Deep Learning

ffgt86

February 26, 2020

This submission is an implementation of Adversially Constrained Autoencoder Interpolation (ACAI) [1]. It is based on three sources: the original paper [1], the authors' TensorFlow implementation [1], and a PyTorch implementation [2].

1 Design

1.1 Autoencoder

Autoencoders are typically 'shallow', consisting of only a single-layer encoder and single-layer decoder, but 'deep' autoencoders have been shown to have numerous advantages [2] and experimentally yield better compression [4]. The autoencoder used here is very deep: 18 layers with the final configuration of hyperparameters.

The encoder f_θ (appendix 3.1) consists of 'blocks' of two consecutive 3×3 convolutional layers (`torch.nn.Conv2d`) followed by 2×2 average pooling (`torch.nn.AvgPool2d`). The number of blocks is computed in such a way to determine the dimension of the latent space. The authors found best results on real-life datasets with a latent space of dimension of 256 $(16, 4, 4)$, which is also used here. After each block is a `LeakyReLU` activation function. Using `LeakyReLU` activation functions with a negative slope of 0.2 appears to be current best practice [**bertholot˙et˙al˙2019**] [3]. `tanh` was also tested, but found to offer no improvement.

The decoder g_ϕ (appendix 3.2) also consists of 'blocks' of 2×2 convolutional layers, also with `LeakyReLU`, and followed by 2×2 nearest neighbour upsampling. The number of channels is halved after each upsampling layer. After the blocks, two more 3×3 convolutions are used, with 3 channels, followed by a final `sigmoid` activation, to convert results to $[0, 1]$ range suitable for display.

Parameters are optimised with Adam [5]. The values for the learning rate `l_r = 1e-3` and weight decay `weight_decay = 1e-5` given in the paper were found to inhibit performance compared to the defaults of `l_r = 1e-5` and `weight_decay = 0`, which are

[1]https://github.com/anonymous-iclr-2019/acai-iclr-2019
[2]https://gist.github.com/kylemcdonald/e8ca989584b3b0e6526c0a737ed412f0

used instead.

Binary-cross entropy loss (`torch.nn.BCELoss`) was found to offer superior results to the standard mean-squared error used in the paper, so was used instead in conjunction with the original regularisation term.

1.2 Discriminator

The discriminator d_ω uses the same network structure as f_θ. The output of $d_\omega(x)$ is a scalar value, computed as the mean of the encoded input $z = f_\theta(x)$. The discriminator uses spectral normalisation [6] to limit variance in discriminator loss, with an over-training ratio controlled by the parameter `disc_train`. Large values (> 5) were found to negatively impact the performance of the autoencoder; 2 was used. However, loss still fluctuates heavily.

1.3 Datasets, samplers, and augmentation

A custom subclass of `torchvision.datasets.CIFAR10` was used. This comprised $10,000$ images. The first 5000 images are horses; the second 5000 are either birds or planes. A custom batch sampler was used, such that, in each batch, the first half comprised horses, and the second half birds or planes.

Standard data augmentations were tested. Random crops and noise were found to decrease the performance of the model. Normalisation is unnecessary when using the built-in transform `torch.transforms.ToTensor`, which converts images to type `torch.FloatTensor`, in the range $[0, 1]$, suitable both for BCE loss functions and sigmoid activation functions. The only augmentation therefore used is a random horizontal flip.

1.4 Training

The model was trained for 150 epochs, with batches of size 64. This was found to offer a good balance between convergence and Colab not disconnecting the runtime. Discriminator loss, though fluctuating considerably between epochs, remained fairly constant. During development, saved images are outputted to Google Drive, where they can be easily examined. This functionality has been disabled for this submission. The model uses GPU hardware acceleration by default. The model was trained twice, once with birds, and once with planes.

2 Results

Below are a selection of the best images produced by my model. They are combinations of either horses and birds or horses and planes. Also included is the α used to combine the latent spaces of the two input images.

3 Appendices

3.1 Encoder

```
Sequential(
  (0): Conv2d(3, 16, kernel_size=(1, 1), stride=(1, 1), padding=(1, 1))
  (1): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (2): LeakyReLU(negative_slope=0.01)
  (3): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (4): LeakyReLU(negative_slope=0.01)
  (5): AvgPool2d(kernel_size=2, stride=2, padding=0)
  (6): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (7): LeakyReLU(negative_slope=0.01)
  (8): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (9): LeakyReLU(negative_slope=0.01)
  (10): AvgPool2d(kernel_size=2, stride=2, padding=0)
  (11): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (12): LeakyReLU(negative_slope=0.01)
  (13): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (14): LeakyReLU(negative_slope=0.01)
  (15): AvgPool2d(kernel_size=2, stride=2, padding=0)
  (16): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (17): LeakyReLU(negative_slope=0.01)
  (18): Conv2d(128, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
)
```

3.2 Decoder

```
Sequential(
  (0): Conv2d(16, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (1): LeakyReLU(negative_slope=0.01)
  (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (3): LeakyReLU(negative_slope=0.01)
  (4): Upsample(scale_factor=2.0, mode=nearest)
  (5): Conv2d(64, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (6): LeakyReLU(negative_slope=0.01)
  (7): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (8): LeakyReLU(negative_slope=0.01)
  (9): Upsample(scale_factor=2.0, mode=nearest)
  (10): Conv2d(32, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (11): LeakyReLU(negative_slope=0.01)
  (12): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (13): LeakyReLU(negative_slope=0.01)
  (14): Upsample(scale_factor=2.0, mode=nearest)
  (15): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (16): LeakyReLU(negative_slope=0.01)
  (17): Conv2d(16, 3, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (18): Sigmoid()
)
```

References

[1] David Berthelot et al. *Understanding and Improving Interpolation in Autoencoders via an Adversarial Regularizer.* 2018. arXiv: 1807.07543 [cs.LG].

[2] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning.* http://www.deeplearningbook.org. MIT Press, 2016.

[3] Ari Heljakka, Arno Solin, and Juho Kannala. “Pioneer Networks: Progressively Growing Generative Autoencoder”. In: *Asian Conference on Computer Vision (ACCV).* 2018.

[4] G. E. Hinton and R.R. Salakhutdinov. “Reducing the Dimensionality of Data with Neural Networks”. In: *Science* 313.5786 (2006), pp. 504–507. DOI: 10.1126/science.1127647.

[5] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization.* 2014. arXiv: 1412.6980 [cs.LG].

[6] Takeru Miyato et al. *Spectral Normalization for Generative Adversarial Networks.* 2018. arXiv: 1802.05957 [cs.LG].