Gradient descent methods in machine learning

Christina J. Edholm, Maryann E. Hohn, Ami E. Radunskaya

1 Lesson 1: Intuition for gradient descent

Suppose we hike up a large mountain. At the top, we become very tired and want to descend the mountain as quickly as possible. What path should we take? By looking at the mountain, we may make an educated guess which direction to go, but what is the best, most optimal, way down? Can we calculate an optimal path to descend the mountain? Take a look at Figure 1 for ideas.

Figure 1: Descending a mountain, follow the path! How is it different if we just went straight down?

In order to calculate our path down the mountain, we need to transform the mountain into something we can study mathematically. That is, we need to create a model of the mountain by generating a function mimicking the three-dimensional surface. When dealing with three-dimensional objects, we can visualize the surface using contour plots. For a visualization of the contour plot for the paraboloid $f(x, y) = 5-x^2-y^2$ we refer to [6] and [8] using [5, 4].

Suppose we are standing at point $P = \left(\frac{1}{2}, \frac{1}{2}, \frac{9}{2}\right)$ on the paraboloid $f(x, y) = 5 - x^2 - y^2$. The gradient of the function is $\nabla f(x, y) = \langle -2x, -2y\rangle$, so the gradient of f at the point P is $\nabla f\left(\frac{1}{2}, \frac{1}{2}\right) = \langle -1, -1\rangle$. Remember, the gradient points in the direction of steepest ascent. We want the direction of steepest *descent*, so we need the opposite direction: $-\nabla f\left(\frac{1}{2}, \frac{1}{2}\right) = \langle 1, 1\rangle$. The level curve which passes through the point P is the one that satisfies $z = \frac{9}{2}$, which is the equation $\frac{9}{2} = 5 - x^2 - y^2$. The tangent line for the paraboloid at the point P in the plane $z = \frac{9}{2}$ is $y = 1 - x$. For a visualization we refer to [9] made using [4]. From an "overhead" view, we can see the relationship between the level curve, gradient, and tangent line. Note, the gradient is orthogonal to the tangent line on the level curve.

Let us apply what we learned to a mountain in our local area. We suggest consulting the USGS or ArcGIS or making a Google search [27, 26, 10, 11].

Exercise 1. *Find a topographical map of your choice. Trace out the gradient descent on the map. Discuss with your group-mates how you know your path is the steepest.*

Our method of finding the gradient descent requires some careful consideration, especially if we are dealing with a mountain range rather than just one mountain. From Figure 2 we can see that while descending a mountain range, we can possibly get trapped in a basin rather than finding the bottom.

Exercise 2. *Let f be a differentiable function and let $\mathbf{u}$ be a unit vector.*

1. *Let θ be the angle between $\nabla f(a, b)$ and $\mathbf{u}$. Explain why the directional derivative of $f(x, y)$ in the direction of $\mathbf{u}$, $D_{\mathbf{u}} f(a, b)$, can be expressed with the formulation below:*

$$D_{\mathbf{u}} f(a, b) = \|\langle f_x(a, b), f_y(a, b)\rangle\| \cos(\theta).$$

Figure 2: Descending a mountain range, we can end up not at the bottom!

2. *At the point (a, b), the only quantity in the equation in Part 1 that changes is θ. Explain why $\theta = 0$ makes $\|\langle f_x(a, b), f_y(a, b)\rangle\| \cos(\theta)$ as large as possible.*

3. *When $\theta = 0$, in what direction does the vector $\mathbf{u}$ point relative to $\nabla f(a, b)$? What does this tell us about the direction of greatest increase of f at (a, b)?*

4. *In what direction, relative to $\nabla f(a, b)$ does f decrease most rapidly at (a, b)?*

2 Lesson 2: The Algorithm for gradient descent

Recall that the gradient of a function f from $\mathbb{R}^n$ to $\mathbb{R}$ is a vector that points in the direction of "steepest ascent," so if we want to find a minimum of a function f, it makes sense to move in the direction opposite to the gradient. That is, we move in the direction of the negative gradient.

The method of *gradient descent* implements this idea as follows. At each iteration, we pick a step size, γ_n (also known as the *learning rate*). We pick some starting value for $\mathbf{x}$, call it $\mathbf{x}_0$. Then, for $n = 0, 1, 2, \ldots$, we define

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \gamma_n \nabla f(\mathbf{x}_n).$$

In other words, at each step, we move a distance $\gamma_n ||\nabla f(\mathbf{x}_n)||$ in the direction of the negative gradient. The choice of step size, γ_n, can be tricky. It needs to be small if $\mathbf{x}_n$ is close to the minimum or the algorithm will overshoot the actual minimum. But, if the step size is too small early on, then the gradient descent is very slow, and the algorithm may never achieve the minimum.

Let us illustrate the algorithm with an example.

Example 1. *Suppose we want to find the minimum of the function*

$$f(x, y) = (x - 47)^2 + (y - 0.1)^2 + 2.$$

A graph of the function is shown in Figure 3.

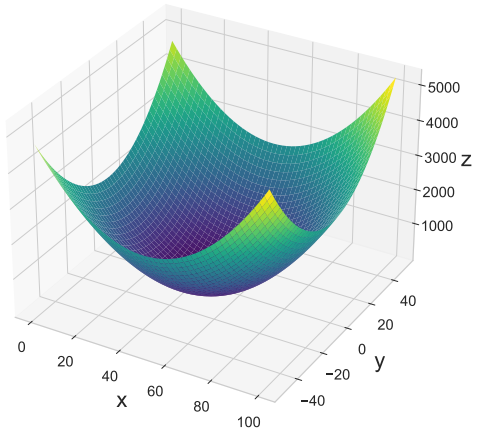

Figure 3: Graph of $f(x, y) = (x - 47)^2 + (y - 0.1)^2 + 2$.

This function has a unique minimum at the vertex of the parabola, where $(x, y) = (47, 0.1)$. To convince yourself that this is, indeed, the minimum, note that $f = A^2 + B^2 + 2 \geq 2$, where $A = (x - 47)$ and $B = (y - 0.1)$. Since $A = 0$ only when $x = 47$ and $B = 0$ only when $y = 0.1$, we conclude that $f(x, y) \geq f(47, 0.1)$ for all values of x and y. Let us see how well the gradient descent algorithm does at finding this minimum. A flow chart representing the algorithm is given in Figure 4.

As a concrete illustration, suppose we decide to use a step size of $\gamma = 0.01$. We will decide to stop if we have not moved by more that $\epsilon = 0.0001$ (the "tolerance") or if we have gone $N = 500$ steps ("maxSteps"). The three values γ, ϵ, and N are the parameters that we set before starting the algorithm.

Since we are trying to find the minimum of $f(x, y) = (x - 47)^2 + (y - 0.1)^2 + 2$, we will move in the direction of the negative gradient of f. First, we compute the gradient.

$$\nabla f(x, y) = \langle 2(x - 47), 2(y - 0.1) \rangle.$$

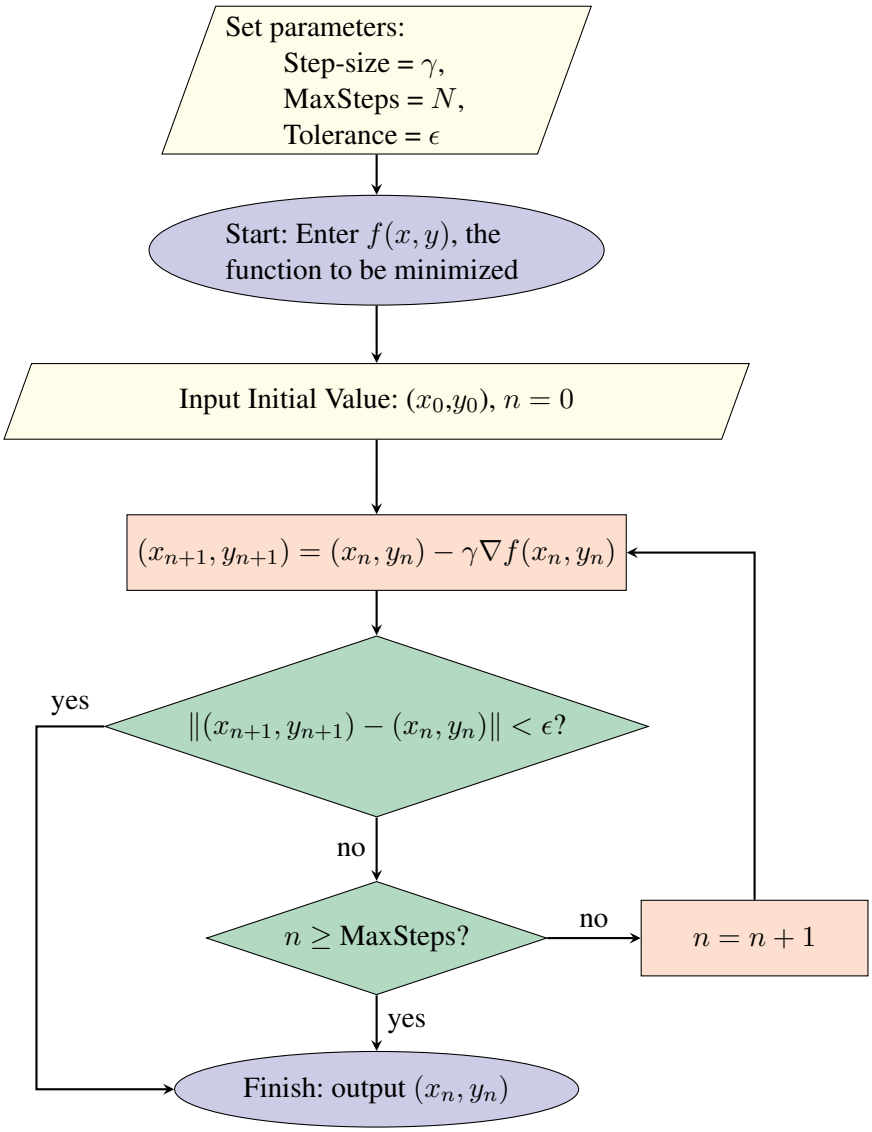

Figure 4: Flow chart describing the gradient descent algorithm for a function of two variables.

So, the main updating step in the algorithm is:

$$\begin{aligned}(x_{n+1}, y_{n+1}) &= (x_n, y_n) - \gamma(2(x_n - 47), 2(y_n - 0.1)) \\ &= (x_n, y_n) - 0.01(2(x_n - 47), 2(y_n - 0.1)).\end{aligned}$$

Finally, to get the algorithm started, we need to input an initial value. Suppose we choose $(x_0, y_0) = (80, 20)$. The first few and final few steps in the algorithm yield:

n	x_n	y_n	$\|(x_{n+1}, y_{n+1}) - (x_n, y_n)\|$	
0	80	20	100.0000	
1	79.3400	19.602000	0.7707	
2	78.6932	19.211960	0.7553	
3	78.0593	18.829721	0.7402	
⋮	⋮	⋮	⋮	
442	47.0045	0.1027	0.0001063	
443	47.0044	0.1026	0.0001041	
444	47.0043	0.1026	0.0001021	
445	47.0042	0.1025	0.0001	
446	47.0041	0.1025	0.000098	$\leftarrow \|\mathbf{x}_{n+1} - \mathbf{x}_n\| < \epsilon$

The algorithm stops at $n = 446$ since the change in the (x, y) value is less than the tolerance, which we had set to $\epsilon = 0.0001$. The entire sequence is shown in Figure 5. The associated Jupyter Notebook can be found on our GitHub [7].

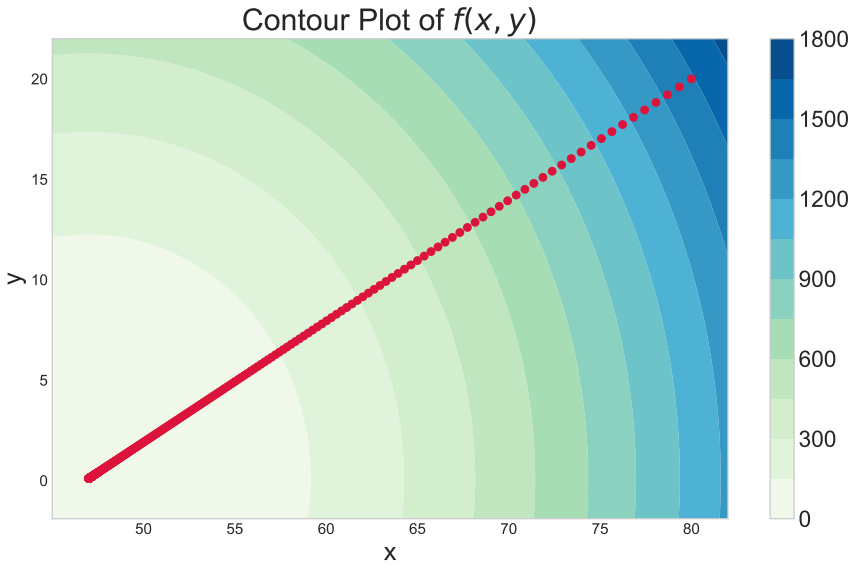

Figure 5: Sequence of x_n and y_n from the gradient descent algorithm plotted on top of the contour plot of $f(x, y) = (x - 47)^2 + (y - 0.1)^2 + 2$. Here, we use step size $\gamma = 0.01$.

Exercise 3. *Use gradient descent to find the minimum of*

$$f(x, y) = 4x^2 - 4xy + 2y^2$$

using a starting value of $(x_0, y_0) = (10, 10)$. *To do this, you will first need to calculate the gradient of* f. *What is the effect of changing the step size (learning rate)? Can you find a step size such that the algorithm does not converge? What is the largest value of* γ *such that the algorithm does converge? Can you adjust this step size intelligently? Explain.*

3 Lesson 3: Gradient descent with linear regression

The goal of regression is typically to predict an output from inputs that have not yet been seen. The technique of *regression* involves, broadly, finding a function that maps inputs to outputs, using a given data set as a guide to determine the function's structure. You may be familiar with the term "least squares fit" applied to a set of data consisting of N real-valued inputs, x^k, and associated outputs, y^k. Note that the superscript identifies the data point. Finding a least squares fit means finding a function, F, that minimizes the sum of the squared distances from the function values, $F(x^k)$ to the data outputs, y^k. In other words, F minimizes the *distance* or *objective* function $D(\mathbf{x}, \mathbf{y}, F)$ defined by

$$D(\mathbf{x}, \mathbf{y}, F) = \sum_{k=1}^{N} (F(x^k) - y^k)^2 = (\mathbf{F}(\mathbf{x}) - \mathbf{y})^T \cdot (\mathbf{F}(\mathbf{x}) - \mathbf{y}) = \|\mathbf{F}(\mathbf{x}) - \mathbf{y}\|^2, \tag{1}$$

where $\mathbf{F}(\mathbf{x})$ is the vector $\langle F(x^1), F(x^2), \ldots, F(x^N)\rangle$. Note that we wrote the function D in three ways. The last notation is useful because it shows that the squared distance function, D, can be written as a *norm*, or the length of the vector of differences between all N inputs and all N outputs.

This is one example of regression as an optimization problem since we want to minimize the function D. The function F is sometimes called a *model* of the data, and it can contain some noise, or randomness, as well as deterministic (non-random) components. To solve a regression problem, we can use gradient descent to find the function F. In this section, we discuss a few examples.

Let us consider the following learning goal: We want the computer to learn the relationship between input variables, $\mathbf{x}^k$, and output variables, y^k, where the inputs live in d dimensions, and the outputs are one-dimensional. To make it easier for the computer, we tell it to restrict itself to a *linear model*. In other words, we want the algorithm to assume that the outputs are linear functions of the inputs, with some normally distributed observational noise, which we denote by η. So, each output, y, is assumed to be:

$$y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_d x_d + \eta = \boldsymbol{\theta}^T \tilde{\mathbf{x}} + \eta \tag{2}$$

where $\boldsymbol{\theta}$ is the vector $\langle \theta_0, \theta_1, \ldots, \theta_d\rangle$, $\tilde{\mathbf{x}}$ is the vector $\langle 1, x_1, x_2, \ldots, x_d\rangle$, and η is a normally distributed random variable with mean zero.

The data (sometimes called a "training set") are a set of outputs: $y^1, \ldots, y^N$, along with the corresponding inputs $\mathbf{x}^1, \ldots \mathbf{x}^N$. The computer's job is to find the unknown *parameter* vector, $\boldsymbol{\theta}$.

In practice, we cannot hope to get the outputs exactly right because there will be some noise, or randomness, in the observations, which is represented by η. We just want to get the best possible guesses for $\boldsymbol{\theta}$, where "best possible" means the θ_i values that minimize the distance from the function output to the data, i.e., we want to minimize the distance function D in equation (1). In the context of equation (1), the function F is a linear function of the inputs, $\mathbf{x}^k$, so finding F is equivalent to finding the coefficients of the linear function, i.e., the vector $\boldsymbol{\theta}$ which includes the intercept θ_0.

Suppose the dimension of our data set is N, and each vector $\mathbf{x}^i$ has d components, so that θ has $d + 1$ components. To simplify the notation, we insert a "1" into the first component of the input vectors, $\mathbf{x}^k$, and call these new vectors $\tilde{\mathbf{x}}^k$. This gives us an easy way to write the right-hand side of equation (2) since:

$$\theta \cdot \tilde{\mathbf{x}}^k = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_d x_d.$$

We will minimize the average distance, in order to keep the values a reasonable size. This is called the *mean squared error* (MSE):

$$D_{MSE}(\boldsymbol{\theta}) = \frac{1}{N} \sum_{k=1}^{N} \|\boldsymbol{\theta}^T \tilde{\mathbf{x}}^k - y^k\|^2.$$

We can solve this minimization problem using gradient descent. In this case, where the function F is linear, we can explicitly write down the the gradient of D:

$$\nabla D_{MSE} = \left\langle \frac{\partial D}{\partial \theta_0}, \frac{\partial D}{\partial \theta_1}, \ldots, \frac{\partial D}{\partial \theta_d} \right\rangle = \frac{2}{N} \sum_{k=1}^{N} \left((\boldsymbol{\theta}^T \tilde{\mathbf{x}}^k - y^k) \tilde{\mathbf{x}}^k \right).$$

We then iterate the gradient descent algorithm, with a suitable starting vector, $\boldsymbol{\theta}^0$ and learning rate, γ, as in the previous Section 2.

Example 2. *Predicting the height of young children using their age.*

Suppose we want to predict a child's height from their age. We have data, [17], giving measurements of heights for various boys between the ages of two and eight [15, 16, 14]. The y-values are the heights measured in centimeters, and the x-values are the ages of the boys in years corresponding to their heights.

Each height and age pair constitutes one training example $(x^{(i)}, y^{(i)})$ in our dataset. There are $N = 114$ training examples, and we will use them to develop a linear regression model. Since the *predictor* data (age) is one-dimensional, the objective function, D_{MSE}, and its gradient are:

$$D_{MSE}(\theta_0, \theta_1) = \frac{1}{N} \sum_{k=1}^{N} (\theta_1 x^k + \theta_0 - y^k)^2,$$

$$\nabla D_{MSE} = \frac{2}{N} \sum_{k=1}^{N} \left((\theta_1 x^k + \theta_0 - y^k), x^k (\theta_1 x^k + \theta_0 - y^k) \right)$$

1. Implement gradient descent using a learning rate of $\gamma = 0.01$.

   Initialize the parameters to $\boldsymbol{\theta} = \mathbf{0}$ (i.e., $\theta_0 = \theta_1 = 0$), and run one iteration of gradient descent from this initial starting point. Record the value of of θ_0 and θ_1 that you get after this first iteration. Did you get $\theta_0 = 1.8285$ and $\theta_1 = 7.8286$?

2. Continue running gradient descent for more iterations until $\boldsymbol{\theta}$ converges. Be sure to create a maximum number of steps that the algorithm may take such as 5000 iterations. After convergence, record the final values of θ_0 and θ_1 that you get.

   After 2067 time steps, the algorithm should stop with $\theta_0 = 68.7328$ and $\theta_1 = 5.9250$.

3. When you have found $\boldsymbol{\theta}$, plot the straight line fit from your algorithm on the same graph as your training data.

   Note that for most machine learning problems, x is very high dimensional, so we may not be able to plot the prediction. However, in this example, we have only one predictor variable, or "feature" (age). Being able to plot this gives a nice sanity-check on our result.

4. Finally, we'd like to make some predictions using the learned values of θ. We will use the model to predict the height for two boys, one of age 3.5 and one of age 7.

   $$\text{Height in cm } = 68.73 + (\text{age in years}) \cdot 5.925.$$

   Using these estimated values of θ_0 and θ_1, the model predicts that a boy of age 3.5 is approximately 89.5 centimeters tall, and a boy of age 7 is approximately 110.2 centimeters tall.

For the Jupyter Notebook associated with Example 2, see our GitHub repository [7].

4 Lesson 4: Classification and gradient descent with logistic regression

In the following lesson, we utilize some concepts from probability theory. The required content is described throughout the lesson. For an additional resource, we direct the reader to “Introduction to Probability" by Joseph Blitzstein and Jessica Hwang, particularly Chapter 2 [2].

Suppose that the set of outputs that we are given (and that we want to predict) come from a finite set of values, rather than from a continuum. As a concrete example, suppose we want to predict whether someone will default on their loan based on their income and the amount that they borrowed. The data set is called “Default-Data” [7, 12]. The data is plotted in Figure 6.

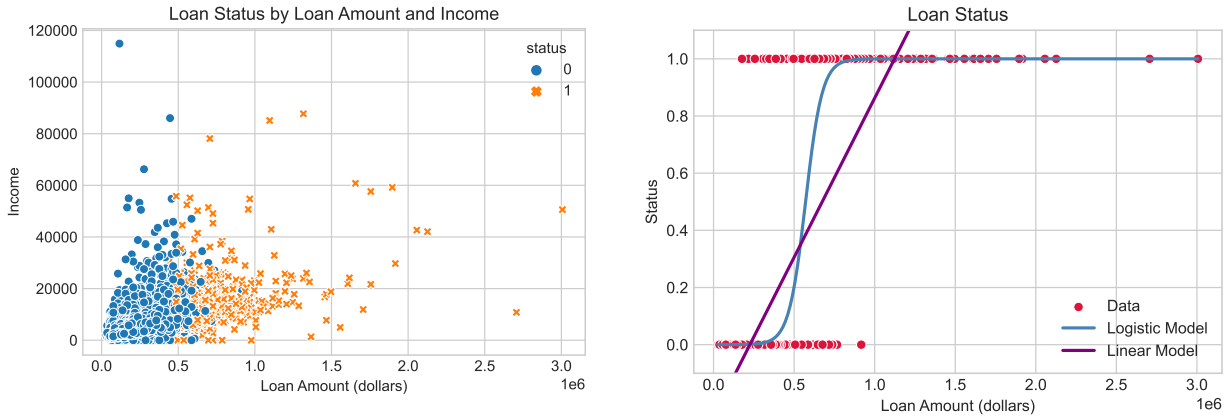

Figure 6: **Left:** Plot of the loan amount versus income from the “Default-Data” set provided [12].The ‘●’ are the non-defaulted loans and the ‘×’ are the defaulted loans. **Right:** A scatter plot of the default yes/no versus the loan amount. We fit the data to a straight line (in purple), and an S-shaped function, the “logistic” curve (in blue).

If we classify those who default as a “$Y = 1$” and those who do not as a “$Y = 0$”, the scatter plot of Y versus the loan amount looks like the plot shown in Figure 6. We fit the data to a straight line (in purple), and an S-shaped function, the “logistic” curve (in blue). In Figure 6, we see that the logistic curve looks more like a switch and fits the data better at the low and high ends.

As in linear regression, we will assume that the function from inputs $\left(X \in \mathbb{R}^d\right)$ to output $(Y \in \{0, 1\})$ has a particularly simple form. But, here, rather than predict the output itself, we will predict the probability that Y is equal to either 0 or 1. We will assume that the conditional probability of the output, Y, is an exponential function of X. In other words, we will assume that the output is modeled as follows: the probability that Y is 1, given the data $\mathbf{x}$, is an exponential function of the data. This is called a *conditional probability*, and we write it as:

$$\Pr(Y = 1|X = \mathbf{x}) = Ce^{\theta_0+\theta_1 x_1+\cdots+\theta_d x_d} = Ce^{\boldsymbol{\theta}^T\tilde{\mathbf{x}}} \tag{3}$$

where $\tilde{\mathbf{x}} = \langle 1, x_1, x_2, \ldots, x_d\rangle$ and $\boldsymbol{\theta} = \langle\theta_0, \theta_1, \ldots, \theta_d\rangle$ are both vectors of length $d+1$, and C is a constant. Since probabilities must sum to one and Y can either be 0 or 1, we must have:

$$\Pr(Y = 0|X = \mathbf{x}) = 1 - \Pr(Y = 1|X = \mathbf{x}).$$

The constant, C, in equation (3) is taken to be $\dfrac{1}{1 + e^{\boldsymbol{\theta}^T\tilde{\mathbf{x}}}}$ so that the probability is always between 0 and 1. This gives the final model:

$$P(Y = 1|X = \mathbf{x}) = \frac{e^{\boldsymbol{\theta}^T\tilde{\mathbf{x}}}}{1 + e^{\boldsymbol{\theta}^T\tilde{\mathbf{x}}}} = \frac{1}{1 + e^{-\boldsymbol{\theta}^T\tilde{\mathbf{x}}}}. \tag{4}$$

As in linear regression, the parameters of the model are a vector, $\boldsymbol{\theta}$. A particular value of $\boldsymbol{\theta}$ will be a vector of length $d+1$: $\langle \theta_0, \theta_1, \ldots, \theta_d \rangle$. From here on, we will write: $p(\mathbf{x}; \boldsymbol{\theta}) = P(Y = 1 | X = \mathbf{x}; \boldsymbol{\theta})$, so that $1 - p(\mathbf{x}; \boldsymbol{\theta}) = P(Y = 0 | X = \mathbf{x}; \boldsymbol{\theta})$.

Suppose we have a training set with N observations of input vectors and their corresponding (scalar) outputs: $(\mathbf{x}^1, y^1), (\mathbf{x}^2, y^2), \ldots, (\mathbf{x}^N, y^N)$. If we assume that they are independent (a typical assumption), then the probability of seeing this set of data is equal to the product of the individual probabilities. For example, suppose we observe the outputs: $y^1 = 1, y^2 = 0, y^3 = 1, y^4 = 1$. Then the *likelihood function*, which gives the probability of seeing the data given the output and the parameter value, is, for this example:

$$\text{Likelihood}(\boldsymbol{\theta}) = \mathcal{L}(\boldsymbol{\theta}) = p(\mathbf{x}^1; \boldsymbol{\theta}) \left(1 - p(\mathbf{x}^2; \boldsymbol{\theta})\right) p(\mathbf{x}^3; \boldsymbol{\theta}) p(\mathbf{x}^4; \boldsymbol{\theta}).$$

To account for small probability values and to leverage the computational benefit of addition over multiplication, we take the logarithm of the likelihood, or the *log-likelihood*:

$$\ell(\boldsymbol{\theta}) = \ln(\mathcal{L}) = \ln p(\mathbf{x}^1; \boldsymbol{\theta}) + \ln \left(1 - p(\mathbf{x}^2; \boldsymbol{\theta})\right) + \ln p(\mathbf{x}^3; \boldsymbol{\theta}) + \ln p(\mathbf{x}^4; \boldsymbol{\theta}).$$

In general, if we have N observations and two categories where y^i is 0 or 1, we have:

$$\mathcal{L}(\boldsymbol{\theta}) = \prod_{i | y^i = 1} p(\mathbf{x}^i; \boldsymbol{\theta}) \prod_{i | y^i = 0} \left(1 - p(\mathbf{x}^i; \boldsymbol{\theta})\right). \tag{5}$$

We can compactly write the log-likelihood, ℓ, as:

$$\begin{aligned} \ell(\boldsymbol{\theta}) &= \sum_{i=1}^{N} \left[y^i \ln p(\mathbf{x}^i; \boldsymbol{\theta}) + (1 - y^i) \ln(1 - p(\mathbf{x}^i; \boldsymbol{\theta})) \right] &(6) \\ &= \sum_{i=1}^{N} \left[y^i \boldsymbol{\theta}^T \tilde{\mathbf{x}}^i - \ln \left(1 + e^{\boldsymbol{\theta}^T \tilde{\mathbf{x}}^i}\right) \right]. &(7) \end{aligned}$$

Since $\boldsymbol{\theta} = \langle \theta_0, \theta_1, \ldots, \theta_d \rangle$, we augment the input vector $\mathbf{x}$ as in linear regression so that its first component is 1, and call this augmented vector $\tilde{\mathbf{x}} = \langle 1, x_1, \ldots, x_d \rangle$. This simplifies the notation in the last equation so that the indices of $\tilde{\mathbf{x}}$ and $\boldsymbol{\theta}$ match up, where the first component of $\tilde{\mathbf{x}}$ is called $\tilde{x}_0$. This first component is always equal to 1.

Our goal is to find the value of the vector, $\boldsymbol{\theta}$, that makes the probability of observing the data as large as possible since that will give a model that best describes the data. In other words, we want to *maximize* the likelihood, which is equivalent to *minimizing* the negative of the log-likelihood:

$$\max(\mathcal{L}) = \min(-\mathcal{L}) = \min(-\ln(\mathcal{L})) = \min(-\ell).$$

To minimize $-\ell(\boldsymbol{\theta})$, we set the gradient equal to the zero-vector. That is, we set all of its partial derivatives to zero. These partial derivatives are sometimes called the *score* equations. Note that, for each θ_k, we get:

$$\frac{\partial \ell}{\partial \theta_k} = \sum_{i=1}^{N} \tilde{x}_k^i (y^i - p(\mathbf{x}^i; \boldsymbol{\theta})) = 0. \tag{8}$$

This gives $d+1$ nonlinear equations in $\boldsymbol{\theta}$. The first score equation is special, since the first component of each $\tilde{\mathbf{x}}$ is equal to 1. This first scale equation is:

$$\frac{\partial \ell}{\partial \theta_0} = 0 \iff \sum_{i=1}^{N} y^i = \sum_{i=1}^{N} p(\mathbf{x}^i; \boldsymbol{\theta}) \tag{9}$$

We interpret this equation as finding the value of $\boldsymbol{\theta}$ such that the *expected* number of outputs that are equal to 1 (the right-hand side of equation (9)) is equal to the observed number of 1's. It automatically follows that the expected number of zeros is the same as the observed number of zeros. This seems like a reasonable goal!

The negative log-likelihood function can be numerically minimized in a number of ways. Some popular ones are gradient-descent and Newton-Raphson. Andrew Ng explains some differences between these methods on his website at [18].

Solving the minimization problem using gradient descent. The *cost* function that we want to minimize is the negative of the log likelihood, whose partial derivatives we calculated in Equation (8). The update rule is

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \gamma^{(t)} \nabla(-\ell) = \boldsymbol{\theta}^{(t)} + \gamma^{(t)} \nabla \ell.$$

We note that since $\nabla \ell$ involves a sum, a large data set could result in large values of the partial derivatives. To control the size of these entries, it is often helpful to use the *mean* log likelihood, i.e., $\frac{1}{N}\nabla\ell$. This is what we recommend for this example and the next, so that the update rule is modified slightly, as written in Equation (10) in the next example.

For the Default Data above, we implemented gradient descent to solve this minimization problem in the Jupyter Notebook file "Example_Default_Data" [7]. In order to get the algorithm to converge, it is helpful to rescale the data so that they are all more or less the same size. We can do this by subtracting the mean of each input category (loan amount and income), and dividing by the standard deviation of that category. The values of the estimated coefficients, θ_i, must then also be rescaled to match the original unscaled variables, as is shown in Equation (11). These rescaling steps are implemented in the Jupyter notebook, and are written out in the implementation steps in the next example.

The algorithm returns the following coefficients of the logistic model, rounded to two signification digits:

$$\theta_0 = -11.40, \quad \theta_1 = 2.00 \times 10^{-5}, \quad \theta_2 = 1.97 \times 10^{-7}$$

which gives the estimate:

$$\text{Probability of defaulting} = \frac{1}{1 + e^{11.4 - 2\cdot 10^{-5} x_1 - 1.97 \cdot 10^{-7} x_2}}$$

where x_1 is the loan amount in dollars, and x_2 is income in dollars. Notice that θ_1 and θ_2 are both very small, with θ_2 smaller by two orders of magnitude. Typically, loan amounts for houses are much larger than income, so this model says that income has a relatively small effect on the probability of defaulting.

Exercise 4. *Explain how to get from Equation (5) to Equation (7), filling in all of the details.*

Exercise 5. *The decision boundary is the line in the state space of the input variables that corresponds to*

$$p(\mathbf{x}; \boldsymbol{\theta}) = 0.5.$$

1. *Show that this is equivalent to the identity:*

$$\boldsymbol{\theta}^T \tilde{\mathbf{x}} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_d x_d = 0.$$

2. *Modify the Jupyter Notebook file "Example_Default_Data" so that it plots the decision boundary on top of the scatter plot of loan amount versus income [7].*

3. *You should get a decision line that is nearly vertical. What does this tell you about the influence of income on the probability of defaulting?*

4. *Plot the loan amount versus the response variable, Y, Loan Status, as in Figure 6. On top of this, plot the probability of defaulting against the loan amount for each of the data points, using the estimated theta-values. If you were a bank, what would the cut-off probability be, i.e., for what value of p would you decide to approve the loan? Discuss. (Note, there is no correct answer!)*

Example 3. *Logistic regression and gradient descent:*

Let's put this idea to work. We will use gradient descent to implement logistic regression in order to solve a classification problem. This is an interactive example: follow along with the Jupyter Notebook in the GitHub repository [7]. Exercise 6 is based on completion of this example.

Data Description. The data for Example 3 is provided in our GitHub repository in a folder called "Lesson_4" [7, 20]. Our mission is to create a model that will estimate the probability of being admitted to a graduate program based on GRE scores and GPA. We have a list of scores for N students (the inputs, $\mathbf{x}^i$), and we know whether or not they were admitted (the outputs, y^i, where a value of "1" means "admitted," and "0" means "not admitted"). This is our *training set*.

The data has three columns, where the first column consists of the outputs y^i and the following columns consist of the inputs, $\mathbf{x}^i$. The second column gives all GRE test scores, and the third column contains the GPA. Before using gradient descent, we will plot the data using different symbols to represent the two classes.

Using the corresponding Jupyter Notebook, load the data and generate a scatter plot of the second column (GRE) against the third column (GPA) using a '×' if they were admitted (if there is a 1 in the first column) and an '•' if they were not admitted (if there is a 0 in the first column). The plot should look like Figure 7.

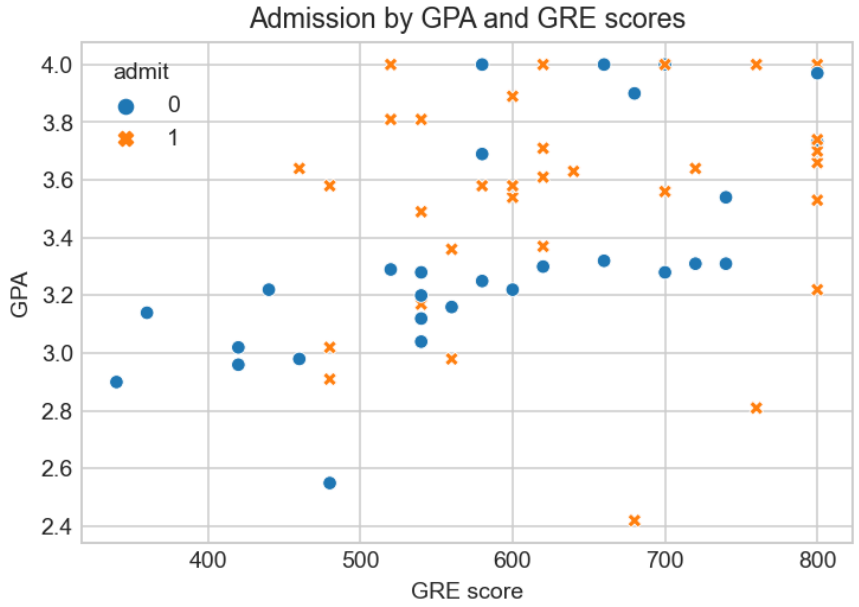

Figure 7: A scatter plot of the second column (GRE) against the third column (GPA) using a '×' if they were admitted (if there is a 1 in the first column) and a '•' if they were not admitted (if there is a 0 in the first column).

The minimization problem. As in the previous example, we want to minimize the negative of the *mean* log likelihood to eliminate large entries, giving the update rule:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \gamma^{(t)} \frac{1}{N} \nabla(-\ell) = \boldsymbol{\theta}^{(t)} + \frac{1}{N} \gamma^{(t)} \nabla \ell. \tag{10}$$

Note that, in the code, we are writing $\boldsymbol{\theta}$, the gradient vector $\nabla \ell$, and the input vector $\mathbf{x}^{(i)}$, as *column* vectors. The superscript of (t) is capturing the different iterations, a role previously denoted as a subscript in Section 2.

Implementation

- Recall that N is the number of students in the dataset. Define $\mathbf{y}$ as the first column of the data matrix, which is an $N \times 1$ column vector of 0's and 1's giving the admission data. Define $\mathbf{x}_1$ to be the data column of GRE values and $\mathbf{x}_2$ to be the data column of GPA values.

- Normalize each column of this input matrix (this helps the algorithm converge). One way to make the inputs more or less the same size is to subtract the mean of each column, and then divide by the standard deviation of each column:

  $$\mathbf{x}_i \mapsto \frac{\mathbf{x}_i - \bar{x}_i}{\sigma_i}$$

  where $\bar{x}_i$ is the mean of the $\mathbf{x}_i$ column, and σ_i is the standard error of the $\mathbf{x}_i$ column. Define $\mathbf{X}$ as the $N \times 2$ matrix consisting of the normalized $\mathbf{x}_1$ and $\mathbf{x}_2$ data columns.

- Initialize the algorithm: Start with a value of $\boldsymbol{\theta}^0$, and set the number of iterations to 0. Fix the rate parameter, γ, the maximum number of iterations, and the convergence threshold, ϵ. For reference, the values we used for these parameters are:

  $$\boldsymbol{\theta}^0 = \mathbf{0}; \quad \gamma = 0.01; \quad \epsilon = 10^{-6}$$

  In this example, we write $\boldsymbol{\theta}^0$ as a 1×3 row vector.

- **Main Loop:** For each value of the parameter vector, $\boldsymbol{\theta}$:

  - Calculate the $N \times 1$ vector of probabilities: $p(\mathbf{X}, \boldsymbol{\theta}) = \dfrac{1}{1 + e^{-\mathbf{X} * \boldsymbol{\theta}}}$.
  - The 1×3 gradient vector, given in Equation (8), can be compactly calculated as:

    $$\nabla \ell = \Big(\mathbf{y} - p(\mathbf{X}, \boldsymbol{\theta})\Big)^T * \mathbf{X}.$$

  - The next value of theta is calculated: $\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^t + \frac{1}{N} \gamma \nabla \ell$.
  - Increment the number of iterations.
  - Test for convergence.

- Continue until the algorithm has converged (the values of $\boldsymbol{\theta}$ change by less than some threshold amount (e.g., continue until $||\boldsymbol{\theta}^{t+1} - \boldsymbol{\theta}^t|| < 10^{-6}$), or the maximum number of iterations has been reached.

See our GitHub repository for associated Jupyter Notebooks [7].

Once the algorithm has converged, use your value of $\boldsymbol{\theta}$ to draw the line that best separates the admitted points from the non-admitted points. If you normalized the input vectors initially, you should convert back to the original variables by transforming the estimated values of $\boldsymbol{\theta}$:

$$\theta_0 + \theta_1 \frac{x_1 - \bar{x}_1}{\sigma_1} + \theta_2 \frac{x_2 - \bar{x}_2}{\sigma_2} = \left(\theta_0 - \frac{\theta_1}{\bar{x}_1}\sigma_1 - \frac{\theta_2}{\bar{x}_2}\sigma_2\right) + \frac{\theta_1}{\sigma_1}x_1 + \frac{\theta_2}{\sigma_2}x_2 = \hat{\theta_0} + \hat{\theta_1}x_1 + \hat{\theta_2}x_2 \qquad (11)$$

and use these transformed values, $\hat{\theta}_0$, $\hat{\theta}_1$, $\hat{\theta}_2$ in what follows in lieu of θ_0, θ_1, θ_2.

This line is called the *decision boundary* for the classification problem, and is given as the points that would be equally likely to be admitted as not admitted:

$$p(\mathbf{x}; \boldsymbol{\theta}) = 0.5 \;\Leftrightarrow\; \frac{1}{1 + e^{-\boldsymbol{\theta}^T \tilde{\mathbf{x}}}} = \frac{1}{2} \;\Leftrightarrow\; \boldsymbol{\theta}^T \tilde{\mathbf{x}} = 0.$$

Exercise 6. *For Example 3, record your observations by answering the following questions:*

1. *What is the final $\boldsymbol{\theta}$ value? How many iterations were required for convergence?*
2. *What is the probability that a student with a score of 680 on the GRE and a GPA of 3.5 will be admitted?*
3. *Which is a better predictor of admission, GRE or GPA? Why?*
4. *Plot the decision boundary on top of a scatter plot of the data, i.e., plot the line $\boldsymbol{\theta}^T \tilde{\mathbf{x}} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 = 0$. How would you use this line if you were on the Graduate Admissions Committee?*

Exercise 7. *Suppose we collect data for a group of students in a statistics class with variables $x_1 =$ hours studied and $x_2 =$ undergrad GPA, and output $y =$ receive an A (1 or 0). We fit a logistic regression and produce estimated coefficients $\theta_0 = -6$, $\theta_1 = 0.05$, $\theta_2 = 1$.*

1. *Estimate the probability that a student who studies for 40 hours and has an undergrad GPA of 3.5 gets an A in the class.*
2. *How many hours would the student in Part 1 need to study to have a 50% chance of getting an A in the class?*

5 Lesson 5: Extensions and Implementations of gradient descent

In this section we present specific extensions and implementations of gradient descent. This is a small sampling intended to show the depth and breadth of possible applications.

5.1 Neural Networks

Introduction and Background. For the background on neural networks and the specific connection to gradient descent, we refer students to the 3Blue1Brown resource [22]. Specifically, there is a whole series of videos, lessons, and Python code on neural networks, [24]. If time allows, we suggest all students watch all videos staggered over many days and have discussions in class. For a more abbreviated lesson, we suggest having students only watch the second video, [23], which outlines how neural networks implement gradient descent.

Extensions to Mathematical Analysis. For upper-level courses, we suggest exploring the mathematics underlying gradient descent on neural networks. Specifically, we propose the following two references: *Lyapunov stability analysis of gradient descent-learning algorithm in network training* (2011) by A. Banakar [1] and *A proof of convergence for gradient descent in the training of artificial neural networks for constant target functions* (2022) by P. Cheridito, A. Jentzen, A. Riekert, and F. Rossmannek [3]. Students in real analysis could engage with this material, and see the connections between the concepts of convergence and stability of solutions of differential equations with new research in neural networks and wavelets.

5.2 Stochastic gradient decent

Recall from Figure 2, we are concerned with getting trapped in valleys when using gradient descent, and not making it to the base of the mountain range. One way we can avoid such a pitfall is to include stochasticity into the gradient descent algorithm, meaning we are going to include some randomness. Specifically, by adding stochasticity, we allow the direction to include some randomness, which means that when we are stuck in a basin, rather than remaining at the local minimum, we might choose a new direction which allows us to leave the local minimum in search of the global minimum. In upper-level mathematics classes, for an in-depth mathematical and statistical analysis of stochastic gradient descent, we suggest the reference *Analysis of stochastic gradient descent in continuous time* (2021) by J. Latz [13].

5.3 Support Vector Machines

We saw in Lesson 4, Section 4, that logistic regression can be used to predict a binary ("yes" or "no") decision. There are many different machine learning *classification* algorithms that are used to predict which class, or category, a sample belongs to based on some input measurements. Clustering, decision trees, and k-nearest neighbors are among these algorithms. Support vector machines are also used in classification problems, and require the solution of an optimization problem. Support vector machines use only numeric predictor variables, and output only binary response variables. The key idea behind support vector machines is to find "optimal" decision boundaries, so this idea ties nicely into the simpler examples presented in Section 4. One way to solve these optimization problems is by using some form of gradient descent. In upper level mathematics classes, for an in-depth study of support vector machines, we suggest the references: *What is a support vector machine?* (2006) by W. S. Noble. [19], *Support vector machine* (2016) by S. Suthaharan [25], or *Support vector machine* (2020) by D. A. Pisner and D. M. Schnyer [21]. Specifically, *Support vector machine* (2020) by D. A. Pisner and D. M. Schnyer [21], covers the background and intuition, along with extensions relating to brain disorders.

References

[1] Ahmad Banakar, *Lyapunov stability analysis of gradient descent-learning algorithm in network training*, International Scholarly Research Notices **2011** (2011).

[2] Joseph Blitzstein and Jessica Hwang, *Introduction to probability*, Available at https://projects.iq.harvard.edu/stat110/home (2023-09-16), 2019.

[3] Patrick Cheridito, Arnulf Jentzen, Adrian Riekert, and Florian Rossmannek, *A proof of convergence for gradient descent in the training of artificial neural networks for constant target functions*, Journal of Complexity (2022), 101646.

[4] Christopher Chudzicki, *Math3d: Online 3d Graphing Calculator*, Available at https://www.math3d.org (2022-02-18), 2022.

[5] desmos, *Desmos Graphing Calculator*, Available at https://www.desmos.com (2022-02-18), 2022.

[6] Christina J. Edholm, Maryann E. Hohn, and Ami E. Radunskaya, *Desmos Graphing Calculator Plot for Lesson 1 Figure 3 Example*, Available at https://www.desmos.com/calculator/kw6ie2jwzi (2022-05-20), 2022.

[7] _____, *Gradient Descent Methods in Machine Learning*, Available at https://github.com/cedholm/Gradient-descent-methods-in-machine-learning (2022-05-20), 2022.

[8] _____, *Math3d: Online 3d Graphing Calculator Plot for Lesson 1 Figure 3 Example*, Available at https://www.math3d.org/e5oNeSrOH (2022-05-20), 2022.

[9] _____, *Math3d: Online 3d Graphing Calculator Plot for Lesson 1 Figure 4 Example*, Available at https://www.math3d.org/264fij5Rh (2022-05-20), 2022.

[10] esri, *ArcGIS: USA Topo Maps*, Available at https://www.arcgis.com/home/item.html?id=99cd5fbd98934028802b4f797c4b1732 (2022-05-20), 2011.

[11] Andrew Green, *Topographic (with Contours) Multisource vector tile layers*, Available at https://www.esri.com/arcgis-blog/products/arcgis-online/announcements/topographic-with-contours-multisource-vector-tile-layers/ (2022-05-20), 2022.

[12] M Yasser H, *Loan Default Dataset*, Available at https://www.kaggle.com/yasserh/loan-default-dataset (2022-02-20), 2022.

[13] Jonas Latz, *Analysis of stochastic gradient descent in continuous time*, Statistics and Computing **31** (2021), no. 4, 1–25.

[14] Richard Borshay Lee, Richard B Lee, and Irven DeVore, *Kalahari hunter-gatherers: Studies of the !kung san and their neighbors*, Harvard University Press, 1976.

[15] Richard McElreath, *Statistical rethinking: A Bayesian course with examples in R and Stan*, Chapman and Hall/CRC, 2020.

[16] _____, *Statistical Rethinking book package*, Available at https://github.com/rmcelreath/rethinking (2022-02-20), 2022.

[17] _____, *Statistical rethinking Howell1 Data on github*, Available at https://github.com/rmcelreath/rethinking/blob/master/data/Howell1.csv (2022-02-20), 2022.

[18] Andrew Ng, *OpenClassroom Machine Learning*, Available at http://openclassroom.stanford.edu/MainFolder/CoursePage.php?course=MachineLearning (2022-02-20), 2022.

[19] William S Noble, *What is a support vector machine?*, Nature biotechnology **24** (2006), no. 12, 1565–1567.

[20] Anup Pandey, *College Admission Data Set*, Available at https://www.kaggle.com/pandanup/college-admission-data-set (2022-02-20), 2022.

[21] Derek A Pisner and David M Schnyer, *Support vector machine*, Machine learning, Elsevier, 2020, pp. 101–121.

[22] Grant Sanderson, *3Blue1Brown*, Available at https://www.3blue1brown.com (2022-02-18), 2022.

[23] _____, *Gradient descent, how neural networks learn*, Available at https://www.3blue1brown.com/lessons/gradient-descent (2022-05-20), 2022.

[24] _____, *Neural Networks*, Available at https://www.3blue1brown.com/topics/neural-networks (2022-05-20), 2022.

[25] Shan Suthaharan, *Support vector machine*, Machine learning models and algorithms for big data classification, Springer, 2016, pp. 207–235.

[26] USGS, *topoView*, Available at https://ngmdb.usgs.gov/topoview/viewer/#4/39.98/-100.06 (2022-05-20), 2022.

[27] _____, *US Topo: Maps for America*, Available at https://www.usgs.gov/programs/national-geospatial-program/us-topo-maps-america (2022-05-20), 2022.