EE-556 Mathematics of Data: From Theory to Computation
Homework 2

WEI Jiheng (319872)

December 6, 2021

1 Minimax problems and GANs

1.1 Theoretical recap

1. $f(x, y) = (x-1)(y-1)$.

$$\nabla f(x,y) = \begin{bmatrix} y-1 \\ x-1 \end{bmatrix}, \quad \nabla^2 f(x,y) = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}. \tag{1}$$

   $\nabla f = \mathbf{0} \implies (x^\star, y^\star) = (1,1)$ is the only first-order stationary point of f. The eigenvalues of the Hessian are $\lambda_1 = 1 > 0$, $\lambda_2 = -1 < 0$, so it is a saddle point.

2. $f(x^\star, y^\star) = f(1,1) = 0$. For all x, y,

$$f(x^\star, y) = (1-1)(y-1) = 0 = f(x^\star, y^\star), \tag{2}$$

$$f(x, y^\star) = (x-1)(1-1) = 0 = f(x^\star, y^\star). \tag{3}$$

   Hence, $(x^\star, y^\star) = (1,1)$ is a solution to the minimax problem $\min_x \max_y f(x,y)$.

3. At each update,

$$x_{k+1} = x_k - \gamma \nabla_x f(x_k, y_k) = x_k - \gamma(y_k - 1), \tag{4}$$

$$y_{k+1} = y_k + \gamma \nabla_y f(x_k, y_k) = y_k + \gamma(x_k - 1). \tag{5}$$

   Denoting the distance of the sequence $\{x_k, y_k\}$ to $(x^\star, y^\star)$ at the k-th iteration by d_k, we have

$$d_k^2 = (x_k - 1)^2 + (y_k - 1)^2, \tag{6}$$

$$\begin{aligned} d_{k+1}^2 &= [x_k - \gamma(y_k - 1) - 1]^2 + [y_k + \gamma(x_k - 1) - 1]^2 \\ &= x_k^2 - 2x_k + y_k^2 - 2y_k + 2 + \gamma^2(x_k - 1)^2 + \gamma^2(y_k - 1)^2 \\ &= (1+\gamma^2)[(x_k - 1)^2 + (y_k - 1)^2] \\ &= (1+\gamma^2) d_k^2. \end{aligned} \tag{7}$$

   Given that $\gamma > 0$, with the non-negativity of a distance, d_{k+1} is strictly greater than d_k iff $d_k \neq 0$ (i.e. the sequence diverges at the rate of $\sqrt{1+\gamma^2}$). This condition implies $d_0 \neq 0$, which means that (x_0, y_0) is not initialised to $(1,1)$. Under this assumption, the distance of the sequence to the stationary point $d_k = O\left((1+\gamma^2)^k\right)$.

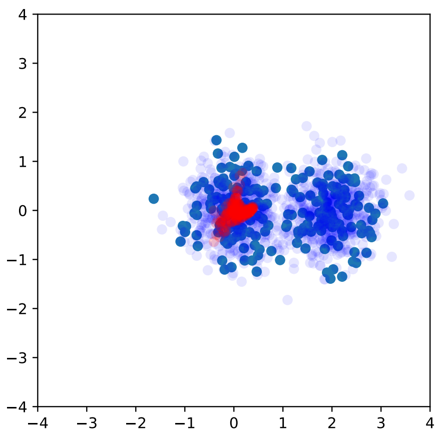
1.2 Practice: GAN

2. - Spectral normalisation: This method normalises each weight matrix $\boldsymbol{Y}^l$ by its spectral norm $\sigma(\boldsymbol{Y}^l)$ which is also its laregest singular value. In this way, all normalised weight matrices will satisfy the 1-Lipschitz constraint $\sigma(\boldsymbol{Y}^l_{\mathrm{SN}}) = 1$. Then, with the inequality $\|g_1 \circ g_2\|_{\mathrm{Lip}} \leqslant \|g_1\|_{\mathrm{Lip}}\|g_2\|_{\mathrm{Lip}}$, the upper bound of the discriminator's Lipschitz norm is given by $\|\mathsf{d}\|_{\mathrm{Lip}} \leqslant \prod_l \sigma(\boldsymbol{Y}^l_{\mathrm{SN}}) = 1$.

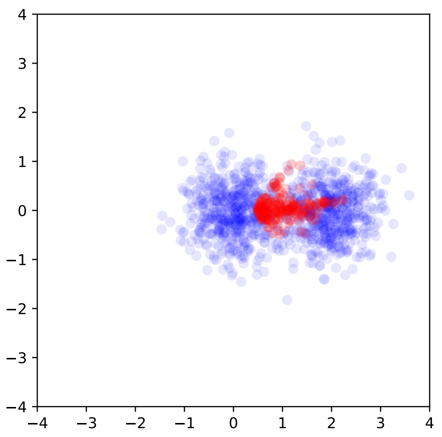
     Instead of the computatinally expensive singular value decomposition, the power iteration method is used for approximating the singular values.
   - Spectral regularisation: The term $\frac{\lambda}{2}\sum_l \sigma(\boldsymbol{Y}^l)^2$, where $\lambda > 0$, is added to the objective function. As a result, instead of setting all $\sigma(\boldsymbol{Y}^l)$ to 1, this method will choose weight matrices with small spectral norm during optimisation. The authors assumed that the neural network use piecewise linear activation functions. The ReLU function at each layer can be considered as a diagonal matrix $\boldsymbol{D}^l$, in which the diagonal entries are either 0 or 1. With the piecewise linearity, in the neighbourhood of input $\boldsymbol{a}$, the network be regarded as a linear transformation $\boldsymbol{Y}\boldsymbol{a} + \boldsymbol{b}$. Then, the spectral norm of $\boldsymbol{Y}$ is bounded by $\sigma(\boldsymbol{Y}) \leqslant \prod_l \sigma(\boldsymbol{D}^l)\sigma(\boldsymbol{Y}^l) \leqslant \prod_l \sigma(\boldsymbol{Y}^l)$.

     Another version of power iteration method is used for approximating the singular values.
   - Weight clipping: This is rather a primitive way to enforce the Lipschitz constraint: simply clipping the weights into the range $[-c, c]$ using a hyperparameter c. Thought it is simply and can still achieve good results, there are traps like slow convergence rate or vanishing gradient if the value c is not properly chosen. Hence, the authors looked forward to other Lipchitz constraining methods.
   - Gradient penalty: Enforce 1-Lipschitz constraint also means to ensure that $\|\nabla_{\boldsymbol{a}}\mathsf{d}(\boldsymbol{a})\| \leqslant 1$, which can be done by penalising the terms whose gradient norm is greater than 1. Like spectral regularisation, a penalty term: $\lambda\mathbb{E}_{\boldsymbol{a}\sim\nu}[(\|\nabla_{\boldsymbol{a}}\mathsf{d}(\boldsymbol{a})\| - 1)^2]$, is added to the objective function. This penalty term takes care of inputs from a special distribution ν which is uniform between the coupling of the samples from data and generator distributions. It is worth noting that this penalty in fact two-sided. It only requires that the gradient norm should be close to 1 but does not care whether it is smaller than 1.

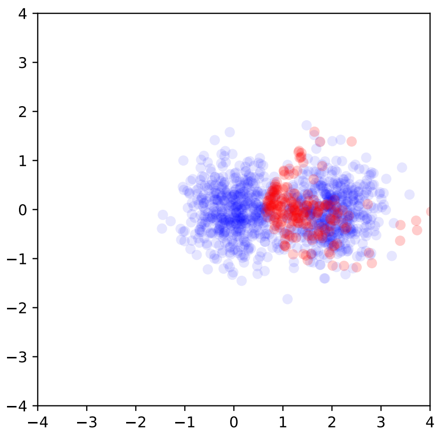
     Though the authors had also definend a one-sided penalty that indeed enforces $\|\nabla_{\boldsymbol{a}}\mathsf{d}(\boldsymbol{a})\| \leqslant 1$, it did not outperform the aforementionned two-sided term.



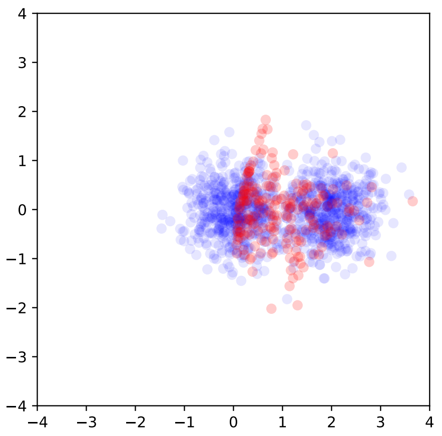
(a)



(b)



(c)

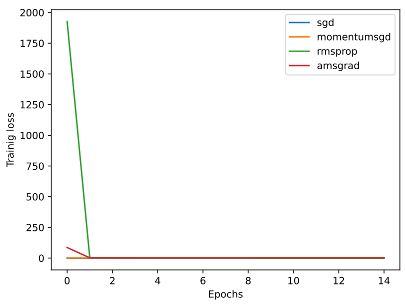


(d)

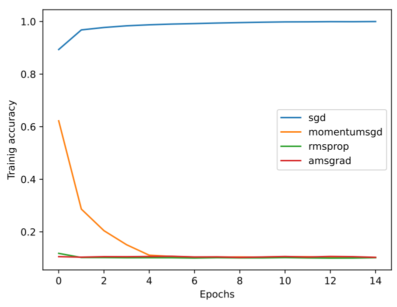
Figure 1: Data and noise distributions

3. Firstly, we can see that the noise samples are tightly clustered and moving toward the mean of the data. During this process, a frontier emerges and limits the noise samples to its right. Also, the points start to spread out. After its “peak” has approximated the data mean, the boundary moves toward the left. The noise continuously mimics the shape of the data. And the end of the training process, the noise has nearly the same shape as the data (which is more obvious in the animation). This means that the data distribution is well learned by the generator.

2 Optimisers of neural network

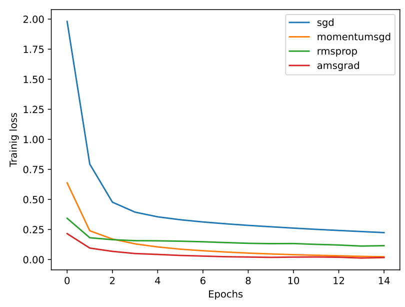


(a) Loss

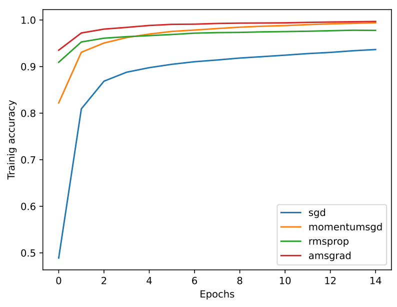


(b) Accuracy

Figure 2: Training results, learning rate = 0.5

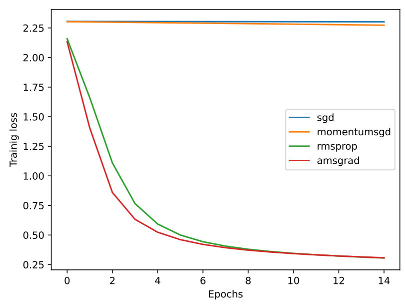


(a) Loss

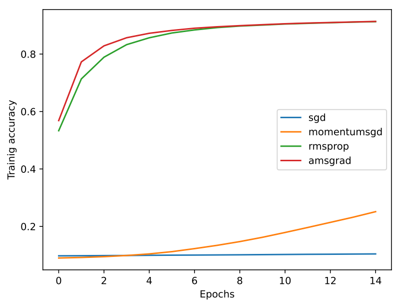


(b) Accuracy

Figure 3: Training results, learning rate = 10^{-2}



(a) Loss



(b) Accuracy

Figure 4: Training results, learning rate = 10^{-5}

Table 1: Averaged accuracies

Learning rate	0.5		10^{-2}		10^{-5}	
	Training	Test	Training	Test	Training	Test
SGD	1.0000	0.9841	0.9362	0.9377	0.1236	0.1222
MomentumSGD	0.1034	0.1048	0.9942	0.9786	0.2740	0.2854
RMSProp	0.1025	0.1094	0.9778	0.9674	0.9126	0.9163
AMSGrad	0.1049	0.0987	0.9962	0.9771	0.9146	0.9174

Among the three imposed learning rates, 10^{-2} is the most suitable one under which all the algorithms work well. Generally, the algorithms prefer more conservative learning rates except for SGD. The larger the step is, the better SGD learns. We can see that when the rate is 0.5, it is the only method that can learn correctly. Also, under this rate, other algorithms achieve low loss but also low accuracy. The reason can be that an algorithm has found a minimum for the training loss that will lead to more validation error (i.e. low accuracy). MomentumSGD is the least robust as it works with neither too large nor too small learning rates. AMSGrad performs slightly better than RMSProp.