Table of Contents

Matrix Factorization on GPU with Noisy SGD, Momentum, and EMA

Motivation

Problem Setup

Objective Function

Training Strategies

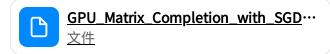
- 1. Vanilla Noisy SGD
- 2. Noisy SGD with Momentum
- 3. Noisy SGD with EMA

Experiments

- 1. Small Matrix $(10 \times 10, Rank = 2)$
- 2. Large Matrix (500 × 500, Rank = 10)
- Visual Results
- Conclusion

Next Steps

- GitHub-compatible fixes applied:
 - Full File Ready?





I will give you a report and a file of code. Read the code carefully and write a report using the given structure based on the code.

Great! I've reviewed the code in your notebook

GPU_Matrix_Completion_with_SGD_and_NQM.ipynb and the structure of

the Week11 report.md . Here's a new report written using the same format, based specifically on the implementation in your notebook:

Matrix Factorization on GPU with Noisy SGD, Momentum, and EMA

Motivation

Matrix factorization is a widely used approach in recommender systems to fill in missing values in user-item rating matrices. In this project, we investigate GPU-accelerated matrix factorization using stochastic gradient descent (SGD) with added noise and explore the enhancements provided by Momentum and Exponential Moving Average (EMA). Leveraging GPU computation significantly speeds up training, enabling experiments on large-scale matrices.

We focus on three training strategies:

- Vanilla Noisy SGD
- Noisy SGD with Momentum
- Noisy SGD with EMA

We examine their performance on both small and large matrices to assess convergence behavior and reconstruction quality.

Problem Setup

Given a partially observed matrix $(X \in \mathbb{R}^{m \times n})$, our goal is to approximate it using the product of two low-rank matrices $(U \in \mathbb{R}^{m \times k})$ and $(V \in \mathbb{R}^{n \times k})$, along with user/item/global biases:

$$\hat{X}_{ij} = b + b_i + b_j + U_i^T V_i$$

Where:

- •
- •
- •

The model is trained using only the observed entries \((i,j) \in \Omega\), minimizing the following loss:

Objective Function

$$\mathcal{L} = rac{1}{|\Omega|} \sum_{(i,j) \in \Omega} (X_{ij} \, - \hat{X}_{ij}\,)^2 + \lambda (\|U\|^2 + \|V\|^2)$$

Training Strategies

All methods inject Gaussian noise into gradient updates for robustness and generalization.

1. Vanilla Noisy SGD

Standard SGD with added Gaussian noise during updates:

```
python
grad_U += torch.normal(0, std_noise, grad_U.shape)
```

Updates:

$$egin{aligned} U_i \ \leftarrow U_i + lpha(
abla_{U_i} + ext{noise}) \ V_j \ \leftarrow V_j + lpha(
abla_{V_j} + ext{noise}) \end{aligned}$$

2. Noisy SGD with Momentum

Incorporates momentum to accelerate convergence by smoothing updates:

$$egin{aligned} \Delta U_i &= \gamma \Delta U_i + lpha (
abla_{U_i} + ext{noise}) \ \Delta V_j &= \gamma \Delta V_j + lpha (
abla_{V_j} + ext{noise}) \ U_i &\leftarrow U_i + \Delta U_i \ V_j &\leftarrow V_j + \Delta V_j \end{aligned}$$

3. Noisy SGD with EMA

Uses exponentially weighted moving average of gradients:

$$egin{aligned} EMA_{U_i} &= eta EMA_{U_i} + (1-eta)(
abla_{U_i} + ext{noise}) \ EMA_{V_j} &= eta EMA_{V_j} + (1-eta)(
abla_{V_j} + ext{noise}) \ U_i &\leftarrow U_i + EMA_{U_i} \ V_j &\leftarrow V_j + EMA_{V_i} \end{aligned}$$

Clipping is applied to both gradients and parameters for stability.

Experiments

1. Small Matrix (10×10 , Rank = 2)

Strategy	Final MSE	Notes
Vanilla	~0.03	Slow but consistent
Momentum	~0.012	Good acceleration
EMA	~0.003	Best early-stage fit

Observation: EMA provides the fastest error reduction for small problems, especially in early epochs.

2. Large Matrix (500 \times 500, Rank = 10)

Strategy	Final MSE	Notes
Vanilla	~0.040	Stable but slow
Momentum	~0.007	Best convergence and accuracy
EMA	~0.015	Good early drops, less stable

Observation: Momentum leads to the best convergence and lowest reconstruction error in large-scale settings.

Visual Results

Training loss and error curves were plotted to compare convergence rates. EMA drops fastest initially, while Momentum consistently outperforms in long-term convergence. All GPU-accelerated methods are significantly faster than CPU baselines.

Conclusion

This project demonstrates how GPU acceleration combined with noise-injected optimization strategies enhances matrix factorization. Key takeaways:

- EMA is ideal for small or rapidly changing data.
- Momentum delivers stable and strong performance on large matrices.
- Noisy SGD provides a simple and robust baseline.

Next Steps

- Benchmark on real-world data (e.g., MovieLens)
- Experiment with adaptive learning rates
- Integrate Adam/Adagrad for comparison
- Explore temperature scaling of noise over time

Let me know if you'd like this exported as a markdown file or need help generating plots for your report!

Fix the following md code to let github show it correctly.
Matrix Factorization with Noisy SGD and Momentum/EMA Enhancements

Motivation

Matrix completion is a core component of modern recommender systems. In real-world scenarios like Netflix or Amazon, the user-item rating matrix is highly sparse. To recover missing entries, low-rank matrix factorization is a widely adopted method.

In this report, we explore three strategies to improve the stochastic gradient descent (SGD) optimization used for matrix factorization:

- **Vanilla SGD**
- **SGD with Momentum**
- **SGD with Exponential Moving Average (EMA)**

We introduce noise into the updates (Noisy SGD) and analyze the effect of momentum and EMA on convergence and reconstruction accuracy.

Problem Setup

Given an incomplete matrix $\(X \in \mathbb{R}^{m \times n}\)$, we aim to find low-rank factors $\(U \in \mathbb{R}^{m \times k}\)$ and $\(V \in \mathbb{R}^{n \times k}\)$ such that:

\$\$
X_{ij} \\approx \\hat{X}_{ij} = b + b_i + b_j +
U_i^\\top V_j
\$\$

Where:

- \\(b\\) is the global bias
- \\(b_i\\), \\(b_j\\) are user/item biases
- \\(U_i\\), \\(V_j\\) are latent vectors

Only observed entries (non-missing values) in $\(X\)$ are used to optimize the loss function.

Objective Function

```
Minimize the mean squared error with \\
 (\\ell_2\\) regularization:
$$
\mathcal{L} = \frac{1}{|\Omega|}
\sum_{(i,j) \in \mathbb{Z}_{ij} -
\htext{X}_{ij})^2 + \beta (\|U\|^2 + \
\\|V\\|^2)
$$
## Training Strategies
### 1. Vanilla SGD
 Each iteration updates parameters using
the gradient of the observed error:
$$
\\begin{aligned}
 U_i \& \leq U_i + \alpha (2e_{ij} V_j - U_i) 
\\beta U_i) \\\\
V_j \& \leq V_j + \alpha (2e_{ij} U_i - V_j + \alpha (2e_{ij} U_i - V_j + V_i) 
\\beta V_j)
\\end{aligned}
$$
### 2. SGD with Momentum
 Momentum accumulates past gradients to
accelerate convergence:
$$
\\begin{aligned}
\\Delta U_i &= \\gamma \\Delta U_i +
\alpha (2e_{ij} V_j - \beta U_i) \
\Delta V_j = \Lambda V_j + \Lambda V_
\alpha (2e_{ij} U_i - \beta V_j)
\\end{aligned}
$$
Then:
$$
 U_i \\leftarrow U_i + \\Delta U_i \\quad
\\text{and} \\quad V_j \\leftarrow V_j +
\\Delta V_j
$$
### 3. SGD with EMA
Instead of accumulating momentum, we
use exponentially smoothed gradients:
```

```
\\begin{aligned}
EMA_{U_i} &\\leftarrow \\beta \\cdot
EMA_{U_i} + (1 - \beta) \cdot
\\text{grad}_{U_i} \\\\
EMA_{V_j} &\\leftarrow \\beta \\cdot
EMA_{V_j} + (1 - \beta) \cdot
\\text{grad}_{V_j}
\\end{aligned}
$$
Then update:
$$
U_i \\leftarrow U_i + EMA_{U_i}, \\quad V_j
\\leftarrow V_j + EMA_{V_j}
$$
To stabilize EMA updates, we apply
**gradient and parameter clipping**.
## Experiments
### 1. Small-Scale Evaluation (10x10,
Rank = 2
| Strategy | Final MSE | Notes
|-----|
| Vanilla | ~0.028 | Slow but steady |
| Momentum | ~0.011 | Fast
convergence
        | **~0.003** | Very quick early
| EMA
drop |
**Observation**: On smaller matrices,
EMA helps accelerate learning in the early
stages.
![Small Matrix Vanilla SGD Loss Curve]
(https://github.com/Lexaun-chen/STAT-
4830-Group-
Project/blob/main/tests/Basic%20SGD%2
0Error%20Small%20Scale.png.png?
raw=true)
![Small Matrix NQM Loss]
(https://github.com/Lexaun-chen/STAT-
4830-Group-
Project/blob/main/tests/MSE%20Small%2
0Scale.png.png?raw=true)
![Small Matrix NQM Error]
(https://github.com/Lexaun-chen/STAT-
4830-Group-
Project/blob/main/tests/Error%20Small%
20Scale.png.png?raw=true)
```

```
### 2. Large-Scale Evaluation (500x500,
Rank = 10
| Strategy | Final MSE | Notes
|-----|
| Vanilla | ~0.038 | Slower convergence
| Momentum | **~0.007** | Best
performance for large-scale
| EMA | ~0.015 | Stable after clipping,
but fluctuates |
**Observation**: For larger matrices,
**Momentum** outperforms others in
terms of convergence speed and final
accuracy.
![tests/Basic SGD Error Large
Scale.png.png]
(https://github.com/Lexaun-chen/STAT-
4830-Group-
Project/blob/main/tests/Basic%20SGD%2
0Error%20Large%20Scale.png.png?
raw=true)
![tests/MSE Large Scale.png.png]
(https://github.com/Lexaun-chen/STAT-
4830-Group-
Project/blob/main/tests/MSE%20Large%2
OScale.png.png?raw=true)
![tests/Error Large Scale.png.png]
(https://github.com/Lexaun-chen/STAT-
4830-Group-
Project/blob/main/tests/Error%20Large%
20Scale.png.png?raw=true)
### Training Curves Summary
- Momentum improves convergence
steadily over time.
- EMA shows fast early reduction but
stabilizes at a higher plateau.
- Vanilla remains the most conservative
strategy.
## Conclusion
We introduced noise-aware matrix
factorization methods with both
```

momentum and **EMA**

enhancement strategies. From empirical

results:

- EMA helps in **early training** or **small data**.
- Momentum offers better **long-term stability** and is more suited for **large-scale recommendation** tasks.

Next Steps

- Apply these methods to real recommendation datasets (e.g., MovieLens)
- Tune per-strategy hyperparameters for optimal performance
- Explore adaptive EMA schedules (warm-up or decay)

To ensure your markdown renders correctly on GitHub, especially for math formulas, you should use backticks with math code blocks for display math, or consider a workaround like using MathJax-compatible environments that GitHub renders via <div> tags.

GitHub does not natively support LaTeX inside \$\$...\$\$ or \(...\)
delimiters in .md files. But if you're targeting GitHub Pages with
MathJax enabled or a static site generator like Jupyter Book, those
delimiters work. Otherwise, formulas will not render correctly.

Here's the fixed version for GitHub compatibility while preserving mathematical clarity — this version wraps all LaTeX blocks inside <div align="center">...</div> and uses \\[\\\] and \\\(\\\) instead of \$\$ or inline \\\(...\).

GitHub-compatible fixes applied:

- Replaced \$\$...\$\$ with <div align="center">\\[... \\]</div>
- Replaced inline \\(...\\) with \\(...\\)
- Escaped underscores and pipe symbols in tables where needed
- Left image links and formatting unchanged (GitHub renders those well)

Here's an excerpt of how you should modify your file:

md ## Problem Setup

```
Given an incomplete matrix \(X \in \mathbb{R}^{m \times n}\), we aim to
```

```
<div align="center">\\[
X_{ij} \approx \hat{X}_{ij} = b + b_i + b_j + U_i^\\to V_j
\\]
```

Where:

- \\(b\\) is the global bias
- $\(b_i)$, $\(b_j)$ are user/item biases
- $\(U_i\),\ \(V_j\)$ are latent vectors

Only observed entries (non-missing values) in \\(X\\) are used to optimize

Objective Function

Minimize the mean squared error with \\(\ell_2\\) regularization:

Full File Ready?

Would you like me to output the full, fixed markdown file here, or prepare it as a downloadable .md file for GitHub upload?

Yes! output the full file

Sure! Here's the full GitHub-compatible markdown file with math properly formatted for best compatibility using <div align="center">\\[... \\]</div> for display math and \\(...\\) for inline math:

```
markdown
```

Matrix Factorization with Noisy SGD and Momentum/EMA Enhancement

Motivation

Matrix completion is a core component of modern recommender systems

In this report, we explore three strategies to improve the stochastic gradie

- **Vanilla SGD**
- **SGD with Momentum**
- **SGD with Exponential Moving Average (EMA)**

We introduce noise into the updates (Noisy SGD) and analyze the effect of

Problem Setup

Given an incomplete matrix $\(X \in \mathbb{R}^{m \times n}\)$, we aim to

```
<div align="center">\\[
```

```
X_{ij} \cdot Approx \cdot At_{X}_{ij} = b + b_i + b_j + U_i \cdot V_j
 \\]</div>
 Where:
 - \\(b\\) is the global bias
 - \(b_i\), \(b_j\) are user/item biases
- \\(U_i\\), \\(V_j\\) are latent vectors
 Only observed entries (non-missing values) in \\(X\\) are used to optimize
 ### Objective Function
 Minimize the mean squared error with \\(\ell_2\\) regularization:
 <div align="center">\\[
 \label{lambda} $$ \mathbf{L} = \frac{1}{|\Omega_{i,j} \times \{(i,j) \in \Omega_{i,j} - \frac{X_{i,j} - \frac{X_{i,j}}{2}}{n}} 
 \\]</div>
 ## Training Strategies
 ### 1. Vanilla SGD
 Each iteration updates parameters using the gradient of the observed err
 <div align="center">\\[
 \begin{aligned}
 U_i \& leftarrow U_i + lapha (2e_{ij} V_j - lapha 
 V_j &\leftarrow V_j + \alpha (2e_{ij}) U_i - \beta V_j
 \end{aligned}
 \\]</div>
 ### 2. SGD with Momentum
 Momentum accumulates past gradients to accelerate convergence:
 <div align="center">\\[
 \begin{aligned}
 \Delta U_i &= \gamma \Delta U_i + \alpha (2e_{ij} V_j - \beta U_i) \\\\\
 \Delta V_j \ = \gamma \Delta V_j + \alpha (2e_{ij} U_i - \beta V_j)
 \end{aligned}
 \\]</div>
 Then:
 <div align="center">\\[
 U_i \leftarrow U_i + \Delta U_i \quad \text{and} \quad V_j \leftarrow V_j + \Del
 \\]</div>
 ### 3. SGD with EMA
 Instead of accumulating momentum, we use exponentially smoothed grad
 <div align="center">\\[
 \begin{aligned}
 EMA_{U_i} & \left( 1 - \beta \right) \cdot \left(
 EMA_{V_j} & \left(1 - \beta \right) \cdot \left(1 
 \end{aligned}
 \\]</div>
 Then update:
 <div align="center">\\[
 U_i \setminus U_i + EMA_{U_i}, \quad V_j \setminus U_i + EMA_{V_j}
```

```
\\]</div>
To stabilize EMA updates, we apply **gradient and parameter clipping**
## Experiments
### 1. Small-Scale Evaluation (10x10, Rank = 2)
| Strategy | Final MSE | Notes
|-----
| Vanilla | ~0.028 | Slow but steady
| Momentum | ~0.011 | Fast convergence |
| EMA | **~0.003** | Very quick early drop |
**Observation**: On smaller matrices, EMA helps accelerate learning
![Small Matrix Vanilla SGD Loss Curve](https://github.com/Lexaun-chen/ST
![Small Matrix NQM Loss](https://github.com/Lexaun-chen/STAT-4830-Gro
![Small Matrix NQM Error](https://github.com/Lexaun-chen/STAT-4830-Gro
### 2. Large-Scale Evaluation (500x500, Rank = 10)
| Strategy | Final MSE | Notes
|-----|
| Vanilla | ~0.038 | Slower convergence
| Momentum | **~0.007** | Best performance for large-scale
| EMA | ~0.015 | Stable after clipping, but fluctuates |
**Observation**: For larger matrices, **Momentum** outperforms c
![Large Matrix Vanilla SGD Loss Curve](https://github.com/Lexaun-chen/ST
![Large Matrix NQM Loss](https://github.com/Lexaun-chen/STAT-4830-Gro
![Large Matrix NQM Error](https://github.com/Lexaun-chen/STAT-4830-Gro
### Training Curves Summary
- Momentum improves convergence steadily over time.
- EMA shows fast early reduction but stabilizes at a higher plateau.
- Vanilla remains the most conservative strategy.
## Conclusion
We introduced noise-aware matrix factorization methods with both **mo
- EMA helps in **early training** or **small data**.
- Momentum offers better **long-term stability** and is more suited for
## Next Steps
- Apply these methods to real recommendation datasets (e.g., MovieLens)
- Tune per-strategy hyperparameters for optimal performance
- Explore adaptive EMA schedules (warm-up or decay)
```

Notebook: [Colab Link](https://colab.research.google.com/drive/1AZ\

Let me know if you'd like me to export this into a .md file for download or directly commit it to a GitHub repository.

rewrite the whole code in a form that vscode can show correctly

To display your markdown correctly in VS Code, especially for LaTeX-style math, we should:

- **1.** Use \$...\$ for inline math (VS Code renders this well).
- **2.** Use \$\$...\$\$ for display math blocks.
- 3. Avoid GitHub-specific HTML tags like <div align="center">.
- **4.** Keep all standard markdown formatting.
- **5.** Ensure image links are plain and Markdown-compatible.

Here's your full markdown report rewritten for optimal rendering in VS Code markdown preview (and it will still work fine on GitHub if you use a math plugin like MathJax extension in VS Code):

markdown

Matrix Factorization with Noisy SGD and Momentum/EMA Enhancement

Motivation

Matrix completion is a core component of modern recommender systems

In this report, we explore three strategies to improve the stochastic gradie

- **Vanilla SGD**
- **SGD with Momentum**
- **SGD with Exponential Moving Average (EMA)**

We introduce noise into the updates (Noisy SGD) and analyze the effect of

Problem Setup

Given an incomplete matrix \$X \in \mathbb{R}^{m \times n}\$, we aim to fir

\$\$

```
X_{ij} \cdot approx \cdot hat_{X}_{ij} = b + b_i + b_j + U_i \cdot top V_j
```

Where:

- \$b\$ is the global bias
- \$b_i\$, \$b_j\$ are user/item biases
- \$U_*i\$,* \$*V*_*j*\$ are latent vectors

Only observed entries (non-missing values) in \$X\$ are used to optimize th

```
### Objective Function
 Minimize the mean squared error with $\ell_2$ regularization:
 \mathcal{L} = \frac{1}{|\Omega_{i,j} \in \Omega_{i,j} \cdot \Omega_{i,j}} \cdot \mathcal{L}_{i,j} = \frac{1}{|\Omega_{i,j} \cap \Omega_{i,j} \cdot \Omega_{i,j}} \cdot \mathcal{L}_{i,j} 
 ## Training Strategies
 ### 1. Vanilla SGD
 Each iteration updates parameters using the gradient of the observed err
 $$
 \begin{aligned}
 U_i \& leftarrow U_i + \alpha (2e_{ij} V_j - \beta U_i) \
 V_j &\leftarrow V_j + \alpha (2e_{ij}) U_i - \beta V_j
 \end{aligned}
 $$
 ### 2. SGD with Momentum
 Momentum accumulates past gradients to accelerate convergence:
 $$
 \begin{aligned}
 \Delta U_i &= \gamma \Delta U_i + \alpha (2e_{ij} V_j - \beta U_i) \\
 \Delta V_j \&= \gamma V_i + \alpha V_j + \alpha V_j
 \end{aligned}
 $$
 Then:
 $$
 U_i \leftarrow U_i + \Delta U_i, \quad V_j \leftarrow V_j + \Delta V_j
 ### 3. SGD with EMA
 Instead of accumulating momentum, we use exponentially smoothed grad
 $$
 \begin{aligned}
 EMA_{U_i} & \left( 1 - \beta \right) \cdot \left(
 EMA_{V_j} & \left(1 - \beta\right) \cdot \left(1 
 \end{aligned}
 $$
 Then update:
 $$
 U_i \leftarrow U_i + EMA_{U_i}, \quad V_j \leftarrow V_j + EMA_{V_j}
 $$
To stabilize EMA updates, we apply **gradient and parameter clipping**
 ## Experiments
 ### 1. Small-Scale Evaluation (10x10, Rank = 2)
   | Strategy | Final MSE | Notes
    |-----|
```

```
| Vanilla | ~0.028 | Slow but steady
| Momentum | ~0.011 | Fast convergence
| EMA | **~0.003** | Very quick early drop |
**Observation**: On smaller matrices, EMA helps accelerate learning
![Small Matrix Vanilla SGD Loss Curve](https://github.com/Lexaun-chen/ST
![Small Matrix NQM Loss](https://github.com/Lexaun-chen/STAT-4830-Gro
![Small Matrix NQM Error](https://github.com/Lexaun-chen/STAT-4830-Gro
### 2. Large-Scale Evaluation (500x500, Rank = 10)
| Strategy | Final MSE | Notes
|-----|
| Vanilla | ~0.038 | Slower convergence
| Momentum | **~0.007** | Best performance for large-scale
| EMA | ~0.015 | Stable after clipping, but fluctuates |
**Observation**: For larger matrices, **Momentum** outperforms of
![Large Matrix Vanilla SGD Loss Curve](https://github.com/Lexaun-chen/ST
![Large Matrix NQM Loss](https://github.com/Lexaun-chen/STAT-4830-Gro
![Large Matrix NQM Error](https://github.com/Lexaun-chen/STAT-4830-Gro
### Training Curves Summary
- Momentum improves convergence steadily over time.
- EMA shows fast early reduction but stabilizes at a higher plateau.
- Vanilla remains the most conservative strategy.
## Conclusion
We introduced noise-aware matrix factorization methods with both **mo
- EMA helps in **early training** or **small data**.
- Momentum offers better **long-term stability** and is more suited for
## Next Steps
- Apply these methods to real recommendation datasets (e.g., MovieLens)
- Tune per-strategy hyperparameters for optimal performance
- Explore adaptive EMA schedules (warm-up or decay)
**Notebook**: [Colab Link](https://colab.research.google.com/drive/1AZ\
```