# EE-556 Homework 2

Edoardo Debenedetti

November 26, 2019

# 1 Minimax problems and GANs

## 1.1 Minimax problem

Consider the function $f : \mathbb{R}^2 \to \mathbb{R}$, where $f(x, y) = xy$.

### 1.1.1 First order stationary points of $f$

In order to find the first order stationary points, we need to find the gradient and see with which $x$ and $y$ it is equal to $(0, 0)$. The gradient of $f$ is:

$$\nabla f(x, y) = \left[\frac{\partial}{\partial x} xy, \frac{\partial}{\partial y} xy\right] = [y, x] \tag{1}$$

Clearly $\nabla f = (0, 0) \iff (x, y) = (0, 0)$. In order to classify this stationary point, we need to find the hessian of $f(0, 0)$:

$$\nabla^2 f(x, y) = \nabla \cdot \nabla f(x, y) = \left[\frac{\partial}{\partial x}, \frac{\partial}{\partial y}\right] \cdot \begin{bmatrix} y \\ x \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \ \forall x, y \in \mathbb{R} \tag{2}$$

The eigenvalues of $\nabla^2 f(x, y)$ are $\lambda_1 = -1$ and $\lambda_2 = 1$, one is positive and one is negative, hence the matrix is not (semi-)positive nor (semi-)negative definite. However, we can do the second derivative test to say something more:

$$Df(x, y) = \frac{\partial^2 f}{\partial^2 x}\frac{\partial^2 f}{\partial^2 y} - \frac{\partial^2 f}{\partial x \partial y}\frac{\partial^2 f}{\partial y \partial x} = 0 \cdot 0 - 4 \cdot 1 = -4 \tag{3}$$

Since $Df(0, 0) < 0$, $(0, 0)$ is a saddle point.

### 1.1.2 Solution of minimax problem $\min_x \max_y f(x, y)$

$(x^\star, y^\star) = (0, 0)$ is a solution to the minimax problem $min_x max_y f(x, y)$ iff

$$\begin{gathered} f(x^\star, y^\star) \geq f(x^\star, y) \ \forall y \in \mathbb{R} \ \wedge \\ f(x^\star, y^\star) \leq f(x, y^\star) \ \forall x \in \mathbb{R} \end{gathered} \tag{4}$$

*Proof.* With $(x^\star, y^\star) = (0, 0)$, we have that:

$$\begin{aligned} f(x^\star, y) = 0 \cdot y = 0 \ \forall y \in \mathbb{R}, \text{ and} \\ f(x, y^\star) = x \cdot 0 = 0 \ \forall x \in \mathbb{R} \end{aligned} \tag{5}$$

Hence, recalling that $f(x^\star, y^\star) = 0$:

$$\begin{aligned} f(x^\star, y^\star) = f(x^\star, y) \ \forall y \in \mathbb{R} \ \wedge \\ f(x^\star, y^\star) = f(x, y^\star) \ \forall x \in \mathbb{R} \end{aligned} \tag{6}$$

However, the equalities can be transformed in to a non-strict inequalities:

$$\begin{aligned} f(x^\star, y^\star) \geq f(x^\star, y) \ \forall y \in \mathbb{R} \ \wedge \\ f(x^\star, y^\star) \leq f(x, y^\star) \ \forall x \in \mathbb{R} \end{aligned} \tag{7}$$

which satisfies eq. (4).

□

### 1.1.3 Gradient descent/ascent divergence

*Proof.* The sequence of iterates $\{x_k, y_k\}_{k=0}^{\infty}$, starting from any point $(x_0, y_0) \neq (0, 0)$ having

$$x_{k+1} = x_k - \gamma \nabla_x f(x_k, y_k), \quad y_{k+1} = y_k + \gamma \nabla_y f(x_k, y_k) \tag{8}$$

But the gradient is known, and the updates can be written as:

$$x_{k+1} = x_k - \gamma y_k, \qquad y_{k+1} = y_k + \gamma x_k, \tag{9}$$

The iterates diverge for any $\gamma > 0$ iff $\|(x_k, y_k)\| \to \infty$, as $k \to \infty$. It is convenient, then, to study the behavior of the squared norm of $\|(x_{k+1}, y_{k+1})\|^2$:

$$\|(x_{k+1}, y_{k+1})\|^2 = x_{k+1}^2 + y_{k+1}^2 \tag{10}$$

We can explicit $x_{k+1}$ and $y_{k+1}$ using eq. (9):

$$\begin{aligned} x_{k+1}^2 + y_{k+1}^2 = (x_k - \gamma y_k)^2 + (y_k + \gamma x_k)^2 = \\ = x_k^2 + \gamma^2 y_k^2 - 2\gamma y_k x_k + y_k^2 + \gamma^2 x_k^2 + 2\gamma y_k x_k = \\ = x_k^2 + y_k^2 + \gamma^2 (x_k^2 + y_k^2) = \\ = (1 + \gamma^2)(x_k^2 + y_k^2) \end{aligned} \tag{11}$$

We can now note that $x_k^2 + y_k^2 = \|(x_k, y_k)\|^2$, then, combining with eq. (11):

$$\|(x_{k+1}, y_{k+1})\|^2 = (1 + \gamma^2)\|(x_k, y_k)\|^2$$

Let us recall that $(x_0, y_0) \neq (0, 0)$, then $\|(x_0, y_0)\|^2 \neq 0$. Taking the ratio between the two subsequent norms, we get:

$$\frac{\|(x_{k+1}, y_{k+1})\|^2}{\|(x_k, y_k)\|^2} = (1 + \gamma^2)$$

Since $\gamma > 0$, $1 + \gamma^2 > 0$, then we can state that:

$$\frac{\|(x_{k+1}, y_{k+1})\|^2}{\|(x_k, y_k)\|^2} > 1, \ \forall k \in \mathbb{N}$$

and then

$$\|(x_{k+1}, y_{k+1})\|^2 > \|(x_k, y_k)\|^2, \ \forall k \in \mathbb{N} \tag{12}$$

Which means that the norm $\|(x_k, y_k)\|^2$ strictly increases each iteration. Moreover, since the ratio between two subsequent squared norms is $1 + \gamma^2$, it means that the norm, and then the sequence of iterates $\{x_k, y_k\}_{k=0}^{\infty}$ diverges at a quadratic $\sqrt{1 + \gamma^2}$ rate.

□

## 1.2 GANs

Given a generator $\mathcal{G}$ and a dual variable $\mathcal{F}$

$$\mathcal{G} := \{g : g(z) = Wz + b\} \qquad \mathcal{F} := \{f : g(x) = v^T x\} \tag{13}$$

Where we want $\mathcal{G}$ to map a standard normal noise distribution on $\mathbb{R}^2$ to a *true* distribution which is multivariate normal on $\mathbb{R}^2$. In our specific case, $W \in \mathbb{R}^{2\times 2}$, and $z, \ x, \ v, \ b \in \mathbb{R}^2$.

The minimax game we want to optimize is:

$$\min_{g\in\mathcal{G}} \max_{f\in\mathcal{F}} \mathbb{E}[f(X) - f(g(Z))] \tag{14}$$

### 1.2.1 $f$ L-Lipischitz constant

Supposing that $\mathbb{R}^2$ is equipped with $\ell_2$-norm $\|(x, y)\|_2^2 = x^2 + y^2$, we can find the L-Lipschitz constant by using Lipschitz smoothness definition, according to which a function $f : \mathbb{R}^p \to \mathbb{R}$ is L-Lipschitz smooth iff

$$\|f(\mathbf{x}_1) - f(\mathbf{x}_2)\| \leq L\|\mathbf{x}_1 - \mathbf{x}_2\|, \ \forall \mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^p \tag{15}$$

where $L$ is the L-Lipschitz smoothness constant. We can then compute the norm of $f \in \mathcal{F}$:

$$\|f(\mathbf{x}) - f(\mathbf{y})\| = \|v^T\mathbf{x} - v^T\mathbf{y}\| = \|v^T(\mathbf{x} - \mathbf{y})\| \tag{16}$$

Using Cauchy-Schwartz inequality:

$$\|v^T(\mathbf{x} - \mathbf{y})\| \leq \|v^T\|\|\mathbf{x} - \mathbf{y}\| \leq L\|\mathbf{x} - \mathbf{y}\|, \iff \|v^T\| \leq L \tag{17}$$

Then the L-Lipschitz smoothness constant of any function $f \in \mathcal{F}$ is $L = \|v^T\|$. The set of functions $f \in \mathcal{F}$ whose $L \leq 1$ are the functions whose $\|v^T\| \leq 1$.

### 1.2.2 1-Lipschitz enforcement

Using eq. (17) we can enforce $f$ to have L-Lipschitz constant smaller or equal to 1 by normalizing the $v$-s whose norm is greater than one. This is done in PyTorch by doing `self.v.data /= norm` where `norm = self.v.norm()`. This is needed to compute Wasserstein distance, according to the Kantorovich-Rubinstein duality theorem:

$$W(\mu, \nu) = \sup \{\mathbb{E}_{x\sim\mu}[f(x)] - \mathbb{E}_{y\sim\nu}[f(y)] : f \text{ is 1-Lipschitz}\} \tag{18}$$

The objective function $\mathbb{E}[f(X) - f(g(Z))]$ can be computed by taking a random sample from the real distribution and the noise, computing the function itself applying the two functions `f` on X (the true distribution) and to the result of `g` of Z (the noise). We can then take the mean of the results (which, in a similar fashion as we did in homework 1, can be proved to be an unbiased estimate of the complete objective function).

### 1.2.3 Training implementation

We now try to implement the training in two different ways:

$$f_{k+1} = f_k + \gamma \mathrm{SG}_f(f_k, g_k) \quad g_{k+1} = g_k - \gamma \mathrm{SG}_g(f_k, g_k) \qquad \text{(simultaneous)} \tag{19}$$

$$f_{k+1} = f_k + \gamma \mathrm{SG}_f(f_k, g_k) \quad g_{k+1} = g_k - \gamma \mathrm{SG}_g(f_{k+1}, g_k) \qquad \text{(alternating)} \tag{20}$$

The main difference between the two implementations is the fact that in the simultaneous one described in eq. (19), after resetting the gradients on both the optimizers, we compute the objective function and its gradient only once and then update both `f` and `g` one after the other. In the case of alternating updates described in eq. (20), we first compute the objective function and its gradient, then update only `f`, next we compute the objective function and its gradient again, and finally update `g` only.

The results of simultaneous updates can be checked statically in figure 1 and dynamically in a GIF on Giphy. The results obtained with alternating updates, instead, can be observed statically in figure 2 and dinamically on Giphy. In both cases we can see that the noise (the red points) gets almost the shape of the real distribution (the blue points), but keeps oscillating around its mean. We will discuss the reason of this in 1.2.5.

### 1.2.4 max of expectation of difference of two distributions

*Proof.* Given two distributions $\mu$ and $\nu$ and the class of functions $f \in \mathcal{F}$ we can prove that

$$\max_{f\in\mathcal{F}} \mathbb{E}_{X\sim\mu}[f(X)] - \mathbb{E}_{Y\sim\nu}[f(Y)] \geq 0 \tag{21}$$

by first swapping the expectation operators with the function $f$, since both are linear:

$$\max_{f\in\mathcal{F}} \mathbb{E}_{X\sim\mu}[f(X)] - \mathbb{E}_{Y\sim\nu}[f(Y)] = \tag{22}$$

$$\max_{f\in\mathcal{F}} f(\mathbb{E}_{X\sim\mu}[X]) - f(\mathbb{E}_{Y\sim\nu}[Y]) = \tag{23}$$

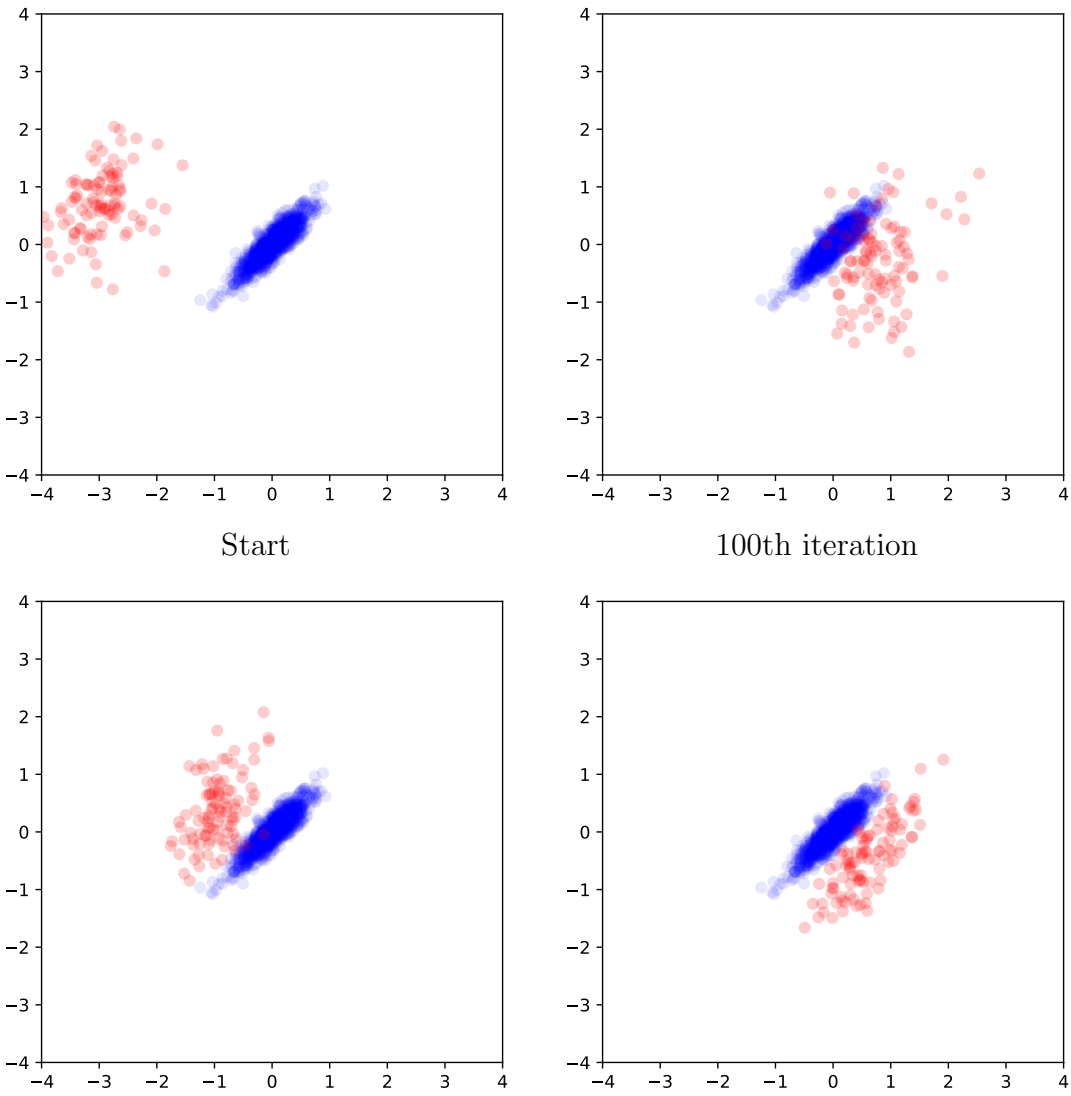



Figure 1: Simultaneous updates GAN training results

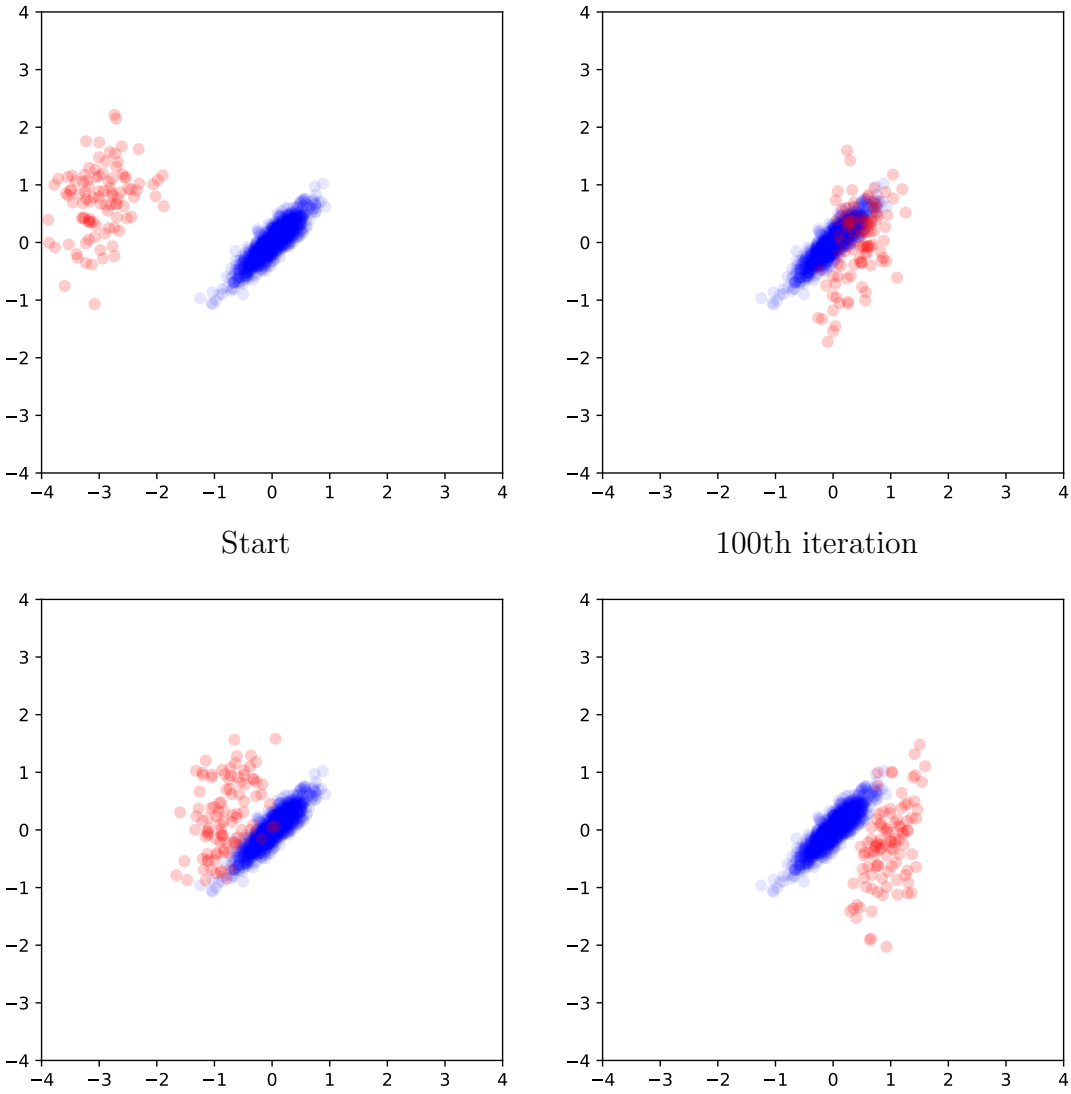



Figure 2: Alternating updates GAN training results

From now on, for the sake of simplicity we will write $\mathbb{E}_{X\sim\mu}[X]$ as $\bar{X}$ and $\mathbb{E}_{Y\sim\nu}[Y]$ as $\bar{Y}$. We can also define the set of vectors $v \in \mathcal{V}_1$ which have norm $\|v\| \leq 1$ and then express eq. (23) as

$$\max_{v\in\mathcal{V}_1} v^T\bar{X} - v^T\bar{Y} = \tag{24}$$

$$\max_{v\in\mathcal{V}_1} v^T(\bar{X} - \bar{Y}) \tag{25}$$

Now we can consider different case, based on the different nature of the distributions. In the case where $\bar{X} - \bar{Y} = 0$, $\max_{v\in\mathcal{V}_1} v^T(\bar{X} - \bar{Y}) = 0$, which satisfies eq. (21). Otherwise, if $\bar{X} - \bar{Y} \lneqq 0$, we can pick as $v \in \mathcal{V}_1$ the normal vector

$$\hat{v} = \frac{\bar{X} - \bar{Y}}{\|\bar{X} - \bar{Y}\|} \tag{26}$$

In such case,

$$\hat{v}^T(\bar{X} - \bar{Y}) = \left(\frac{\bar{X} - \bar{Y}}{\|\bar{X} - \bar{Y}\|}\right)^T (\bar{X} - \bar{Y}) = \tag{27}$$

$$= \frac{\|\bar{X} - \bar{Y}\|^2}{\|\bar{X} - \bar{Y}\|} = \|\bar{X} - \bar{Y}\| \geq 0 \tag{28}$$

Since eq. (23) requires the max to be greater or equal than 0, we just need to find one $f$ that satisfies such condition, which is satisfied with $f = \hat{v}^T x$. However, since $\hat{v}$ is parallel to $\bar{X} - \bar{Y}$ and since is the largest vector that satisfies the 1-Lipschitz condition, is the vector that maximizes the product $v^T(\bar{X} - \bar{Y})$ which means

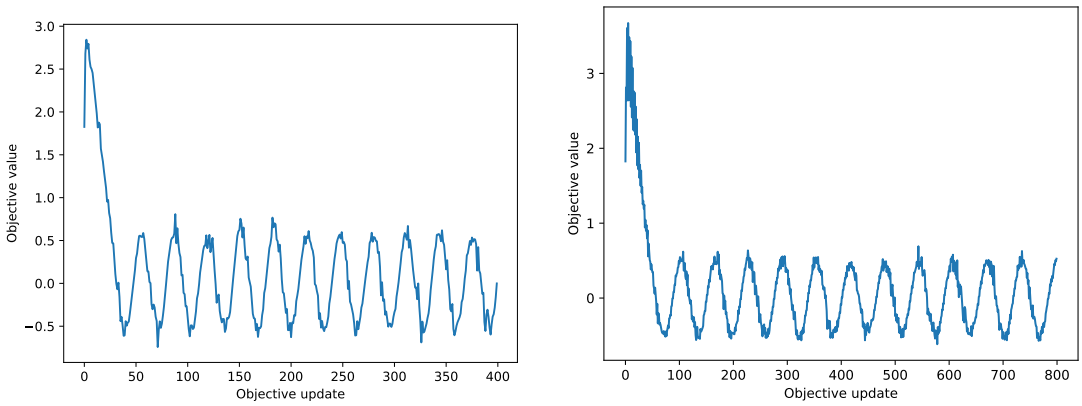
$$\hat{v} \in \{v : \max_{v\in\mathcal{V}_1} v^T(\bar{X} - \bar{Y})\} \tag{29}$$

□

### 1.2.5 0 max with equal first momentum

*Proof.* If we consider the left side of the inequality defined in (1.2.4), and we assume the first moment of $\mu$ and $\nu$ to coincide, then $\mathbb{E}_{X\sim\mu}[X] = \mathbb{E}_{Y\sim\nu}[Y]$, since $X$ and $Y$ are two random variables drawn from $\mu$ and $\nu$. This leads to the case in eq. (25) where $\bar{X} - \bar{Y} = 0$ (recalling that $\mathbb{E}_{X\sim\mu}[X] = \bar{X}$ and $\mathbb{E}_{Y\sim\nu}[Y] = \bar{Y}$). Hence $\max_{v\in\mathcal{V}_1} v^T(\bar{X} - \bar{Y}) = 0$, which satisfies the required hypothesis. □

**Update non-convergence study** In the case of the simultaneous update, we update $f$ and $g$ together, which means that we simultaneously try to minimize Wasserstein distance over the generator $g$ and to compute the Wasserstein distance maximizing the difference between the expectations over the dual variable $f$. Instead, in the case of alternating updates, we first maximize the difference between the expectations over the dual variable $f$ to compute Wasserstein distance, and then we try to minimize Wasserstein distance over the generator $g$.




Simultaneous updates objective Alternating updates objective

Figure 3: GAN training objective behavior

We can see from figure 2 that the real behavior of the generated data is oscillating around the mean, which means that $\mathbb{E}_{Z\sim\nu}[f(g(Z))]$ oscillates around $\mathbb{E}_{X\sim\mu}[f(X)]$, and then, since $f$ is linear, we can say that $\mathbb{E}_{Z\sim\nu}[g(Z)]$ oscillates around $\mathbb{E}_{X\sim\mu}[X]$. This is evident also in figure 3, where we can see how the objective oscillates around 0 in both kinds of updates. Using alternating updates lead to a more oscillatory behavior, but the trend is the same. This is caused by the fact that, when the noise $Z$ is mapped to a distribution that has the same expectation as $X$ does, then $\max_{f\in\mathcal{F}} \mathbb{E}_{X\sim\mu}[f(X)] - \mathbb{E}_{Z\sim\nu}[f(g(Z)] = 0$, according to what we proved above. However, $g$ will still be updated to minimize the objective function (that can get negative values), leading to values smaller than 0, until the dual variable counteracts the observed behavior, causing the observed oscillatory behavior. A possible solution to this problem could be to take in consideration also higher order momenta for the distributions (such as variance).

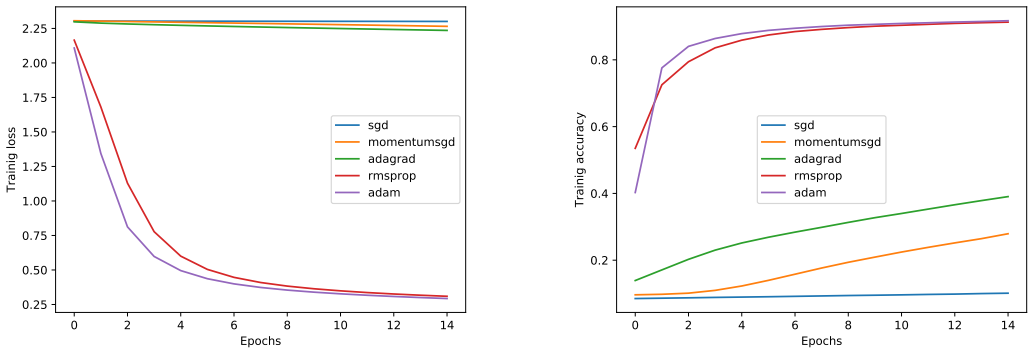
# 2 Optimizers of Neural Networks

From the plots in figures 4, 5, 6 and from the tables 1, 2, 3, we can see that, in general, the best performing learning rate is $10^{-2}$. In fact, the only two methods that converge fast enough with l.r. $10^{-5}$ are ADAM and RMSProp. On the other side, the only two algorithms that do not suffer from the larger 0.5 learning rate are regular SGD and AdaGrad. Moreover, it is worth noting that with 0.5 l.r. the loss decreases quite fast for all the methods, but the accuracy does not increase for all of them. This phenomenon could be caused by overfitting on the loss function. In a classification task like the one our neural network is performing is more important to use the accuracy than the loss as a performance metric (even though accuracy could have strong limitations due to class proportions in the dataset in use).

In general, adaptive gradient methods perform slightly better and are more robust to different learning rates, even though we have some noteworthy differences among them. As a matter

| | 1st run | | 2nd run | | 3rd run | | Mean | |
|---|---|---|---|---|---|---|---|---|
| Algorithm | Training | Test | Training | Test | Training | Test | Training | Test |
| SDG | 0.1251 | 0.122 | 0.1177 | 0.115 | 0.1012 | 0.095 | **0.1147** | **0.1107** |
| SDGm | 0.1915 | 0.2005 | 0.107 | 0.1101 | 0.2792 | 0.2887 | **0.1926** | **0.1998** |
| RMSProp | 0.9134 | 0.9162 | 0.9131 | 0.918 | 0.9129 | 0.9167 | **0.9131** | **0.9170** |
| ADAM | 0.9162 | 0.9183 | 0.917 | 0.9201 | 0.9171 | 0.9194 | **0.9168** | **0.9193** |
| AdaGrad | 0.3218 | 0.3205 | 0.3969 | 0.4084 | 0.3904 | 0.3964 | **0.3697** | **0.3751** |

Table 1: Results obtained with learning rate $10^{-5}$



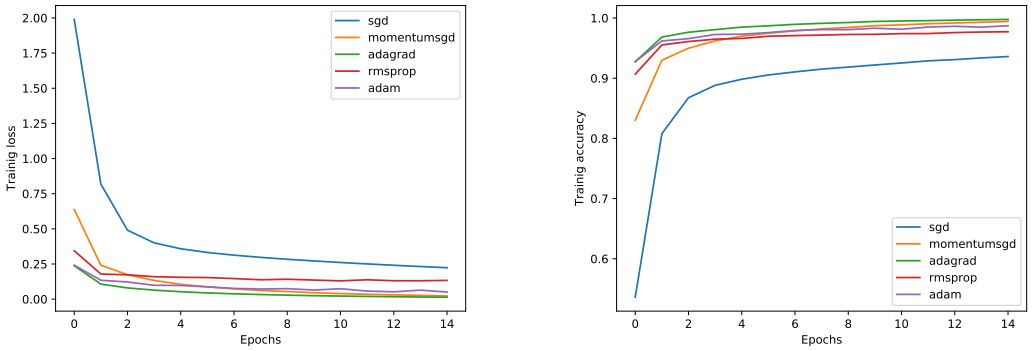

Loss Accuracy

Figure 4: Performance with learning rate $10^{-5}$

of fact, AdaGrad does not perform well enough with a too small learning rate, which can be caused by the stronger weight given to older gradients. Older gradients could lead to a more *conservative* behavior that slows down AdaGrad's convergence. On the other side, this *conservative* behavior improves AdaGrad's performance with the significantly larger 0.5 stepsize, for which we get acceptable enough performance.

| | 1st run | | 2nd run | | 3rd run | | Mean | |
|---|---|---|---|---|---|---|---|---|
| Algorithm | Training | Test | Training | Test | Training | Test | Training | Test |
| SDG | 0.9366 | 0.9372 | 0.9366 | 0.9396 | 0.936 | 0.9392 | **0.9364** | **0.9387** |
| SDGm | 0.994 | 0.9792 | 0.9938 | 0.9791 | 0.9943 | 0.981 | **0.9940** | **0.9798** |
| RMSProp | 0.9783 | 0.9676 | 0.9771 | 0.9617 | 0.9772 | 0.9617 | **0.9775** | **0.9637** |
| ADAM | 0.985 | 0.9688 | 0.9858 | 0.9687 | 0.9872 | 0.9737 | **0.9860** | **0.9704** |
| AdaGrad | 0.9978 | 0.9824 | 0.9975 | 0.9803 | 0.9977 | 0.9812 | **0.9977** | **0.9813** |

Table 2: Results obtained with learning rate $10^{-2}$

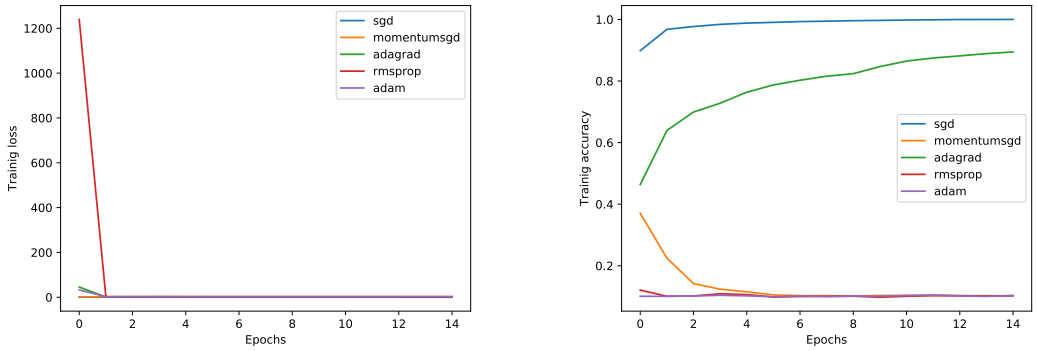



Loss Accuracy

Figure 5: Performance with learning rate $10^{-2}$

| | 1st run | | 2nd run | | 3rd run | | Mean | |
|---|---|---|---|---|---|---|---|---|
| Algorithm | Training | Test | Training | Test | Training | Test | Training | Test |
| SDG | 0.9999 | 0.985 | 1 | 0.985 | 1 | 0.9839 | **1.0000** | **0.9846** |
| SDGm | 0.2101 | 0.2144 | 0.1031 | 0.1135 | 0.1022 | 0.101 | **0.1385** | **0.1430** |
| RMSProp | 0.1033 | 0.101 | 0.0995 | 0.1034 | 0.1027 | 0.1135 | **0.1018** | **0.1060** |
| ADAM | 0.1031 | 0.1136 | 0.1036 | 0.0982 | 0.1031 | 0.1135 | **0.1033** | **0.1084** |
| AdaGrad | 0.8418 | 0.8481 | 0.8972 | 0.8876 | 0.8944 | 0.8879 | **0.8778** | **0.8745** |

Table 3: Results obtained with learning rate 0.5




Loss Accuracy

Figure 6: Performance with learning rate 0.5