# ***Assignment 5***

# Model saving, saliency map, Adversarial attack,

# GAN, Transfer learning, Meta Learning

Hui Xue

[hui.xue@nih.gov](mailto:hui.xue@nih.gov)

1. **Multi-choice and short-answer questions**
2. Which of the following statements are incorrect for the generative model and discriminative model?
3. Generative model cannot give us an estimation of decision boundary.
4. Discriminative model is suitable if datasets have missing items.
5. Generative model aims to model the data distribution p(x).
6. Discriminative model aims to model the decision boundary p(y|x).
7. You trained a GAN model to make MRI images of a beating heart. Suppose your model was trained successfully and can make realistic-looking heart images. Which of the following is wrong?
8. Generator learned to approximate the data distribution of MRI heart images well.
9. Generator is able to produce “new” heart images which are not acquired in real.
10. After training, discriminator is able to differentiate MRI heart images from MRI head images.
11. The GAN loss is reduced after the training and approach a constant level.
12. Which of the following are plausible losses for a GAN?
13. Adversarial examples are samples which can fool a neural network. Which of the following statements are correct about the adversarial examples?
    * 1. Adversarial examples always look like a normal image, but make the model make wrong predictions.
      2. Real images which were physically acquired include all samples with realistic looking.
      3. Adversarial examples are theoretic and cannot happen in real scenario.
      4. The fast gradient sign method uses the gradient of loss to the input x to compute perturbation leading to incorrect model prediction.
14. Which of the following model visualization methods require to run the backprop multiple times?
    * 1. Occlusion
      2. Saliency map
      3. Smoothing Grad
      4. Grad-CAM
15. You have a model to predict the outcome from COVID X-ray images. The size of X-ray dataset is 100,000. You get a new task to predict the COVID outcome from CT lung images. Due to the limitation of data source, there are only 10,000 CT images. How will you approach the problem? Select all of the following which may be applicable:
    * 1. Mix X-ray and CT images and train a big model.
      2. Train a big model on X-ray datasets and use it to do feature-extraction transfer learning on CT data.
      3. Train a big model with X-ray images and 8,000 CT images and tested on 2,000 CT images.
      4. Train a smaller model on CT images. X-ray images may not be helpful here.
16. As shown in the lecture, one problem of training GAN is the model collapse. It means the generator can produce good quality samples, but many samples may be very similar. Remember in the standard GAN optimization, we try to solve this minmax problem:

Explain why only solving this optimization problem cannot prevent model collapse?

1. In the lecture, we introduced the self-supervised learning (SSL) where the self-supervisory signal is generated from the unlabeled data. A key concept is the positive and negative samples. What is the advantage of this way to create self-supervisory signals, compared to the usage of a set of pretext tasks? The most prominent SSL publications focus more on the classification tasks. Discuss whether the SSL can be applied to other tasks, e.g. object detection or segmentation?
2. The MAML algorithm in meta-learning requires an inner update loop:

Here is the model parameter. MAML requires to compute the update on using a gradient step and evaluates its performance on the new task/data. It is solved as a joint optimization problem.

In the lecture, we showed this optimization problem will not lead to computation of hessian matrix (as it is impractical), but only requires vector-matrix product.

In fact, if we performed more than one inner update step, this conclusion stills holds:

Derive the gradient of this loss to and show there is no need to compute and store Hessian matrix, but only Hessian-vector product.

1. **Model saving and format.** In the lecture, we used Pytorch to train the model. What we did not discuss is how to deploy the model. Pytorch offered its native way to save and load models as the python pickle files: <https://pytorch.org/tutorials/beginner/saving_loading_models.html>

However, this method requires to deploy both model files and the saved model. In other words, if we do the training on computer A and want to call model on computer B. The model definition needs to be copied, together with model weights. The reason is that Pytorch uses the dynamic computational graph and the saved weights contain no information about how to reassemble the computational graph. Please read more on this topic: <https://pytorch.org/blog/model-serving-in-pyorch/>

So is there a better way to deploy Pytorch model?

First, you can use torch.jit.trace (<https://pytorch.org/tutorials/beginner/Intro_to_TorchScript_tutorial.html>), which will create an internal representation of the model graph and record all tensor operations. After traced the model, it can be saved using torch.jit.save. Models saved in this way can be loaded and called in other runtime environment, e.g. on mobile devices or in C++.

Second, you can use the [Open Neural Network Exchange (ONNX)](https://onnx.ai/) and the ONNX Runtime. Please read the tutorial about exporting Pytorch model to Onnx format: <https://pytorch.org/tutorials/advanced/super_resolution_with_onnxruntime.html>. This exporting process will also create the intermediate representation and record computational steps. You can use different backends to run the ONNX model. For high performance inference on nvidia GPUs, you can use TensorRT (https://developer.nvidia.com/blog/speeding-up-deep-learning-inference-using-tensorflow-onnx-and-tensorrt/).

For both formats, conceptually, the model is converted to “static”, so it will not be required to keep model file around, which is convenient for deployment management.

Your task is to implement the model saving from Pytorch to torchscript and ONNX formats and also create functions to load model and perform inference. Please read and understand a5\_model\_saving\_loading.py, where a pre-trained model has been provided to you with the example data for inference.

Please submit the script outputs for model inference and saved plots. Run the model as “python3 a5\_model\_saving\_loading.py --format onnx” for ONNX format and “python3 a5\_model\_saving\_loading.py --format torchscript” for the torch script format. When only cpus are used for inference, is ONNX faster?

1. **Model visualization.** In the lecture, we reviewed a number of model visualization methods. A popular approach is the saliency map. Please read a5\_model\_visualization.py and complete functions to compute saliency maps. A pre-trained model is provided to you, together with testing samples. You are required to implement the standard saliency map and the Smoothing-Grad version. Please submit the example saliency maps for testing examples.
2. **Fast Gradient Sign Method.** In the lecture, the FGS method is introduced as an algorithm to create the adversarial examples for a trained model. In this problem, you will implement the FGS method and create adversarial examples. We will use the Cifar10 datasets. A pre-trained model (the small resnet for Cifar10 in A3) is provided to you. Please read a5\_fast\_gradient\_sign.py and implement this algorithm. Please test your implementation on the testing samples and compute the accuracy with different level of attacking. Please submit a plot of vs. the accuracies when the perturbation is increased. Also, please submit some attacking examples (the plotting code is provided to you).
3. **GAN.** In this problem, you are required to implement and train generative adversarial network. The Fashion-MNIST dataset will used. The Fashion-MNIST is a dataset of clothes from the [Zalando](https://jobs.zalando.com/tech/). It has a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 grayscale image, associated with a label from 10 classes.

<https://github.com/zalandoresearch/fashion-mnist>

1. Please read a5\_gan.py and implement the non-saturated form of GAN loss for the minimization:

Here is the discriminator with parameter . The is the generator with parameters . is the loss for discriminator for minimization. is the non-saturated loss for generator. is the real sample. is the random vector sampled from uniform norm distribution. What is the minimal value of and ?

Train the model for 10 epochs. The code is set up to sample from generator in every epoch and save the results to result folder. Submit a few sample images for different number of training steps.

Note the wandb is used to log the sampled images during training, which is quite convenient to monitor GAN training progress.

In this problem, the torchvision.datasets (<https://pytorch.org/vision/stable/datasets.html>) is used to download images, instead of supplying data files. Torchvision has archives for a number of common datasets. Please check the website to learn more.

1. One problem with the vanilla GAN using Jensen-Shannon divergence loss is its instability in training. You can get a sense by changing the learning rate or optimization method from the default value and check the final samples. The fundamental problem is there are two networks to compute each other. In the vanilla GAN setup, the JS divergence is not guaranteed to be continuous to the generator parameters, which can cause difficulties in training.

One possible way to improve stability in GAN training is to change its loss function. One effective way is the so-called “Wasserstein GAN with gradient penalty” (WGAN-GP, <https://arxiv.org/pdf/1704.00028.pdf>). In the WGAN-GP set up, the output of discriminator is not a probability, but a measure how good the sample is. For a real sample, D(x) should output a large positive number. For a fake sample, D(x) should output a smaller number. The losses for minimization are :

Here , a linear combination of real and fake samples. is the derivative of discriminator to the tensor . is the L2 norm of this derivative.

Finish the WGAN-GP loss in gan.py and train the GAN with “python3 a5\_gan.py --loss\_type wgan”.

Submit a few sample images for different number of training steps.

1. The GAN model can be extended to conditional on class category . Given a class from 0 to 9, for ten clothing classes, the GAN loss is now conditioned on :

To implement the conditioning on generator , one method is to append the class one-hot vector(10x1 vector) to the .

To enforce the conditioning on the discriminator, the discriminator network outputs the logits for all 10 classes as a tensor of [B, 10] and . This is equivalent to pick the score corresponding to the correct class.

This definition of is to optimize the discriminator as a binary classification problem. Another possible way is to let classify the generated images from 10 classes. In practice, the binary classification setup works much better than multi-class setup. Can you think about the explanation?

Read the code in a5\_conditional\_gan.py and finish the conditional GAN loss in gan.py. Pay attention to the output of discriminator network.

Train the model for 10 epochs and submit the sampled images generated at different training steps. Every column in the saved samples should belong to the same cloth class.

***Congratulations! You completed the*** [***Deep Learning Crash Course***](http://www.deeplearningcrashcourse.org)***!***

***Thank you for the hardworking and I hope the experience is rewardable!***

***Please send feedbacks to*** [***hui.xue@nih.gov***](mailto:hui.xue@nih.gov)