# Deep Learning Assignment 11 in FS 2022

## Adversarial Training

Microsoft Forms Document:
https://forms.office.com/r/e3n9gYXsL5

Manuel Günther

Distributed: Friday, May 20, 2022
Discussed: Friday, May 27, 2022

In this assignment we will implement a simple strategy to make deep networks more robust toward adversarial samples – which is to train with them. We will use the MNIST dataset and a small convolutional network, and we generate adversarial samples with the Fast Gradient Sign (FGS) method.

## 1 Dataset and Model

For our experiments we will come back to the dataset and the model that we have used before. Particularly, we will train and test our methods on the MNIST dataset, using the same deep network structure as in previous exercises.

**Task 1: Dataset** Instantiate the training and validation set data loaders for the MNIST dataset. Select appropriate batch sizes and parameters.

**Task 2: Classification Network** Implement a network with two convolutional and two fully-connected layers. Each convolution with kernel size $5 \times 5$ is followed by a $2 \times 2$ maximum pooling and a ReLU activation. Select appropriate numbers of input and output channels for the convolutions. The first fully-connected layer reduces the feature map to a certain size, which can be selected freely, while the second layer produces $O = 10$ logits.

## 2 Image Manipulations

We implement two different ways to manipulate the images for generating additional training data. The first way is to compute adversarial samples using the Fast Gradient Sign method. Here, we want to adapt the image such that we maximize the loss between the network output and the target. For this purpose, we need to compute the FGS adversarial samples by using the derivative of the loss with respect to the image:

$$\mathbf{X}_{\text{FGS}} = \mathbf{X} + \alpha \operatorname{sign}(\nabla_{\mathbf{X}}) \qquad \text{with} \qquad \nabla_{\mathbf{X}} = \frac{\partial \mathcal{J}^{\text{CCE}}}{\partial \mathbf{X}} \tag{1}$$

The second type of manipulation is to add simple noise that represents the same type and magnitude of manipulations as FGS, i.e., we want our noisy image to be:

$$\mathbf{X}_{\text{noise}} = \mathbf{X} + \alpha \{-1, 1\}^{D \times E} \tag{2}$$

where $D$ and $E$ are the width and the height of the original image.

In order to represent actual images, we will restrict the pixel values of both $\mathbf{X}_{\text{FGS}}$ and $\mathbf{X}_{\text{noise}}$ to be in range $[0, 1]$ by clipping any value that is outside that range.

**Task 3: Fast Gradient Sign** Implement a function that computes the gradient of the given loss function w.r.t. the input. Compute the adversarial sample using (1), and restrict the output values to be in range $[0, 1]$.

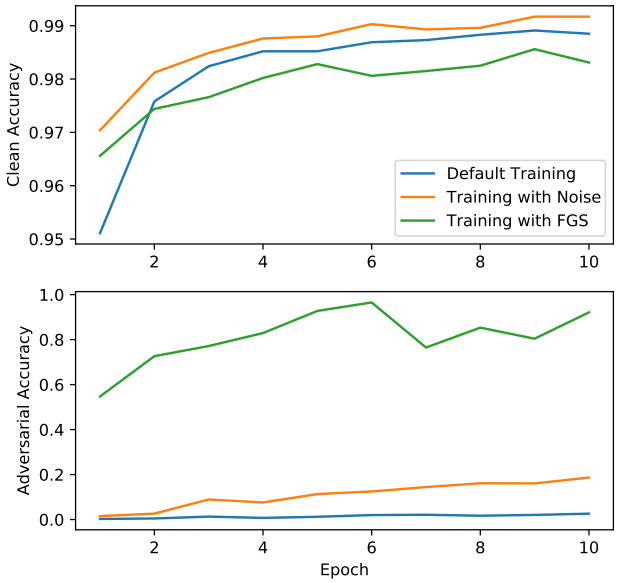


Figure 1: Exemplary plots for clean and adversarial accuracy of the three types of training.

**Task 4: Random Noise** Implement a function that computes the noisy sample according to (2), and restrict the output values to be in range $[0, 1]$.

# 3 Training and Evaluation

We will implement three different variations of the training procedure, which will be combined into one function. In all three variations, we will iterate over all batches in the training set and train the network with standard categorical cross entropy loss. Variation 1 will only perform the standard training. Variation 2 will generate FGS adversarial samples for all images in the batch, and perform an additional weight update step after training with the adversarial samples, while assigning the original targets. In variation 3, we will also train with additional data, but instead of generating adversarial samples, we will just add noise to our images.

For evaluation, we will compute two different measures. First, we compute the classification accuracy of the network on the clean, unperturbed images of the validation set. Additionally, we will create adversarial samples via our FGS technique for all correctly classified validation set samples and assess how many of these adversarial samples are classified as their original class by the network.

**Task 5: Training Loop** For a given network, loss and optimizer, implement a function that trains the network for one epoch on the training data. If desired by the `add_additional_samples` parameter, implement variation 1, 2 or 3. Perform update steps where they fit.

**Task 6: Validation Loop** For a given network and loss function, iterate over the validation set and compute the classification accuracy on the original validation set samples. Also compute how many of the adversarial samples are still classified as the original class by the network.

**Task 7: Training of Three Networks** Instantiate three networks, one to train with only clean samples, one to additionally train with adversarial samples and one to additionally train with noise. We will make use of SGD with an appropriate learning rate $\eta$ and categorical cross-entropy loss. Train each of the three networks for 10 epochs (or more) using their specific data extension. Compute and store the validation set accuracy on the clean and adversarial samples – note that adversarial samples will be generated for each network separately – after each training epoch.

**Task 8: Plotting Accuracies** Generate two plots. In the first plot, compare the accuracies on the clean images of the validation set over the 10 epochs. In the second plot, compare the stability of the networks w.r.t. adversarial samples, i.e., how many adversarial samples can change the classification of the network. Exemplary plots can be found in Figure 1.