Machine Learning Algorithms

The fundamental goal of a machine learning model is to find a relationship between a set of input data points, called features, and an output data point, called a label. For example, a model might be designed to find a relationship between a person's credit score, income, net worth, and other values -- the features -- and how much money they will be able to pay back to the bank if they take out a loan -- the label.
To do this, the creators of a machine learning model first curate a set of data points, known as training data, in which values for the feature variables are paired with values for the label. Then, an algorithm is used to find a relationship between the known datapoints for the feature variables and the known datapoints for the label variable; then, as long as the data in the training set is representative of the actual data in the real world, the relationship that was found can be applied to values for the feature variables that were found in real-world contexts to find values for the label variables that aren't already known.
The algorithm a model is trained by functions by minimizing the output of a chosen loss function, which defines the how error-prone a model is. One example of a commonly used loss function is the L_2 loss function, which defines the error as equal to the square of the difference between a label's value and the value that the model's relationship predicts for the label based on the given feature values -- so, if y is the label's actual value $O(x)$ is the model's predicted value for y given the feature variables' values x, then $L_2(O, y) = (O - y)^2$.
(Google, 2025)

Gradient Descent

One simple method for finding correlations between features and labels is gradient descent. This algorithm finds a number corresponding to each feature, called weights, and one additional number, called a bias, such that when all of the feature values are multiplied by their weights and the bias is added, the loss is minimized, and the resulting number is as close to the corresponding label value as possible.
A gradient descent function can also apply an activation function to this output number. An activation function is any nonlinear function that takes in a single number as an input and gives out a single number as an output; processing the output of a gradient descent model through an activation function allows the model to find more complex, nonlinear relationships.
For example, one activation function which is particularly easy to understand and compute is the leaky rectified linear unit function $f(x) = \frac{x+0.9|x|}{1.9}$, which returns x without any change if it is positive but divides it by 10 if it is negative. Another popular activation function is the sigmoid function $f(x) = \frac{1}{1+e^x}$, which always returns a number between 0 and 1; this function is particularly helpful when the labels that a model is trying to predict are boolean (i.e. either 0 or 1).
Gradient descent works by taking advantage of the fact that because the functions that are used to calculate the loss using the weights and bias of the model are known, it is possible to calculate the derivative of the loss function with respect to these weights. Then, because these derivatives indicate how changing different weights would affect a model's loss, the weights can be changed in such a way that the loss will be decreased. So, gradient descent can be carried out by repeating the following three steps until the model has begun achieving acceptably low loss values across as many feature-label pairs as possible:

1. Pick a feature-label pairing from the training data set
2. Calculate the partial derivative of the loss for this pairing with respect to each weight
3. For each weight, subtract the loss for this weight found in the previous step from that weight's value
   Note that if the weights of the model change too much in each iteration of step 3, then the model might overshoot the label, causing the loss to increase instead of decreasing. Additionally, the model could be

changed so much in each step that the effects of previous iterations of the step would become irrelevant, meaning that the model might match each feature-label pairing well after it was trained on it, but not other pairings it had been trained on previously -- so, it could only accurately predict one data point in the training set at a time. So, before each weight is modified by the loss, that loss must be decreased according to a scalar whose value is known as the learning rate. Finding the optimal learning rate is important for training the neural network, as if the learning rate is too slow, the model will take a long time to find an accurate relationship, but if it is too high, the relationship it finds might be inaccurate and not take all training data into account.
(Google, 2025)

Calculation of the Derivative of Loss with Respect to Weight

Let $w_1, \ldots, w_t$ be weights and w_0 be a bias, $x_1, \ldots, x_t$ be feature values and $x_0 = 1$, $f(x)$ be an activation function, y be a label value, and $L(O, y)$ be a loss function. Then the output of the model will be $O(x) = f(1w_0 + \sum_{n=1}^{t} w_n x_n) = f(\sum_{n=0}^{t} w_n x_n)$, and the derivative of loss with respect to weight can be calculated as:

$$
\begin{aligned}
&\frac{\partial L}{w_t} \\
&= \frac{\partial}{\partial w_t} L(O(x), y) \\
&= \frac{\partial}{\partial w_t} L(f(\sum_{n=0}^{t} w_n x_n), y) \\
&L'(f(\sum_{n=0}^{t} w_n x_n), y) \frac{\partial}{\partial w_t} f(\sum_{n=0}^{t} w_n x_n) \\
&L'(O(x), y) f'(\sum_{n=0}^{t} w_n x_n) \frac{\partial}{\partial w_t} \sum_{n=0}^{t} w_n x_n \\
&L'(O(x), y) f'(\sum_{n=0}^{t} w_n x_n) x_n
\end{aligned}
$$

For example, with the L_2 loss function and no activation function:

$$
\begin{aligned}
L(O, y) &= (O - y)^2 \\
L'(O, y) &= 2O - 2y \\
f(x) &= x \\
f'(x) &= 1 \\
\frac{\partial L}{w_t} &= L'(O(x), y) f'(\sum_{n=0}^{t} w_n x_n) x_n \\
\frac{\partial L}{w_t} &= (2O(x) - 2y)(1) x_n \\
\frac{\partial L}{w_t} &= 2x_n(\sum_{n=0}^{t} (w_n x_n) - y)
\end{aligned}
$$

(Google, 2025)

Neural Networks

An issue with the gradient descent method is that it only finds linear relationships between feature variables and the input to the model's activation function. So, if gradient descent was used to try to find the relationship $y = (x_1)^2 + x_2$, where y is the label variable and x_1 and x_2 are feature variables, it would not find an accurate approximation -- it could find the approximation $y = (1x_1 + 0x_2)^2$ or the approximation $y = (0x_1 + 1x_2)$ if a

quadratic or linear, respectively, activation function was used, but neither of these relationships would be accurate outside of a limited range of values.
The solution to this problem is to create synthetic features, which are features derived from other features, and allow the model to find a relationship between this expanded set of feature variables and the label variables. So, if $x_3 = (x_1)^2$, then the model in the above example could find the approximation $y = (0x_1 + 1x_2 + 1x_3)$, which is a much better approximation for $y = (x_1)^2 + x_2$.
However, it is impractical to create every possible synthetic feature that might be relevant to a machine learning model's approximation for a label value. So, instead of trying to create a synthetic feature for every possible combination of nonlinearities and non-synthetic features, most advanced neural networks use neurons, which are synthetic features that can be modified by the model's algorithm to find the synthetic features that will minimize loss.
Each neuron acts similarly to a simple neuron network programmed via gradient descent. It has its own set of weights and a bias which it applies to generate a linear combination of the feature variables, and then applies an activation function to introduce nonlinearity into the equation. The weights and bias can then be modified by an algorithm to reduce the approximation's error.
Additionally, neurons can use other neurons as inputs to find more complex relationships between feature and label data. To this end, neurons are grouped into sets called hidden layers which each take the outputs of the neurons in the previous hidden layer, processes them according to their weights, biases, and activation functions, and outputs a set of data points which can be used by later hidden layers. This process of taking a set of feature data points and processing it through a series of hidden layers is called forward propagation.
(Google, 2025)

Forward Propagation Calculation

Let $o_{a,n}$ be the output of the ath neuron in the nth hidden layer of a neural network where each layer has t_n total neurons, $f_{a,n}(x)$ be the activation function used by that neuron, and $w_{a,b,n}$ be a weight used by that neuron which will be multiplied with the output of the bth neuron in the previous layer -- i.e. $o_{b,n-1}$ -- with $w_{a,0,n}$ being that neuron's bias. Then the output of layer n can be calculated as follows:

$$\begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_{t_n} \end{bmatrix} = \begin{bmatrix} w_{1,0,n} & w_{1,1,n} & \dots & w_{1,t_{n-1},n} \\ w_{2,0,n} & w_{2,1,n} & \dots & w_{2,t_{n-1},n} \\ \vdots & \vdots & & \vdots \\ w_{t_n,0,n} & w_{t_n,1,n} & \dots & w_{t_n,t_{n-1},n} \end{bmatrix} \begin{bmatrix} 1 \\ o_{1,n-1} \\ o_{2,n-1} \\ \vdots \\ o_{t_{n-1}} \end{bmatrix}$$

$$\begin{bmatrix} o_{1,n} \\ o_{2,n} \\ \vdots \\ o_{t_n,n} \end{bmatrix} = \begin{bmatrix} f_{1,n}(s_1) \\ f_{2,n}(s_2) \\ \vdots \\ f_{t_n,n}(s_{t_n}) \end{bmatrix}$$

(Google, 2025)

Back Propagation

Because the calculations involved with neural networks are more complex than the calculations involved with simple gradient-descent models, finding the derivative of the loss function with respect to an arbitrary weight directly is more challenging than finding the equation used for gradient descent above. Instead of a direct calculation, this derivative is found via several intermediate steps:

1. Find the derivative of the loss function with respect to the outputs of the final layer of neurons
2. Find the derivative of the output of a neuron with respect to the outputs of the neurons in the next layer

    1. Repeat step 2 for each layer, beginning with the final layer and ending with the first layer
3. Find the derivative of the weights of a neuron with respect to the output of that neuron
   The values found in these steps can then be used to calculate the derivative of the loss with respect to a weight.
   (Wikipedia, 2025)

Back Propagation Calculation

Let $o_{a,n}$, $f_{a,n}(x)$, $w_{a,b,n}$, and t_n have the same meanings as in the calculations that were performed for forward propagation, l be the total number of layers in the neural network, and $L(O,y)$ be the loss function for the neural network. Then, let $s(a,n)$ be the sum of a neuron's weights times the outputs of the neurons in the previous hidden layer (i.e. $s(a,n)=\sum_{m=1}^{t_{n-1}} w_{a,m,n}o_{m,n-1}$) so that $o_{a,n}=f_{a,n}(s(a,n))$.
Then, according to the chain rule, the derivative of loss with respect to a particular weight can be found via the following system of equations:

$$\begin{cases} \frac{\partial L}{o_{a,n}} = \frac{\partial L}{\partial o_{a,n}} & n=l \\ \frac{\partial L}{o_{a,n}} = \sum_{m=1}^{t_{n+1}} \frac{\partial L}{\partial o_{m,n+1}} \frac{\partial o_{m,n+1}}{o_{a,n}} & 1 \le n < l \\ \frac{\partial L}{\partial w_{a,b,n}} = \frac{\partial L}{\partial o_{a,n}} \frac{\partial o_{a,n}}{\partial w_{a,b,n}} \end{cases}$$

Which can be evaluated as follows:

$$\begin{aligned} &\frac{\partial L}{o_{a,n}} \quad n=l \\ &= \frac{\partial}{\partial o_{a,n}} L(y, f_{\text{output},n+1}(s(\text{output},n+1)) \\ &= L'(y, f_{\text{output},n+1}(s(\text{output},n+1))) \frac{\partial}{\partial o_{a,n}} f_{\text{output},n+1}(s(\text{output},n+1)) \\ &= L'(y, f_{\text{output},n+1}(s(\text{output},n+1))) f'_{\text{output},n+1}(s(\text{output},n+1)) \frac{\partial}{\partial o_{a,n}} s(\text{output},n+1) \\ &= L'(y, f_{\text{output},n+1}(s(\text{output},n+1))) f'_{\text{output},n+1}(s(\text{output},n+1)) \frac{\partial}{\partial o_{a,n}} \sum_{m=0}^{t_n} w_{\text{output},m,n} o_{m,n} \\ &= L'(y, f_{\text{output},n+1}(s(\text{output},n+1))) f'_{\text{output},n+1}(s(\text{output},n+1)) w_{\text{output},a,n} \end{aligned}$$

$$\begin{aligned} &\frac{\partial L}{o_{a,n}} \quad 1 \le n \le l \\ &= \sum_{m=1}^{t_{n+1}} \frac{\partial L}{\partial o_{m,n+1}} \frac{\partial o_{m,n+1}}{o_{a,n}} \\ &= \sum_{m=1}^{t_{n+1}} \frac{\partial L}{\partial o_{m,n+1}} \frac{\partial}{o_{a,n}} f_{m,n+1}(s(m,n+1)) \\ &= \sum_{m=1}^{t_{n+1}} \frac{\partial L}{\partial o_{m,n+1}} f'_{m,n+1}(s(m,n+1)) \frac{\partial}{o_{a,n}} s(m,n+1) \\ &= \sum_{m=1}^{t_{n+1}} \frac{\partial L}{\partial o_{m,n+1}} f'_{m,n+1}(s(m,n+1)) \frac{\partial}{o_{a,n}} \sum_{p=0}^{t_n} w_{m,p,n+1} o_{p,n} \\ &= \sum_{m=1}^{t_{n+1}} \frac{\partial L}{\partial o_{m,n+1}} f'_{m,n+1}(s(m,n+1)) w_{m,a,n+1} \end{aligned}$$

$$
\begin{aligned}
&\frac{\partial L}{\partial w_{a,b,n}} \\
&= \frac{\partial L}{\partial o_{a,n}} \frac{\partial o_{a,n}}{\partial w_{a,b,n}} \\
&= \frac{\partial L}{\partial o_{a,n}} \frac{\partial}{\partial w_{a,b,n}} f_{a,n}(s(a,n)) \\
&= \frac{\partial L}{\partial o_{a,n}} f'_{a,n}(s(a,n)) \frac{\partial}{\partial w_{a,b,n}} s(a,n) \\
&= \frac{\partial L}{\partial o_{a,n}} f'_{a,n}(s(a,n)) \frac{\partial}{\partial w_{a,b,n}} \sum_{m=0}^{t_{n-1}} w_{a,m,n} o_{m,n-1} \\
&= \frac{\partial L}{\partial o_{a,n}} f'_{a,n}(s(a,n)) o_{b,n-1}
\end{aligned}
$$

This results in the final sequence of equations:

$$
\begin{cases}
\frac{\partial L}{o_{a,n}} = L'(y, f_{\text{output},n+1}(s(\text{output}, n+1))) f'_{\text{output},n+1}(s(\text{output}, n+1)) w_{\text{output},a,n} & n = l \\
\frac{\partial L}{o_{a,n}} = \sum_{m=1}^{t_{n+1}} \frac{\partial L}{\partial o_{m,n+1}} f'_{m,n+1}(s(m, n+1)) w_{m,a,n+1} & 1 \leq n < l \\
\frac{\partial L}{\partial w_{a,b,n}} = \frac{\partial L}{\partial o_{a,n}} f'_{a,n}(s(a,n)) o_{b,n-1}
\end{cases}
$$

(Wikipedia, 2025)

Example Code

An example of a neural network implementing backpropagation can be found here:
https://github.com/MasterJLord/linalg_ML

Dimensionality Reduction

An important step in the process of creating a machine learning model is feature engineering. This is the process of modifying the data that a model is trained on to make it easier to learn from. So, removing outliers from a training data set and binning numerical variables into categorical variables are both examples of feature engineering.
One important technique that is used in feature engineering is dimensionality reduction. This is the process of changing the basis of a dataset to a lower dimension one, so that the model does not have to optimize as many weights as it would otherwise. For example, dimensionality reduction might reduce the following dataset in the standard basis:

$$
\begin{bmatrix}
1 & 3 & 2 \\
2 & -3 & -4 \\
2 & 5 & 3 \\
\vdots & \vdots & \vdots
\end{bmatrix}
$$

into the following simpler dataset with fewer variables:

$$
\begin{bmatrix}
1 & 2 \\
2 & -4 \\
2 & 3 \\
\vdots & \vdots
\end{bmatrix}
$$

by finding the following basis:

$$
\begin{bmatrix}
1 & 1 & 0 \\
0 & 1 & 1
\end{bmatrix}
$$

By converting this reduced dataset into the original standard basis, we can see that it is a close approximation of the original dataset, although not perfect:

$$\begin{bmatrix} 1 & 2 \\ 2 & -4 \\ 2 & 3 \\ \vdots & \vdots \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 3 & 2 \\ 2 & -2 & -4 \\ 2 & 5 & 3 \\ \vdots & \vdots & \vdots \end{bmatrix}$$

Therefore, the process of dimensionality reduction is finding a simpler basis that will result in the closest approximation and the least data loss when the data is converted into it.
(Google, 2025)

Principal Component Analysis

A common method for dimensionality reduction is principal component analysis, or PCA. This method is performed in two steps:

1. Find the product of the transpose of the dataset times the original dataset, then
2. Find the eigenvectors of this product matrix with the highest eigenvalues.
   So, if the training dataset is A, then the basis found by PCA will be the n highest-eigenvalued eigenvectors of A^tA.
   Note that this strategy assumes that the rows of the training dataset matrix represent datapoints, and the columns of the dataset represent variables. If this situation is reversed so that the columns of the dataset represent datapoints and the rows represent variables, as it was in the example dataset above, then the method of PCA must be applied to the transpose of the matrix rather than the matrix itself -- so, the basis will be the highest-eigenvalued n eigenvectors of $(A^t)^t(A^t) = AA^t$.
   (Laity, 2018)

Singular Value Decomposition

One flaw in principal component analysis is that it involves multiplying A by its own transpose. Calculating this matrix can be problematic because it involves multiplying the values of A by themselves and each other: if some values of A are as high or as low or as precise as the number format that they are being stored in can handle, then calculating their products might cause number overflows or underflows, or a loss of precision. For example, if a computer is storing each value in the matrix A below as a short integer to save space, then it can only store numbers between $-32\,768$ and $32\,767$; therefore, calculating A^tA will result in multiple products that cannot be stored properly:

$$A = \begin{bmatrix} 11 & 281 \\ -147 & 313 \\ 40 & 96 \end{bmatrix}$$
$$A^tA = \begin{bmatrix} 23\,330 & -39\,080 \\ -39\,080 & 186\,146 \end{bmatrix}$$

Fortunately, there is a way to find the eigenvalues of A^tA without actually multiplying the two matrices together. This is by finding the singular value decomposition, or SVD, of A.
The singular value decomposition is a type of matrix decomposition that breaks A down into three matrices called V, D, and U^t with the following properties:

- D is a diagonal matrix with the square roots of the eigenvalues of AA^t (which are also the values of A^tA) on its main diagonal, sorted from highest to lowest

- V is an orthogonal matrix whose columns are eigenvectors of AA^t, sorted from highest eigenvalue to lowest eigenvalue
- U^t is an orthogonal matrix whose columns are eigenvectors of A^tA, sorted from highest eigenvalue to lowest eigenvalue
- $VDU^t = A$
  Because U^t is composed of the eigenvectors of A^tA as well as the eigenvalues that correspond to them, any method that finds the singular value decomposition of A will also find the principal component analysis of that matrix. This is important because it is possible to find the singular value decomposition of a matrix without multiplying A by A^t, and therefore, it is possible to calculate the PCA and reduce the dimensionality of a matrix with a reduced risk of suffering calculation errors.
  (J. M. ain't a mathematician, 2013)

Proofs of the Validity of PCA and SVD

The proofs of the validity of Principal Component Analysis and Singular Value Decomposition rely on several other theorems relating to projection matrices, traces, the norm of Frobenius, symmetric matrices, and orthogonal matrices. I will prove these other theorems here, before finally proving that PCA works and the SVD can be found for any matrix.

Definition of a Projection Matrix

A projection matrix is a matrix B' such that, for a given subspace B in $\mathbb{R}^m$ and any mxn matrix A, $\text{proj}_B(A) = B'A$

Definition of a Trace

The trace of a matrix is equal to the sum of the elements on the main diagonal of a square matrix; so:
$\text{tr}(A) = \sum_{i=1}^{n} A_{i,i}$
(Penney, 2016)

Definition of a Norm of Frobenius

The norm of Frobenius of an mxn matrix A and an mxn matrix B, written $\langle A, B \rangle$, is the sum of the products of all of the elements in A and all of the elements in the same position in B: so, $\langle A, B \rangle = \sum_{i=1}^{m} \sum_{j=1}^{n} A_{i,j} B_{i,j}$.
(Johnston, 2020)

Definition of a Symmetric Matrix

A symmetric matrix is a matrix that is equal to its transform. So, a matrix A is symmetric if and only if $A = A^t$.
(Penney, 2016)

Definition of a Partially Orthogonal Matrix

A partially orthogonal matrix is a non-square matrix whose column vectors form an orthonormal matrix. So, by applying the same proof that is used to a non-square matrix A if partially orthogonal if and only if $AA^t = A^tA = I$.
(Penney, 2016)

Definition of the Reduced SVD

The reduced singular value decomposition is a set of 3 matrices: the diagonal matrix D and the partially orthogonal matrices V and U^t and the diagonal matrix D such that $VDU^t = A$.
(Penney, 2016)

Theorem 1: Calculation of a Projection Matrix

Let A be an mxn matrix with columns $C_1, \ldots, C_n$ and let B be a k-dimensional subspace in $\mathbb{R}^m$ with the orthogonal basis $B_1, \ldots, B_k$.
The matrix B' made by projecting each column in A onto B can be written as follows:

$$\begin{aligned}
&\text{proj}_B(A) \\
&= \left[\sum_1^k \tfrac{C_1 \cdot B_i}{B_i \cdot B_i} B_i \quad \ldots \quad \sum_1^k \tfrac{C_n \cdot B_i}{B_i \cdot B_i} B_i\right] \\
&= \sum_1^k \left[\tfrac{1}{B_i \cdot B_i}(C_1 \cdot B_i)B_i \quad \ldots \quad \tfrac{1}{B_i \cdot B_i}(C_n \cdot B_i)B_i\right] \\
&= \sum_1^k \left[\tfrac{1}{B_i \cdot B_i} B_i(B_i \cdot C_1) \quad \ldots \quad \tfrac{1}{B_i \cdot B_i} B_i(B_i \cdot C_n)\right] \\
&= \sum_1^k \left[\tfrac{1}{B_i \cdot B_i} B_i(B_i^t C_1) \quad \ldots \quad \tfrac{1}{B_i \cdot B_i} B_i(B_i^t C_n)\right] \\
&= \sum_1^k \left[\tfrac{1}{B_i \cdot B_i}(B_i B_i^t)C_1 \quad \ldots \quad \tfrac{1}{B_i \cdot B_i}(B_i B_i^t)C_n\right] \\
&= (\sum_1^k \frac{1}{B_i \cdot B_i} B_i B_i^t)\,[C_1 \quad \ldots \quad C_n] \\
&= (\sum_1^k \frac{B_i B_i^t}{B_i \cdot B_i})A
\end{aligned}$$

So, the projection matrix onto B will equal $\sum_1^k \frac{B_i B_i^t}{B_i \cdot B_i}$.

Theorem 2: Symmetry of a Projection Matrix

From theorem 1, if B' is a projection matrix:

$$\begin{aligned}
&B'_{j,k} \\
&= (\sum_{i=1}^k B_i B_i^t)_{j,k} \\
&= \sum_{i=1}^k (B_i)_j (B_i)_k \\
&= \sum_{i=1}^k (B_i)_k (B_i)_j \\
&= (\sum_{i=1}^k B_i B_i^t)_{k,j} \\
&= B'_{k,j}
\end{aligned}$$

So, $B' = (B')^t$.

Theorem 3: Square of a Projection Matrix

Because $B' = \sum_1^k \frac{B_i B_i^t}{B_i \cdot B_i}$, each column in B' can be written as a linear combination of the vectors in B. Therefore, B' is already in B, and so $\text{proj}_B(B') = B'$, so $B'B' = B'$.

Theorem 4: Distributive Property of a Trace

Let A and B be square matrices. Then, from theorem 1:

$$\begin{aligned}
&\operatorname{tr}(A+B) \\
&=\sum_{i=1}^{n}((A+B)_{i,i}) \\
&=\sum_{i=1}^{n}(A_{i,i}+B_{i,i}) \\
&=\sum_{i=1}^{n}(A_{i,i})+\sum_{i=1}^{n}(B_{i,i}) \\
&=\operatorname{tr}(A)+\operatorname{tr}(B)
\end{aligned}$$

So, $\operatorname{tr}(A+B)=\operatorname{tr}(A)+\operatorname{tr}(B)$.

Theorem 5: Commutative Property of a Trace

Let A be an mxn matrix and and B is an nxm matrix:

$$\begin{aligned}
&\operatorname{tr}(AB) \\
&=\sum_{i=1}^{m}(\sum_{j=1}^{n}A_{i,j}B_{j,i}) \\
&=\sum_{i=1}^{m}(\sum_{j=1}^{n}B_{j,i}A_{i,j}) \\
&=\sum_{j=1}^{n}(\sum_{i=1}^{m}B_{j,i}A_{i,j}) \\
&=\operatorname{tr}(BA)
\end{aligned}$$

So, $\operatorname{tr}(AB)=\operatorname{tr}(BA)$.

Theorem 6: Norm of Frobenius and Trace

$\langle A,B\rangle$ is equal to the trace of AB^t:

$$\begin{aligned}
&\operatorname{tr}(AB^t) \\
&=\sum_{i=1}^{m}((AB^t)_{i,i}) \\
&=\sum_{i=1}^{m}(\sum_{j=1}^{n}(A_{i,j}B^t_{j,i})) \\
&=\sum_{i=1}^{m}(\sum_{j=1}^{n}(A_{i,j}B_{i,j})) \\
&=\langle A,B\rangle
\end{aligned}$$

(Johnston, 2020)

Theorem 7: AA^t and symmetric matrices

For any matrix A: $(AA^t)^t=(A^t)^t(A)^t=AA^t$
Therefore, AA^t (and, consequently, A^tA, which is $(A^t)(A^t)^t$) will be symmetric matrices.
(Penney, 2016)

Theorem 8: Dot Product of Symmetric Matrices

Let A be a symmetric matrix. Then, because $x \cdot y = x^t y$:

$$\begin{aligned} &Ax \cdot y \\ &= (Ax)^t y \\ &= x^t A^t y \\ &= x \cdot A^t y \\ &= x \cdot Ay \end{aligned}$$

This means that, if A is symmetric, then $Ax \cdot y = x \cdot Ay$.
(Penney, 2016)

Theorem 9: Orthogonality of Eigenvectors of Symmetric Matrices

Let A be a symmetric matrix and x and y be eigenvectors of A so that $Ax = \lambda_1 x$ and $Ay = \lambda_2 y$. Then, either $\lambda_1 = \lambda_2$ or $\lambda_1 \neq \lambda_2$.
If $\lambda_1 = \lambda_2$: x and y (along with any other eigenvectors which also share this eigenvalue) form a basis for A's λ_1-eigenspace. An orthonormal basis can be chosen for this vector space using the Gram Schmidt Process, and since these vectors will also be in the eigenspace, they will also be λ_1-eigenvectors of A.
Alternatively, if $\lambda_1 \neq \lambda_2$, then theorem 8 can be used to prove that $(\lambda_1 - \lambda_2)(x \cdot y) = 0$, as follows:

$$\begin{aligned} &\lambda_1 (x \cdot y) \\ &= (\lambda_1 x) \cdot y \\ &= (Ax) \cdot y \\ &= x \cdot (Ay) \\ &= x \cdot (\lambda_2 y) \\ &= \lambda_2 (x \cdot y) \end{aligned}$$

$$\begin{aligned} \lambda_1 (x \cdot y) &= \lambda_2 (x \cdot y) \\ (\lambda_1 - \lambda_2)(x \cdot y) &= 0 \end{aligned}$$

This means that either $\lambda_1 - \lambda_2$ or $x \cdot y$ must equal 0. Because $\lambda_1 \neq \lambda_2$, $\lambda_1 - \lambda_2$ must not equal 0. This means that $x \cdot y$ must equal 0, so x and y are orthogonal.
Therefore, it is possible to pick eigenvectors for A such that all eigenvectors of A are orthogonal.
(Penney, 2016)

Theorem 10: The Spectral Theorem

Let A be an nxn matrix. Then, let W be an A-invariant subspace, i.e. a subspace such that for any vector x in W, Ax is also in W.
In this case, A is a linear transformation that maps each vector $x_{\text{pre }1}, x_{\text{pre }1}, \ldots$ in $\mathbb{R}^n$ that belongs to the subspace W onto another vector $x_{\text{post }1}, x_{\text{post }2}, \ldots$ in $\mathbb{R}^n$ that also belongs to the subspace W. We can then convert each of these vectors into the basis of W, naming them $x'_{\text{pre }1}, x'_{\text{pre }2}, \ldots$ and $x'_{\text{post }1}, x'_{\text{post }2}, \ldots$, and define a linear transformation B that maps $x'_{\text{pre }1}$ onto $x'_{\text{post }1}$ and $x'_{\text{pre }2}$ onto $x'_{\text{post }2}$ and so on just as A would have, but is only applicable to vectors in that basis, not to vectors in $\mathbb{R}^n$. Since every matrix has at least one eigenvector, B has at least one eigenvector $x'_{\text{pre }i}$. So, $x'_{\text{post }i} = \lambda x'_{\text{pre }i}$. We can then convert this vector in W back to the original vector in $\mathbb{R}^n$ by multiplying it by the point matrix $P_{W \to \mathbb{R}^n}$:

$$\begin{aligned} P_{W \to \mathbb{R}^n} x'_{\text{post }i} &= P_{W \to \mathbb{R}^n} \lambda x'_{\text{pre }i} \\ P_{W \to \mathbb{R}^n} x'_{\text{post }i} &= \lambda P_{W \to \mathbb{R}^n} x'_{\text{pre }i} \\ x_{\text{post }i} &= \lambda x_{\text{pre }i} \end{aligned}$$

Therefore, A has at least one eigenvector in W.
The orthogonal complement $W_1^{\perp}$ of W to this eigenvector will also be A-invariant if and only if, for each x in $W_1^{\perp}$,

Ax is also in $W_1^\perp$; this is the case if and only if, for each x in W that is orthogonal to $x_{\text{pre}\,i}$, Ax is also in W and also orthogonal to $x_{\text{pre}\,i}$. It is the case that Ax will be in W because x was in W and W is A-invariant, and as long as A is symmetric, it is the case that x is orthogonal to $x_{\text{pre}\,i}$ because $Ax \cdot y = x \cdot y$ for symmetric matrices:
$Ax \cdot x_{\text{pre}\,i} = x \cdot Ax_{\text{pre}\,i} = x \cdot \lambda x_{\text{pre}\,i} = \lambda(x \cdot x_{\text{pre}\,i}) = 0$. This means that after we pick an eigenvector e_1 of A in the A-invariant subspace W and take the orthogonal complement $W_1^\perp$ of W to e_1, we will be able to pick an eigenvector e_2 of A in the A-invariant subspace $W_1^\perp$ and take the orthogonal complement $W_2^\perp$ of $W_1^\perp$ to e_2, and so on, repeating this process as many times as we would like.
By using the Gram Schmidt process, we can pick an orthogonal basis $W_1, \ldots, W_a$ for each $W_p^\perp$ such that W_1 is the chosen eigenvector. Then, the orthogonal complement $W_p^\perp$ to the chosen eigenvector (i.e. W_1) in $W_{p-1}^\perp$ will have the basis $W_2, \ldots, W_a$ and will therefore have a multiplicity exactly one less than the multiplicity of $W_{p-1}^\perp$. This means that we can repeat the process m times before we we reach a vector space $W_m^\perp$ with multiplicity 0, i.e. a vector space with no elements in it, where m is the multiplicity of the original vector subspace W. This will ultimately leave us with m orthogonal eigenvectors of A.
Because A is symmetric, it must be square, meaning that it maps vectors in $\mathbb{R}^n$ onto vectors in $\mathbb{R}^n$, so $\mathbb{R}^n$ is A-invariant. $\mathbb{R}^n$ has multiplicity n, and therefore we can find n linearly independent, orthogonal eigenvectors of A. Finally, this means that A is diagonalizable, and so we can find the diagonal matrix $D = Q^{-1}AQ = Q^tAQ$ where the orthogonal matrix Q is composed of the eigenvectors of A and the elements on the main diagonal of D are the eigenvalues of A corresponding to the eigenvectors in Q. These Q and D lead to the expression QDQ^t, which is the spectral decomposition of A, and is guaranteed to exist for any A as long as A is a symmetric matrix.

Furthermore, let λ be a non-real eigenvalue of the real matrix A, and let $x = [a_1 + b_1 i \quad \ldots \quad a_n + b_n i]^t$ be the corresponding eigenvector. Let $\bar{x}$ be the complex conjugate of x. Because the complex conjugate of every complex eigenvector of a real matrix A is also an eigenvector of A, $\bar{x}$ is also an eigenvector of A. Then, we can find $x \cdot \bar{x} = (a_1 + b_1 i)(a_1 - b_1 i) + \cdots + (a_n i = +b_n i)(a_n - b_n i) = a_1^2 + b_1^2 + \cdots + a_n^2 + b_n^2$. This dot product can be zero only when each of its terms is 0, which can only be the case only if $x = \bar{x} = 0$. Since eigenvectors can never be zero, this means that complex eigenvectors of real matrices can never be orthogonal to their complex conjugates. Since eigenvectors of symmetric matrices with different eigenvalues must always be orthogonal to each other due to theorem 9, and complex conjugate eigenvectors always have complex conjugate (i.e. differing) eigenvalues, this means that real symmetric matrices will have exclusively real eigenvectors. So, Q and D are guaranteed to not only exist, but also be real.
(Penney, 2016)

Theorem 11: Proof of Principal Component Analysis

Let A be an mxn matrix and let B be an n'-dimensional subspace in $\mathbb{R}^m$. Let $B_1, \ldots, B_{n'}$ be an orthonormal basis for B.
The nearest matrix to A whose columns are all in B is $\text{proj}_B(A)$. If B' is the projection matrix of B, then $\text{proj}_B(A) = B'A$. So, the euclidean distance between A and the nearest matrix to A whose columns are in B is the square root of the sum of the squares of the elements of $A - B'A$. This is equal to the norm of Frobenius of

$A - B'A$ with itself. Using theorems 2, 3, 4, 5, and 6, this can be reduced as follows:

$$\begin{aligned}
&\langle A - B'A, A - B'A \rangle \\
&= \mathrm{tr}((A - B'A)(A - B'A)^t) \\
&= \mathrm{tr}((A - B'A)(A^t - (B'A)^t)) \\
&= \mathrm{tr}((A - B'A)(A^t - A^t B'^t)) \\
&= \mathrm{tr}((A - B'A)(A^t - A^t B')) \\
&= \mathrm{tr}(AA^t - AA^t B' - B'AA^t + B'AA^t B') \\
&= \mathrm{tr}(AA^t) - \mathrm{tr}(AA^t B') - \mathrm{tr}(B'AA^t) + \mathrm{tr}(B'AA^t B') \\
&= \mathrm{tr}(AA^t) - \mathrm{tr}(B'AA^t) - \mathrm{tr}(B'AA^t) + \mathrm{tr}(B'B'AA^t) \\
&= \mathrm{tr}(AA^t) - 2\mathrm{tr}(B'AA^t) + \mathrm{tr}(B'AA^t) \\
&= \mathrm{tr}(AA^t) - \mathrm{tr}(B'AA^t)
\end{aligned}$$

Therefore, the subspace B will be the basis such that $\mathrm{proj}_B(A)$ is as close to A as possible when $\mathrm{tr}(AA^t) - \mathrm{tr}(B'AA^t)$ is as low as possible. Since $\mathrm{tr}(AA^t)$ cannot be decreased by changing B', this will be the case when $\mathrm{tr}(B'AA^t)$ is as high as possible.

From the theorem 10, there exists matrices Q and D such that D is a diagonal matrix with the eigenvalues of AA^t on its main diagonal, Q is an orthogonal matrix made of the corresponding eigenvectors, and $QDQ^t = AA^t$. Then, if Q_j is the jth column of Q, it is possible to reduce $\mathrm{tr}(B'AA^t)$ as follows by using theorems 1, 4, and 5:

$$\begin{aligned}
&\mathrm{tr}(B'AA^t) \\
&= \mathrm{tr}(B'QDQ^t) \\
&= \mathrm{tr}(\sum_{i=1}^{n'} \frac{B_i B_i^t}{||B_i||^2} QDQ^t) \\
&= \mathrm{tr}(\sum_{i=1}^{n'} \frac{B_i B_i^t}{1^2} QDQ^t) \\
&= \sum_{i=1}^{n'} \mathrm{tr}(B_i B_i^t QDQ^t) \\
&= \sum_{i=1}^{n'} \mathrm{tr}(Q^t B_i B_i^t QD) \\
&= \sum_{i=1}^{n'} \mathrm{tr}(Q^t B_i (Q^t B_i)^t D) \\
&= \sum_{i=1}^{n'} \mathrm{tr}(\begin{bmatrix} (Q^t B_i)_1 \\ \vdots \\ (Q^t B_i)_m \end{bmatrix} \begin{bmatrix} (Q^t B_i)_1 & \dots & (Q^t B_i)_m \end{bmatrix} \begin{bmatrix} \lambda_1 & \dots & 0 \\ \vdots & & \vdots \\ 0 & \dots & \lambda_m \end{bmatrix}) \\
&= \sum_{i=1}^{n'} \mathrm{tr}(\begin{bmatrix} (Q^t B_i)_1 \\ \vdots \\ (Q^t B_i)_m \end{bmatrix} \begin{bmatrix} (Q^t B_i)_1 \lambda_1 & \dots & (Q^t B_i)_m \lambda_1 \end{bmatrix}) \\
&= \sum_{i=1}^{n'} \mathrm{tr}(\begin{bmatrix} (Q^t B_i)_1 (Q^t B_i)_1 \lambda_1 & \dots & (Q^t B_i)_1 (Q^t B_i)_m \lambda_m \\ \vdots & & \vdots \\ (Q^t B_i)_m (Q^t B_i)_1 \lambda_1 & \dots & (Q^t B_i)_m (Q^t B_i)_m \lambda_m \end{bmatrix})
\end{aligned}$$

$$= \sum_{i=1}^{n'} \mathrm{tr}(\begin{bmatrix} (Q^t B_i)_1^2 \lambda_1 & \dots & (Q^t B_i)_1 (Q^t B_i)_m \lambda_m \\ \vdots & & \vdots \\ (Q^t B_i)_m (Q^t B_i)_1 \lambda_1 & \dots & (Q^t B_i)_m^2 \lambda_m \end{bmatrix})$$

$$= \sum_{i=1}^{n'} \sum_{j=1}^{m} (Q^t B_i)_j^2 \lambda_j$$

$$= \sum_{j=1}^{m} (\sum_{i=1}^{n'} (Q^t B_i)_j^2) \lambda_j$$

$$= \sum_{j=1}^{m} (||\begin{bmatrix} Q_j \cdot B_1 & \dots & Q_j \cdot B_{n'} \end{bmatrix}||^2) \lambda_j$$

$$= \sum_{j=1}^{m} (||Q_j^t \begin{bmatrix} B_1 & \dots & B_{n'} \end{bmatrix}||^2) \lambda_j$$

Let $c_j = ||Q_j^t \begin{bmatrix} B_1 & \dots & B_{n'} \end{bmatrix}||^2$, so that $\mathrm{tr}(B' A A^t) = \sum_{j=1}^{m} (c_j \lambda_j)$.
Because $B_1, \dots, B_{n'}$ is orthonormal, $\begin{bmatrix} B_1 & \dots & B_{n'} \end{bmatrix}$ is a partially orthogonal matrix. Let $B_{n'+1}, \dots, B_m$ be normal vectors orthogonal to $B_1, \dots, B_{n'}$ so that $\begin{bmatrix} B_1 & \dots & B_{n'} & B_{n'+1} & \dots & B_m \end{bmatrix}$ is an orthogonal matrix. Then:

$$||Q_j^t \begin{bmatrix} B_1 & \dots & B_{n'} & B_{n'+1} & \dots & B_m \end{bmatrix}||$$

$$= \sqrt{Q_j^t B_1 + \dots + Q_j^t B_n + Q_j^t B_{n'+1} + \dots + Q_j^t B_m}$$

$$= \sqrt{c_j + Q_j^t B_{n'+1} + \dots + Q_j^t B_m}$$

Therefore, $||Q_j^t \begin{bmatrix} B_1 & \dots & B_{n'} & B_{n'+1} & \dots & B_m \end{bmatrix}|| \geq c_j$. Additionally:

$$||Q_j^t \begin{bmatrix} B_1 & \dots & B_{n'} & B_{n'+1} & \dots & B_m \end{bmatrix}||$$

$$= ||(Q_j^t \begin{bmatrix} B_1 & \dots & B_{n'} & B_{n'+1} & \dots & B_m \end{bmatrix})^t||$$

$$||\begin{bmatrix} B_1 & \dots & B_{n'} & B_{n'+1} & \dots & B_m \end{bmatrix}^t Q_j||$$

So, $||\begin{bmatrix} B_1 & \dots & B_{n'} & B_{n'+1} & \dots & B_m \end{bmatrix}^t Q_j|| \geq c_j$. Because $\begin{bmatrix} B_1 & \dots & B_{n'} & B_{n'+1} & \dots & B_m \end{bmatrix}$ is an orthogonal matrix, its transpose will be orthogonal as well, and so because multiplication by an orthogonal matrix does not modify a vector's magnitude, $||Q_j|| \geq c_j$. Because Q is an orthogonal matrix, each column in it is normalized, and so $||Q_j|| = 1$. For that reason, and because c_j is a square of a magnitude and therefore cannot be negative, $0 \leq c_j \leq 1$.
Next, because Q is orthogonal and therefore does not modify the magnitude of vectors multiplied by it, and because the columns $B_1, \dots, B_{n'}$ are orthonormal and therefore their magnitudes are all 1, another property of

$c_1, \ldots, c_j$ can be found:

$$\begin{aligned}
&\sum_{j=1}^{m}(c_j) \\
&= \sum_{j=1}^{m} ||Q_j^t \begin{bmatrix} B_1 & \ldots & B_{n'} \end{bmatrix}||^2 \\
&= || \begin{bmatrix} ||Q_1^t \begin{bmatrix} B_1 & \ldots & B_{n'} \end{bmatrix}|| \\ \vdots \\ ||Q_m^t \begin{bmatrix} B_1 & \ldots & B_{n'} \end{bmatrix}|| \end{bmatrix} ||^2 \\
&= ||Q^t \begin{bmatrix} B_1 & \ldots & B_{n'} \end{bmatrix}||^2 \\
&= || \begin{bmatrix} B_1 & \ldots & B_{n'} \end{bmatrix}||^2 \\
&= || \begin{bmatrix} 1_1 & \ldots & 1_{n'} \end{bmatrix}||^2 \\
&= \sqrt{(\sum_{j=1}^{n'} 1)^2} \\
&\sum_{j=1}^{n'} 1 \\
&= n'
\end{aligned}$$

The sum $\sum_{j=1}^{m}(c_j \lambda_j)$ will be highest when the c_j's corresponding to the n' highest λ_j's are as high as possible, and from these two properties (i.e. $0 \leq c_j \leq 1$ and $\sum_{j=1}^{m}(c_j) = n'$), this will be the case when those n' c_j's are 1 and all other c_j's are 0. So, $\text{tr}(B'AA^t)$ is highest when it is equal to the sum of the n' highest eigenvalues of AA^t.
If $B_1, \ldots, B_n$ are orthogonal eigenvectors of AA^t, then Q can be chosen such that $Q_i = B_i$ for $1 \leq i \leq n'$. Then, because Q is an orthonormal matrix, a column c_j of Q will either be equal to B_i when $i = j$, in which case $c_j \cdot B_i = c_j \cdot c_j = 1$, or orthogonal to B_i, in which case $c_j \cdot B_i = 0$. So, $Q^t B_i$ will be equal to $\begin{bmatrix} 0_1 & \ldots & 0_{i-1} & 1 & 0_{i+1} & \ldots & 0_m \end{bmatrix}^t$. Using this information combined with theorems 1, 4, and 5, the trace of $B'AA^t$ can be found to be equal to the sum of the eigenvalues corresponding to the eigenvectors in B:

$$\begin{aligned}
&\text{tr}(B'AA^t) \\
&= \text{tr}(B'QDQ^t) \\
&= \text{tr}(\sum_{i=1}^{n'} \frac{B_i B_i^t}{||B_i||^2} QDQ^t) \\
&= \text{tr}(\sum_{i=1}^{n'} B_i B_i^t QDQ^t) \\
&= \sum_{i=1}^{n'} \text{tr}(B_i B_i^t QDQ^t) \\
&= \sum_{i=1}^{n'} \text{tr}(Q^t B_i B_i^t QD) \\
&= \sum_{i=1}^{n'} \text{tr}(Q^t B_i (Q^t B_i)^t D)
\end{aligned}$$

$$= \sum_{i=1}^{n'} \text{tr}(\begin{bmatrix} 0_1 \\ \vdots \\ 0_{i-1} \\ 1 \\ 0_{i+1} \\ \vdots \\ 0_m \end{bmatrix} \begin{bmatrix} 0_1 & \dots & 0_{i-1} & 1 & 0_{i+1} & \dots & 0_m \end{bmatrix} \begin{bmatrix} \lambda_1 & 0 & \dots & 0 \\ 0 & \lambda_2 & \dots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & \lambda_m \end{bmatrix})$$

$$= \sum_{i=1}^{n'} \text{tr}(\begin{bmatrix} 0_1 \\ \vdots \\ 0_{i-1} \\ 1 \\ 0_{i+1} \\ \vdots \\ 0_m \end{bmatrix} \begin{bmatrix} 0_1 & \dots & 0_{i-1} & \lambda_i & 0_{i+1} & \dots & 0_m \end{bmatrix})$$

$$= \sum_{i=1}^{n'} \text{tr}(\begin{bmatrix} 0_1 & \dots & 0 & 0 & 0 & \dots & 0 \\ \vdots & & \vdots & \vdots & \vdots & & \vdots \\ 0 & \dots & 0_{i-1} & 0 & 0 & \dots & 0 \\ 0 & \dots & 0 & \lambda_i & 0 & \dots & 0 \\ 0 & \dots & 0 & 0 & 0_{i+1} & \dots & 0 \\ \vdots & & \vdots & \vdots & \vdots & & \vdots \\ 0 & \dots & 0 & 0 & 0 & \dots & 0_m \end{bmatrix})$$

$$= \sum_{i=1}^{n'} (\lambda_i + 0)$$

$$= \sum_{i=1}^{n'} \lambda_i$$

So, if a subspace B is defined by a basis composed of orthonormal eigenvectors of AA^t, $\text{tr}(B'AA^t)$ will be as high as possible, and consequently $\langle A - B'A, A - B'A \rangle$ will be as low as possible. This leads to the following conclusion:
If a subspace B is defined by a basis composed of orthogonal eigenvectors of AA^t, then there exists no other basis D such that $\text{proj}_D(A)$ is closer to A than $\text{proj}_B(A)$ is. Thus, principal component analysis is proved.

Note that the A in the above proof will actually be A^t if a matrix has its rows as data points and its columns as variables instead of the other way around; so, it is often necessary to find an orthogonal basis for the eigenvectors of $(A^t)(A^t)^t = A^tA$ instead.
(Laity, 2018)

Theorem 12: Existence of the Reduced SVD

From theorems 7 and 9, it is possible to find the diagonal matrix D^2 made of the nonzero eigenvalues of A^tA and the partially orthogonal matrix U made of the corresponding eigenvectors such that $UD^2U^t = A^tA$. Then, let $V = AUD^{-1}$.

From this, we can prove that $V^tV = I$, and therefore V is partially orthogonal just as U is:

$$\begin{aligned} & V^tV \\ &= (AUD^{-1})^t(AUD^{-1}) \\ &= (D^{-1})^tU^tA^tAUD^{-1} \\ &= (D^{-1})(U^tA^tAU)D^{-1} \\ &= (D^{-1})D^2D^{-1} \\ &= I \end{aligned}$$

Finally, we can prove that $A = VDU^t$:

$$\begin{aligned} & VDU^t \\ &= AUD^{-1}DU^t \\ &= AUU^t \\ &= A \end{aligned}$$

This VDU^t is then the singular value decomposition for A. The elements of the diagonal of D (i.e. the square roots of the eigenvalues of A^tA) are known as the singular values of A.
Note that for this decomposition to truly qualify as a Singular Value Decomposition, the eigenvectors must be sorted from highest to lowest eigenvalue, so $D_{n,n} \geq D_{n-1,n-1}$. however, this is not required to prove the theorem, and VDU^t will equal A even if this is not the case.
One additional fact to note is that V will be composed of the eigenvectors of AA^t and D^2 will be the eigenvalues of AA^t as well as of A^tA. This can be proven as follows:

$$\begin{aligned} & AA^tV \\ &= AA^tAUD^{-1} \\ &= AUD^2D^{-1} \\ &= AUD^{-1}D^2 \\ &= VD^2 \end{aligned}$$

(Penney, 2016)

Theorem 13: Singular Value Decomposition

From theorem 12, $A = VDU^t$. we can then extend D by adding zeroes and extend V and U by adding the eigenvectors of AA^t and A^tA with eigenvalues of 0 (i.e. the nullspaces of those matrices):

$$
\begin{aligned}
& V'D'(U')^t \\
&= \begin{bmatrix} V_{1,1} & \dots & V_{1,r} & V'_{1,r+1} & \dots & V'_{1,m} \\ \vdots & & \vdots & \vdots & & \vdots \\ V_{m,1} & \dots & V_{m,r} & V'_{m,r+1} & \dots & V'_{m,m} \end{bmatrix} \begin{bmatrix} D_1 & \dots & D_r & 0 & \dots & 0 \\ 0 & \dots & 0 & 0 & \dots & 0 \\ \vdots & & \vdots & \vdots & & \vdots \\ 0 & \dots & 0 & 0 & \dots & 0 \end{bmatrix} \begin{bmatrix} U_{1,1} & \dots & U_{1,r} & U'_{1,r+1} & \dots & U'_{1,n} \\ \vdots & & \vdots & \vdots & & \vdots \\ U_{n,1} & \dots & U_{n,r} & U'_{n,r+1} & \dots & U'_{n,n} \end{bmatrix}^t \\
&= \begin{bmatrix} [V_{1,1} & \dots & V_{1,r}] \cdot D_1 + 0 & \cdots & [V_{1,1} & \dots & V_{1,r}] \cdot D_r + 0 & 0 & \cdots & 0 \\ & & \vdots & & & & \vdots & \vdots & & \\ [V_{m,1} & \dots & V_{m,r}] \cdot D_1 + 0 & \cdots & [V_{m,1} & \dots & V_{m,r}] \cdot D_r + 0 & 0 & \cdots & 0 \end{bmatrix} \begin{bmatrix} U_{1,1} & \dots & U_{1,r} & U'_{1,r+1} & \dots & U'_{1,n} \\ \vdots & & \vdots & \vdots & & \vdots \\ U_{n,1} & \dots & U_{n,r} & U'_{n,r+1} & \dots & U'_{n,n} \end{bmatrix}^t \\
&= \begin{bmatrix} (VD)_1 & 0 & \dots & 0 \\ \vdots & \vdots & & \vdots \\ (VD)_r & 0 & \dots & 0 \end{bmatrix} \begin{bmatrix} U_{1,1} & \dots & U_{1,r} & U'_{1,r+1} & \dots & U'_{1,n} \\ \vdots & & \vdots & \vdots & & \vdots \\ U_{n,1} & \dots & U_{n,r} & U'_{n,r+1} & \dots & U'_{n,n} \end{bmatrix}^t \\
&= \begin{bmatrix} (VD)_1 \cdot [U_{1,1} & \dots & U_{1,r}] + 0 & \dots & (VD)_1 \cdot [U_{n,1} & \dots & U_{n,r}] + 0 \\ & & \vdots & & & & \vdots & \\ (VD)_r \cdot [U_{1,1} & \dots & U_{1,r}] + 0 & \dots & (VD)_r \cdot [U_{n,1} & \dots & U_{n,r}] + 0 \end{bmatrix} \\
&= VDU^t
\end{aligned}
$$

This equivalency is also true if any other values for $V'_{r+1,\dots,m}$ and $U'_{r+1,\dots,n}$ are used as opposed to the 0-eigenvalued eigenvectors of AA^t and A^tA; however, the singular value decomposition is defined to use those eigenvectors. So, a different $V'D'(U')^t$ would still equal A, but would not still be the singular value decomposition of A.

(Penney, 2016)

Works Cited

Google. (2025, December 9). Machine learning. Google. https://developers.google.com/machine-learning/crash-course

J. M. ain't a mathematician. (2013, April 12). Why is SVD on X preferred to eigendecomposition of $XX^\top$ in PCA? https://math.stackexchange.com/questions/359397/why-is-svd-on-x-preferred-to-eigendecomposition-of-xx-top-in-pca

Johnston, N. (2020, July 16). Advanced Linear Algebra - Lecture 18: The Trace and the Frobenius Inner Product. Youtube. https://www.youtube.com/watch?v=-tvsZ7Un8_g

Laity, J. (2018, October 18). Principal component analysis: pictures, code and proofs. Joellaity.com. https://joellaity.com/2018/10/18/pca.html

Penney, R. C. (2016). Linear algebra : ideas and applications – Fourth edition. John Wiley & Sons, Inc. ISBN 978-1-118-90958-4

Wikipedia contributors. (2025, November 15). Backpropagation. Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=Backpropagation&oldid=1322271807