Collaborative Filtering

ECON 4810

General Procedure to Model Fitting

1. Hyperparameters
2. Initialize parameters
3. Main loop
   a. Fit parameters
   b. Update loss

SVD with Alternating Least Squares

1. Hyperparameters
2. Initialize parameters

```
k = No. of latent factors
```

These parts change when the <u>*model*</u> change

P = random(n,k)
Q^T = random(k,i)
$\hat{X} = PQ^T$
loss_prev = mean$\left[(X - \hat{X})^2\right]$

3. Main loop
   a. Fit parameters with least squares
   b. Update loss

```
while loss_delta is not zero:
```

$Q^T = (P^T P)^{-1} P^T X$
$P = XQ(Q^T Q)^{-1}$
$\hat{X} = PQ^T$

This part changes when the *method of fitting* the model change

loss = mean$\left[(X - \hat{X})^2\right]$
loss_delta = loss - loss_prev
loss_prev = loss

SVD with Alternating Least Squares

- SVD with ALS is very fast, typically converge within a dozen epochs.
- The use of least squares means the process is easily parallelized, so scaling up is not a problem
- ALS might not be feasible if you want $X \neq PQ^T$.

Gradient Descent

- Gradient descent is a very general optimization algorithm, usable with pretty much all models.
- It uses the <u>first derivative</u> of the loss function with respect to a parameter to adjust the parameter.

Gradient Descent

- Gradient descent uses the first derivative of the loss function with respect to a parameter to adjust the parameter.
- E.g. suppose our model is

$$\hat{y} = \alpha + x$$

This is a (simplified) regression task, which we can solve by minimizing the squared error:

$$c = (y - \hat{y})^2$$

- For illustration purpose, let us assume the data is generated by $y = 5 + x$, so that if $x = 1, y = 6$.

Gradient Descent

Error $\epsilon \quad = y - \hat{y}$

Loss $\quad = \epsilon^2$

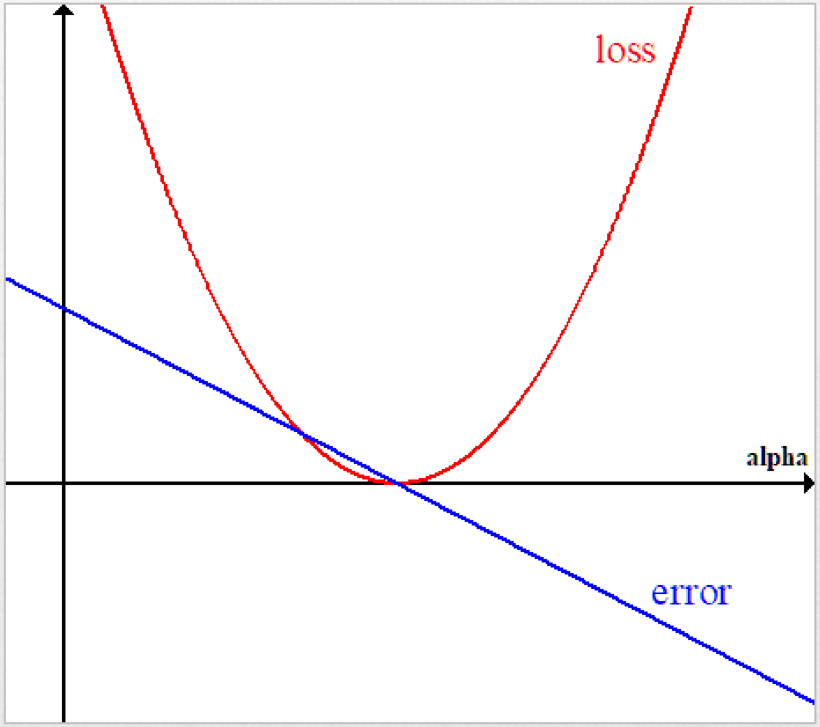
First derivative of loss function:

$$\frac{d}{d\alpha}\epsilon^2$$

$$= 2\epsilon \frac{d}{d\alpha}[y - (\alpha + x)]$$

$$= -2\epsilon$$

This is the marginal effect, or **gradient**, of α on loss.



Gradient Descent

We have an initial guess of what α is (usually just a random number.)

This gives us an initial prediction $\hat{y}$ and corresponding error and loss.

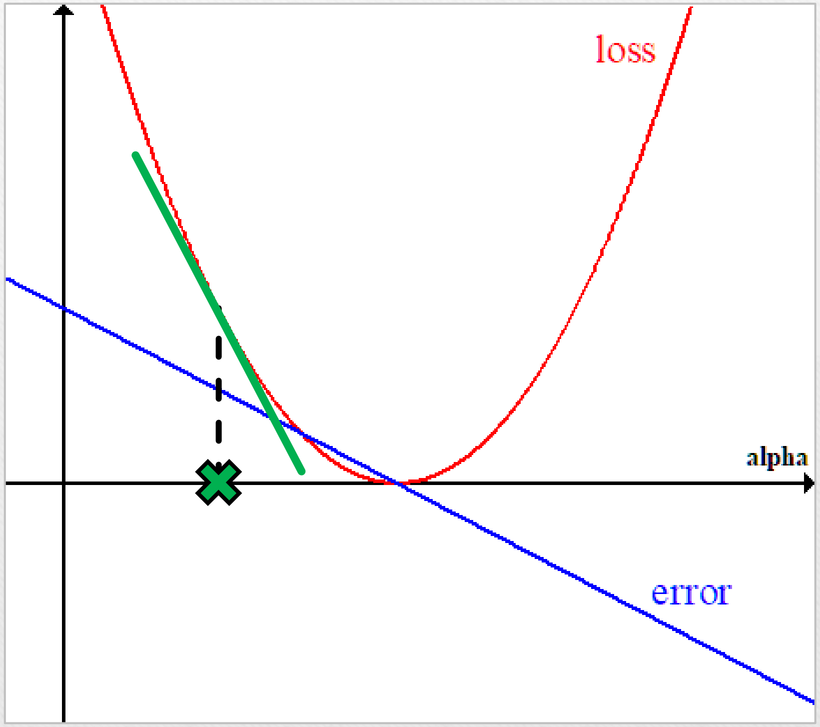
e.g.

$\hat{\alpha}_0 = 2$

$\hat{y}_0 = 2 + 1 = 3$

$\epsilon_0 = y - \hat{y} = 6 - 3 = 3$

$\frac{d}{d\alpha}\epsilon^2 = -2\epsilon = -6$



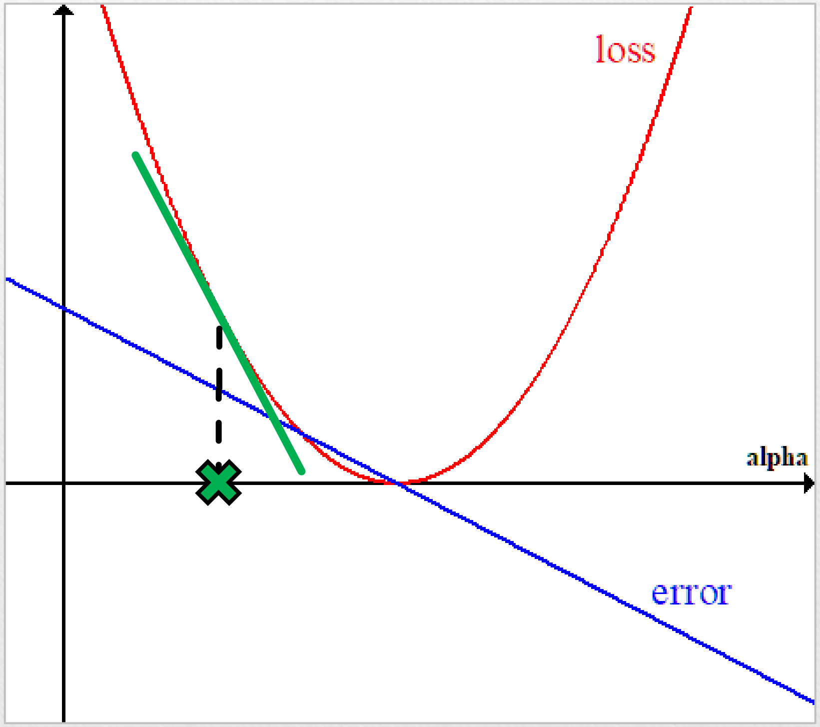
Gradient Descent

$\epsilon_0 = 3$

$\frac{d}{d\alpha}\epsilon^2 = -6 > 0$

In our example, ϵ is positive, so the gradient is negative.

This means loss is decreasing in alpha.

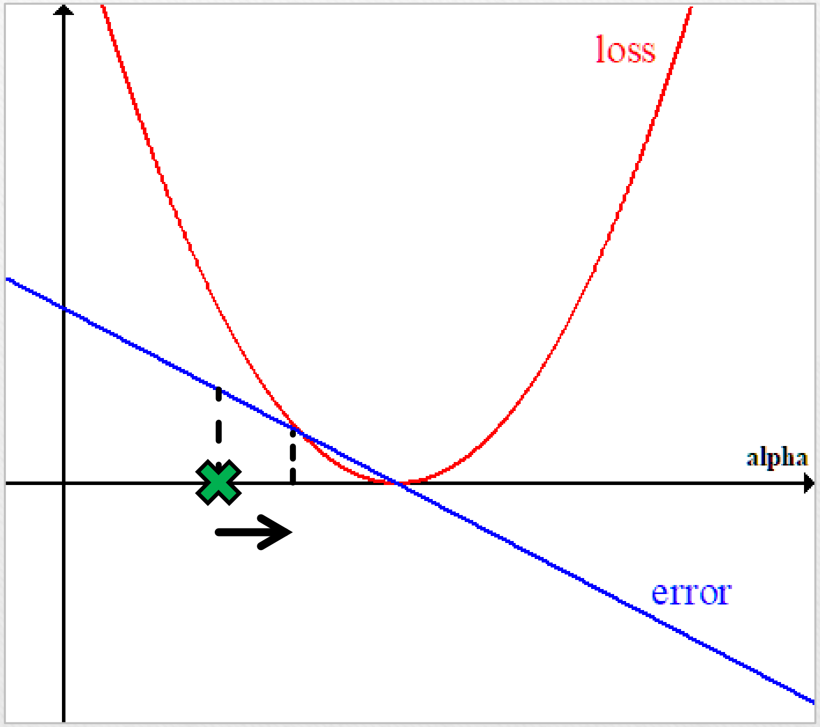


Gradient Descent

This makes sense, because if

$$y > \hat{y} = \alpha + x$$

increasing α will bring $\hat{y}$ closer to y.

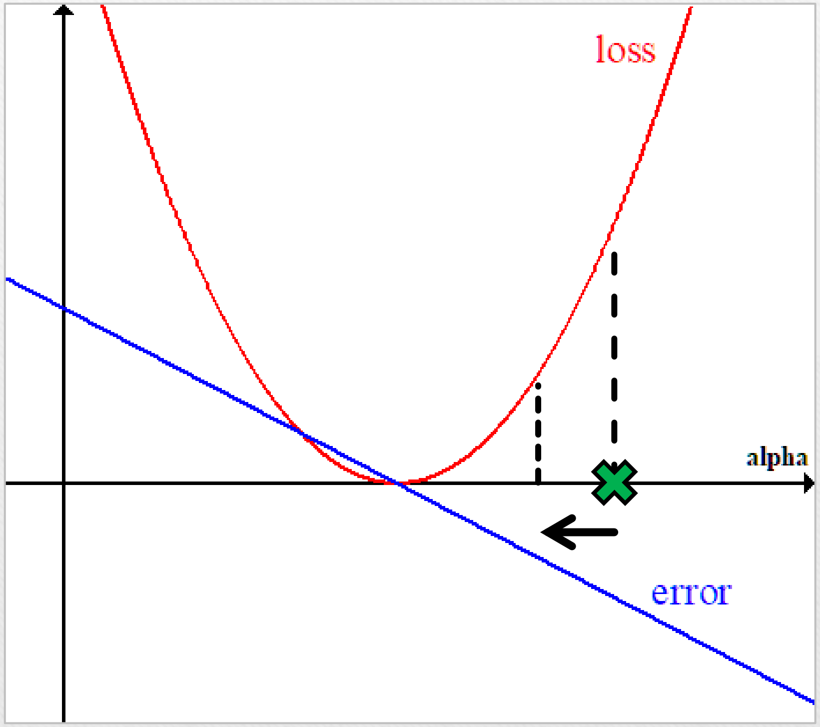


Gradient Descent

Conversely, if $\epsilon < 0$, then

$$y < \hat{y} = \alpha + x$$

decreasing α will bring $\hat{y}$ closer to y.



Gradient Descent

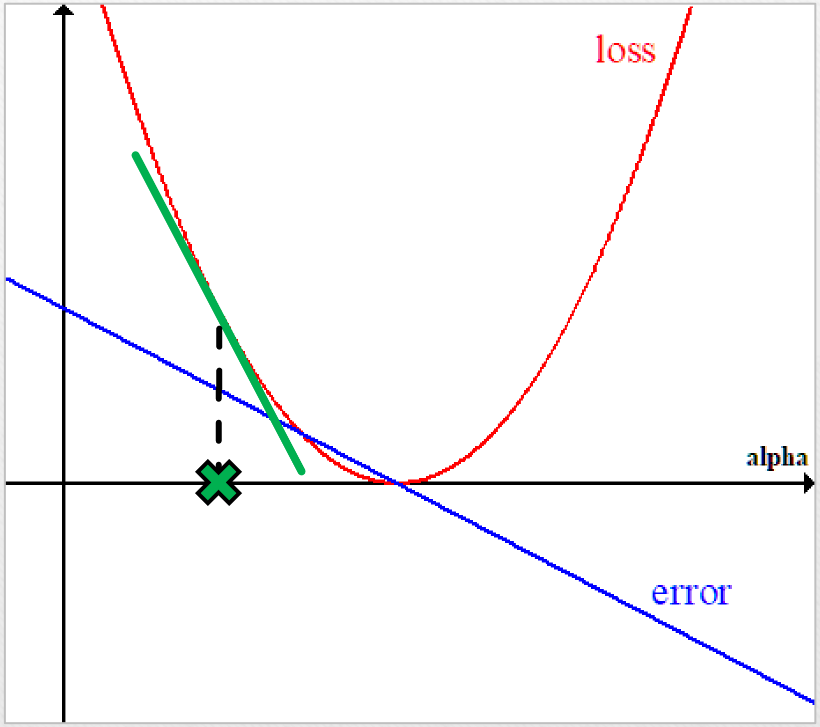
The gradient tells us the direction we need to adjustment our parameter.

The amount we need to adjust is, to a first-order approximation, inversely proportional to the gradient.

E.g.

$$\frac{d}{d\alpha}\epsilon^2 = -6$$

α needs to be bigger.

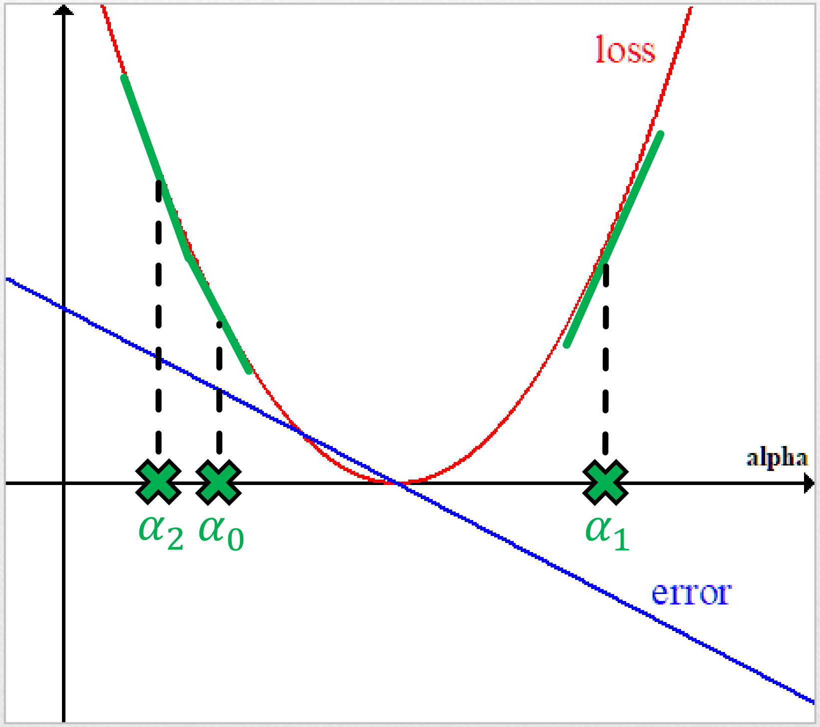


Gradient Descent

If we adjust the parameter by the exact amount of the gradient, we might overshoot:

$$\hat{\alpha}_1 = \hat{\alpha}_0 - \frac{d}{d\alpha}\epsilon^2$$
$$= 2 - (-6)$$
$$= 8 > 5 = \text{true } \alpha$$

Overshooting could result in our parameters bouncing around the true value, never to converge.



Gradient Descent

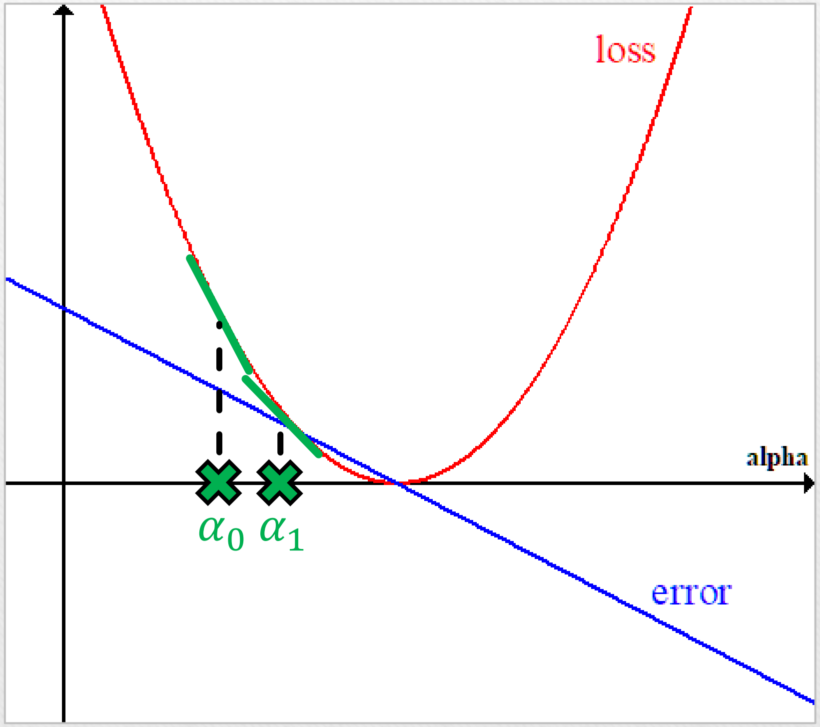
To prevent overshooting, we moderate the gradient by a *learning rate* γ:

$$\hat{\alpha}_1 = \hat{\alpha}_0 - \gamma \frac{d}{d\alpha} \epsilon^2$$

E.g. if $\gamma = 0.1$:

$$\begin{aligned}\hat{\alpha}_1 &= \hat{\alpha}_0 - \gamma \frac{d}{d\alpha} \epsilon^2 \\ &= 2 - 0.1(-6) \\ &= 2.6 < 5 = \text{true } \alpha\end{aligned}$$

Now we are not overshooting, but it will take us more iterations to get to the true α value.



Gradient Descent

- The gradient tells us the direction we need to adjustment our parameter. In the above example, the amount we need to adjust is, to a first-order approximation, proportional to $-dc/d\alpha$.
- Moderating the adjustment by a learning rate γ, we have the following update rule:

$$\alpha_t = \alpha_{t-1} - \gamma \left.\frac{dc}{d\alpha}\right|_{\alpha_{t-1}}$$

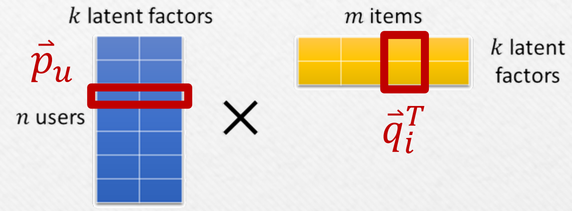
Or more typical in computer science:

$$\alpha \leftarrow \alpha - \gamma \frac{dc}{d\alpha}$$

- With more than one sample, we take the average.

SVD and Gradient Descent

- In SVD, we have $\hat{X} = PQ^T$
- User u's preference for item i is $\hat{x}_{ui} = \vec{p}_u \vec{q}_i^T$
- Loss function is $\mathrm{L} = \sum \epsilon_{ui}^2 = \sum (\vec{x}_{ui} - \hat{x}_{ui})^2$
- Take derivative of loss function w.r.t. to $\vec{p}_u$ and $\vec{q}_i^T$ (note that they are both vectors):



$$\frac{\partial L}{\partial p_{u,1}} = 2\epsilon_{ui}(-q_{i,1}), \frac{\partial L}{\partial p_{u,2}} = 2\epsilon_{ui}(-q_{i,2}), \ldots$$

So we have $\frac{\partial L}{\partial \vec{p}_u} = 2\epsilon_{ui}(-\vec{q}_i^T) = -2\epsilon_{ui}\vec{q}_i$

Similarly, $\frac{\partial L}{\partial \vec{q}_i} = 2\epsilon_{ui}(-\vec{p}_u) = -2\epsilon_{ui}\vec{p}_u$

- Update rule is $\vec{p}_u = \vec{p}_u - \gamma(-2\epsilon_{ui}\vec{q}_i)$, $\vec{q}_i = \vec{q}_i - \gamma(-2\epsilon_{ui}\vec{p}_u)$
- Combine every u and i, we have $P = P + \gamma 2\mathrm{E}Q$ and $Q^T = Q^T + \gamma 2 P^T \mathrm{E}$

SVD with Gradient Descent

1. Hyperparameter

k = No. of latent factors

2. Random initial values

P = random(n,k)
Q^T = random(k,i)
$\hat{X} = PQ^T$
loss_prev = mean$\left[(X - \hat{X})^2\right]$

3. Main loop

while loss_delta is not zero:

   a. Fit parameters with gradient descent

   $E = X - \hat{X}$
   $Q^T = Q^T + \gamma P^T E$
   $P = P + \gamma E Q$
   $\hat{X} = PQ^T$

   b. Update loss

   loss = mean$\left[(X - \hat{X})^2\right]$
   loss_delta = loss - loss_prev
   loss_prev = loss

Simon Funk's SVD

- User u's preference for item i is $\hat{x}_{ui} = \mu + b_u + b_i + \vec{p}_u \vec{q}_i^T$, where μ the constant term, b_u is the user fixed effect and b_i the item fixed effect.
- Compute gradients:

$$\frac{\partial L}{\partial \vec{p}_u} = 2\epsilon_{ui}(-\vec{q}_i^T) = -2\epsilon_{ui}\vec{q}_i$$

$$\frac{\partial L}{\partial \vec{q}_i} = 2\epsilon_{ui}(-\vec{p}_u) = -2\epsilon_{ui}\vec{p}_u$$

$$\frac{\partial L}{\partial \mu} = \frac{\partial L}{\partial b_u} = \frac{\partial L}{\partial b_i} = 2\epsilon_{ui}(-\vec{p}_u) = -2\epsilon_{ui}$$

- Update rule is

$$\vec{p}_u = \vec{p}_u - \gamma(-2\epsilon_{ui}\vec{q}_i), \vec{q}_i = \vec{q}_i - \gamma(-2\epsilon_{ui}\vec{p}_u)$$

$$\mu = \mu - \gamma(-2\epsilon_{ui}), b_u = b_u - \gamma(-2\epsilon_{ui}), b_i = b_i - \gamma(-2\epsilon_{ui})$$

Stochastic Gradient Descent

- Learning is quite slow if we only update model weights after we go through all data.
- We could instead update every time after we gone through a given number of samples. This is **Stochastic Gradient Descent (SGD).**
- Because we are not using all data, we could be updating towards the wrong direction sometimes, but on average the updates will be correct. Hence, *stochastic*.

Stochastic Gradient Descent

- The stochastic nature is actually a good property because it helps the model escape from local optima.