Day-97, Mar 7, 2025 (Falgun 23, 2081)

1. Binary Labels: Cats, Likes and Clicks
2. Movie Normalization
3. Tensorflow Implementation of Collaborative filtering (AutoDiff).
4. Explanation and Summary (using LLMs chatGPT).

Source: Unsupervised, Recommenders, Reinforcement Learning Coursera Course.

Binary labels

Movie	Alice(1)	Bob(2)	Carol(3)	Dave(4)
Love at last	1	1	0	0
Romance forever	1	?	?	0
Cute puppies of love	?	1	0	?
Nonstop car chases	0	0	1	1
Swords vs. karate	0	0	1	?

Didn't see.

Here 1 means like and 0 means dislike.

Tom's must be recommended to them.

Or it could mean that she explicitly hit like or

Example applications

1. Did user j purchase an item after being shown?
2. Did user j fav/like an item?
3. Did user j spend at least 30sec with an item?
4. Did user j click on an item?

Meaning of ratings:
1 - engaged after being shown item
0 - did not engage after being shown item
? - item not yet shown

From regression to binary classification

- Previously:
  - Predict $y^{(i,j)}$ as $\boldsymbol{w}^{(j)} \cdot \boldsymbol{x}^{(i)} + \boldsymbol{b}^{(j)}$

→ For binary labels:
predict that the probability of $y^{(i,j)} = 1$
is given by $g(w^{(j)} \cdot x^{(i)} + b^{(j)})$
where $g(z) = \frac{1}{1+e^{-z}}$

Previously we were predicting label yij as wj.xi+b.

DeepLearning.AI Stanford ONLINE Andrew Ng

① Binary Labels
1 → like
0 → Dislike
? → Not Seen

② Update Model

③ Cost function Change:
MSE to
Binary Cross Entropy.

$P(y_{ij} = 1) \to 1$

$P(y_{ij} = 1) \to 0.$

$$J = -\sum_{(i,j) \in R} [y_{ij} \log f(x) + (1 - y_{ij}) \log (1 - f(x))]$$

From regression to binary classification

- Previously:
  - Predict $y^{(i,j)}$ as $w^{(j)} \cdot x^{(i)} + b^{(j)}$
- For binary labels:
  Predict that the probability of $y^{(i,j)} = 1$ is given by $g(w^{(j)} \cdot x^{(i)} + b^{(j)})$

where $g(z) = \frac{1}{1+e^{-z}}$

Cost function for binary application

Previous cost function:

$$\frac{1}{2} \sum_{(i,j):r(i,j)=1} (\underbrace{w^{(j)} \cdot x^{(i)} + b^{(j)}}_{f(x)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^{n} (x_k^{(i)})^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^{n} (w_k^{(j)})^2$$

Loss for binary labels $y^{(i,j)}$: $f_{(w,b,x)}(x) = g(w^{(j)} \cdot x^{(i)} + b^{(j)})$

$$L(f_{(w,b,x)}(x), y^{(i,j)}) = -y^{(i,j)} \log(f_{(w,b,x)}(x)) - (1 - y^{(i,j)}) \log(1 - f_{(w,b,x)}(x))$$

Loss for single example

$$J(w, b, x) = \sum_{(i,j):r(i,j)=1} L(f_{(w,b,x)}(x), y^{(i,j)})$$

$g(w^{(j)} \cdot x^{(i)} + b^{(j)})$

DeepLearning.AI Stanford ONLINE Andrew Ng

Cost function for binary application

Previous cost function:

$$\frac{1}{2}\sum_{(i,j):r(i,j)=1}\left(\underbrace{w^{(j)}\cdot x^{(i)}+b^{(j)}}_{f(x)}-y^{(i,j)}\right)^2+\frac{\lambda}{2}\sum_{i=1}^{n_m}\sum_{k=1}^{n}\left(x_k^{(i)}\right)^2+\frac{\lambda}{2}\sum_{j=1}^{n_u}\sum_{k=1}^{n}\left(w_k^{(j)}\right)^2$$

Loss for binary labels $y^{(i,j)}$: $f_{(w,b,x)}(x) = g(w^{(j)}\cdot x^{(i)}+b^{(j)})$

Logistic Regression.

Loss for a single example.

$$\mathcal{L}\left(f_{(w,b,x)}(x), y^{(i,j)}\right) = -y^{(i,j)}\log\left(f_{(w,b,x)}(x)\right) - \left(1-y^{(i,j)}\right)\log\left(1-f_{(w,b,x)}(x)\right)$$

$$J(w,b,x) = \sum_{(i,j):r(i,j)=1} \mathcal{L}\left(f_{(w,b,x)}(x), y^{(i,j)}\right)$$

$g(w^{(j)}\cdot x^{(i)}+b^{(j)})$

Cost for all examples

When you now have binary labels,

Explanation of the Lecture Text: Collaborative Filtering with Binary Labels

This lecture explains how to **extend collaborative filtering** (which originally used numerical ratings) to work with **binary labels** (e.g., "liked" or "not liked"). The extension follows a similar transition as going from **linear regression** to **logistic regression**.

---

1. Understanding Binary Labels in Collaborative Filtering

Previously, collaborative filtering used a **numerical rating system** (e.g., 1 to 5 stars). However, in many real-world applications, user feedback is **binary**:

- **1 (liked/engaged)** → The user liked or interacted with the item.
- **0 (not liked/not engaged)** → The user did not like or did not interact with the item.
- **? (unknown)** → The user has **not yet seen** the item.

Examples of Binary Labels in Different Applications

1. **Streaming Services (e.g., Netflix, YouTube)**
   - **1** → The user watched the entire movie.
   - **0** → The user stopped watching early.
   - **?** → The user hasn't seen the movie.
2. **E-commerce (e.g., Amazon)**
   - **1** → The user purchased the product.
   - **0** → The user didn't buy the product.
   - **?** → The product was never shown to the user.
3. **Social Media (e.g., Instagram, Facebook)**
   - **1** → The user liked/shared/commented on a post.
   - **0** → The user ignored the post.
   - **?** → The user was never shown the post.
4. **Online Advertising**
   - **1** → The user clicked on an ad.
   - **0** → The user saw the ad but didn't click.
   - **?** → The ad was not shown to the user.

---

2. Adapting the Collaborative Filtering Algorithm

Previously, the algorithm predicted a rating using:

$$y_{ij} = w_j \cdot x_i + b_j$$

where:

- w_j represents user **preferences**.
- x_i represents item **features**.
- b_j is a bias term.

This is similar to **linear regression**.

However, binary labels require us to **predict probabilities** rather than continuous values. This leads t
a **logistic regression-like model**:

$$P(y_{ij} = 1) = g(w_j \cdot x_i + b_j)$$

where:

- $g(z) = \frac{1}{1+e^{-z}}$ is the **sigmoid function** (same as in logistic regression).
- This ensures predictions are in the range **(0,1)** and represent probabilities.

Why the Sigmoid Function?

- The sigmoid function converts any real number into a probability (between 0 and 1).
- If the probability is close to **1**, the model predicts **engagement (liked)**.
- If the probability is close to **0**, the model predicts **no engagement (not liked)**.

---

3. Cost Function for Binary Labels

Previously, collaborative filtering used the **Mean Squared Error (MSE)** loss:

$$J = \sum_{i,j \text{ where } r_{ij}=1} (y_{ij} - \hat{y}_{ij})^2$$

But for **binary classification**, we replace this with the **Binary Cross-Entropy (Log Loss)**:

$$J = -\sum_{i,j \text{ where } r_{ij}=1} [y_{ij} \log f_{ij} + (1 - y_{ij}) \log(1 - f_{ij})]$$

where:

- $f_{ij} = g(w_j \cdot x_i + b_j)$ is the predicted probability.
- This is the same cost function used in **logistic regression** and **binary classification in neural networks**.

---

4. Summary of the Generalized Algorithm

1. **Replace the linear rating prediction with a logistic (sigmoid) function.**
2. **Use the binary cross-entropy loss function** instead of MSE.
3. **Train the model** to minimize this new loss function.
4. **Use the predicted probability** to make recommendations:
   - If $P(y_{ij} = 1)$ is high → recommend the item.
   - If $P(y_{ij} = 1)$ is low → don't recommend the item.

---

5. Why This Approach is Powerful

- **More realistic**: Many platforms don't use explicit ratings but instead rely on user behavior.
- **Scalable**: Works for large-scale datasets (e.g., millions of users/items).
- **Flexible**: Can be applied to **movies, products, ads, and social media content**.

This approach significantly expands the range of applications for collaborative filtering! 🚀

Binary Cross Entropy (BCE) → A loss function for binary Classification (labels 0 or 1)

$$J = -\frac{1}{N} \sum [y \log(\hat{y}) + (1-y) \log(1-\hat{y})]$$

Mean Normalization

Users who have not rated any movies

Movie	Alice(1)	Bob (2)	Carol (3)	Dave (4)	Eve (5)	
Love at last	5	5	0	0	?	0
Romance forever	5	?	?	0	?	0
Cute puppies of love	?	4	0	?	?	0
Nonstop car chases	0	0	5	4	?	0
Swords vs. karate	0	0	5	?	?	0

Matrix Conversion

$$\begin{bmatrix} 5 & 5 & 0 & 0 & ? \\ 5 & ? & ? & 0 & ? \\ ? & 4 & 0 & ? & ? \\ 0 & 0 & 5 & 4 & ? \\ 0 & 0 & 5 & ? & ? \end{bmatrix}$$

$$\min_{\substack{w^{(1)},\ldots w^{(n_u)} \\ b^{(1)},\ldots b^{(n_u)} \\ x^{(1)},\ldots x^{(n_m)}}} \frac{1}{2} \sum_{(i,j):r(i,j)=1} \left(w^{(j)} \cdot x^{(i)} + b^{(j)} - y^{(i,j)}\right)^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^{n} \left(w_k^{(j)}\right)^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^{n} \left(x_k^{(i)}\right)^2$$

$w^{(5)} \Rightarrow \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ $\quad b^{(5)} \Rightarrow 0$ $\quad w^{(5)} \cdot x^{(i)} + b^{(5)}$

Mean Normalization

1) Better Predictions
2) Faster optimizations
3) More Reasonable Recommendation

Users who have not rated any movies

Movie	Alice(1)	Bob (2)	Carol (3)	Dave (4)	Eve (5)
Love at last	5	5	0	0	?
Romance forever	5	?	?	0	?
Cute puppies of love	?	4	0	?	?
Nonstop car chases	0	0	5	4	?
Swords vs. karate	0	0	5	?	?

$$\begin{bmatrix} 5 & 5 & 0 & 0 & ? \\ 5 & ? & ? & 0 & ? \\ ? & 4 & 0 & ? & ? \\ 0 & 0 & 5 & 4 & ? \\ 0 & 0 & 5 & 0 & ? \end{bmatrix}$$

$$\min_{w^{(1)},\ldots w^{(n_u)},\ b^{(1)},\ldots b^{(n_u)},\ x^{(1)},\ldots x^{(n_m)}} \frac{1}{2}\sum_{(i,j):r(i,j)=1}\left(w^{(j)}\cdot x^{(i)} + b^{(j)} - y^{(i,j)}\right)^2 + \frac{\lambda}{2}\sum_{j=1}^{n_u}\sum_{k=1}^{n}\left(w_k^{(j)}\right)^2 + \frac{\lambda}{2}\sum_{i=1}^{n_m}\sum_{k=1}^{n}\left(x_k^{(i)}\right)^2$$

$w^{(5)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ $b^{(5)} = 0$ $w^{(5)}\cdot x^{(i)} + b^{(5)}$

Steps

1) Average rating μ
2) Subtract μ from each rating to normalize the data and train algorithm.
3) Add μ back when making predictions to ensure reasonable ratings.

Mean Normalization

$$\begin{bmatrix} 5 & 5 & 0 & 0 & ? \\ 5 & ? & ? & 0 & ? \\ ? & 4 & 0 & ? & ? \\ 0 & 0 & 5 & 4 & ? \\ 0 & 0 & 5 & 0 & ? \end{bmatrix} \begin{matrix} 2.5 \\ 2.5 \\ 2 \\ 2.25 \\ 1.25 \end{matrix}$$

$$\mu \Rightarrow \begin{bmatrix} 2.5 \\ 2.5 \\ 2 \\ 2.25 \\ 1.25 \end{bmatrix}$$

$$\begin{bmatrix} 2.5 & 2.5 & -2.5 & -2.5 & ? \\ 2.5 & ? & ? & -2.5 & ? \\ ? & ? & -2 & ? & ? \\ -2.25 & -2.25 & 2.75 & 1.75 & ? \\ -1.25 & -1.25 & 3.75 & -1.25 & ? \end{bmatrix}$$

Predicting $y^{(i,j)}$ using $w^{(j)}, b^{(j)}, x^{(i)}$

for user j, on movie i predict

$$w^{(j)} \cdot x^{(i)} + b^{(j)} + \mu_i$$

User 5 (Eve):

$$w^5 \Rightarrow \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad b^{(5)} \Rightarrow 0$$

$$\underbrace{w^{(5)} \cdot x^{(1)} + b^{(5)}}_{0} + \mu_1 \Rightarrow 2.5$$

Mean Normalization in Recommendation Systems

Mean normalization helps a collaborative filtering algorithm make better predictions, especially for new users who haven't rated any movies yet. Without mean normalization, new users default to zero-star predictions, which is not useful.

Key Steps:

1. **Compute the Mean Rating for Each Movie**
   For each movie i, compute the mean rating over all users who rated it:

$$\mu_i = \frac{1}{|\{j \mid r(i,j)=1\}|} \sum_{j:r(i,j)=1} y_{i,j}$$

where:

- $y_{i,j}$ is the rating given by user j to movie i.
- $r(i,j) = 1$ if user j rated movie i, otherwise 0.

2. **Normalize the Ratings**

$$y'_{i,j} = y_{i,j} - \mu_i$$

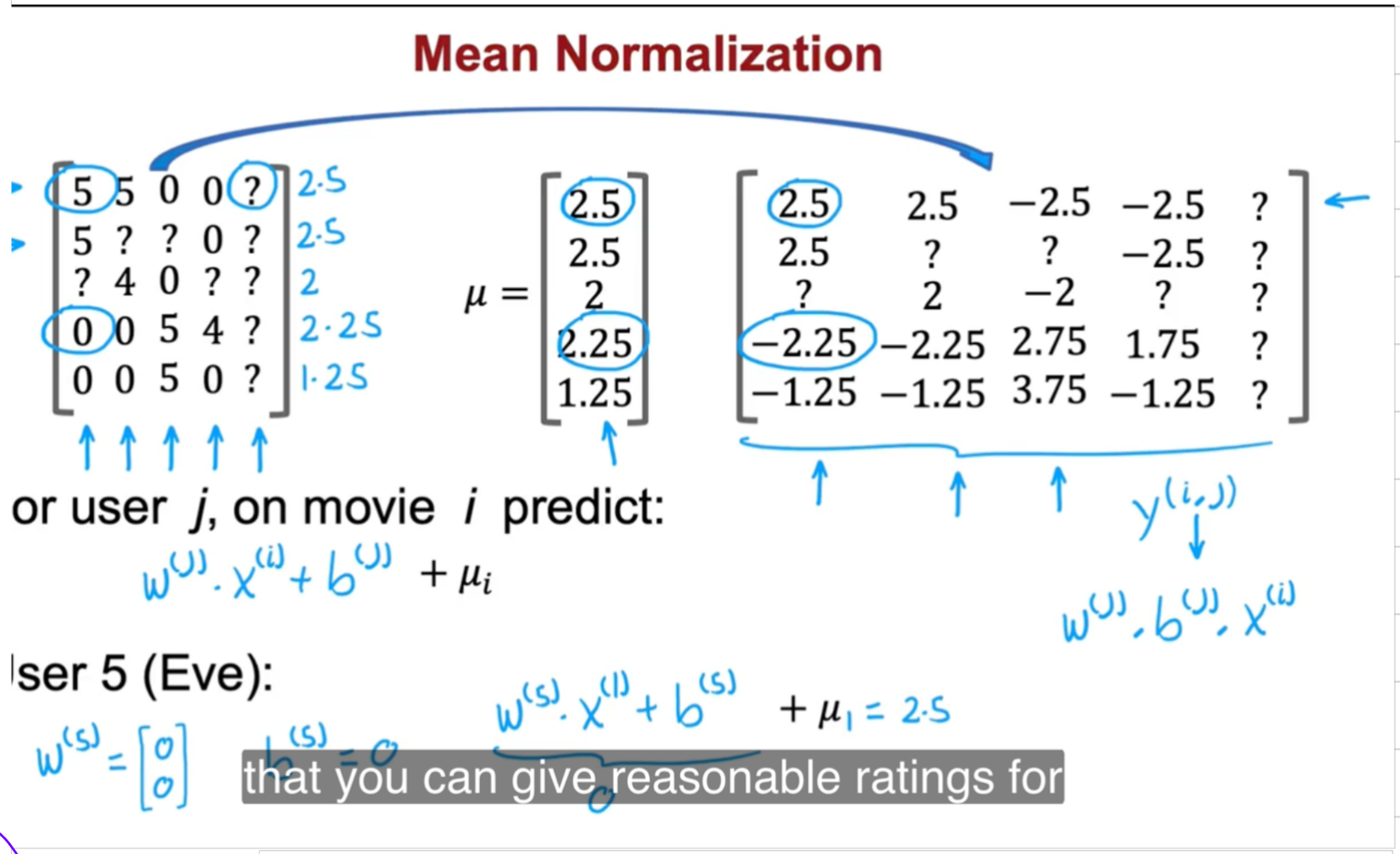
This shifts the ratings to have an average of zero.

3. **Collaborative Filtering Model**
   The cost function to minimize is:

$$J(\mathbf{X}, \Theta) = \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{n} r(i,j) \left((\theta_j^T x_i + b_j) - y'_{i,j} \right)^2 + \frac{\lambda}{2} \sum_{j=1}^{n} \|\theta_j\|^2 + \frac{\lambda}{2} \sum_{i=1}^{m} \|x_i\|^2$$

where:

- x_i represents movie features.
- θ_j represents user preference features.
- b_j is the user bias term.



4. **Predict Ratings for a New User**

$$\hat{y}_{i,j} = \theta_j^T x_i + b_j + \mu_i$$

Adding back μ_i ensures the predicted rating is in the original scale (e.g., 0–5 stars).

Benefits of Mean Normalization:

- Prevents new users from defaulting to **zero-star** predictions.
- Makes predictions **more realistic and reasonable**.
- Helps the optimization algorithm **converge faster**.

Tensorflow Implementation of Collaborative filtering:

(Tensorflow → rich library & frameworks, inbuilt APIs, functions.

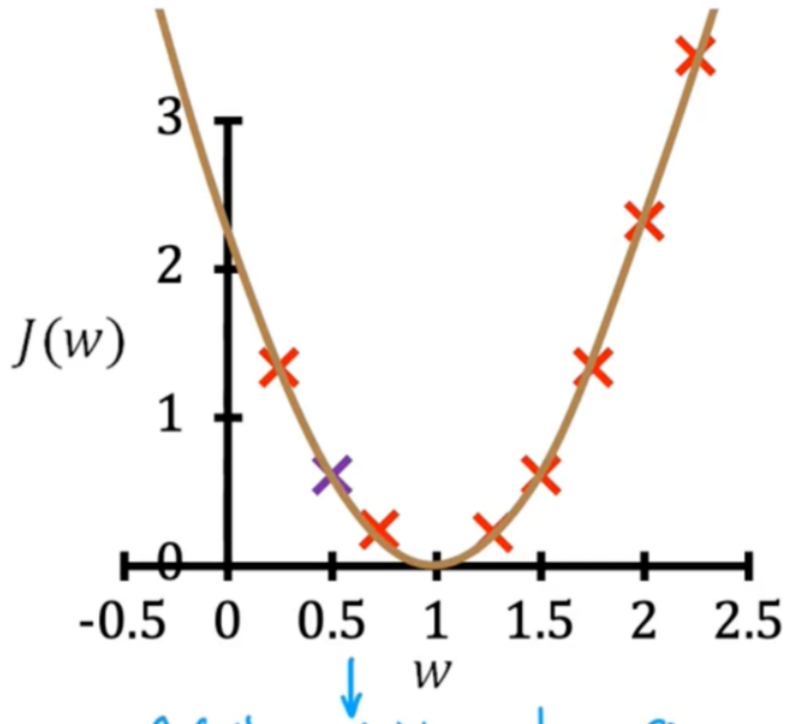
Derivatives in ML

Gradient descent algorithm

$w = w - \alpha \frac{d}{dw} J(w,b)$

Learning Rule
Derivative

$b = b - \alpha \frac{\partial}{\partial b} J(w,b)$ ← $b = 0$.



$f(x) = w \cdot x$ $b = 0$

which looked like this,
where w gets repeatedly updated as w minus

DeepLearning.AI Stanford ONLINE Andrew Ng

Auto Diff. → whose library knows to Compute $f'(x)$ and ∇ descent.

$$J = \overbrace{(wx}^{f(x)} - \overbrace{1}^{y})^2$$

Gradient descent algorithm

Repeat until convergence

$$w = w - \alpha \frac{d}{dw} J(w,b)$$

Fix b = 0 for this example

Custom Training Loop

```
w = tf.Variable(3.0)
x = 1.0
y = 1.0 # target value
alpha = 0.01

iterations = 30
for iter in range(iterations):
    # Use TensorFlow's Gradient tape to record the steps
    # used to compute the cost J, to enable auto differentiation.
    with tf.GradientTape() as tape:
        fwb = w*x
        costJ = (fwb - y)**2

    # Use the gradient tape to calculate the gradients
    # of the cost with respect to the parameter w.
    [dJdw] = tape.gradient( costJ, [w] )

    # Run one step of gradient descent by updating
    # the value of w to reduce the cost.
    w.assign_add(-alpha * dJdw)
```

tf.variables are params we want to optimize.

Automate recognize cost function.

f(x)

→ TF knows to do $\frac{\partial J}{\partial w}(w)$.

this update over here.

DeepLearning.AI Stanford ONLINE Andrew Ng

Implementation in TensorFlow

Gradient descent Algorithm

Repeat until Convergence

$$w = w - \alpha \frac{\partial}{\partial w} J(w,b,x)$$

$$b = b - \alpha \frac{\partial}{\partial b} J(w,b,x)$$

$$x = x - \alpha \frac{\partial}{\partial x} \left(J(w,b,x) \right)$$

```
# Instantiate an optimizer.
optimizer = keras.optimizers.Adam(learning_rate=1e-1)

iterations = 200
for iter in range(iterations):
    # Use TensorFlow's GradientTape
    # to record the operations used to compute the cost
    with tf.GradientTape() as tape:

        # Compute the cost (forward pass is included in cost)
        cost_value = cofiCostFuncV(X, W, b, Ynorm, R,
                                   num_users, num_movies, lambda)

    # Use the gradient tape to automatically retrieve
    # the gradients of the trainable variables with respect to
    #     the loss
    grads = tape.gradient( cost_value, [X,W,b] )

    # Run one step of gradient descent by updating
    # the value of the variables to minimize the loss.
    optimizer.apply_gradients( zip(grads, [X,W,b]) )
```

Dataset credit: Harper and Konstan. 2015. The MovieLens Datasets: History and Context

this is the syntax you can use.

DeepLearning.AI Stanford ONLINE Andrew Ng

This video explains how to implement **collaborative filtering** using **TensorFlow**, leveraging **automatic differentiation (Auto Diff)** to optimize the cost function without manually computing derivatives. Below is a structured breakdown with key equations.

Gradient Descent in TensorFlow

Basic Gradient Descent Update

The weight w is updated iteratively using the gradient descent rule:

$$w := w - \alpha \frac{\partial J}{\partial w}$$

If bias b is also optimized, the update rule includes:

$$b := b - \alpha \frac{\partial J}{\partial b}$$

Example: Simple Cost Function

Given a cost function for a single training example:

$$J = (wx - 1)^2$$

The derivative w.r.t. w is:

$$\frac{\partial J}{\partial w} = 2(wx - 1)x$$

TensorFlow automatically computes this derivative using **gradient tape**.

Collaborative Filtering Algorithm in TensorFlow

The collaborative filtering cost function is given by:

$$J(\mathbf{X}, \Theta) = \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{n} r(i,j) \left((\theta_j^T x_i + b_j) - y'_{i,j} \right)^2 + \frac{\lambda}{2} \sum_{j=1}^{n} \|\theta_j\|^2 + \frac{\lambda}{2} \sum_{i=1}^{m} \|x_i\|^2$$

where:

- x_i is the feature vector for movie i.
- θ_j is the feature vector for user j.
- b_j is the user bias term.
- $r(i,j)$ is 1 if user j rated movie i, otherwise 0.
- $y'_{i,j} = y_{i,j} - \mu_i$ is the **mean-normalized rating**.

Using **Auto Diff**, TensorFlow automatically computes:

$$\frac{\partial J}{\partial x_i}, \quad \frac{\partial J}{\partial \theta_j}, \quad \frac{\partial J}{\partial b_j}$$

and updates parameters using gradient descent or the **Adam optimizer**.

Key TensorFlow Steps

1. Define variables:

```python
w = tf.Variable(3.0)
x = 1.0
y = 1.0
alpha = 0.01
```

Adaptive - Gradient Based optimizations

↓

Dynamically adjusts the learning rate for each parameter based on past gradients.

Key TensorFlow Steps

1. Define variables:


```python
w = tf.Variable(3.0)
x = 1.0
y = 1.0
alpha = 0.01
```


2. Use **gradient tape** for automatic differentiation:

```python
with tf.GradientTape() as tape:
    f_x = w * x
    J = (f_x - y) ** 2
dJdw = tape.gradient(J, w)
```

3. Update parameters using **gradient descent** or **Adam optimizer**:

```python
optimizer = tf.keras.optimizers.Adam(learning_rate=0.01)
optimizer.apply_gradients([(dJdw, w)])
```

By leveraging **TensorFlow's automatic differentiation**, the collaborative filtering model efficiently optimizes the recommendation system without manually computing gradients. 🚀

Adam optimizer → Combines momentum and adaptive learning rates for efficient and stable convergence and is a adaptive-Gradient Based optimizer.