Predicting adversarial network for creation of loss series:

Problem definition:

When training a DNN, we are required to give the DNN a loss value after every iteration.

Then the gradients of that loss value are computed on the DNN and an optimizer step can be done to get the DNN closer to a state, that will result in a smaller loss.  
The problem with that approach is, sometimes we are trying the train the DNN to perform a series of actions or predictions and we can compute the loss only after a series of many steps.

If we would try to compute the gradients of the DNN after such a gib number of activations, the gradients will be totally lost due to vanishing gradients.

(and computation time of that gradient will be extremely long)

Proposed method:

We would use 2 DNN

We will denote them NN A and NN B.  
the process we are trying to optimize is:

The loss of A will be defined as:

The loss of B will be defined as:

The notation means applying the loss function we have, at the end of the process.

The idea:

The NN B will create a series of losses, that will converge in the end to the loss we have at the end of the process.

The NN A will use B as a loss function.

If the series of losses is N iterations long.

We have created a series of numbers L.  
and exists a function f, so:

and also

and so, if we will optimize the NN A, to minimize , that would result to minimizing

implementation search of MRI reconstruction sampling trajectory:

we will use 3 DNN.

Reconstruction NN = U-net model

Sampling NN = U-net model followed by a soft-max layer.  
then the maximal value is subtracted from the entire matrix.  
let’s denote (max\_value – second\_max\_value)  
 is added to the sampling matrix .  
and so the maximal value in the Unet-result is positive, the second largest is 0 and the rest are negative.

Then the entire matrix is divided by .

and so the maximal value in the Unet-result is 1, the second largest is 0 and the rest are negative.  
then a Relu function is applied to leave only the largest value as 1, and the rest at 0.

Adversarial NN: a series of convolutions and then a series of linear layers, resulting with a scalar.  
  
training the model:

First, we will do an initial train of the remonstration model.  
we will train the reconstruction model with random sampling of the K-space with samples.  
N – the number of samples we are trying to sample from the K-space.

For every epoch in the entire train:

Adversarial epoch:

Train the Adversarial NN to predict its own result in the next iteration of the NN A.  
using the loss defines as:

And for every batch, add the last sample:

It is imperative for the series of losses to converge to the actual loss on the last sample, where a loss function is defined.

In the case of the MRI reconstruction problem, the loss will be defined as using the reconstruction NN on the subsampled K-space over the entire trajectory.

Train the sampler NN:

Use the loss of:

Train the Remonstration NN to fit better to the sampling NN:

Do multiple epochs of:

Find the full sampling trajectory for an image.  
use the trajectory to subsample the K-space.  
calculate the loss of the reconstruction NN over the subsampled K-space.  
train the reconstruction NN.