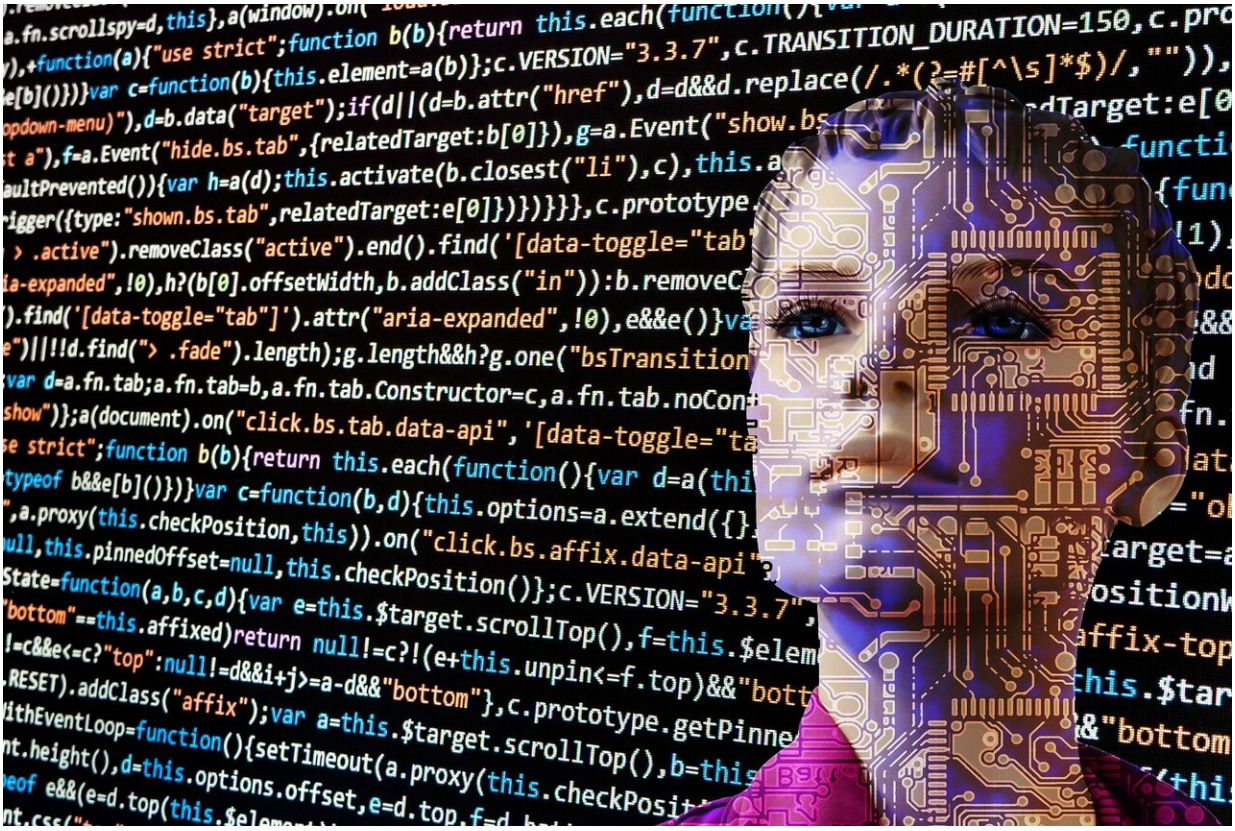
A faster way to optimize deep learning models

May 31 2024, by Alvin Lee



Credit: CC0 Public Domain

AI and its related terms are now fairly well known. Many people have heard of terms such as "neural network" and may even associate "CNN" with "Convolutional Neural Network" instead of the news organization.

Those with more than a passing interest in AI might even know about AlexNet, the pioneering CNN architecture that revolutionized image recognition and deep learning in 2012.

What is lesser known is the use of optimizers or optimization algorithms, which help improve the performance of AI models. For example, computer vision AI models would need optimizers that receive data input (a visual image) and correctly 'predict' that data, i.e., correctly identify an image of a panda as "panda" instead of "bear" or "koala".

"Panda" would be the ground truth that the AI model should correctly predict every time, while the difference between the AI prediction and the ground truth is quantified into a figure called training loss.

"Given a task, an AI model will take input samples and output its prediction. Without training, an AI model often cannot predict correctly, and thus perform poor on the task," explains Zhou Pan, Assistant Professor of Computer Science at SMU. "An optimizer is to update an AI model's parameters so that the AI model can make correct predictions."

"The primary role of an optimizer is to feed training samples into the AI model, then compute the training loss, i.e., the discrepancy between model's prediction and the ground truth prediction, and finally adjust the model parameter to minimize the training loss."

Solving overshoot

Different types of deep learning networks require different optimizers, often with the most suitable one selected only after multiple trials that are often costly and time-consuming.

In simple words, an optimizer does its job when an AI model's output

corresponds to the lowest point on an approximately V-shaped curve charting training loss, which is often referred to as the convergence point. This is where the model has learned the optimal set of parameters, such that further training iterations do not significantly improve its performance on the task at hand.

A key obstacle to efficient optimization is something called the "Overshoot Issue", whereby an optimizer produces predictions corresponding to the other side of the V-shaped curve, which requires recalibration to bring the prediction back to the contours of the curve.

Professor Zhou's latest project, titled "Adan: Adaptive Nesterov Momentum Algorithm for Faster Optimizing Deep Models," attempts to solve the overshoot issue.

He explains, "The Adan optimizer can accelerate the process to find good model parameters of a model. At each training iteration, like other optimizers, Adan also feeds data into the model and then computes the training loss, and finally computes the gradients of model parameters.

"But when it uses the gradient to update the parameter, it will first take a step to update the model parameter, looking at whether the current model parameter update is good or not. If it is, then it will update the model parameter in a larger step; otherwise, it will take a small step to update the parameter slowly. This ensures the parameter update is always in the right way, and thus guarantees faster convergence speed."

An Epoch(al) achievement

Improvements in neural network training can be measured in epochs, where one epoch is a complete pass or cycle through the entire training dataset.

Professor Zhou expects Adan to outdo existing state of the art (SoTA) optimizers for major deep learning tasks such as visual, language, and reinforcement learning such as that which underpinned AlphaGo, the AI model that beat the world's top-ranked human player in the ancient board game, Go, in 2017.

"Overall, Adan can use half of training iteration to achieve comparable performance of SoTA optimizers," Professor Zhou elaborates.

"For vision tasks, on the ViT and Swin models for supervised image classification task, Adan can use 150 training epochs to achieve similar performance as the SoTA optimizer, AdamW, which trains 300 epochs. On the MAE model for self-supervised image classification tasks, Adan can use 800 training epochs to achieve similar performance as the SoTA optimizer, AdamW, which trains 1,600 epochs.

"For language tasks, on GPT2, Adan can use 150k training iterations to achieve similar performance as the SoTA optimizer, Adam, which trains 150k training iterations; on Transformer-XL, Adan can use 100k training iterations to achieve the same performance as the SoTA optimizer, Adam, which trains 200k training iterations."

For RL, or reinforcement learning tasks, Adan works on four games, namely Ant, Half Cheetah, Humanoid, and Walker2d. For simplicity, one often calls them MuJoCo games. These games are designed to control the body of a robot to finish different activities in a 3D environment stably and robustly, like walking and running.

"On RL, by using the same training iterations, Adan always achieves higher performance than the SoTA optimizer, Adam, on the four tested game tasks," says Professor Zhou.