School of Information Technology and Engineering

Home work №1

Done by: **Kasym Talgat**
Checked by: **Sergei Lytkin**

Almaty 2024

1 Task 1.1 (1 point)

Let $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, $\boldsymbol{B} \in \mathbb{R}^{n \times m}$. Prove that $\operatorname{tr}(\boldsymbol{AB}) = \operatorname{tr}(\boldsymbol{BA})$. Using this property, calculate $\operatorname{tr}(\boldsymbol{uv}^{\mathsf{T}})$ if $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^{n}$, $\boldsymbol{u} \perp \boldsymbol{v}$.

	1	2	…	n
1	a_{11}	a_{12}	…	a_{1n}
2	a_{21}	a_{22}	…	a_{2n}
3	a_{31}	a_{32}	…	a_{3n}
⋮	⋮	⋮	⋱	⋮
m	a_{m1}	a_{m2}	…	a_{mn}

	1	2	…	m
1	b_{11}	b_{12}	…	b_{1m}
2	b_{21}	b_{22}	…	b_{2m}
3	b_{31}	b_{32}	…	b_{3m}
⋮	⋮	⋮	⋱	⋮
n	b_{n1}	b_{n2}	…	b_{nm}

1) We multiply A and B and get new matrix with m rows and m columns
2) We multiply B and A and get new matrix with n rows and n columns

$AB =$

	1	2	…	m
1	$a_{11}b_{11} + \cdots + a_{1n}b_{n1}$	…	…	…
2	…	$a_{21}b_{12} + \cdots + a_{2n}b_{n2}$	…	…
⋮	⋮	⋮	⋱	⋮
m	…	…	…	$a_{m1}b_{1m} + \cdots + a_{mn}b_{nm}$

$BA =$

	1	2	…	n
1	$b_{11}a_{11} + \cdots + b_{1m}a_{m1}$	…	…	…
2	…	$b_{21}a_{12} + \cdots + b_{2m}a_{m2}$	…	…
⋮	⋮	⋮	⋱	⋮
n	…	…	…	$b_{n1}a_{1n} + \cdots + b_{nm}a_{mn}$

3) The trace of a square matrix A, denoted as tr(A), is the sum of the elements on its main diagonal (the diagonal from the top-left to the bottom-right of the matrix).
1) $tr(AB) = a_{11}b_{11} + \cdots + a_{1n}b_{n1} + a_{21}b_{12} + \cdots + a_{2n}b_{n2} + a_{m1}b_{1m} + \cdots + a_{mn}b_{nm}$
1) $tr(BA) = b_{11}a_{11} + \cdots + b_{1m}a_{m1} + b_{21}a_{12} + \cdots + b_{2m}a_{m2} + b_{n1}a_{1n} + \cdots + b_{nm}a_{mn}$

We can observe a pattern: the sum of the first elements of each row of AB is equal to the sum of the first column of BA, and the same applies to the other elements.

4) Let u and v be vectors with n elements and we know that u and v orthogonal. Orthogonal means that their dot product is equal to zero.

$u_1v_1 + u_2v_2 + \cdots + u_nv_n = 0$

$\sum_{i=1}^{n} u_i v_i = 0$

5) v^T has 1 row and n columns.
6) Multiplying u and v^T gives us matrix with 1 row and 1 column. And that cell will be equal to dot product of v and u, which means zero. That means $\text{tr}(uv^T) = 0$

2 Task 1.2 (1 point)

Let $\boldsymbol{A} \in \mathbb{R}^{m \times n}$. Prove that $\boldsymbol{N(A)} = \boldsymbol{N(A^\top A)}$, i.e., matrices $\boldsymbol{A}$ and $\boldsymbol{A^\top A}$ have same nullspaces.

1. **First, show that $N(A) \subseteq N(A^T A)$:**
   - If $x \in N(A)$, then $Ax = 0$.
   - Now apply $A^T A$ to x:
   $$A^T Ax = A^T(Ax) = A^T(0) = 0$$
   - This shows $x \in N(A^T A)$, so $N(A) \subseteq N(A^T A)$.
2. **Now, show that $N(A^T A) \subseteq N(A)$:**
   - If $x \in N(A^T A)$, then $A^T Ax = 0$.
   - Multiply both sides by x^T:
   $$x^T A^T Ax = 0$$
   - This simplifies to:
   $$\|Ax\|^2 = 0$$
   - Since the norm of a vector is zero only if the vector itself is zero, we conclude $Ax = 0$, meaning $x \in N(A)$.
3. **Conclusion:**
   Since $N(A) \subseteq N(A^T A)$ and $N(A^T A) \subseteq N(A)$, we have:
   $$N(A) = N(A^T A)$$

3 Task 1.3 (0.5 point)

Two matrices A and B are called similar if there exists an invertible matrix P such that:
$$B = P^{-1} AP$$

Show that similar matrices have equal determinants: $\det \boldsymbol{A} = \det \boldsymbol{B}$ if $\boldsymbol{A} \sim \boldsymbol{B}$.

Rules that we use

1. $\det(AB) = \det(A)\det(B)$ for any two square matrices A and B.
2. $\det(P^{-1}) = \frac{1}{\det(P)}$ for any invertible matrix P.

Now, using the fact that $B = P^{-1}AP$, let's compute $\det(B)$:

$$\det(B) = \det(P^{-1}AP)$$

Using the property of determinants for a product of matrices:

$$\det(B) = \det(P^{-1})\det(A)\det(P)$$

Now, applying the second property, $\det(P^{-1}) = \frac{1}{\det(P)}$:

$$\det(B) = \left(\frac{1}{\det(P)}\right)\det(A)\det(P)$$

The terms $\frac{1}{\det(P)}$ and $\det(P)$ cancel out, leaving:

$$\det(B) = \det(A)$$

4 Task 1.4 (Sherman—Morrison formula, 1.5 points)

The task asks to prove the Sherman–Morrison formula:

$$(A + uv^T)^{-1} = A^{-1} - \frac{A^{-1}uv^TA^{-1}}{1 + v^TA^{-1}u}$$

where $A \in \mathbb{R}^{n\times n}$ is an invertible matrix, $u, v \in \mathbb{R}^n$, and $v^TA^{-1}u \neq -1$.

5 Proof Outline:

The Sherman–Morrison formula provides a way to compute the inverse of a rank-1 update to an invertible matrix A, i.e., the inverse of $A + uv^T$. To prove this, we proceed as follows:

Step 1: Start with the inverse expression

We seek to prove that:

$$(A + uv^T)\left(A^{-1} - \frac{A^{-1}uv^TA^{-1}}{1 + v^TA^{-1}u}\right) = I$$

where I is the identity matrix.

Step 2: Expand the left-hand side

Multiply the two terms on the left-hand side:

$$(A + uv^T)\left(A^{-1} - \frac{A^{-1}uv^TA^{-1}}{1 + v^TA^{-1}u}\right)$$

First, distribute $A + uv^T$ over both terms inside the parentheses.

- For the first term:

$$(A + uv^T)A^{-1} = AA^{-1} + uv^TA^{-1} = I + uv^TA^{-1}$$

- For the second term:

$$(A + uv^T)\left(-\frac{A^{-1}uv^TA^{-1}}{1 + v^TA^{-1}u}\right)$$

Break this into two parts:

$$A\left(-\frac{A^{-1}uv^TA^{-1}}{1 + v^TA^{-1}u}\right) = -\frac{uv^TA^{-1}}{1 + v^TA^{-1}u}$$

and

$$uv^T\left(-\frac{A^{-1}uv^TA^{-1}}{1 + v^TA^{-1}u}\right) = -\frac{uv^TA^{-1}uv^TA^{-1}}{1 + v^TA^{-1}u}$$

Step 3: Combine the terms

Now, combining all the results:

$$I + uv^TA^{-1} - \frac{uv^TA^{-1}}{1 + v^TA^{-1}u} - \frac{uv^TA^{-1}uv^TA^{-1}}{1 + v^TA^{-1}u}$$

Notice that the first two terms involving uv^TA^{-1} can be simplified:

$$uv^TA^{-1}\left(1 - \frac{1}{1 + v^TA^{-1}u}\right)$$

Simplifying the term inside the parentheses:

$$1 - \frac{1}{1 + v^T A^{-1} u} = \frac{v^T A^{-1} u}{1 + v^T A^{-1} u}$$

So the expression becomes:

$$I + \frac{uv^T A^{-1}(v^T A^{-1} u)}{1 + v^T A^{-1} u} - \frac{uv^T A^{-1} uv^T A^{-1}}{1 + v^T A^{-1} u}$$

These two terms involving $uv^T A^{-1} u$ cancel out, leaving:

$$I$$

Conclusion

We have shown that:

$$(A + uv^T)\left(A^{-1} - \frac{A^{-1} uv^T A^{-1}}{1 + v^T A^{-1} u}\right) = I$$

Thus, the inverse of $A + uv^T$ is indeed:

$$(A + uv^T)^{-1} = A^{-1} - \frac{A^{-1} uv^T A^{-1}}{1 + v^T A^{-1} u}$$

as required.

6 Task 1.5 (programming, 1.5 points)

Apply k-NN algorithm to MNIST dataset and measure its performance:

- Split the dataset into train and test parts.
- Train several models with different hyperparameters (take $1 \leq k \leq 20$ and different distance metrics ($p = 1$, $p = 2$, $p = +\infty$)).
- Visualize several test samples and their predictions (see code below).
- Plot train and test accuracies for each model on the same graph.
- Find a model with the best test accuracy.

To split a dataset into training and testing subsets, we commonly use the train_test_split function from libraries like scikit-learn in Python. This function randomly divides the data into two parts: one for training the model and the other for testing its performance.

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=42)
```

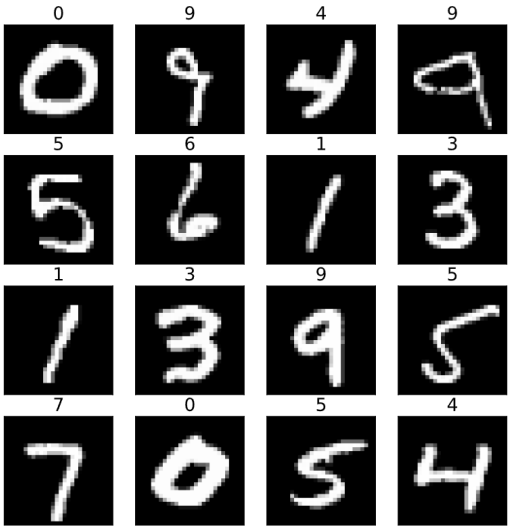
```
for p in [1, 2, np.inf]:
    for k in range(1, 21):
        knn = KNeighborsClassifier(n_neighbors=k, p=p)
        knn.fit(X_train, Y_train)

        # Evaluate on test set
        Y_test_pred = knn.predict(X_test)
        Y_train_pred = knn.predict(X_train)

        acc_test = accuracy_score(Y_test, Y_test_pred)
        acc_train = accuracy_score(Y_train, Y_train_pred)

        print(f"k={k}, p={p}, Accuracy: {acc_test:.4f}")
        print(f"k={k}, p={p}, Accuracy: {acc_train:.4f}")
        arr_acc_test.append(acc_test)
        arr_acc_train.append(acc_train)
```

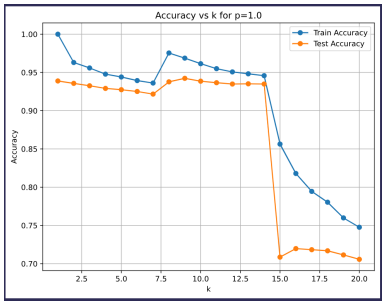
```python
from sklearn.datasets import fetch_openml
X, Y = fetch_openml('mnist_784', return_X_y=True, parser='auto')
X = X.astype(float).values / 255
Y = Y.astype(int).values
X.shape, Y.shape
```

```
((70000, 784), (70000,))
```

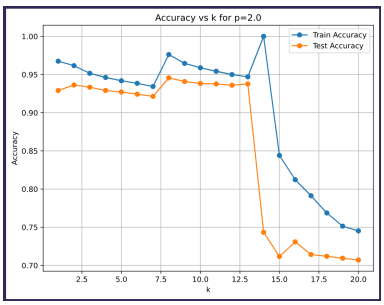
Visualize it:

```python
import numpy as np
import matplotlib.pyplot as plt

%config InlineBackend.figure_format = 'svg'

def plot_digits(X, y_true, y_pred=None, n=4, random_state=123):
    np.random.seed(random_state)
    indices = np.random.choice(np.arange(X.shape[0]), size=n*n, replace=False)
    plt.figure(figsize=(10, 10))
    for i in range(n*n):
        plt.subplot(n, n, i+1)
        plt.xticks([])
        plt.yticks([])
        plt.grid(False)
        plt.imshow(X[indices[i]].reshape(28, 28), cmap='gray')
        # plt.imshow(train_images[i], cmap=plt.cm.binary)
        if y_pred is None:
            title = str(y_true[indices[i]])
        else:
            title = f"y={y_true[indices[i]]}, ŷ={y_pred[indices[i]]}"
        plt.title(title, size=20)
    plt.show()

plot_digits(X, Y)
```

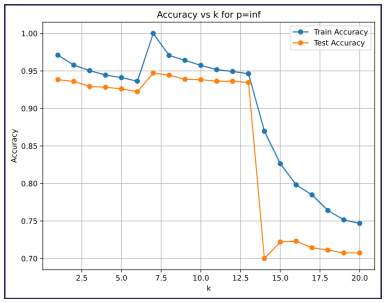




(a) p=1



(b) p=2



(c) p=inf

Figure 2: Train and test accuracies

7 Task 1.6 (programming, 1.5 points)

Compare the performance of different algorithms for calculation matrix product

First of all we should write 4 function loops for calculating sum of diagonals

$$C_{ik} = \sum_{j=1}^{n} A_{ij} B_{jk}$$

mat_mul_loop, mat_mul_inner, mat_mul_outer and A@B

```python
import numpy as np

def mat_mul_loop(A, B):
    m, n = A.shape
    n, p = B.shape
    C = np.zeros((m, p))
    for i in range(m):
        for j in range(p):
            for k in range(n):
                C[i, j] += A[i, k] * B[k, j]
    return C

def mat_mul_inner(A, B):
    m, n = A.shape
    n, p = B.shape
    C = np.zeros((m, p))
    for i in range(m):
        C[i, :] = np.dot(A[i, :], B)
    return C

def mat_mul_outer(A, B: np.array) -> np.array:
    result = np.zeros((A.shape[0], B.shape[1]))
    for j in range(A.shape[1]):
        result += np.outer(A[:, j], B[:, j])
    return result

def mat_mul_np(A, B: np.array) -> np.array:
    return A @ B
```

Then we test our functions for returning the same result:
Testing each function using timeit
And finally we visualize it on graph:

```python
def TestMatrices(n_max):
    for n in range(1, n_max):
        m = np.random.randint(1, n_max)
        p = np.random.randint(1, n_max)
        A = np.random.randn(m, n)
        B = np.random.randn(n, p)
        prod_loop = mat_mul_loop(A, B)
        prod_inner = mat_mul_inner(A, B)
        prod_outer = mat_mul_outer(A, B)
        prod_np = mat_mul_np(A, B)
        assert np.allclose(prod_loop, prod_inner)
        assert np.allclose(prod_loop, prod_np)
        assert np.allclose(prod_loop, prod_outer)
```

```
%%timeit
mat_mul_loop(A, B)
```

✓ 3.9s

```
502 ms ± 48.7 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

```
%%timeit
mat_mul_inner(A, B)
```

✓ 2.7s

```
3.42 ms ± 136 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

```
%%timeit
mat_mul_outer(A, B)
```

✓ 14.5s

```
1.78 ms ± 58.3 µs per loop (mean ± std. dev. of 7 runs, 1,000 loops each)
```

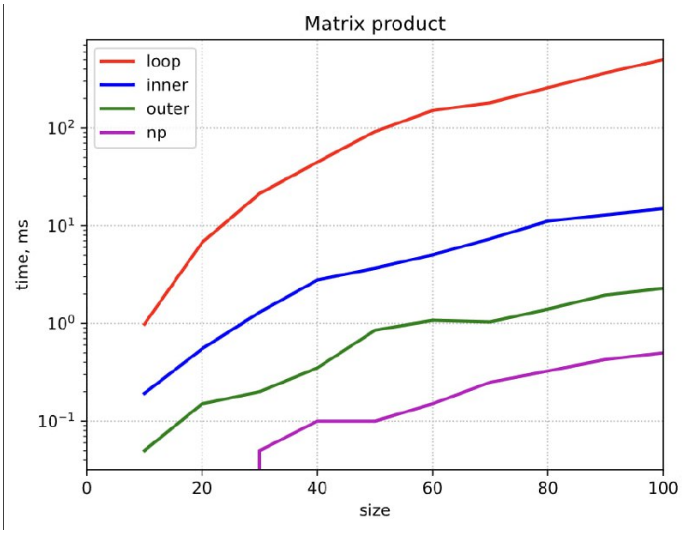
```
%%timeit
mat_mul_np(A, B)
```

✓ 5.5s

```
68 µs ± 3.17 µs per loop (mean ± std. dev. of 7 runs, 10,000 loops each)
```

8 Conclusion

Overall, by using NumPy, developers can save time in both development and execution, making it an essential tool for data analysis, scientific computing, and machine learning in Python.


Matrix product
loop
inner
outer
np
time, ms
size
10^2
10^1
10^0
10^{-1}
0
20
40
60
80
100