Deep learning in audio, homework 5, anti-spoofing

Alexey Slizkov
17.12.2023

1 The mode

RawNet2 with SincConv of type S1.

2 A lot of time spent on...

Debugging a very stupid bug. Loss was good, but EER score was awful. I was trying to find the bug in my implementation of the model for a long time, but to no avail. I've gone over many permutations of batchnorms and skip-connections and so on, minor details which didn't play a role, which agrees with my intuition, as well as details of SincConv, such as whether we mirror or not, what are the gaps (0 or 30/50), etc. It turned out, the bug was not in the model, but in the computation of EER: the provided function `calculate_eer` computes EER from two arguments, I thought the predictions of bonafide on real bonafide, and the predictions of spoof on real spoof. It turned out it was expecting the predictions of bonafide on real bonafide, and the prediction of *bonafide* on real spoof.

Although this project contains approximately one hundred runs, most of them were debugging; the following ones are important.

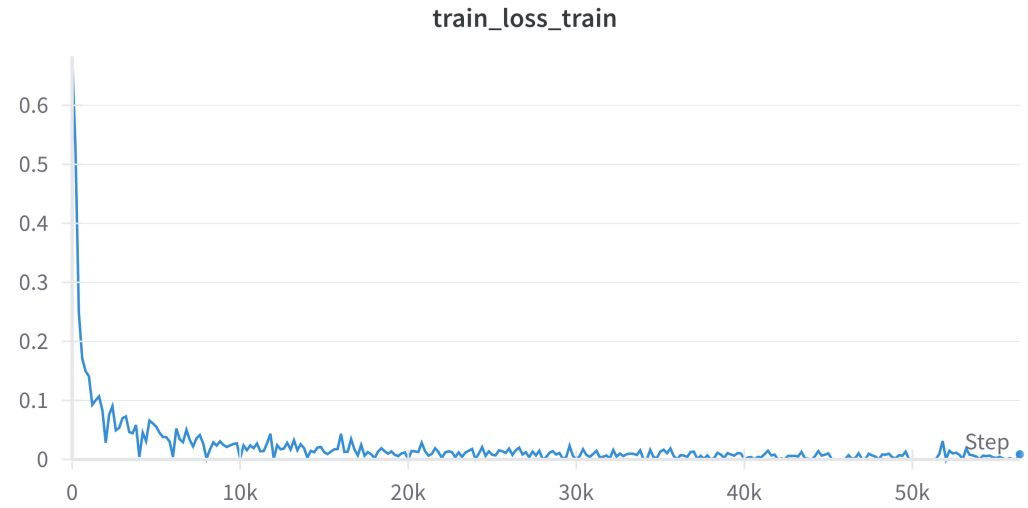
3 Main run

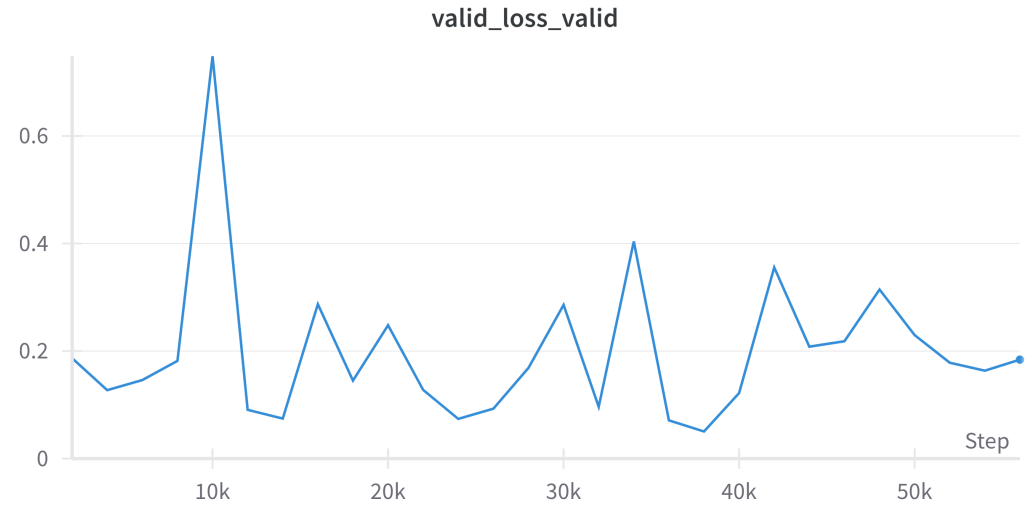
Link: https://wandb.ai/elexunix/as_project/runs/8rmrzw80/overview?workspace=user-elexunix.

Uses Peh's hyperparameters and other recommendations.

The model has 16578658 parameters. Its checkpoint can be found at `default-test-model/main_model_best.pth` To train, run `python3 train.py`, to test, run `python3 test.py`.

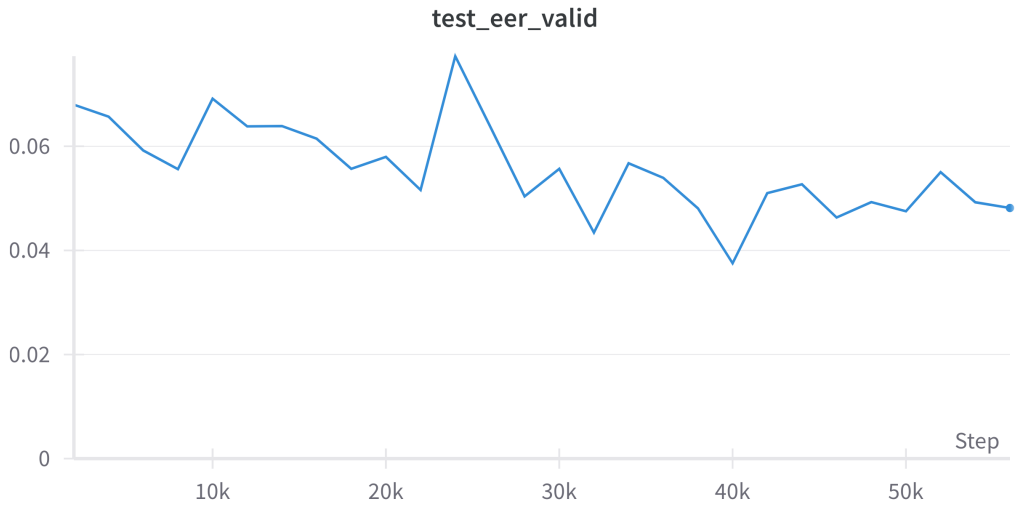
Test EER score of the model: 0.04446.

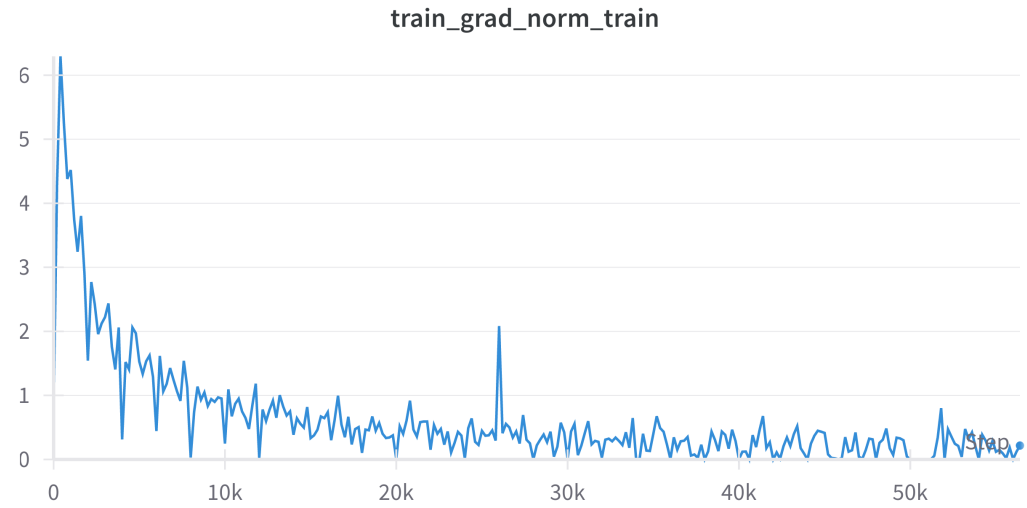




Sorry for the last word of title, it should be ignored, but hard to correct now since this is a picture from wandb.


test_loss_valid
0.4
0.3
0.2
0.1
0
Step
10k
20k
30k
40k
50k


test_eer_valid
0.06
0.04
0.02
0
Step
10k
20k
30k
40k
50k



4 Ablations

4.1 One GRU layer instead of three

Link: https://wandb.ai/elexunix/as_project/runs/8yv1t0zm?workspace=user-elexunix.

Test EER score of the model: 0.07859.

It follows that the impact is noticeable, but not huge, and maybe long training can lead to the desired score, but I don't know. It seems a bit worse, though.

The paper doesn't mention the number of these GRU layers (I didn't notice).

4.2 Remove weights from cross entropy

More precisely, set them to 1. 1. instead of 1. 9.

Link: https://wandb.ai/elexunix/as_project/runs/mcy64xah?workspace=user-elexunix.

Test EER score of the model: 0.05096.

In fact, I expected this would matter more...

4.3 Remove abs from the sinc layer

Simply removing abs from from the formulas will not be anymore explainable that way, so it will probably not work good. Let's see.

Link: https://wandb.ai/elexunix/as_project/runs/h7havmc4?workspace=user-elexunix.

Test EER score of the model: 0.05740.

I was not right, it seems to work.

The paper doesn't comment on this matter.

4.4 Zero weight decay

This will of course lead to overfitting, even with our wd of 10^{-4} we have a strong one. The number of parameters is vastly more than the number of answers in the train part (the ratio is approximately 1000), training without huge regularization is strongly recommended against for your and your model's health.

Link: https://wandb.ai/elexunix/as_project/runs/l98p8btf?workspace=user-elexunix.

Test EER score of the model: 0.06581.

Actually, I thought it would be worse.

4.5 Conv HPs as in the old paper

Sinc filter length is already 1024.

Number of channels in this experiment is reduced from 128 to 20, from 512 to 128.

This is the recommendation from the old paper (but not the new one).

Link: https://wandb.ai/elexunix/as_project/runs/n3xx0owb?workspace=user-elexunix

Test EER score of the model: 0.10102.

This was obviously not trained for too long, it would be very long.

4.6 Non-zero min_low_hz, min_band_hz

In this setup, we try setting min_low_hz, min_band_hz both to 50. This will probably lead to poor processing of frequencies below approximately 50Hz, which is not that low and probably human voice has some reverbations in these ranges as well...

Link: https://wandb.ai/elexunix/as_project/runs/9hev2bm3?workspace=user-elexunix.

Test EER score of the model: 0.07884.

The paper doesn't discuss these values.

The results seem to agree with the prediction in this case.

5 Another sinc layer type

The sinc layer type used in the main run was S1 (equal spacing in Mels). I have also implemented S2 (equal spacing in Hertz)

Link for the test run: https://wandb.ai/elexunix/as_project/runs/gch8vhg3?workspace=user-elexunix.

Test EER score of the model (this was not trained for too long, as this is just a demonstration that it learns): 0.05766.

6 Full description of new and modified code

6.1 hw_code/collate_fn/collate.py

New function `as_collate_fn` has been defined. This function is used to collate fields in dataset items that have the same length. It takes in a list of dictionaries

representing dataset items. It uses torch default collate.

6.2 hw_code/datasets/__init__.py

Minor import changes.

6.3 hw_code/losses/__init__.py

Minor import-related changes.

6.4 hw_code/model/__init__.py

Import-related changes: new folders for each hw are created, and our folder is hw_code/model/hw5as.

6.5 hw_code/utils/object_loading.py

Essentially, `pin_memory=True` was added (may require more RAM, but may help with speed).

6.6 hw_code/utils/trainer/__init__.py

Minor import-related changes.

6.7 hw_code/config/config.json

Oh...

6.7.1 Changes in top-level keys

- `sr`: The value of `sr` has been changed from 22050 to 16000.

6.7.2 Changes in model configuration

- `type`: The value of `type` has been changed from `"HiFiGAN"` to `"RawNet2Model"`.

6.7.3 Changes in model arguments

- `n_mels`: The key `n_mels` has been removed.
- `lrelu_slope`: The key `lrelu_slope` has been removed.

6.7.4 Changes in training configuration

- `batch_size`: The value of `batch_size` has been changed from 16 to 32.

6.7.5 Changes in training dataset configuration

- `type`: The value of `type` has been changed from `"LJSpeechDataset"` to `"ASVSpoofDataset"`.
- `segment_size`: The value of `segment_size` has been changed from 8192 to 64000.

6.7.6 Changes in validation configuration

- `batch_size`: The value of `batch_size` has been changed from 16 to 256. Yes! It just works faster this way.

6.7.7 Changes in validation dataset configuration

- `type`: The value of `type` has been changed from `"LJSpeechDataset"` to `"ASVSpoofDataset"`.

6.7.8 Changes in test configuration

- `batch_size`: The value of `batch_size` has been changed to 256.
- `num_workers`: The value of `num_workers` has been changed from 0 to 0.

6.7.9 Changes in test dataset configuration

- `type`: The value of `type` has been changed from `"LJSpeechDataset"` to `"ASVSpoofDataset"`.

6.7.10 Changes in optimizer configuration

- `lr`: The value of `lr` has been changed from 3e-4 to 1e-4.
- `weight_decay`: The value of `weight_decay` has been changed to 1e-4.

6.7.11 Changes in learning rate scheduler configuration

- `type`: The value of `type` has been changed from `"OneCycleLR"` to `"ExponentialLR"`.
- `gamma`: The value of `gamma` has been set to 1.

6.7.12 Changes in trainer configuration

- `log_interval`: The value of `log_interval` has been changed from 50 to 200.
- `monitor`: The value of `monitor` has been changed from `"min val_total_loss"` to `"min valid_loss"`.

6.7.13 Changes in WandB configuration

- `wandb_project`: The value of `wandb_project` has been changed from `"nv_project"` to `"as_project"`.
- `len_epoch`: The value of `len_epoch` has been changed from 400 to 2000.

6.8 ASTrainer class description

It is actually easier to describe what is in the new trainer than how it differs from the previous one... There are really a lot of differences. So here the description of ASTrainer class goes:

The `ASTrainer` class is a subclass of the `BaseTrainer` class and is responsible for training the Anti-Spoofing model. Here is a detailed description of the class and its methods:

6.8.1 Constructor (__init__)

The constructor initializes the `ASTrainer` class with the following parameters:

- `model`: The Anti-Spoofing model.
- `metrics`: The list of metrics to evaluate during training.
- `optimizer`: The optimizer for model parameter updates.
- `config`: The configuration dictionary.
- `device`: The device (CPU or GPU) to use for training.
- `dataloaders`: A dictionary of dataloaders for the training, validation, and test datasets.
- `lr_scheduler`: The learning rate scheduler.
- `len_epoch`: Length of an epoch (number of iterations). If not provided, the length of the training dataloader is used.
- `skip_oom`: A boolean flag indicating whether to skip batches that result in out-of-memory errors.

6.8.2 Helper Methods

- `move_batch_to_device()`: Moves the batch data to the given device (CPU or GPU).
- `_clip_grad_norm()`: Clips the gradient norm of the model parameters if `grad_norm_clip` is defined in the trainer configuration.
- `_progress()`: Returns a formatted string showing the progress of the current epoch during training.
- `get_grad_norm()`: Calculates and returns the gradient norm of the model parameters.

6.8.3 Training Methods

- `process_batch()`: Processes a batch of data by moving it to the device, performing forward pass, calculating the loss, updating the metrics, and backpropagating the gradients if in training mode.

- `_train_epoch()`: The main training logic for an epoch. It iterates over the training dataloader, processes each batch, and updates the metrics. It also logs the train metrics and returns them as a log.
- `_validation_epoch()`: Performs the validation after training an epoch. It evaluates the model on the validation dataset, updates the validation metrics, and logs them.
- `_test_epoch()`: Performs the testing after training an epoch. It evaluates the model on the test dataset, updates the test metrics, and logs them. It also computes the Equal Error Rate (EER) using the `compute_eer()` function.

6.8.4 Training Process

The training process follows these steps:

1. The model is set to train mode, and the train metrics are reset.
2. For each batch in the training dataloader:
   (a) The batch is processed using the `process_batch()` method.
   (b) The train metrics are updated.
   (c) If the batch index is divisible by the log step, the train metrics are logged.
   (d) If the batch index is greater than or equal to the length of the epoch, the training loop is exited.
3. The validation and test epochs are run using the `_validation_epoch()` and `_test_epoch()` methods, respectively. The metrics from each of these epochs are added to the log.
4. The log is returned, containing average loss and metrics for the epoch.

This class provides a complete implementation of the training and evaluation process for the Anti-Spoofing model.

6.9 ASVspoof Loss Function Description

The ASVspoof loss function is defined in the file `hw_code/losses/asvspoof_losses.py`. Here is a description of the loss function:

- `CE_weights`: It is a tensor that contains the weights for the classes in the cross-entropy loss. In this case, the tensor contains the weights `[1., 9.]`, where the first class has weight 1 and the second class has weight 9. This tensor is stored on the CUDA device.
- `loss(predicted, target)`: This function calculates the cross-entropy loss between the predicted and target tensors. The `predicted` tensor represents the predicted class probabilities, and the `target` tensor represents the ground truth class labels. The loss is computed using the `F.cross_entropy()` function from the `torch.nn.functional` module, with the weights `CE_weights`. The calculated loss is returned.

The ASVspoof loss function computes the cross-entropy loss with weighted classes for a classification task, where the first class has a weight of 1 and the second class has a weight of 9. This weighted loss can help in handling class imbalance and assigning more importance to certain classes during training.

6.10 ASVSpoofDataset Class Description

The `ASVSpoofDataset` class represents the dataset for the ASVspoof challenge. Here is a detailed description of the class and its methods:

6.10.1 Constructor (__init__)

The constructor initializes the `ASVSpoofDataset` class with the following parameters:

- `segment_size`: The size of each audio segment to extract.
- `data_dir`: The path to the directory containing the dataset. If not provided, the default path is used.
- `part`: Indicates the part of the dataset to load, i.e., "train", "valid", or "test".
- `max_size`: The maximum number of samples to load from the dataset.
- `shuffle`: A boolean value indicating whether to shuffle the dataset.
- `*args, **kwargs`: Additional arguments that can be passed to the constructor.

6.10.2 Methods

- `__getitem__(self, index)`: This method returns a sample from the dataset at the specified index. It loads the audio file using the `torchaudio.load()` function, retrieves the audio tensor, and trims it to the desired segment size. The trimmed audio tensor and the corresponding label are returned as a dictionary.
- `__len__(self)`: This method returns the length of the dataset, which is equal to the number of audio files available.

The `ASVSpoofDataset` class represents the ASVspoof dataset for the given part ("train", "valid", or "test"). It reads the audio files from the specified directory and provides access to the audio data and corresponding labels. During initialization, the dataset is checked to ensure that it contains the expected number of samples for the specified part.

The dataset is designed to work with audio files in FLAC format. The `torchaudio.load()` function is used to load the audio files, and the resulting audio tensors are processed to extract the desired segment size.

If a maximum size is specified, the dataset will be truncated to the specified number of samples. The dataset can also be shuffled during initialization by setting the `shuffle` parameter to `True`.

The dataset is designed to be used with PyTorch's `torch.utils.data` module, allowing for easy integration with data loaders and training pipelines. By accessing individual samples using the `__getitem__` method, training, validation, and testing procedures can be performed using this dataset.

6.11 hw_code/model/hw5as/__init__.py: empty file

6.12 SincConvEpic Class Description

The `SincConvEpic` class represents the SincConvEpic module, which performs Sinc convolution on audio waveforms. Here is a more detailed description of the class and its methods, including the mathematical details:

6.12.1 Constructor (__init__)

The constructor initializes the `SincConvEpic` class with the following parameters:

- `in_channels`: The number of input channels in the audio waveforms. This should be 1 since audio typically has a single channel.
- `out_channels`: The number of output channels (filters) in the Sinc convolution layer.
- `kernel_size`: The size of the SincConv kernel.
- `sample_rate`: The sample rate of the audio waveforms.

The constructor initializes the instance variables `out_channels`, `kernel_size`, `sample_rate`, and `cnt` (which is half of the kernel size).

The frequency range of the Sinc filters is determined by the sample rate. The frequencies are equally spaced in the Mel scale between 0 and half of the sample rate. The frequencies are initialized as trainable parameters using `nn.Parameter`. The trainable frequencies are represented by two sets of parameters: `trainable_f1s` and `trainable_f2s`.

6.12.2 Methods

- `forward(self, waveforms)`: This method performs the forward pass of the SincConvEpic module. It takes the input waveforms and applies the SincConv operation. The trainable Sinc filter frequencies (`trainable_f1s` and `trainable_f2s`) are used to compute the frequencies for each output channel. The sinc functions are computed by applying the hamming window to the sampled time. The band-pass filters are obtained by subtracting the sinc function of frequency 1 from the sinc function of frequency 2. Finally, the input waveforms are convolved with the band-pass filters using the `F.conv1d()` function. The resulting output is returned.

Additionally, the class defines the following static methods:

- `hz_to_mel(hz)`: This method converts frequencies from Hertz (Hz) to the Mel scale. It uses the formula: $\text{Mel} = 2595 \log_{10}(1 + \frac{\text{hz}}{700})$.

- `mel_to_hz(mel)`: This method converts frequencies from the Mel scale to Hertz (Hz). It uses the formula: $\text{Hz} = 700 \left(10^{\frac{\text{mel}}{2595}} - 1\right)$.

The `SincConvEpic` class provides the functionality of applying Sinc convolution on audio waveforms. It allows for customization of the number of output channels, kernel size, and sample rate, making it suitable for various audio processing tasks.

The SincConv operation involves creating band-pass filters using sinc functions. The frequency range of the filters is determined by the trainable parameters. By using trainable frequencies, the SincConvEpic module can adapt to different input and output requirements.