National Tsing Hua University
Fall 2023 11210IPT 553000
Deep Learning in Biomedical Optical Imaging
Homework 3

AUTHOR

柯志明

Student ID: 111003804

1. Task A: Reduce Overfitting (30 pts)

There are several methods to address overfitting, including dropout, weight decay, data augmentation, reducing model complexity, early stopping, and batch normalization, among others. I have experimented with each of these methods and their combinations in my code. However, only a few of them have proven effective in mitigating overfitting in this case. I will illustrate the weight-decay model as the example below.

Discussion (20 pts)

Weight Decay – A weight-decay model is implemented by applying an L2 penalty to the weights, with λ=0.01 selected. Upon examining the green lines (representing the weight-decay model) in Figure 2, it becomes evident that performance gaps between the training and validation data persist, although they are smaller than those of the original ConvModel. Similar results were obtained when experimenting with various values of λ, indicating no significant improvement in mitigating the overfitting issue concerning the training and validation data.

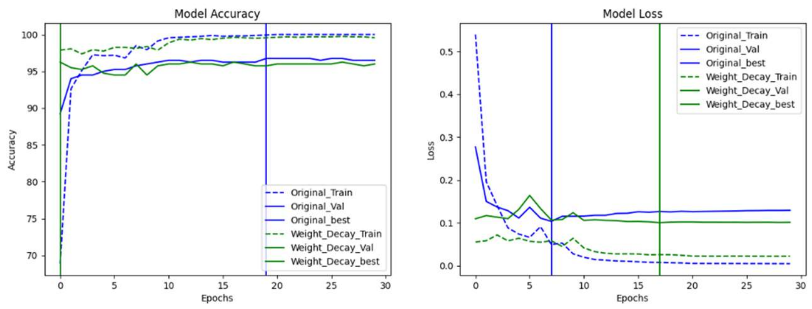


Fig. 2 displays the validation (solid) vs. training performance (dashed), including the accuracy curve (a) and the loss curve (b) for both the original ConvModel (blue) and the weight-decay ConvModel (green) with $\lambda = 0.01$. The vertical lines indicate the epochs at which the best validation performance occurs for each model.

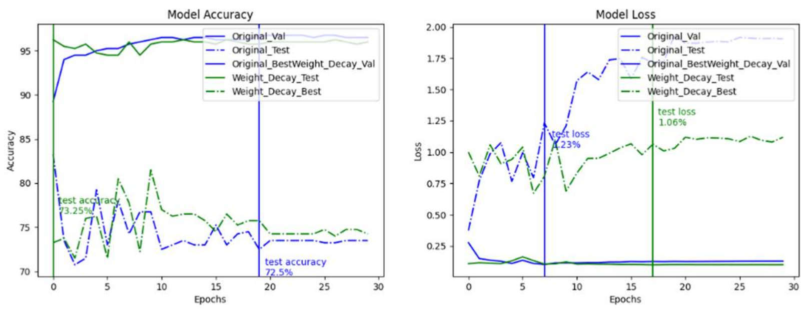


Fig. 3 displays the validation (solid) vs. testing performance (dash-dot), including the accuracy curve (a) and the loss curve (b) for both the original ConvModel (blue) and the weight-decay ConvModel (green) with $\lambda = 0.01$. The vertical lines indicate the epochs at which the best validation performance occurs for each model. Additionally, the testing performances are shown corresponding to the epochs at which their best validation performances occur.

Upon examining Figure 3, we observe that the weight-decay model exhibits significantly lower testing loss compared to the original model, while its testing accuracy is slightly higher than that of the original model. This evidence illustrates that weight decay can enhance the model's ability to generalize to unseen data.

Weight decay, often referred to as L2 regularization, is a powerful tool in addressing overfitting in machine learning models. Its influence on different aspects of the model and its ability to generalize to unseen data can be summarized as follows:

(1) **Training Loss Increase**: Weight decay introduces a penalty term in the loss function that encourages the model to maintain small weights. Consequently, the model produces smaller weight values during training. This may lead to a rise in the training loss compared to a model without weight decay. The regularization term discourages the model from fitting the training data too closely.

(2) **Smoothing Decision Boundaries**: Weight decay encourages the model to establish smoother decision boundaries. In classification tasks, this implies that the model is less likely to create intricate, closely fitted decision regions tailored to the training data. Instead, it aims for smoother, less sensitive decision boundaries, which ultimately leads to better generalization.

(3) **Balancing Bias and Variance**: Weight decay is a form of bias introduced into the model's learning process. While it might slightly increase bias by preventing the model from fitting the training data too closely, it reduces variance by preventing overfitting. This balance can lead to better overall model performance on unseen data.

Implementation Visualization (10 pts)

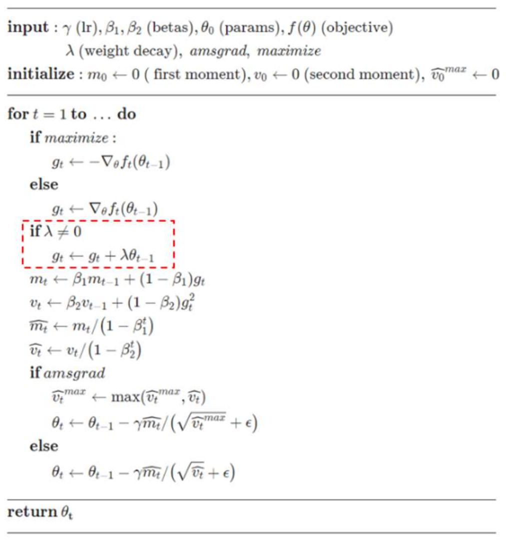
The weight-decay model shares the same architecture as the original ConvModel. Therefore, we do not distinguish their differences in the visualized architectures. Instead, we present the algorithm itself. All settings remain the same, except for the addition of the L2 penalty term to the loss function. This is represented as:

$$Loss_{weight-deca}\ (w,b) = Loss_{original}(w,b) + \frac{\lambda}{2}\|w\|_2$$

To implement weight decay, we need to determine the value of λ and assign it as the weight_decay argument when initializing an instance of the torch.optim.Adam class, as shown in the following line:

```
optimizer = torch.optim.Adam(model.parameters(), lr=1e-3, weight_decay=λ)
```

The specific optimization steps for weight decay are highlighted in the dashed red rectangle in the following torch.optim.Adam algorithm:



```
input : $\gamma$ (lr), $\beta_1, \beta_2$ (betas), $\theta_0$ (params), $f(\theta)$ (objective)
        $\lambda$ (weight decay), $amsgrad$, $maximize$
initialize : $m_0 \leftarrow 0$ ( first moment), $v_0 \leftarrow 0$ (second moment), $\widehat{v_0}^{max} \leftarrow 0$

for $t = 1$ to ... do
    if $maximize$ :
        $g_t \leftarrow -\nabla_\theta f_t(\theta_{t-1})$
    else
        $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$
    if $\lambda \neq 0$
        $g_t \leftarrow g_t + \lambda\theta_{t-1}$
    $m_t \leftarrow \beta_1 m_{t-1} + (1-\beta_1)g_t$
    $v_t \leftarrow \beta_2 v_{t-1} + (1-\beta_2)g_t^2$
    $\widehat{m_t} \leftarrow m_t/(1-\beta_1^t)$
    $\widehat{v_t} \leftarrow v_t/(1-\beta_2^t)$
    if $amsgrad$
        $\widehat{v_t}^{max} \leftarrow \max(\widehat{v_t}^{max}, \widehat{v_t})$
        $\theta_t \leftarrow \theta_{t-1} - \gamma\widehat{m_t}/(\sqrt{\widehat{v_t}^{max}} + \epsilon)$
    else
        $\theta_t \leftarrow \theta_{t-1} - \gamma\widehat{m_t}/(\sqrt{\widehat{v_t}} + \epsilon)$

return $\theta_t$
```

2. Task B: Performance Comparison between CNN and ANN (40 pts)

Comparing Convolutional Neural Networks (CNNs) and Artificial Neural Networks (ANNs) involves evaluating their feature extraction capabilities, training speed, and model performance.

Feature Extraction Capabilities (20 pts)

CNN: CNNs excel in feature extraction from structured data like images, time series, and audio. They leverage convolutional layers that apply filters to local regions, allowing them to capture spatial hierarchies and patterns. This ability to learn hierarchical features is crucial for image-related tasks like image classification and object detection.

ANN: ANNs, specifically fully connected networks, lack the specialized layers found in CNNs for feature extraction from structured data. They perform well with structured data but aren't optimized for tasks involving grid-like data, making them less effective for image-related tasks.

Training Speed - CNNs generally exhibit faster training speeds compared to ANNs when dealing with large amounts of structured data. This efficiency can be attributed to the convolutional layers, which reduce the number of parameters and connections in the network, thereby expediting the optimization process. However, in cases where the dataset is relatively small, CNNs may not be as fast as conventional computer systems. The reason behind this is that they need more computational power to get worked up.

We conducted the training process with 30 epochs for both CNN (ConvModel) and ANN (LinearModel), which were defined in lab3, and recorded the time each model took. CNN required 70.37 seconds to complete training, while ANN only took 35.24 seconds. This is despite the fact that the number of parameters in CNN is approximately half that of ANN.

Model Performance - In Figure 4, the validation accuracy of the CNN is significantly higher than that of the ANN. When considering early stopping at the best validation accuracy, the CNN achieves a test accuracy of 72.5%, while the ANN achieves 70.75%.

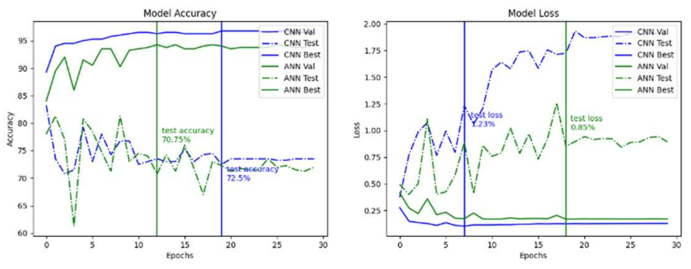


Fig. 4 displays the validation (solid) vs. testing performance (dash-dot), including the accuracy curve (a) and the loss curve (b) for both the original ConvModel (blue) and the LinearModel (green).

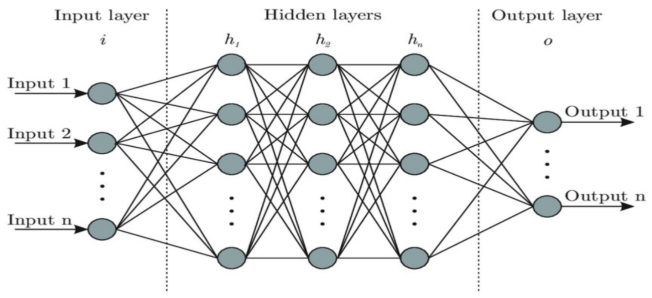
Architecture Description (20 pts)

ANN Architecture: ANNs consist of multiple layers that work together to process and transform data. The main layers in an ANN include the input layer, hidden layers, and the output layer. Here's an explanation of each of these layers:

Input Layer: The input layer receives the feature vectors and simply consists of nodes equal to the number of input features. These nodes pass the input directly to the hidden layer.

Hidden Layers: These layers comprise nodes connected to every input layer node, each with its weight. Nodes in the hidden layer apply an activation function to their weighted inputs, introducing non-linearity.

Output Layer: This layer contains nodes representing the model's predictions. In classification, it may employ a softmax activation function to output class probabilities.



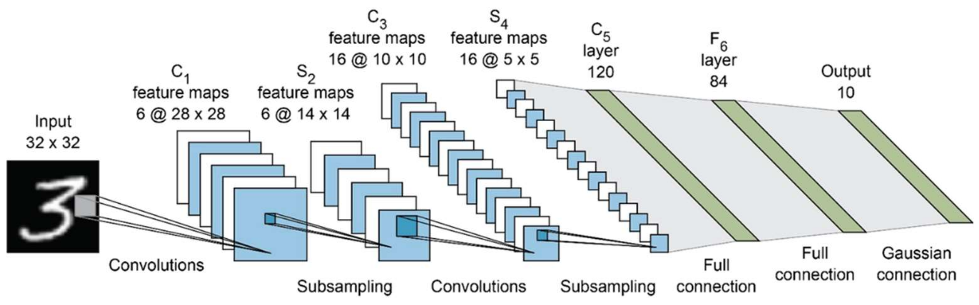
CNN Architecture: The CNN architecture is specifically designed for image data and involves convolutional and pooling layers, along with fully connected layers.

Convolutional Layers: The CNN starts with one or more convolutional layers. These layers consist of convolutional filters that slide over the input image, extracting features such as edges, corners, and textures. Each convolutional layer applies multiple filters, and the output from these filters is passed to the next layer.

Pooling Layers: After each convolutional layer, there is typically a pooling layer, often max-pooling. Pooling reduces the spatial dimensions of the feature maps while retaining the most important information. It helps make the network translation-invariant.

Fully Connected Layers: Following the convolutional and pooling layers, there are one or more fully connected layers. These layers connect all neurons to those in the previous layer, like a traditional neural network.

Output Layer: Similar to the ANN, the output layer of the CNN produces the final prediction.



3. Task C: Global Average Pooling in CNNs (30 pts)

Explanation (10 pts): Global Average Pooling (GAP) is a technique used in CNNs to reduce spatial dimensions before the fully connected layer. It replaces the need for manually determining feature numbers, as it transforms the entire feature map into a single value for each feature, often by calculating the average value. GAP simplifies the architecture and helps in regularizing the model. Instead of specifying the number of features, GAP creates a fixed-size output, making the architecture adaptable to different input sizes. It captures the essence of features across the spatial dimensions, reducing overfitting and improving generalization.

Increase Performance (20 pts): Global Average Pooling (GAP) significantly reduces the feature size of the last convolutional layer, consequently abstracting less information that is fed forward to the fully connected layers. This has prompted me to consider increasing the channel size instead. The model ConvGAP_1024 is identical to the ConvGAP model, except for the modification of the last convolutional layer's channel size, which has been increased from 32 to 1024. In Figure 5, it is evident that both the validation and testing performances exhibit uniform improvements in terms of accuracy and loss.

Increasing the number of channels in the last convolutional layer of a Convolutional Neural Network (CNN) can be a powerful strategy for enhancing performance, particularly in the context of complex tasks or diverse datasets. In this context, each channel within a convolutional layer serves as a specialized feature detector. By augmenting the number of channels, we effectively equip the network with a more comprehensive set of feature detectors. This expansion allows the network to capture a broader spectrum of patterns, textures, and shapes within the data. The experiment depicted in Figure 5 provides compelling evidence of the efficacy of this strategy.

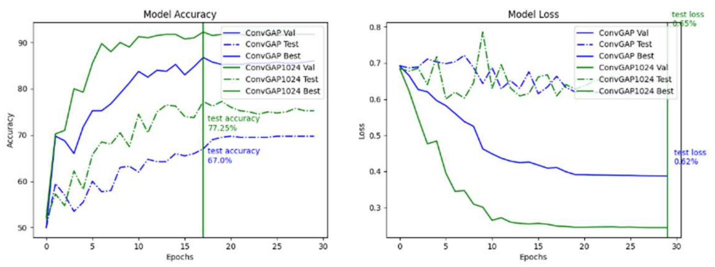


Fig. 5 displays the validation (solid) vs. testing performance (dash-dot), including the accuracy curve (a) and the loss curve (b) for both the original ConvGAP (blue) and the ConvGAP_1024 (green).