École polytechnique fédérale de Lausanne

Mathematics of data: from theory to computation

Laboratory for information and inference systems

Fall 2021

Homework 2

Bruno Rodriguez Carrillo

Sciper 326180

Professor:
Volkan Cevher

Head TA's:
Fabian Latorre
Ali Kavis

December 5th, 2021

1 Minimax problems and GANs

Consider a stylized function $f : \mathbb{R}^2 \longrightarrow \mathbb{R}$ such that we have $f(x, y) = (x - 1)(y - 1)$

1. We have that the gradient of $f(x, y)$ is computed as:

$$\nabla f(x, y) = \left(\frac{\partial}{\partial x} f(x, y), \frac{\partial}{\partial y} f(x, y) \right)^T = (\nabla_x f(x, y), \nabla_y f(x, y))^T = (y - 1, x - 1)^T$$

   Since we are interested in computing the stationary points, we equal the gradient to zero; thus we have the following stationary point only:

$$y - 1 = 0 \text{ and } x - 1 = 0 \longrightarrow (x^*, y^*) = (1, 1)$$

   and the Hessian matrix is:

$$H_{f(x,y)} = \begin{bmatrix} \frac{\partial^2}{\partial x^2} f(x, y) & \frac{\partial^2}{\partial x \partial y} f(x, y) \\ \frac{\partial^2}{\partial y \partial x} f(x, y) & \frac{\partial^2}{\partial y^2} f(x, y) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

   Then, computing the eigenvalues of $H_{f(x,y)}$, we have $\lambda(H_{f(x,y)}) = \{-1, 1\}$. As a result, the point $(x^*, y^*) = (1, 1)$ corresponds to a saddle point.

2. We use the given saddle point inequality:

$$f(x^*, y) \leq f(x^*, y^*) \leq f(x, y^*)$$

   using $(x^*, y^*) = (1, 1)$, we have:

$$f(x^*, y) = 0, f(x, y^*) = 0 \text{ for any } x, y \in \mathbb{R}. \text{ Moreover } f(x^*, y^*) = 0$$

   Then, the previous inequality holds. In our case, the equality holds and the point $(x^*, y^*) = (1, 1)$ is a solution to the $\min_x \max_y f(x, y)$ problem.

3. We consider now a first-order iterative method to perform updates on the variables x and y. More precisely, we consider the following gradient descent/ascent updates:

$$x_{k+1} = x_k - \gamma \nabla_x f(x, y) \text{ and } y_{k+1} = y_k + \gamma \nabla_y f(x, y)$$

   for $\gamma > 0$. Making use of the values for the gradients computed preciously, we have that the $k + 1$ iteration for x and y are given by:

$$x_{k+1} = x_k - \gamma(y_k - 1) \text{ and } y_{k+1} = y_k + \gamma(x_k - 1)$$

   We now compute the distance from (x_{k+1}, y_{k+1}) to the point (x^*, y^*) as follows:

$$\|\mathbf{z}_{k+1} - \mathbf{z}^*\|_2^2 = |x_{k+1} - z_1^*|^2 + |y_{k+1} - z_2^*|^2$$

with $\mathbf{z}_{k+1} = (x_{k+1}, y_{k+1})^T$ and $\mathbf{z}^* = (z_1^*, z_2^*)^T = (1,1)^T$.

Substituting the values for x_{k+1} and y_{k+1}, simplifying the right hand side, we get:

$$(x_k^2 - 2x_k + 1 + y_k^2 - 2y_k + 1)(\gamma^2 + 1) = \left((x_k - 1)^2 + (y_k - 1)^2\right)(\gamma^2 + 1)$$

Consequently, we have:

$$\|\mathbf{z}_{k+1} - \mathbf{z}^*\|_2^2 = \|\mathbf{z}_k - \mathbf{z}^*\|_2^2(\gamma^2 + 1)$$

We then can observe that it is possible to compute $\|\mathbf{z}_k - \mathbf{z}^*\|_2^2$ using the same procedure as before. Finally, by recursion, we obtain:

$$\|\mathbf{z}_{k+1} - \mathbf{z}^*\|_2 = \sqrt{\|\mathbf{z}_0 - \mathbf{z}^*\|_2^2(\gamma^2 + 1)^k} = \sqrt{(\gamma^2 + 1)^k} \cdot \|\mathbf{z}_0 - \mathbf{z}^*\|_2$$

We observe that since k increases at each iteration, so does $\sqrt{(\gamma^2 + 1)^k}$ and as a result for the method to converge, we need $\mathbf{z}_0 = \mathbf{z}^*$, which in turn shows that the sequence $\{x_{k+1}, y_{k+1}\}$ diverges if $\mathbf{z}_0 \neq \mathbf{z}^*$.

Now, if we select the origin $\mathbf{z}^* = (0,0)$, we have:

$$\|\mathbf{z}_{k+1}\|_2 = \sqrt{(\gamma^2 + 1)^k} \cdot \|\mathbf{z}_0\|_2$$

Then, the rate at which the the distance of the sequence $\{x_k, y_k\}$ to the origin grows as $(\gamma^2 + 1)^{k/2}$ while increasing the number of iterations k. It is worth pointing out that such a rate also corresponds to the rate at which the distance of the same sequence to the saddle point grows.

We illustrate the saddle point in the following picture:

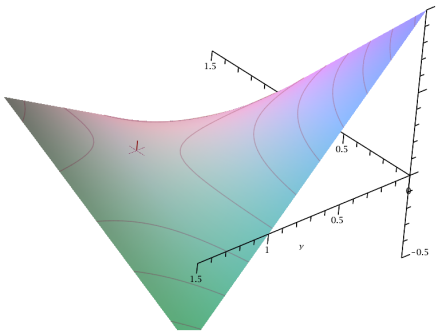


Figure 1: Plot of $f(x, y)$ and its saddle point

1.1 Practice: GAN

1. See attached code for implementation.

2. See attached code for implementation.

   As stated in the reference paper, in the code we call the enforce_lipschitz function before computing the gradient step. To compute the estimate of the stochastic estimate of the objective function:

   $$\min_{h\in\mathcal{H}} \max_{d\in\mathcal{F}} \mathbb{E}\left[d(\mathbf{a})\right] - \mathbb{E}\left[d(h(\omega))\right]$$

   We compute a simple arithmetic average: $\frac{1}{N} \cdot \sum_{i=1}^{N} f(\mathbf{a}) - f(g(\omega))$

   since we know that $\mathbf{a} \sim \hat{\mu}_n$ is distributed with respect to the empirical estimate $\hat{\mu}_n$ of the true distribution $\hat{\mu}^\sharp$ and $\omega \sim p_\Omega$ is the noise of the distribution that we push-forward with the generator neuronal network $h_x(\omega) \sim h_x^\sharp p_\Omega$.

   Similarly, we implement weight clipping by using the in-built PyTorch function p.clamp_().

3. The learning rate used is the pre-defined value in Pytorch for ADAM, $lr = 0.001$.

   From the reference paper, we know that the largest singular value of a matrix $\mathbf{W}$ is also its spectral norm

   $$\sigma(\mathbf{W}) = \max_{\mathbf{h}:\|\mathbf{h}\|_2\leq 1} \|\mathbf{W}\mathbf{h}\|_2$$

   And to compute it, we use the power iteration method. This starts with two random vectors $\mathbf{u}_0$ and $\mathbf{v}_0$ and we apply the following in a iterartive manner:

   $$\mathbf{v}_{n+1} = \frac{\mathbf{W}^T\mathbf{u}_n}{\|\mathbf{W}^T\mathbf{u}_n\|_2} \ , \ \mathbf{u}_{n+1} = \frac{\mathbf{W}\mathbf{v}_{n+1}}{\|\mathbf{W}\mathbf{v}_{n+1}\|_2} \ , \ \mathbf{W} = \mathbf{W}/\mathbf{u}_{n+1}^T\mathbf{W}\mathbf{v}_{n+1}$$

   and we note that : $\sigma(\mathbf{W}) = \lim_{n\to\infty} \mathbf{u}_n^T\mathbf{W}\mathbf{v}_n$

   For spectral normalization we have the following plots when using the default parameters provided with the codes:

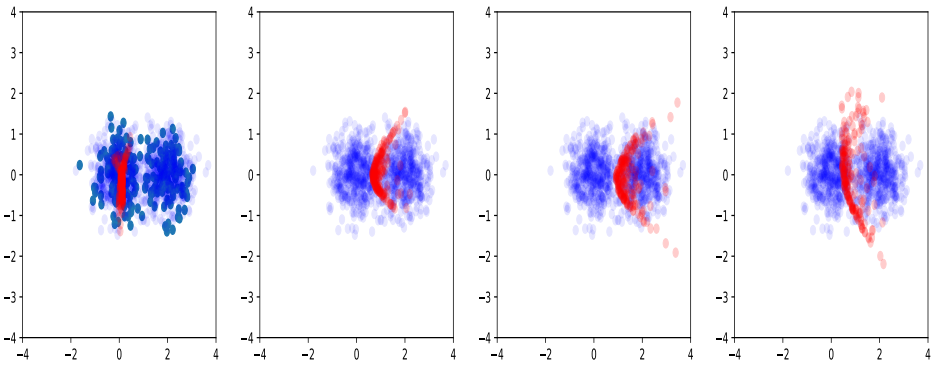


Figure 2: Spectral normalization, default parameters.

If we change the number of iterations and number of neurons in the hidden layer to 1000 both, we have the following results:

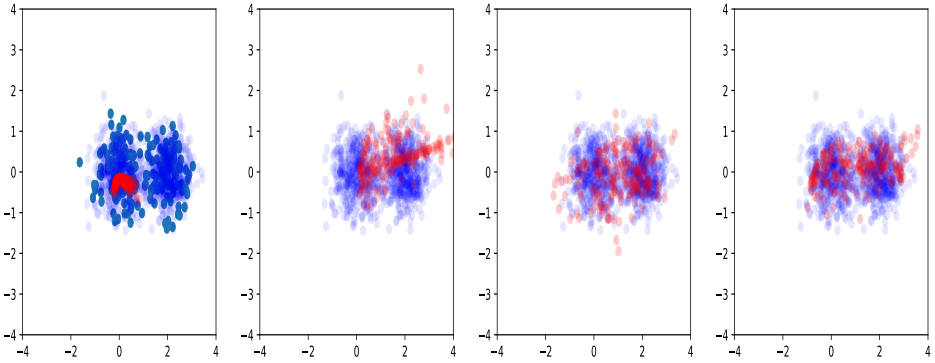


Figure 3: Spectral normalization, modified parameters.

On the other hand, now we use weight clipping and use the default parameters provided with the code; we have

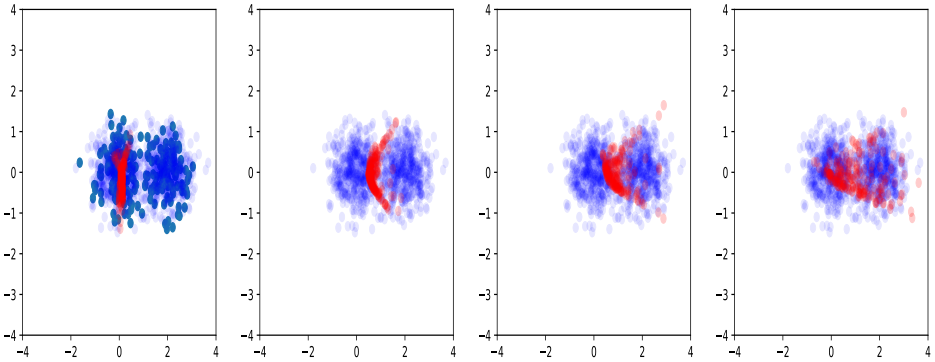


Figure 4: Weight clipping, default parameters.

Similarly, when we set the number of iterations and number of neurons in the hidden layer to 1000 both, we have the following results:

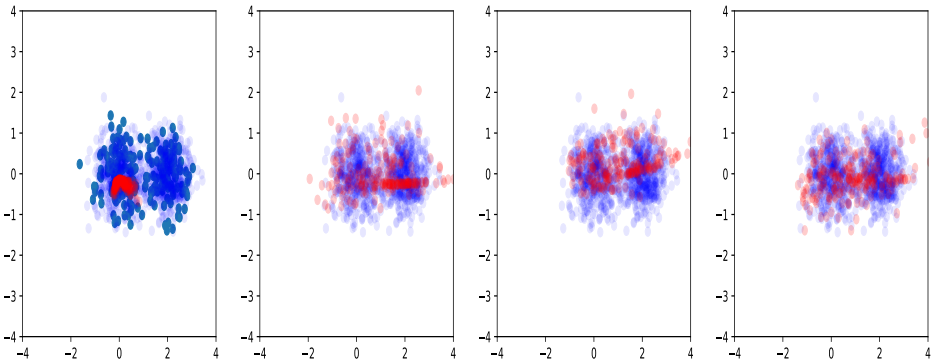


Figure 5: Weight clipping, modified parameters.

From the previous plots, we observe the following:

(a) First we should point out that the blue points corresponds to data points that we want to classify and from the plots, we may say that there exist two clusters; our model's role is to learn the parameters of a mixture of two Gaussian distributions. We recall that our model has the task of finding the mean value, the covariance matrix and the probability of a point belonging to either cluster.

(b) Regarding the implementation, when we use 400 iterations and 100 neurons in the single hidden layer (these values are the ones we modified to test model's performance), we clearly see that the model is not capable of deciding what the cluster of each point is. Interestingly, our model then is able to come up with a distribution centered between the two clusters. This behavior is observed for both spectral normalization and weight clipping.

(c) However, when we increased the number of neurons in the hidden layer and the number of iterations to 1000 each, the model improves its learning capabilities. This is observed in how the red points become nearly equally distributed into both clusters. We may thus say that the model actually learns that there exist two clusters and it assigns almost the same number of points to either cluster. As with the previous point, for spectral normalization and weight clipping, the results are quite similar.

As observed, in our exercise spectral normalization and weight clipping we obtain quite similar results. However, we shall mention that, based on the reference paper for this homework, weight clipping may have the following drawback; citing the paper *weight clipping simply truncates each element of weight matrices* $\mathbf{W}$ *so that its absolute value is bounded above by a constant* $c > 0$. *Unfortunately, weight clipping suffers from the same problem as weight normalization and Frobenius normalization. With weight clipping with the truncation value* c*, the value* $\mathbf{Wx}$ *for a fixed unit vector* $\mathbf{x}$ *is maximized when the rank of* $\mathbf{W}$ *is one, and the training will again favor the discriminators that use only select few features.*

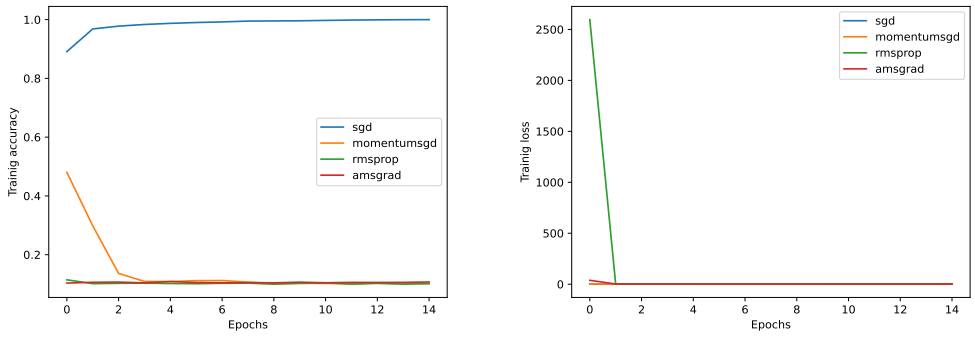
2 Optimizer of neuronal networks

1. See attached code for implementation.
2. See attached code for implementation.
3. We have the following chart summarizing the results for the given learning rates lr. It is important to mention that each accuracy is averaged from three independent runs for each accuracy.

lr	method	test accuracy
0.50	SGD	0.9841
	momentum SGD	0.1022
	RMS prop	0.1046
	AMS grad	0.10651

Table 1: Test accuracy for different optimizers, $lr = 0.5$.

with plots



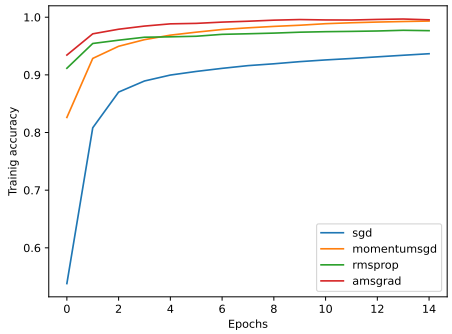
(a) Accuracy. (b) Loss.

Figure 6: Training accuracy and loss for different optimizers, $lr = 0.5$.

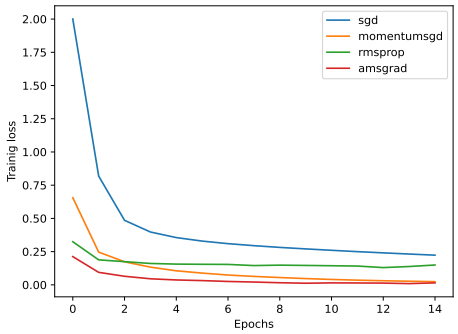
lr	method	test accuracy
$1e^{-2}$	SGD	0.9384
	momentum SGD	0.9796
	RMS prop	0.9605
	AMS grad	0.9788

Table 2: Test accuracy for different optimizers, $lr = 1e^{-2}$.

with plots



(a) Accuracy.



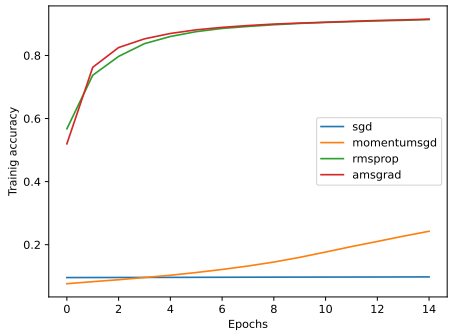
(b) Loss.

Figure 7: Training accuracy and loss for different optimizers, $lr = 1e^{-2}$.

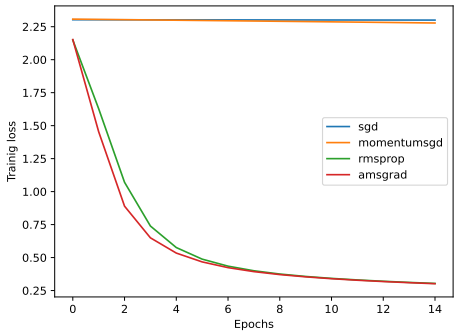
lr	method	test accuracy
$1e^{-5}$	SGD	0.1081
	momentum SGD	0.2871
	RMS prop	0.9170
	AMS grad	0.8984

Table 3: Test accuracy for different optimizers, $lr = 1e^{-5}$.

with plots



(a) Accuracy.



(b) Loss.

Figure 8: Training accuracy and loss for different optimizers, $lr = 1e^{-5}$.

(a) From figure (6), we can observe that the standard mini-batch SGD method outperforms greatly adaptive-learning rate and with-momentum methods

(b) From figure (7), the learning rate $1e{-}2$ seems to be such that the four methods give similar accuracy

(c) From figure (8), the learning rate $1e{-}2$ may be the one for which RMSProp and AMSGrad outperform the SGD and SGD with momentum more evidently. For such a learning rate, there is a clear distinction between adaptive-learning-rate methods and the rest.

From the previous results, it looks like adaptive-learning-rate methods like RMSProp and AMSGrad perform better when the global learning rate is small. Here, when we mention 'small', we want to say that the learning rates $1e{-}2$ and $1e{-}5$ are evidently much smaller than 0.50.

One final comment we can make is the fact that the standard minibatch SGD method behaves better than the rest when the learning rate is the largest and that adaptive-learning-rate methods give better results when the step we take in each iteration of the gradient computation is small.