Applications of TensorFlow

Rok Hribar and Primož Godec

September 2019

What can TensorFlow do?

1. It can perform numerical operations on data (in a parallel way – multi-core, GPU).

```
import tensorflow as tf

A = tf.Variable( [[1.0, 2.0], [3.0, 4.0]] )
B = tf.Variable( [[5.0, 6.0], [7.0, 8.0]] )

C = tf.matmul(A, B)          # matrix multiplication
D = A - B*C                  # elementwise operations
cos_D = tf.cos(D)            # elementwise math functions
sum_D = tf.reduce_sum(D)     # sum of all D components
max_D = tf.reduce_max(D)     # max component of D
svd_D = tf.linalg.svd(D)     # singular value decomposition
```

```
C = tf.matmul(A, B)
# <tf.Tensor: id=17, shape=(2, 2), dtype=float32,
#  numpy=array([[19., 22.], [43., 50.]], dtype=float32)>

cos_D = tf.cos(D)
# <tf.Tensor: id=30, shape=(2, 2), dtype=float32,
#  numpy=array([[ 0.96945935, -0.36729133],
#               [-0.89988   ,  0.9873345 ]],
#              dtype=float32)>

max_D = tf.reduce_max(D)
# <tf.Tensor: id=26, shape=(), dtype=float32,
#  numpy=-94.0>

C.numpy()
# array([[19., 22.], [43., 50.]], dtype=float32)
```

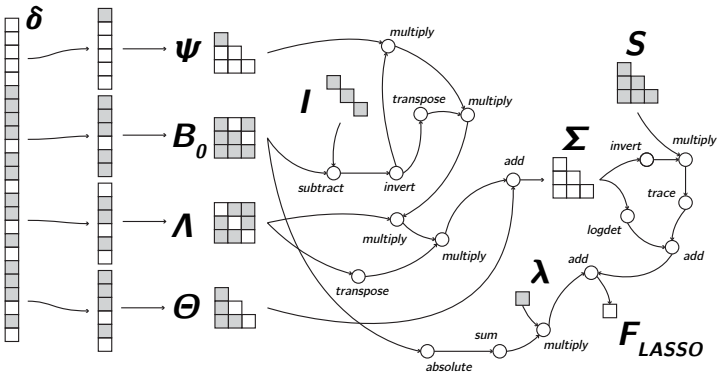
```
svd_D = tf.linalg.svd(D)
# (<tf.Tensor: id=27, shape=(2,), dtype=float32,
#   numpy=array([520.9103, 2.9102921], dtype=float32)>,
#
#  <tf.Tensor: id=28, shape=(2, 2), dtype=float32,
#   numpy=array([[-0.30792360,  0.95141107],
#                [-0.95141107, -0.30792360]],
#               dtype=float32)>,
#
#  <tf.Tensor: id=29, shape=(2, 2), dtype=float32,
#   numpy=array([[0.59984480,  0.80011636],
#                [0.80011636, -0.59984480]],
#               dtype=float32)>)
```

What can TensorFlow do?

1. It can perform numerical operations on data (in a parallel way – multi-core, GPU).

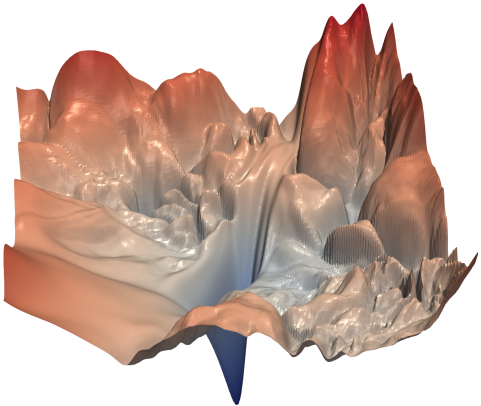
What can TensorFlow do?

1. It can perform numerical operations on data (in a parallel way – multi-core, GPU).
2. It can calculate derivatives using automatic differentiation.



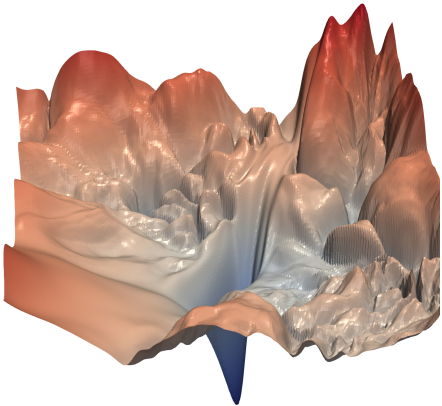
Why do we need derivatives?

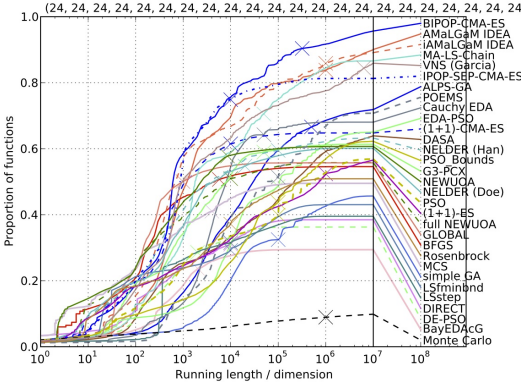
Knowing in which direction “down” is, can help us when solving optimization problems.



Why do we need derivatives?

Knowing in which direction "down" is, can help us when solving optimization problems.





- So, various competitions show that evolutionary algorithms outpreform gradient based optimization algorithms.
- **However**, all those competitions use functions of "low" dimension (≤ 100) and gradient based optimization excels in high dimensions.

- So, various competitions show that evolutionary algorithms outpreform gradient based optimization algorithms.
- **However**, all those competitions use functions of "low" dimension (≤ 100) and gradient based optimization excels in high dimensions.

How many local minima are there with respect to dimension?

- So, various competitions show that evolutionary algorithms outpreform gradient based optimization algorithms.
- **However**, all those competitions use functions of "low" dimension (≤ 100) and gradient based optimization excels in high dimensions.

How many local minima are there with respect to dimension?

$$\begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

- So, various competitions show that evolutionary algorithms outpreform gradient based optimization algorithms.
- **However**, all those competitions use functions of "low" dimension (≤ 100) and gradient based optimization excels in high dimensions.

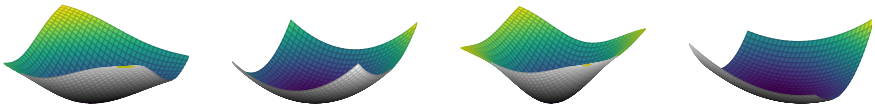
How many local minima are there with respect to dimension?

$$\begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

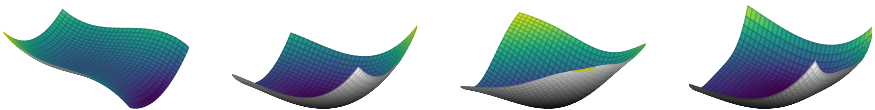
If eigenvalues of Hessian matrix are randomly distributed, then probability that a stationary point is a local minimum is $\mathbf{2^{-n}}$.

- Saddle points are exponentially more common than local minima.

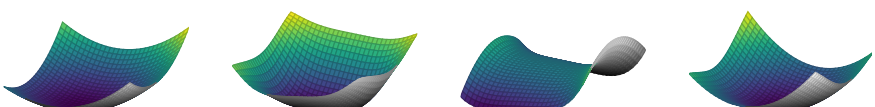
x_1, x_2 x_3, x_4 x_5, x_6 x_7, x_8



x_9, x_{10} x_{11}, x_{12} x_{13}, x_{14} x_{15}, x_{16}



x_{17}, x_{18} x_{19}, x_{20} x_{21}, x_{22} x_{23}, x_{24}



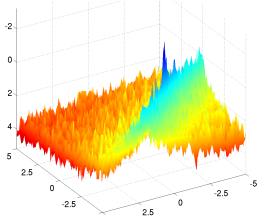
Ways to calculate derivatives of a program

1. Numerical differentiation

$$\frac{\partial}{\partial x_1} f(x_1, x_2, \dots) \approx \frac{f(x_1 + h, x_2, \dots) - f(x_1 - h, x_2, \dots)}{2h}$$

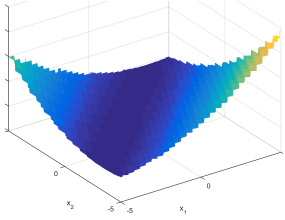
Very efficient for

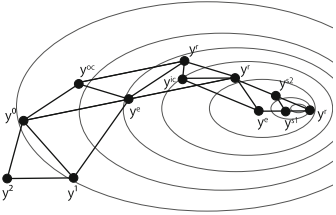
- Noisy functions
- Locally flat functions



Algorithms that use it

- Nelder-Mead algorithm
- OpenAI evolution strategy



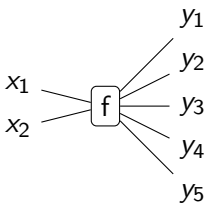


Ways to calculate derivatives of a program

2. Symbolic differentiation

$$\frac{\partial}{\partial x} \log\left(1 + \exp\left(ax + b\right)\right) = \frac{a \exp\left(ax + b\right)}{1 + \exp\left(ax + b\right)}$$

Very efficient in case function has large number of outputs

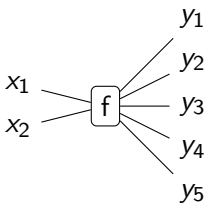


Ways to calculate derivatives of a program

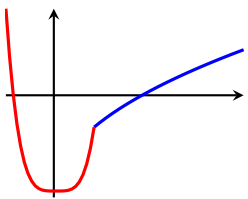
2. Symbolic differentiation

$$\frac{\partial}{\partial x} \log\left(1 + \exp\left(ax + b\right)\right) = \frac{a \exp\left(ax + b\right)}{1 + \exp\left(ax + b\right)}$$

Very efficient in case function has large number of outputs



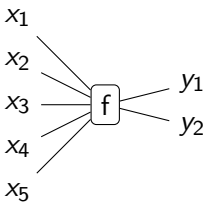

```
if f(x, data) > 0:
    g(x, data)
else:
    h(x, data)
```



Ways to calculate derivatives of a program

3. Automatic differentiation
   - Sort of a hybrid between simbolic and numerical differentiation
   - There exist forward and reverse automatic differentiation – `TensorFlow` uses reverse automatic differentiation

Very efficient in case function has large number of inputs

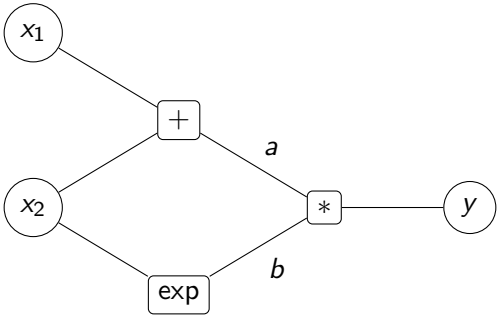


Example:

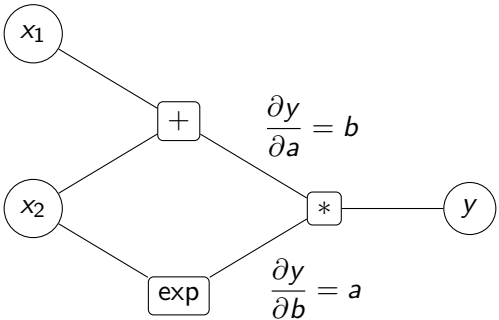
$$f(x_1, x_2, \ldots, x_n) = x_1 \cdot x_2 \cdot \ldots \cdot x_n$$

$$\nabla f = \begin{bmatrix} x_2 \cdot x_3 \cdot \ldots \cdot x_n \\ x_1 \cdot x_3 \cdot \ldots \cdot x_n \\ \vdots \\ x_1 \cdot x_2 \cdot \ldots \cdot x_{n-1} \end{bmatrix}$$

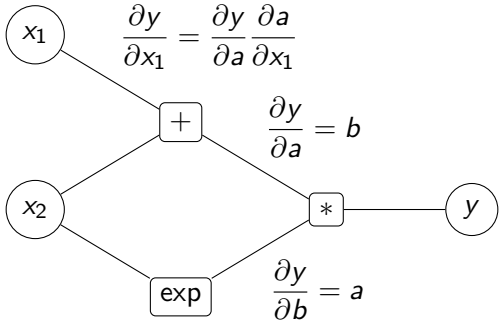
$$y = (x_1 + x_2) \exp(x_2)$$



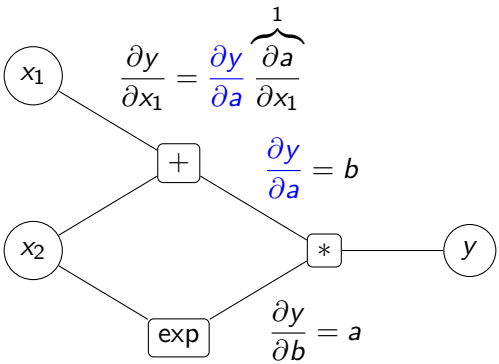
$$y = (x_1 + x_2)\exp(x_2)$$


x_1
$+$
$\frac{\partial y}{\partial a} = b$
x_2
$*$
y
exp
$\frac{\partial y}{\partial b} = a$

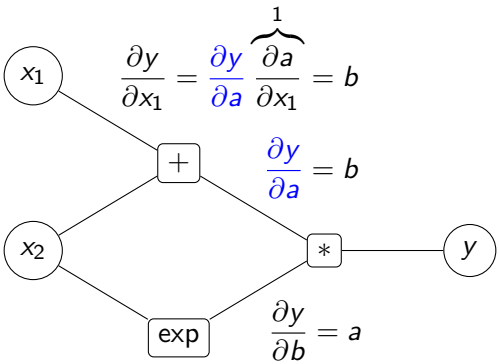
$$y = (x_1 + x_2)\exp(x_2)$$


x_1
$\frac{\partial y}{\partial x_1} = \frac{\partial y}{\partial a}\frac{\partial a}{\partial x_1}$
+
$\frac{\partial y}{\partial a} = b$
x_2
*
y
exp
$\frac{\partial y}{\partial b} = a$

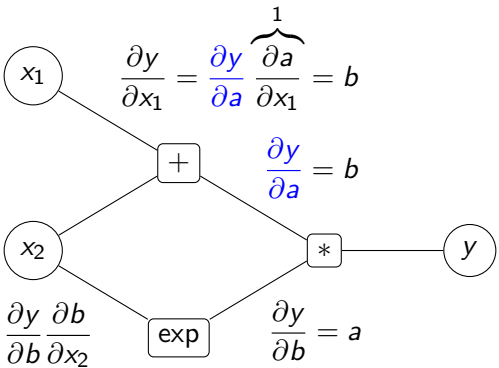
$$y = (x_1 + x_2)\exp(x_2)$$


x_1
x_2
+
*
exp
y
$\frac{\partial y}{\partial x_1} = \frac{\partial y}{\partial a}\overbrace{\frac{\partial a}{\partial x_1}}^{1}$
$\frac{\partial y}{\partial a} = b$
$\frac{\partial y}{\partial b} = a$

$$y = (x_1 + x_2)\exp(x_2)$$

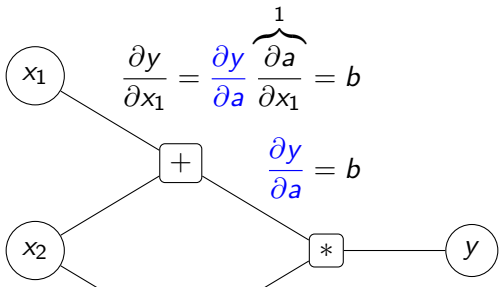

x_1
x_2
+
*
exp
y
$\frac{\partial y}{\partial x_1} = \frac{\partial y}{\partial a}\overbrace{\frac{\partial a}{\partial x_1}}^{1} = b$
$\frac{\partial y}{\partial a} = b$
$\frac{\partial y}{\partial b} = a$

$$y = (x_1 + x_2)\exp(x_2)$$

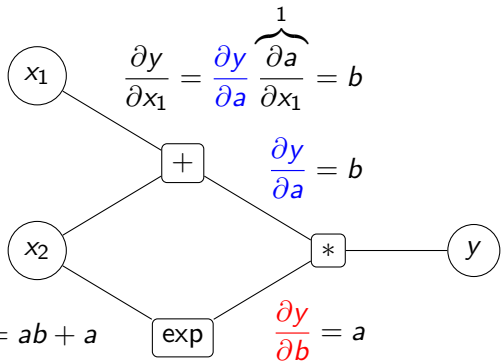


$$\frac{\partial y}{\partial x_2} = \frac{\partial y}{\partial a}\frac{\partial a}{\partial x_2} + \frac{\partial y}{\partial b}\frac{\partial b}{\partial x_2}$$

$$y = (x_1 + x_2)\exp(x_2)$$


x_1
x_2
+
*
exp
y
$\frac{\partial y}{\partial x_1} = \frac{\partial y}{\partial a}\overbrace{\frac{\partial a}{\partial x_1}}^{1} = b$
$\frac{\partial y}{\partial a} = b$
$\frac{\partial y}{\partial b} = a$
$\frac{\partial y}{\partial x_2} = \frac{\partial y}{\partial a}\underbrace{\frac{\partial a}{\partial x_2}}_{a} + \frac{\partial y}{\partial b}\underbrace{\frac{\partial b}{\partial x_2}}_{1}$

$$y = (x_1 + x_2)\exp(x_2)$$



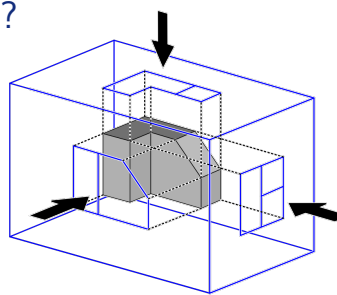
$$\frac{\partial y}{\partial x_2} = \frac{\partial y}{\partial a}\underbrace{\frac{\partial a}{\partial x_2}}_{a} + \frac{\partial y}{\partial b}\underbrace{\frac{\partial b}{\partial x_2}}_{1} = ab + a$$

The same thing in TensorFlow

```
import tensorflow as tf

x1 = tf.Variable(3.1)
x2 = tf.Variable(-1.4)

with tf.GradientTape() as tape: # Save graph to tape
    tape.watch([x1, x2])        # Watch for x1 and x2
    f = (x1+x2)*tf.exp(x2)      # Calculate f(x1, x2)

df = tape.gradient(f, [x1, x2])
# [<tf.Tensor: id=22, shape=(), dtype=float32, numpy
    =0.24659698>,
#  <tf.Tensor: id=25, shape=(), dtype=float32, numpy
    =0.66581184>]
```

What do tensors have to do with anything?

What do tensors have to do with anything?



```
import tensorflow as tf
from numpy.random import rand

interior = tf.Variable(rand(16, 16, 16))

with tf.GradientTape() as tape:
    tape.watch(interior)
    up_down    = tf.reduce_sum(interior, axis=0)
    left_right = tf.reduce_sum(interior, axis=1)
    in_out     = tf.reduce_sum(interior, axis=2)
    error = ...

grad = tape.gradient(error, interior)
```

Gradient descent

```
while True:
    weights_grad = evaluate_gradient(loss_fun, data,
    weights)
    weights += - step_size * weights_grad # perform
    parameter update
```

Types:

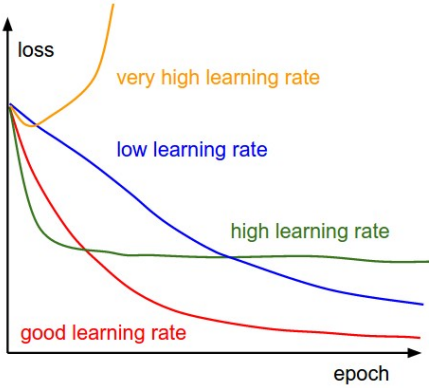
- Mini-batch gradient descent
- Stochastic gradient descent
- Gradient descent on whole data

Simple linear model

We will create a simple linear model and train it using the gradient descent.

1. Define the model
2. Define a loss function
3. Generate training data
4. Fit the model to training data

Simple linear model



Optimizers

- Vanilla update (`tf.keras.optimizers.SGD`)
- Momentum update
- Nesterov Momentum

- Adagrad (`tf.keras.optimizers.Adagrad`)
- RMSprop (`tf.keras.optimizers.RMSprop`)
- Adam (`tf.keras.optimizers.Adam`)

Optimizers are available in `tf.keras.optimizers` module

Optimizers - SGD

Vanilla update

```
x += - learning_rate * dx
```

Momentum update

```
v = mu * v - learning_rate * dx # integrate velocity
x += v # integrate position
```

Nesterov Momentum

```
x_ahead = x + mu * v
# evaluate dx_ahead (the gradient at x_ahead instead
of at x)
v = mu * v - learning_rate * dx_ahead
x += v
```

Optimizers - Adaptive

Adagrad

```
cache += dx**2
x += - learning_rate * dx / (np.sqrt(cache) + eps)
```

RMSprop

```
cache = decay_rate * cache + (1 - decay_rate) * dx**2
x += - learning_rate * dx / (np.sqrt(cache) + eps)
```

Adam

```
m = beta1*m + (1-beta1)*dx
v = beta2*v + (1-beta2)*(dx**2)
x += - learning_rate * m / (np.sqrt(v) + eps)
```

Optimizers - example

We will change the previous linear example that it will use Adam optimizer now.

Matrix factorization example

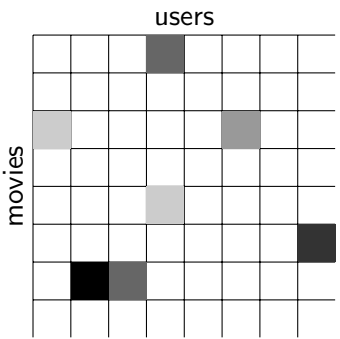
Suppose we have movie ratings from various people for set of movies they have watched.

user id	movie id	rating
4160	14501	5
182	14502	2
6649	14502	3
17240	14502	1
115	14503	4
⋮	⋮	⋮

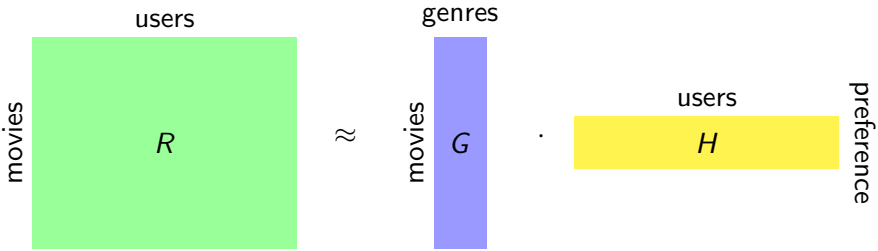
Matrix factorization example

Suppose we have movie ratings from various people for set of movies they have watched.

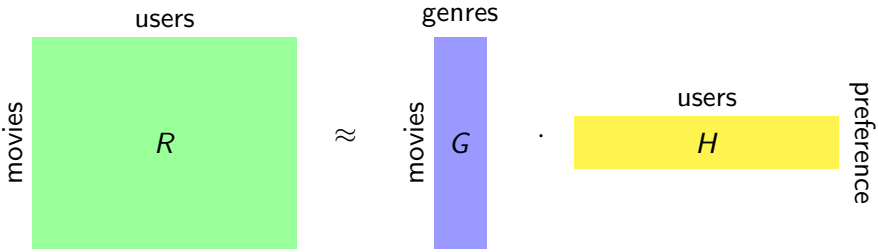
user id	movie id	rating
4160	14501	5
182	14502	2
6649	14502	3
17240	14502	1
115	14503	4
⋮	⋮	⋮



Matrix factorization example



Matrix factorization example



$$\text{error} = \|W \odot (R - G \cdot H)\| = \min. \qquad G, H \geq 0$$

```
# Initialize variables.
G = tf.Variable(...)
H = tf.Variable(...)

# Choose a gradient based optimizer.
optimizer = tf.keras.optimizers.Adam()

# Perform gradient descent.
for i in range(num_steps):
    with tf.GradientTape() as tape:
        tape.watch([G, H])
        absG = tf.abs(G)
        absH = tf.abs(H)
        dR = R - tf.matmul(absG, absH)
        loss = tf.reduce_sum(tf.square(dR))
    dG, dH = tape.gradient(loss, [G, H])
    optimizer.apply_gradients([[dG, G], [dH, H]])
```