means that the model will perform better against outside data that it was not trained for. A model without SAM can have an issue with overfitting making it fit well to the data that it was trained for, but it performs a lot worse against outside data. During training, SAM also provides a more optimal and stable path for the model. Another benefit that we found during this project is that SAM can be applied to multiple models, making it a great tool to add to your models to help improve accuracy, reduce loss, and match to outside data better.

References

https://colab.research.google.com/github/sayakpaul/Sharpness-Aware-Minimization-TensorFlow/blob/main/SAM.ipynb

https://arxiv.org/pdf/2010.01412.pdf

https://github.com/google-research/sam

https://arxiv.org/pdf/1609.04836.pdf

https://keras.io/guides/customizing_what_happens_in_fit/

https://github.com/davda54/sam

https://www.tensorflow.org/guide/keras/custom_layers_and_models#the_model_class

https://arxiv.org/pdf/2110.08529.pdf

https://www.inference.vc/sharp-vs-flat-minima-are-still-a-mystery-to-me/

https://en.wikipedia.org/wiki/Dual_norm

https://arxiv.org/abs/2104.10645v1