HDDA Recommendation systems via approximate matrix factorization

Dmitry Beresnev
Vsevolod Klyushev

Innopolis University — IU — 2024

Initial problem

We need to solve the following problem:

$$\min_{U \in \mathbb{R}^{m \times r}, V \in \mathbb{R}^{r \times n}} \|W \circ (X - UV)\|_F^2 \tag{1}$$

where:

- X — target matrix ($m \times n$);
- W — binary mask;
- r — rank of factorization;

Gradients

In order to use gradient methods, we need to derive it with respect to each parameter (U and V) (full derivation might be found here):

$$\frac{\partial \|W \circ (X - UV)\|_F^2}{\partial U} = -2(W \circ X)V^T + 2(W \circ (UV))V^T \tag{2}$$

$$\frac{\partial \|W \circ (X - UV)\|_F^2}{\partial V} = -2U^T(W \circ X) + 2U^T(W \circ (UV)) \tag{3}$$

Improved problem

As a modified problem, we decided just to add some regularization. This brings us to the following forms:

$$\min_{U \in \mathbb{R}^{m*r}, V \in \mathbb{R}^{r*n}} \|W \circ (X - UV)\|_F^2 + \lambda \|U\|_F^2 + \lambda \|V\|_F^2 \tag{4}$$

$$\frac{\partial(\|W \circ (X - UV)\|_F^2 + \lambda \|U\|_F^2 + \lambda \|V\|_F^2)}{\partial U} = -2(W \circ X)V^T + 2(W \circ (UV))V^T + 2\lambda U \tag{5}$$

$$\frac{\partial(\|W \circ (X - UV)\|_F^2 + \lambda \|U\|_F^2 + \lambda \|V\|_F^2)}{\partial V} = -2U^T(W \circ X) + 2U^T(W \circ (UV)) + 2\lambda V \tag{6}$$

where λ — regularization parameter

Constant + decreasing

We can take constant step, or such step, that would decrease with increase of iterations. For example, $\gamma = \frac{1}{\sqrt{k}}$, where k — number of iterations.

Estimate $1/L$

Algorithm 1 Estimate $1/L$

Input: θ (point), $\nabla f(\theta)$ (gradient function), p (desired direction), β (multiplier), t (max iterations)

```
α ← 1
for i = 1 to t do
    θ_i ← θ + αp
    if ‖∇f(θ) − ∇f(θ_i)‖ > (1/α)‖θ − θ_i‖ then
        α ← β · α
        if α < ϵ then
            return α
        end if
    else
        return α
    end if
end for
return α
```

Armijo

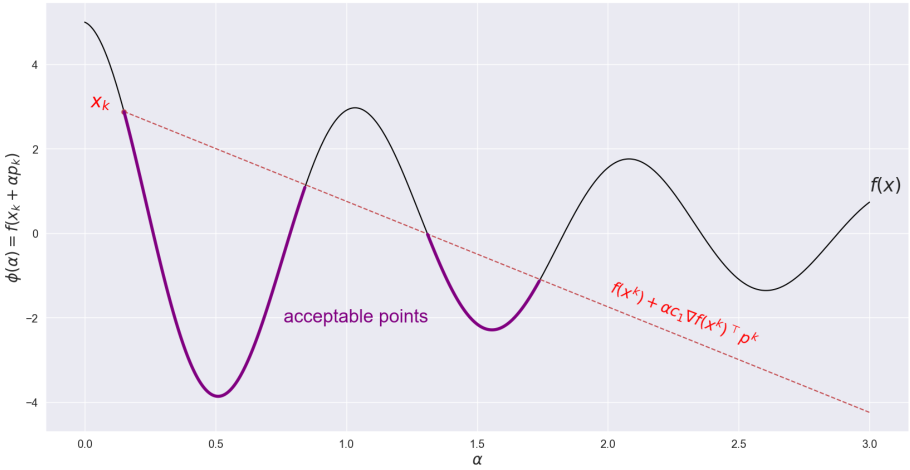

Figure: Armijo rule

Armijo

Algorithm 2 Armijo Step

Input: θ (point), $f(\theta)$ (objective function), p (desired direction), β (multiplier), $c_1 > 0$, t (max iterations)

$\alpha \leftarrow 1$
for $i = 1$ **to** t **do**
    $\theta_i \leftarrow \theta + \alpha p$
    if $f(\theta_i) > f(\theta) + c_1\alpha\langle\nabla_\theta f(\theta), p\rangle$ **then**
        $\alpha \leftarrow \beta \cdot \alpha$
        if $\alpha < \epsilon$ **then**
            return α
        end if
    else
        return α
    end if
end for
return α

Curvature condition

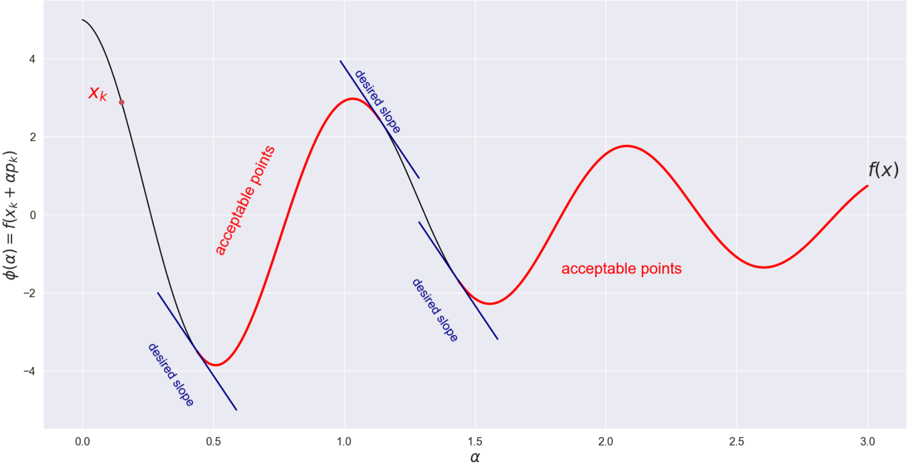

Figure: Curvature condition

Bisection Weak Wolfe

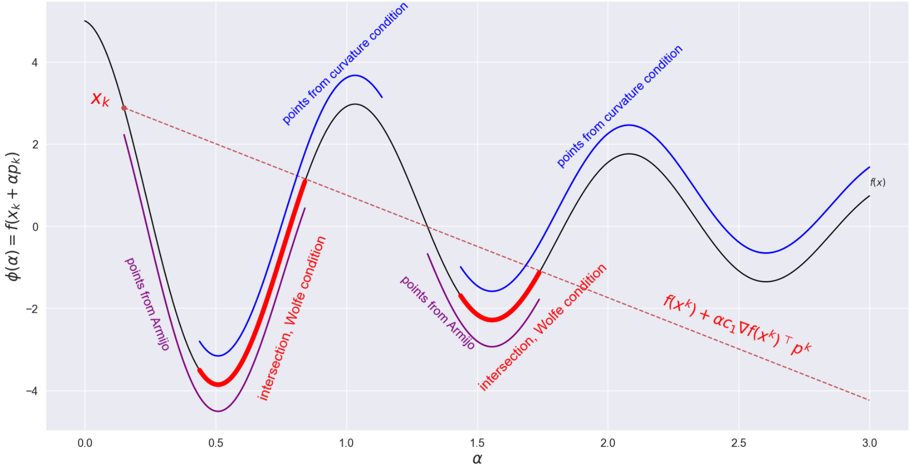

Figure: Weak Wolfe conditions

Bisection Weak Wolfe

Algorithm 3 Bisection Weak Wolfe Step

Input: θ (point), $f(\theta)$ (objective function), p (desired direction), β (multiplier), $c_1 > 0$, $c_2 > c_1$, t (max iterations)

```
α ← 1
a ← 0                                                        ▷ Lower bound
b ← +∞                                                       ▷ Upper bound
for i = 1 to t do
    θ_i ← θ + αp
    if f(θ_i) > f(θ) + c_1 α ⟨∇_θ f(θ), p⟩ then              ▷ Armijo condition
        b ← α
        α ← (1/2)(a + b)                                     ▷ Decrease step
    else if ⟨p, ∇_θ f(θ_i)⟩ < c_2 ⟨p, ∇_θ f(θ)⟩ then          ▷ Curvature condition
        a ← α
        if b = +∞ then
            α ← 2a
        else
            α ← (1/2)(a + b)                                 ▷ Increase step
        end if
    else
        return α
    end if
end for
return α
```

Strong Wolfe

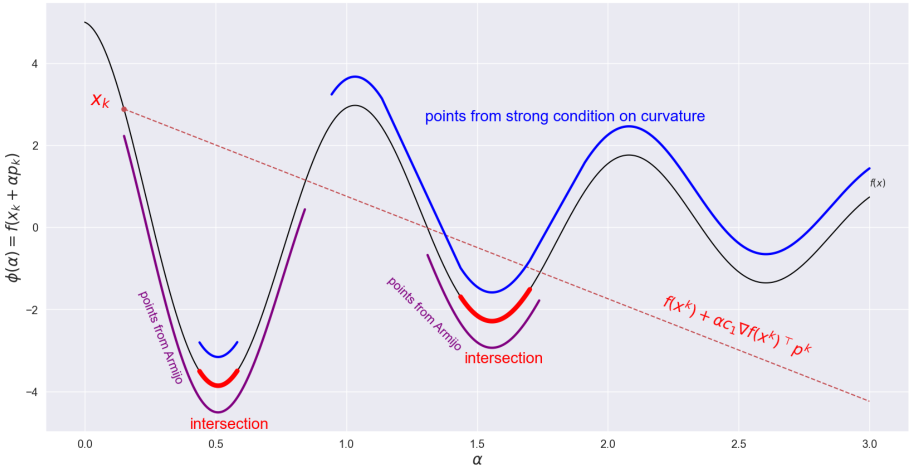

Figure: Strong Wolfe conditions

Strong Wolfe

Algorithm 4 Strong Wolfe Step

Input: θ (point), $f(\theta)$ (objective function), p (desired direction), β (multiplier), $c_1 > 0$, $c_2 > c_1$, t (max iterations)

$\alpha \leftarrow 1$
for $i = 1$ **to** t **do**
  $\theta_i \leftarrow \theta + \alpha p$
  if $\left(f(\theta_i) > f(\theta) + c_1\alpha\langle\nabla_\theta f(\theta), p\rangle\right)$ **or** $\left(\|\langle p, \nabla_\theta f(\theta_i)\rangle\| > c_2\|\langle p, \nabla_\theta f(\theta)\rangle\|\right)$ **then**
    $\alpha \leftarrow \beta \cdot \alpha$
    if $\alpha < \epsilon$ **then**
      return α
    end if
  else
    return α
  end if
end for
return α

Gradient Descent

Algorithm 5 Gradient Descent optimizer

Input: θ_0 (parameters to optimize), $f(\theta)$ (objective function), $\mathcal{L}(p)$ (step size choosing strategy)

for $t = 1$ **to** $\ldots$ **do**

$p_t \leftarrow -\nabla_\theta f_t(\theta_{t-1})$ ▷ Step direction

Choose step size γ according to $\mathcal{L}(p_t)$

$\theta_t \leftarrow \theta_{t-1} + \gamma p_t$

end for

return θ_t

Adaptive Gradient Descent

Algorithm 6 Adaptive Gradient Descent

Input: x_0 (parameters to optimize), $f(x)$ (objective function)
Initialize: $\lambda_0 > 0$ (small start step), $\theta_0 \leftarrow +\infty$

$x_1 \leftarrow x_0 - \lambda_0 \nabla f(x_0)$
for $t = 1$ **to** $\dots$ **do**
  $\lambda_t = \min\left\{ \sqrt{1 + \theta_{t-1}}\lambda_{t-1}, \frac{\|x_t - x_{t-1}\|}{2\|\nabla f(x_t) - \nabla f(x_{t-1})\|} \right\}$
  $x_{t+1} \leftarrow x_t - \lambda_t \nabla f(x_t)$
  $\theta_t \leftarrow \frac{\lambda_t}{\lambda_{t-1}}$
end for
return x_t

Heavy Ball

Algorithm 7 Heavy Ball optimizer

Input: θ_0 (parameters to optimize), $f(\theta)$ (objective function), β (momentum), $\mathcal{L}(p)$ (step size choosing strategy)

for $t = 1$ **to** $\dots$ **do**

  $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$

  if $\beta \neq 0$ **then**

    if $t > 1$ **then**

      $b_t \leftarrow \beta b_{t-1} + g_t$

    else

      $b_t \leftarrow g_t$

    end if

    $g_t \leftarrow b_t$

  end if

  $p_t \leftarrow -g_t$ ▷ Step direction

  Choose step size γ according to $\mathcal{L}(p_t)$

  $\theta_t \leftarrow \theta_{t-1} + \gamma p_t$

end for

return θ_t

Nesterov

Algorithm 8 Nesterov optimizer

Input: θ_0 (parameters to optimize), $f(\theta)$ (objective function), β (momentum), $\mathcal{L}(p)$ (step size choosing strategy)

for $t = 1$ **to** $\dots$ **do**
  $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$
  if $\beta \neq 0$ **then**
    if $t > 1$ **then**
      $b_t \leftarrow \beta b_{t-1} + g_t$
    else
      $b_t \leftarrow g_t$
    end if
    $g_t \leftarrow g_t + \beta b_t$
  end if
  $p_t \leftarrow -g_t$ ▷ Step direction
  Choose step size γ according to $\mathcal{L}(p_t)$
  $\theta_t \leftarrow \theta_{t-1} + \gamma p_t$
end for
return θ_t

AdaGrad

Algorithm 9 AdaGrad optimizer

Input: θ_0 (parameters to optimize), $f(\theta)$ (objective function), $\mathcal{L}(p)$ (step size choosing strategy)

Initialize: $s_0 \leftarrow 0$ (cumulative square sum)

for $t = 1$ **to** $\ldots$ **do**

$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$

$s_t \leftarrow \alpha s_{t-1} + g_t^2$

$p_t \leftarrow -g_t/(\sqrt{s_t} + \epsilon)$ ▷ Step direction

Choose step size γ according to $\mathcal{L}(p_t)$

$\theta_t \leftarrow \theta_{t-1} + \gamma p_t$

end for

return θ_t

RMSProp

Algorithm 10 RMSProp optimizer

Input: θ_0 (parameters to optimize), $f(\theta)$ (objective function), α (alpha), $\mathcal{L}(p)$ (step size choosing strategy)

Initialize: $v_0 \leftarrow 0$ (square average)

for $t = 1$ **to** $\ldots$ **do**

$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$

$v_t \leftarrow \alpha v_{t-1} + (1-\alpha) g_t^2$

$p_t \leftarrow -g_t / (\sqrt{v_t} + \epsilon)$ ▷ Step direction

Choose step size γ according to $\mathcal{L}(p_t)$

$\theta_t \leftarrow \theta_{t-1} + \gamma p_t$

end for

return θ_t

Adam

Algorithm 11 Adam optimizer

Input: θ_0 (parameters to optimize), $f(\theta)$ (objective function), β_1, β_2 (alpha), $\mathcal{L}(p)$ (step size choosing strategy)

Initialize: $m_0 \leftarrow 0$ (first moment), $v_0 \leftarrow 0$ (second moment)

for $t = 1$ **to** $\ldots$ **do**

$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$

$m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$

$v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$

$\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$

$\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$

$p_t \leftarrow -\hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$ ▷ Step direction

Choose step size γ according to $\mathcal{L}(p_t)$

$\theta_t \leftarrow \theta_{t-1} + \gamma p_t$

end for

return θ_t

BFGS

Algorithm 12 BFGS optimizer

Input: θ_0 (parameters to optimize), $f(\theta)$ (objective function), $\mathcal{L}(p)$ (step size choosing strategy)

Initialize: $H_0 \leftarrow I$

for $t = 1$ **to** $\dots$ **do**

$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$

$p_t \leftarrow -H_{k-1} g_t$ ▷ Step direction

Choose step size γ according to $\mathcal{L}(p_t)$ ▷ γ should satisfy Wolfe conditions

$s_t \leftarrow \gamma p_t$

$\theta_t \leftarrow \theta_{t-1} + s_t$

$y_t \leftarrow \nabla_\theta f_t(\theta_t) - g_t$

$H_t \leftarrow H_{t-1} + \frac{(s_t^T y_t + y_t^T H_{t-1} y_t)(s_t s_t^T)}{(s_t^T y_t)^2} - \frac{H_{t-1} y_t s_t^T + s_t y_t^T H_{t-1}}{s_t^T y_t}$

end for

return θ_t

Experiments. Gradient descent

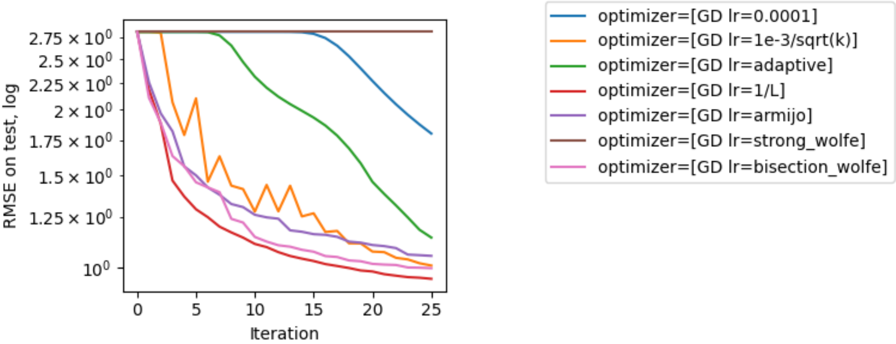

Figure: Gradient descent

Experiments. Heavy Ball

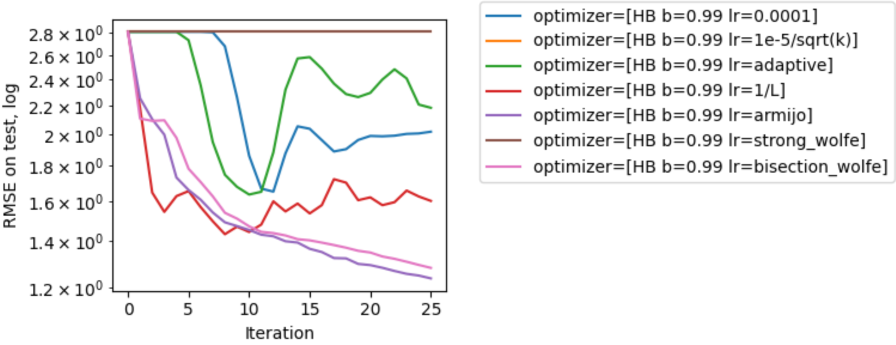

Figure: Heavy Ball

Experiments. Nesterov

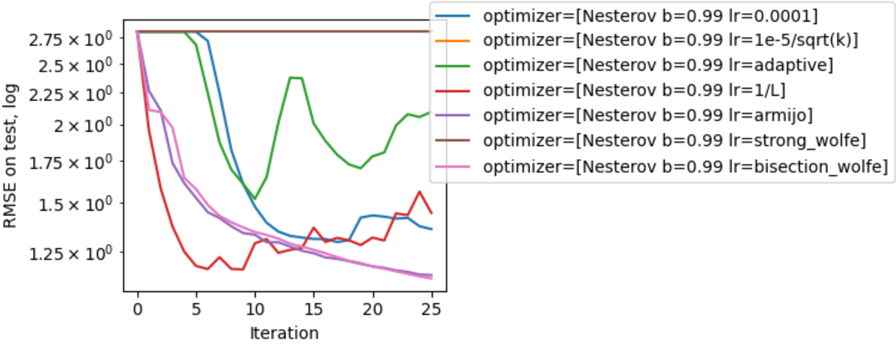

Figure: Nesterov

Experiments. RMSprop

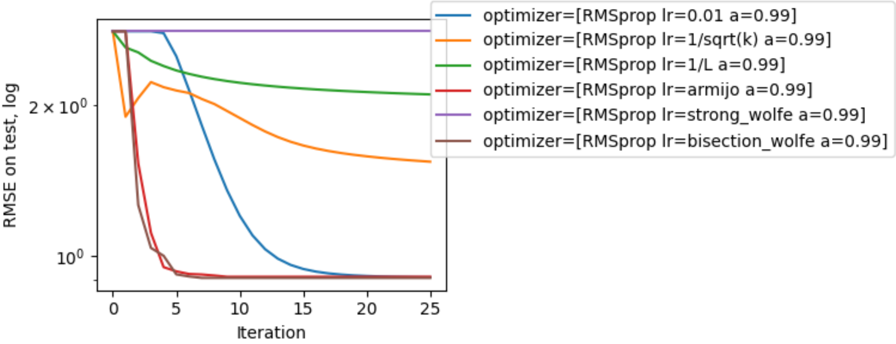

Figure: RMSprop

Experiments. AdaGrad

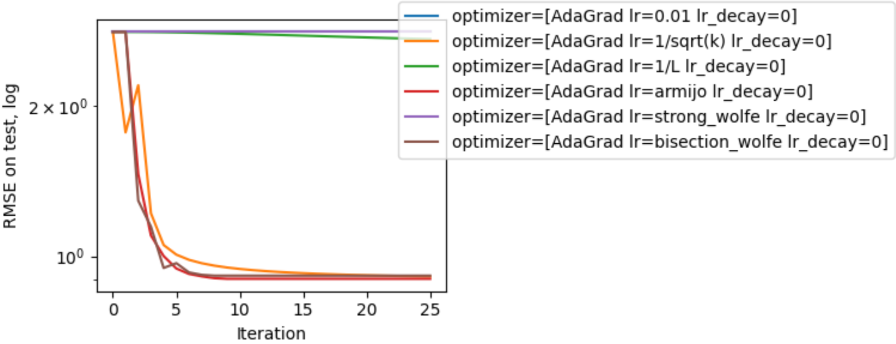

Figure: AdaGrad

Experiments. Adam

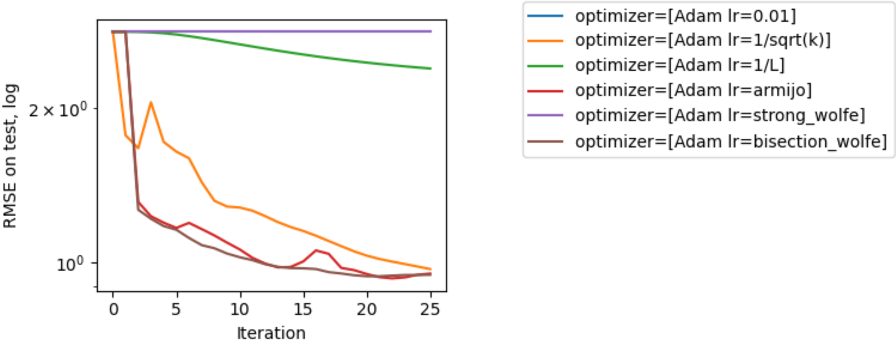

Figure: Adam

Experiments. Comparison

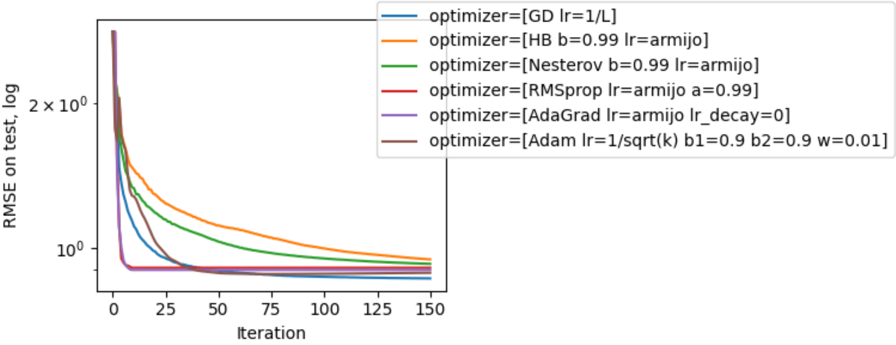

Figure: Comparison

Experiments. Best model

Our best model with RMSE score 0.86 have the following parameters: $r = 10$, GD optimizer with estimate 1/L strategy and $\lambda = 2$

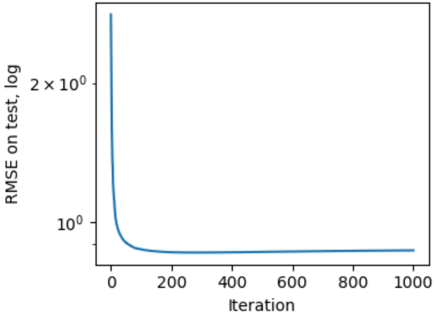

Figure: Best model

Main idea

Instead of making update for entire U or V simultaneously, we can make updates row by row (column by column for V). The reasons are the following:

- The objective function becomes $f : \mathbb{R}^d \to \mathbb{R}$, so we can apply methods like BFGS
- There will be more updates, and such updates will be more diverse: we will use just updated values for new updates

Problem formulation

Therefore, the new problem with fixed V becomes

$$\min_{U_i^\top \in \mathbb{R}^r} \|W_i^\top \circ (X_i^\top - U_i^\top V)\|^2 + \lambda \|U_i^\top\|^2, \ \forall i \in \{1, 2, \dots, m\} \tag{7}$$

and the new problem with fixed U:

$$\min_{V_j \in \mathbb{R}^r} \|W_j \circ (X_j - UV_j)\|^2 + \lambda \|V_j\|^2, \ \forall j \in \{1, 2, \dots, n\} \tag{8}$$

Gradients

Gradient of **??** is

$$\begin{aligned} &\frac{\partial(\|W_i^\top \circ (X_i^\top - U_i^\top V)\|^2 + \lambda \|U_i^\top\|^2)}{\partial U_i^\top} = \\ &= -2(W_i^\top \circ X_i^\top)V^T + 2(W_i^\top \circ (U_i^\top V))V^T + 2\lambda U_i^\top \end{aligned} \tag{9}$$

and the gradient of **??**:

$$\begin{aligned} &\frac{\partial(\|W_j \circ (X_j - UV_j)\|^2 + \lambda \|V_j\|^2)}{\partial V_j} = \\ &= -2U^T(W_j \circ X_j) + 2U^T(W_j \circ (UV_j)) + 2\lambda V_j \end{aligned} \tag{10}$$

Algorithm

Algorithm 13 Vector Gradient Descent

Input: $X, W \in \mathbb{R}^{m \times n}$ — given initial and binary matrices, $U \in \mathbb{R}^{m \times r}, V \in \mathbb{R}^{r \times n}$ — arbitrary matrices, k_U, k_V — small integers such that $\frac{m}{n} \approx \frac{k_U}{k_V}$

$d \leftarrow k_U + k_V$
repeat
  for $t = 0$ **to** $n + m - 1$ **do**
    $r \leftarrow t$ **mod** d
    if $r < k_U$ **then**
      $i \leftarrow k_U \cdot (t$ **div** $d) + r + 1$
      if $i > m$ **then**
        continue
      end if
      Update $U_i^\top$ using **??**
    else

Algorithm Vector Gradient Descent. Continue

```
...
repeat
    for t = 0 to n + m − 1 do
        ...
        if i > m then
            ...
        else
            j ← k_V · (t div d) + r − k_U + 1
            if j > n then
                continue
            end if
            Update V_j using ??
        end if
    end for
until convergence

return U, V =0
```

Experiments. Results

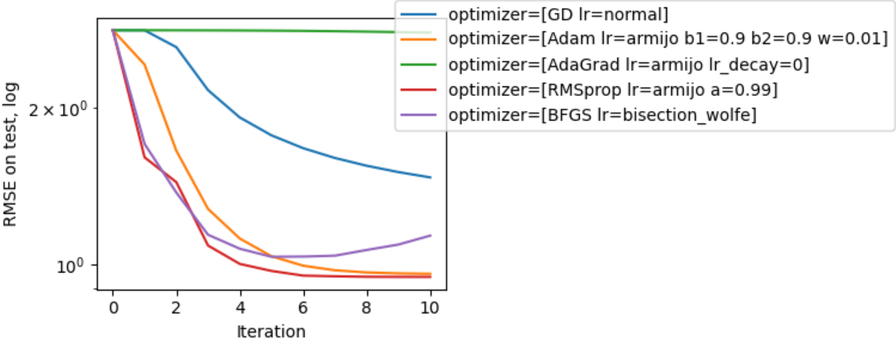

Figure: Vector Gradient Descent

NNMF. Problem formulation

We want to solve initial problem (**??**), but with non-negativity constraints:

$$\min_{U \in \mathbb{R}^{m \times r}, V \in \mathbb{R}^{r \times n}} \|W \circ (X - UV)\|_F^2 \tag{11}$$

s.t. $U, V \geq 0$
In order to solve such problem more easily, we need to derive multiplicative updates for U and V.

MU updates

With the idea from Lee and Seung paper we defined our MU updates in such way:

$$U \leftarrow U \circ \frac{((W \circ X)V^T)}{((W \circ (UV))V^T + \epsilon)} \tag{12}$$

$$V \leftarrow V \circ \frac{(U^T(W \circ X))}{(U^T(W \circ (UV)) + \epsilon)} \tag{13}$$

Experiments. Results

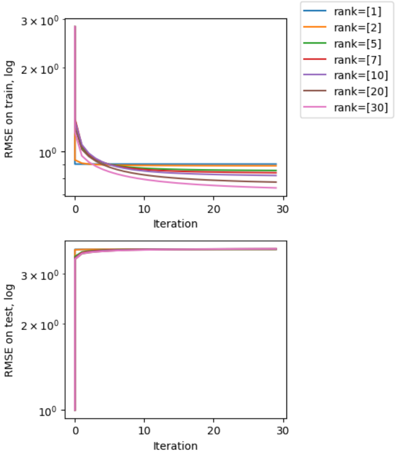

Figure: NNMF

Neural Network. Setup

As an alternative approach we decided to train simple neural network with such parameters:

- 4 layers $(23 \times 64, 64 \times 128, 128 \times 64, 64 \times 1)$
- ReLU as activation function after each layer
- MSELoss criterion
- Adam optimizer with $\alpha = 0.001$
- Early stopping (if on test set loss is not decreasing for 3 iterations)
- Maximum number of training epochs $= 50$

Input/Output

As input to our model we decided to use:

- Min-max scaled ‘user_id’
- Min-max scaled ‘movie_id’
- One-hot encoded ‘genres’ (we have 18 unique genres)
- Binary encoded ‘gender’
- Min-max scaled first two user features (‘feature_1’ and ‘feature_2‘)

In output we have just one number - rating for specific pair of user and film.
As a result we achieved average loss on test set 1.1

Overall Results

We tested several approaches: Gradient descent, Vector Gradient descent, Non-Negative Matrix Factorization and Neural Network.

- Both Neural Network and Vector Gradient descent showed their applicability for recommendation system task, however, they have been outperformed by other models
- NNMF is inapplicable for recommendation system task mostly because of the mask in the problem formulation. Mask makes NNMF to set almost all unknown values to 1, which is very bad decision
- It happens to be that Gradient descent method showed the best overall performance (both in terms of time and score). We were able to achieved RMSE score 0.86 on test data

Conclusions

Thanks!