BST 261: Data Science II

Lecture 3

Heather Mattie

Department of Biostatistics
Harvard T.H. Chan School of Public Health
Harvard University

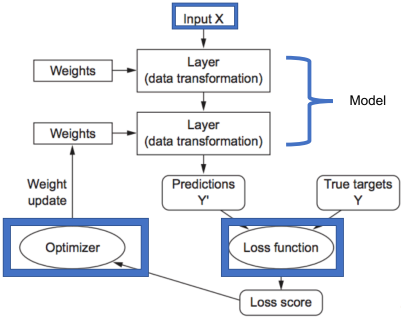
March 26, 2018

Machine Learning Basics

Learning Algorithms

- A machine learning algorithm is an algorithm that is able to learn from data
- But what do we mean by learning?
- "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E" (Mitchell, 1997)
- Nearly all learning algorithms can be described as particular instances of a simple recipe: combine a **dataset**, **model**, **cost function**, and **optimization procedure**
- By realizing that we can replace any of these components mostly independently from others, we can obtain a wide range of algorithms

Machine Learning Recipe



Learning Algorithms: Task T

- The process of “learning" itself is not the task
- Instead, learning is our means of attaining the ability to perform the task
- Tasks are usually described in terms of how the system should process an **example**
- An example is a collection of features from some object or event we want the system to process
- We typically represent an example as a **column** vector $\boldsymbol{x} \in \mathbb{R}^n$ where x_i is feature i of $\boldsymbol{x}$

ID	age	smoke	toxemia	bmi	gestation
1	25	1	0	27.3	32
2	31	0	1	28.4	37
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

- Common machine learning tasks: classification, classification with missing inputs, regression, transcription, machine translation, anomaly detection, synthesis and sampling, imputation of missing values, denoising, density estimation

Learning Algorithms: Task T

Classification

- Need to specify which of k categories some input belongs to
- Learning algorithm produces a function $f : \mathbb{R}^n \to \{1, \dots, k\}$
- When $y = f(\boldsymbol{x})$, the model assigns an input described by vector $\boldsymbol{x}$ to a category identified by numeric code y

-Example: classifying handwritten digits as 0 - 9 (10 possible classes or categories)

- Another variant of this problem is where f outputs a probability distribution over classes
- Example of a classification task is object recognition, where the input is an image and the output is a numeric code identifying the object in the image

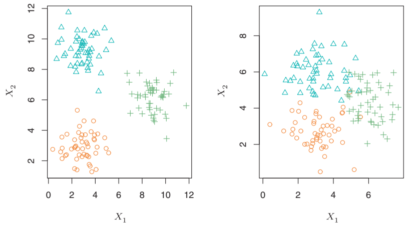


Figure: Well-separated groups (left) and overlapping groups (right). Algorithm needs to determine class boundaries (not shown). Source: ISL

Learning Algorithms: Task T

Regression

- Asked to predict a numerical value given some input
- Learning algorithm produces a function $f : \mathbb{R}^n \to \mathbb{R}$
- Similar to classification, except that the format of output is different (continuous)
- Example of a regression task is the prediction of annual income from individual-level covariates, such as years of education

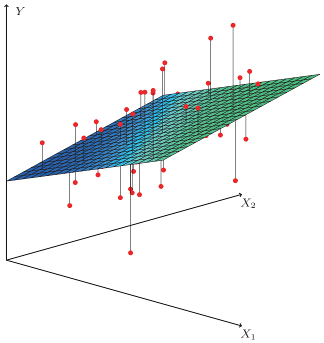


Figure: Least squares regression plane for two predictors and one response. Source: ISL

Learning Algorithms: Performance Measure P

- To evaluate the ability of a learning algorithm, we must design a quantitative measure of its performance
- Usually this performance measure P is specific to the task T being carried out
- For tasks such as classification, we often measure **accuracy**, the proportion of examples for which the model produces the correct output
- Equivalently can use **error rate**, the proportion of examples for which the model produces an incorrect output
- The **0 - 1 loss** on a particular example is 0 if it is correctly classified and 1 if it is not
- The term **expected 0 - 1 loss** is often used to refer to the error rate
- For others tasks, such as regression, need a different performance metric that gives the model a continous-valued score for each example
    - Example: RMSE, MSE, etc.

Learning Algorithms: Experience E

- Machine learning algorithms can be broadly categorized as unsupervised or supervised depending on what kind of data (experience) is available
- **Unsupervised learning algorithms** experience a dataset containing features and attempt to learn useful properties of the structure of the dataset
- Examples include learning the probability distribution that generated a dataset, or clustering a dataset into clusters of similar examples
- **Supervised learning algorithms** experience a dataset containing features, but in addition, each example is associated with a **label** or **target**
- Classic example is the Iris dataset by Fisher, which consists of measurements of 150 iris plants (sepal length, petal width, etc.) and their species

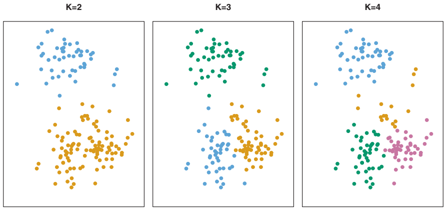


Figure: Simulated dataset with K-means clustering using $K \in \{2, 3, 4\}$. Source: ISL

Learning Algorithms: Example

- The goal is to build a system that can take vector $\boldsymbol{x} \in \mathbb{R}^n$ as input and predict the value of a scalar $y \in \mathbb{R}$ as its output
- We define the output to be

$$\hat{y} = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b \tag{1}$$

- Here $\boldsymbol{w} \in \mathbb{R}^n$ is a vector and b is a scalar, and are referred to as attributes of a layer in a neural network
- $\boldsymbol{w}$ and b are called weights or **trainable parameters**; $\boldsymbol{w}$ is the *kernel* attribute and b is the *bias* attribute
- $\hat{y}$ denotes the value that our model predicts for y
- The produced output $\hat{y}$ is a linear combination of the features, i.e., the parameter or coefficient w_i multiplies the feature x_i, b is added, and the contributions from all the features are summed up
- Our task T in this case: predict y from $\boldsymbol{x}$ by outputting $\hat{y} = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b$; try to learn $\boldsymbol{w}$ and b so $\hat{y}$ is accurate

Example: Logistic Regression

- Suppose we want to predict if a baby will be low birthweight i.e. babies who are born weighing less than 2,500 grams (5 pounds, 8 ounces)
- If the predicted probability of being low birthweight is high, we classify that baby as low birthweight with a label of 1, and 0 otherwise
- We are given several covariates including the mother's age, bmi, smoking status, toxemia status, etc.

ID	age	smoke	toxemia	bmi	gestation
1	25	1	0	27.3	32
2	31	0	1	28.4	37
⋮	⋮	⋮	⋮	⋮	⋮

Example: Logistic Regression

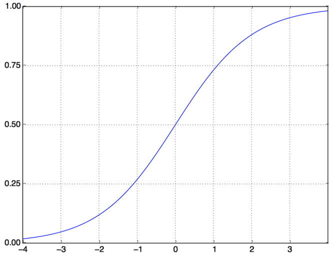
- Given a feature vector $\boldsymbol{x} \in \mathbb{R}^n$, we want to calculate $\hat{y} = P(y = 1|\boldsymbol{x})$, with $0 \leq \hat{y} \leq 1$
- A single training example is denoted (x, y)
- A set of m training examples is denoted $\{(x^{(1)}, y^{(1)}), ..., (x^{(m)}, y^{(m)})\}$ and we want to calculate $\hat{y}^{(i)} \approx y^{(i)}$
- Each training example feature vector is made into a column vector and these column vectors are combined to create a feature matrix $\boldsymbol{X}^{n\times m}$
- The outcomes $y^{(i)}$ are combined to form an outcome vector $\boldsymbol{Y}^{1\times m}$

$$\boldsymbol{X} = \begin{bmatrix} | & | & \dots & | \\ | & | & \dots & | \\ x^{(1)} & x^{(2)} & \dots & x^{(m)} \\ | & | & \dots & | \\ | & | & \dots & | \end{bmatrix}, \boldsymbol{Y} = \begin{bmatrix} y^{(1)} & y^{(2)} & \dots & y^{(m)} \end{bmatrix} \tag{2}$$

Example: Logistic Regression

- Parameters: $\boldsymbol{w} \in \mathbb{R}^n$, $b \in \mathbb{R}$
- Output: $\hat{y} = \sigma(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b) = \sigma(z)$
- Logistic sigmoid function:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{3}$$



Example: Logistic Regression

- Now we need to specify a loss or error function to train our model
- For logistic regression we will use

$$\mathscr{L}(\hat{y}, y) = -[ylog(\hat{y}) + (1-y)log(1-\hat{y})] \tag{4}$$

- If $y = 1$: $\mathscr{L}(\hat{y}, y) = -log(\hat{y})$
- If $y = 0$: $\mathscr{L}(\hat{y}, y) = -log(1-\hat{y})$
- This is just for 1 training example. If we have m training examples, the loss function is now called the **cost** function and would be

$$\begin{aligned} J(\boldsymbol{w}, b) &= \frac{1}{m}\sum_{i=1}^{m}\mathscr{L}(\hat{y}, y) && (5) \\ &= -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}log(\hat{y}^{(i)}) + (1-y^{(i)})log(1-\hat{y}^{(i)})] && (6) \end{aligned}$$

- This is known as the **binary cross-entropy loss function**
- We want to find $\boldsymbol{w}, b$ that minimize $J(\boldsymbol{w}, b)$
- How do we do this?

Gradient-Based Optimization in Statistical Learning

Gradient-Based Optimization

- Many learning algorithms involve some sort of optimization
- Optimization refers to the task of either minimizing or maximizing some function $f(\boldsymbol{x})$ by altering the value of $\boldsymbol{x}$
- Optimization problems are typically formulated in terms of minimizing $f(\boldsymbol{x})$, and maximization can be accomplished by minimizing $-f(\boldsymbol{x})$
- The function to be minimized is called the **objective function**, **criterion**, **cost function**, **loss function**, or **error function**
- The value that minimizes a function is denoted by $\boldsymbol{x}^* = \arg\min f(\boldsymbol{x})$
- In some cases, such as with principal components analysis, analytical optimization is possible
- Of all the many optimization problems involved in deep learning, the most difficult is neural network training

Gradient-Based Optimization

- Optimization makes use of familiar concepts from calculus
- Suppose we have a function $y = f(x)$ where both x and y are real numbers
- The **derivative** of this function is denoted as $f'(x)$ or $\frac{dy}{dx}$
- The derivative $f'(x)$ gives the slope of $f(x)$ at the point x, specifying how a small change in x affects the value of y: $f(x+\epsilon) \approx f(x) + \epsilon f'(x)$
- We can therefore use the derivative to minimize a function because it tells us how to change x in order to decrease the value of y
- Reducing the value of $f(x)$ by moving x in small steps with the opposite sign of the derivate is called **gradient descent**

Gradient-Based Optimization

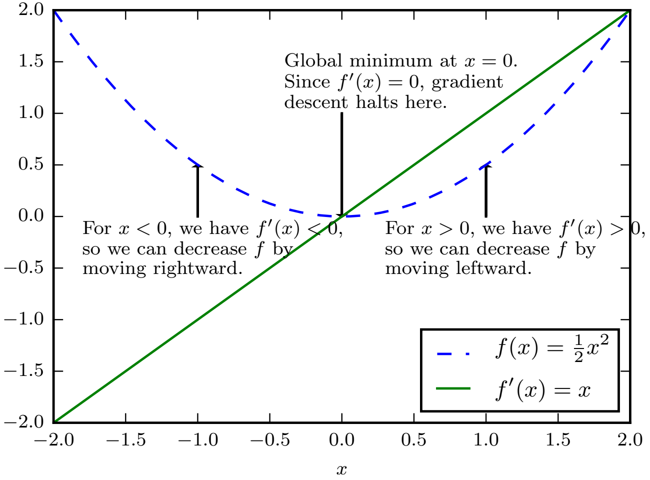


Figure: Illustration of how the gradient descent algorithm uses the derivatives of a function to follow the function downhill to a minimum. Source: DL.

Gradient-Based Optimization

- Points where $f'(x) = 0$ are called **critical points** or **stationary points**
- At these points, the derivative provides no information about which direction to move
- A **local minimum** is a point where $f(x)$ if lower than at all neighboring points, so it is no longer possible to decrease $f(x)$ by making infinitesimal steps
- A **local maximum** is a point where $f(x)$ if higher than at all neighboring points, so it is no longer possible to increase $f(x)$ by making infinitesimal steps
- Some critical points are neither maxima nor minima, and they are known as **saddle points**

Gradient-Based Optimization

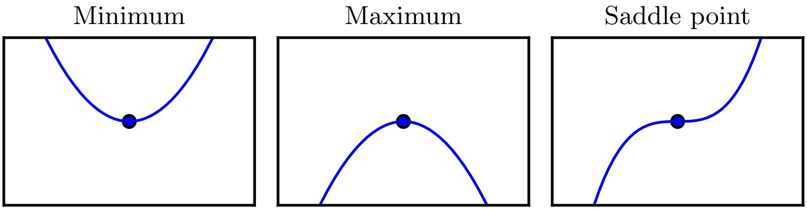


Figure: Three types of critical points in one dimension. A local minimum is lower than the neighboring points; a local maximum is higher than the neighboring points; a saddle point has neighbors that are both higher and lower than the point itself. Source: DL.

Gradient-Based Optimization

- A point that obtains the absolute lowest value of $f(x)$ is a **global minimum**
- There may be one global minimum or multiple global minima
- It is also possible for there to be local minima that are not globally optimal
- It is common in many settings to settle for a value of f that is very low but not necessarily minimal

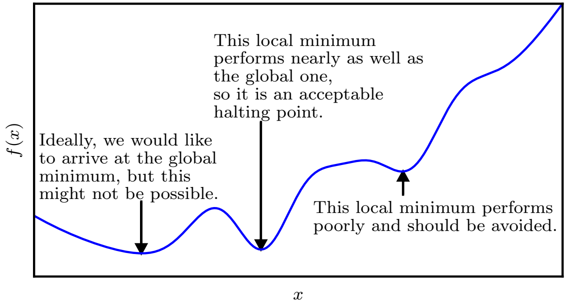


Figure: Approximate minimization. Source: DL.

Gradient-Based Optimization

- We often minimize functions that have multiple inputs: $f : \mathbb{R}^n \to \mathbb{R}$
- For the concept of minimization to make sense, there must still be only one scalar output, i.e., f must be scalar-valued
- For functions with multiple inputs, we must make use of **partial derivatives**
- The partial derivative $\frac{\partial}{\partial x_i} f(\boldsymbol{x})$ quantifies how f changes as only the variable x_i increases at point $\boldsymbol{x}$
- The **gradient** generalizes the notion of derivative to the case where the derivative is with respect to a vector
- The gradient of f, denoted $\nabla_{\boldsymbol{x}} f(\boldsymbol{x})$, is the vector containing all the partial derivatives, where element i of the gradient is the partial derivative of f with respect to x_i
- In multiple dimensions, critical points are points where every element of the gradient is equal to zero, i.e., the gradient is equal to the zero-vector $\mathbf{0}$

Gradient-Based Optimization

- To minimize f, we would like to find the direction in which f decreases the fastest
- One can show that the gradient points directly uphill, and the negative gradient points directly downhill
- We can therefore decrease f by moving in the direction of the negative gradient, which is known as the **method of steepest descent** or **gradient descent**
- Steepest descent proposes a new point $\boldsymbol{x}'$ as

$$\boldsymbol{x}' = \boldsymbol{x} - \epsilon \nabla_{\boldsymbol{x}} f(\boldsymbol{x}) \tag{7}$$

- Here ϵ is the **learning rate**, a positive scalar determining the size of the step
- A popular approach is to set ϵ to a small constant
- Although gradient descent is limited to optimization in continous spaces, the general concept of repeatedly making a small move towards better configurations can be generalized to discrete spaces
- Ascending an objective function of discrete parameters is called **hill climbing**

Stochastic Gradient Descent

- Stochastic gradient descent is an extension of the gradient descent algorithm, and it powers nearly all of deep learning
- A recurring problem is that large training sets are necessary for good generalization, but they are also computationally more expensive
- The cost function used by a machine learning algorithm often decomposes as a sum over training examples of some per-example loss function
- In the context of maximum likelihood estimation, we know that the cost function can be written as an expectation with respect to the empirical distribution $\hat{p}_{\text{data}}$:

$$\boldsymbol{\theta}_{\text{ML}} = \arg\max_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{x}\sim\hat{p}_{\text{data}}} \log p_{\text{model}}(\boldsymbol{x};\boldsymbol{\theta}) \tag{8}$$

- For example, the negative conditional log-likelihood of the training data can be written as

$$J(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{x},y\sim\hat{p}_{\text{data}}} L(\boldsymbol{x},y,\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m} L(\boldsymbol{x}^{(i)},y^{(i)},\boldsymbol{\theta}) \tag{9}$$

- Here $L(\boldsymbol{x},y,\boldsymbol{\theta}) = -\log p(y|\boldsymbol{x};\boldsymbol{\theta})$ is the per-example loss

Stochastic Gradient Descent

- For additive cost functions, gradient descent requires computing

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}} L(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta}) \tag{10}$$

- The problem is that the computational cost of this operation is $O(m)$, so if the training set size is very large, the time to compute a single gradient step becomes prohibitively long
- The insight of stochastic gradient descent is that the gradient is an expectation over $\hat{p}_{\mathrm{data}}$, and it may be approximately estimated using a small set of samples
- On each step of the algorithm, we sample a **minibatch** of examples $\mathbb{B} = \{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m')}\}$ drawn uniformly from the training set
- The minibatch size m' is typically chosen to be relatively small, ranging from one to a few hundred, and it is usually held fixed as the training set size m grows

Stochastic Gradient Descent

- The estimate of the gradient is formed using samples from the minibatch $\mathbb{B}$

$$\boldsymbol{g} = \frac{1}{m'} \nabla_{\boldsymbol{\theta}} \sum_{i=1}^{m'} L(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta}) \tag{11}$$

- The stochastic gradient descent algorithm then follows the estimated gradient downhill with learning rate ϵ

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \boldsymbol{g} \tag{12}$$

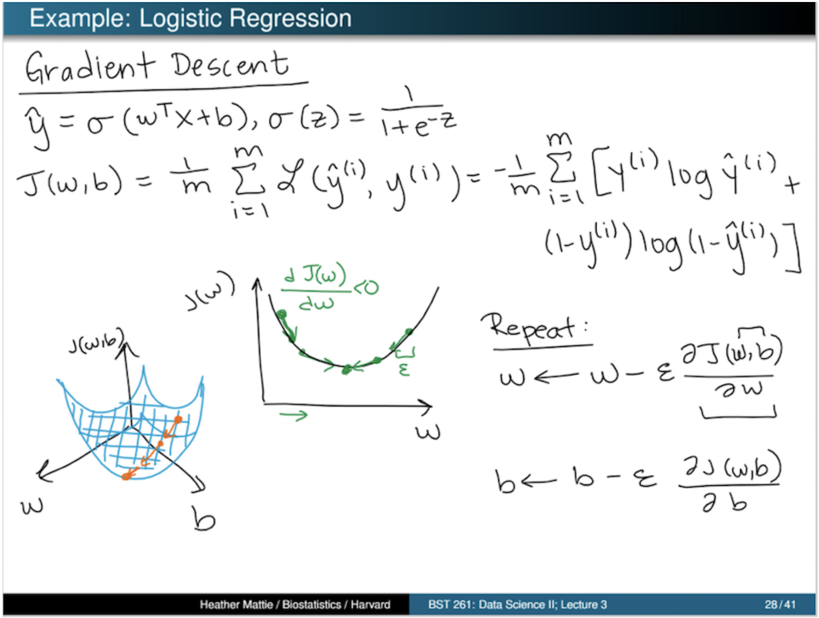
- While the use of gradient descent to nonconvex optimization problems was regarded unprincipled, today we know that machine learning models work well when trained with gradient descent
- The optimization algorithm is not guaranteed to arrive at even a local minimum, but in practice it often finds a low enough value of the cost function to be useful
- SGD and its variants are probably the most used optimization for machine learning in general and for deep learning in particular

Stochastic Gradient Descent

- Gradient descent (GD) follows the gradient of the entire training set downhill
- Stochastic gradient descent (SGD) follows the gradient of randomly selected minibatches downhill
- SGD is considerably faster than GD, but it also introduces a source of noise not present in GD, which is the noise due to random sampling of m' training examples
- While the gradient of the total cost function becomes small and then $\mathbf{0}$ as we approach and reach a minimum in GD, the sampling noise in SGD does not vanish
- To overcome this, one makes the learning rate ϵ adaptive, so that instead of using a fixed learning rate, we gradually decrease the learning rate over time
- It is common to decay the learning rate at iteration k, denoted ϵ_k, linearly until iteration τ, after which ϵ is left to a constant value (here $\alpha = k/\tau$):

$$\epsilon_k = (1-\alpha)\epsilon_0 + \alpha\epsilon_\tau \tag{13}$$

- The parameters to choose are ϵ_0, ϵ_τ, and τ
- It is common to plot the cost function as a function of time to monitor learning

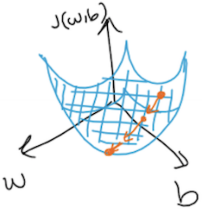


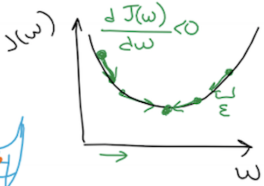
Example: Logistic Regression

Gradient Descent

$$\hat{y} = \sigma(w^T x + b), \quad \sigma(z) = \frac{1}{1+e^{-z}}$$

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) = -\frac{1}{m}\sum_{i=1}^{m} \left[y^{(i)} \log \hat{y}^{(i)} + (1-y^{(i)}) \log(1-\hat{y}^{(i)}) \right]$$





Repeat:

$$w \leftarrow w - \varepsilon \underbrace{\frac{\partial J(\overbrace{w,b})}{\partial w}}$$

$$b \leftarrow b - \varepsilon \frac{\partial J(w,b)}{\partial b}$$

Heather Mattie / Biostatistics / Harvard BST 261: Data Science II; Lecture 3 28 / 41

Example: Logistic Regression

$z = w^T x + b$

$\hat{y} = a = \sigma(z) = \frac{1}{1+e^{-z}}$

$\mathcal{L}(a,y) = -[y \log(a) + (1-y)\log(1-a)]$

$\rightarrow$: forward propagation

$\leftarrow$: backward propagation

$$\begin{tikzcd}
x_1 \arrow[r] & \boxed{z = w_1 x_1 + w_2 x_2 + b} \arrow[r] & \boxed{\hat{y} = a = \sigma(z)} \arrow[r] & \mathcal{L}(a,y) \\
w_1 \arrow[ur, shift left] & & & \\
x_2 \arrow[uur] & & & \\
w_2 \arrow[uuur, shift left] & & & \\
b \arrow[uuuur, shift left] & & &
\end{tikzcd}$$

$\frac{\partial \mathcal{L}(a,y)}{\partial z} = \boxed{\frac{\partial \mathcal{L}}{\partial a}} \underbrace{\boxed{\frac{\partial a}{\partial z}}}_{a(1-a)} \leftarrow \frac{\partial \mathcal{L}(a,y)}{\partial a} = \boxed{\frac{-y}{a} + \frac{1-y}{1-a}}$

$\swarrow J$

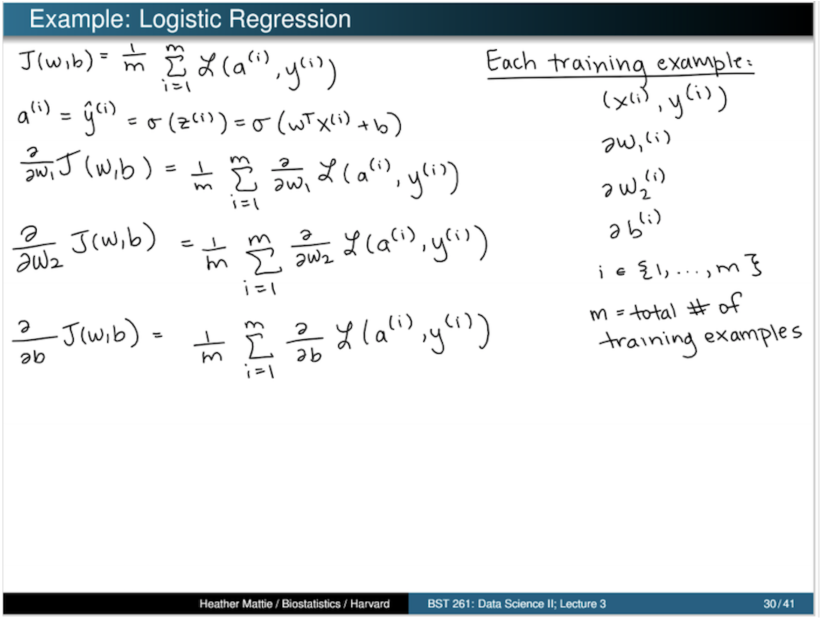
$\frac{\partial \mathcal{L}}{\partial w_1} = \frac{\partial \mathcal{L}}{\partial z} \frac{\partial z}{\partial w_1}$

Update: $w_1 \leftarrow w_1 - \varepsilon \frac{\partial \mathcal{L}}{\partial w_1}$

• similarly, update w_2 and b

Note: this is for one training example

Note: $\mathcal{L}(\,)$ refers to the loss function (one training example)
$J(\,)$ refers to the cost fuction (all m training examples)



Example: Logistic Regression

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}(a^{(i)}, y^{(i)})$$

$$a^{(i)} = \hat{y}^{(i)} = \sigma(z^{(i)}) = \sigma(w^T x^{(i)} + b)$$

$$\frac{\partial}{\partial w_1} J(w,b) = \frac{1}{m}\sum_{i=1}^{m}\frac{\partial}{\partial w_1}\mathcal{L}(a^{(i)}, y^{(i)})$$

$$\frac{\partial}{\partial w_2} J(w,b) = \frac{1}{m}\sum_{i=1}^{m}\frac{\partial}{\partial w_2}\mathcal{L}(a^{(i)}, y^{(i)})$$

$$\frac{\partial}{\partial b} J(w,b) = \frac{1}{m}\sum_{i=1}^{m}\frac{\partial}{\partial b}\mathcal{L}(a^{(i)}, y^{(i)})$$

Each training example:

$(x^{(i)}, y^{(i)})$

$\partial w_1^{(i)}$

$\partial w_2^{(i)}$

$\partial b^{(i)}$

$i \in \{1, \dots, m\}$

m = total # of training examples

Heather Mattie / Biostatistics / Harvard BST 261: Data Science II; Lecture 3 30 / 41

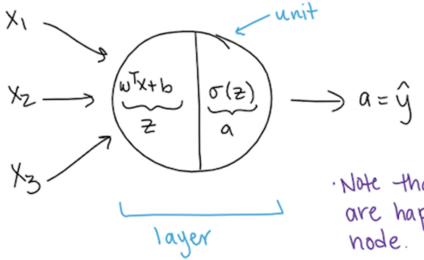
Activation and Loss Functions

- We'll get into activation functions for hidden layers soon - for now, this is a handy list when choosing the last-layer activation function and loss function

Problem Type	**Last-layer activation**	**Loss Function**
Binary Classification	`sigmoid`	`binary_crossentropy`
Multiclass, single-label classification	`softmax`	`categorical_crossentropy`
Multiclass, multilabel classification	`sigmoid`	`binary_crossentropy`
Regression to arbitrary values	None	`mse`
Regression to values between 0 and 1	`sigmoid`	`mse` or `binary_crossentropy`

Neural Network Representation

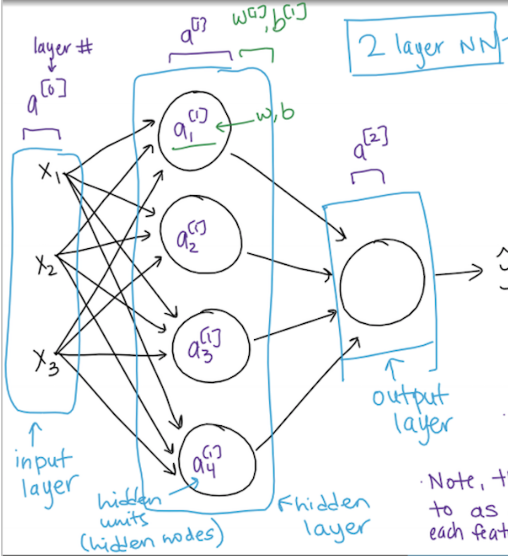
The example we've seen so far can be represented as a neural network with one layer and one node or unit.



· Note that 2 calculations are happening within the node.

· Note: this is what is referred to as the output layer - we don't have any hidden layers in this example.

Neural Network Representation



$\boxed{\text{2 layer NN}}$ - the input layer (feature vector) is <u>not</u> counted as a layer.

$$a^{[1]} = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \\ a_4^{[1]} \end{bmatrix}$$

· Here, $a^{[2]}$ is a scalar $= \hat{y}$

· Note, this network is referred to as a "dense" network since each feature is connected to each node.

Neural Network Representation

$$W^{[1]} = \begin{bmatrix} w^{[1]}_{1,1} & w^{[1]}_{2,1} & w^{[1]}_{3,1} \\ w^{[1]}_{1,2} & w^{[1]}_{2,2} & w^{[1]}_{3,2} \\ w^{[1]}_{1,3} & w^{[1]}_{2,3} & w^{[1]}_{3,3} \\ w^{[1]}_{1,4} & w^{[1]}_{2,4} & w^{[1]}_{3,4} \end{bmatrix}$$

- For one training example:

$$\begin{tikzcd} & w^{[\ell]}_{j,k} & \text{layer \#} \arrow[l] \\ \text{feature \#} \arrow[ur] & & \text{hidden unit \#} \arrow[ul] \end{tikzcd}$$

- For m training examples:

$$\begin{tikzcd} \text{layer \#} \arrow[r] & w^{[\ell](i)}_{j,k} & \text{training example \#} \arrow[l] \\ \text{feature \#} \arrow[ur] & & \text{hidden unit \#} \arrow[ul] \end{tikzcd}$$

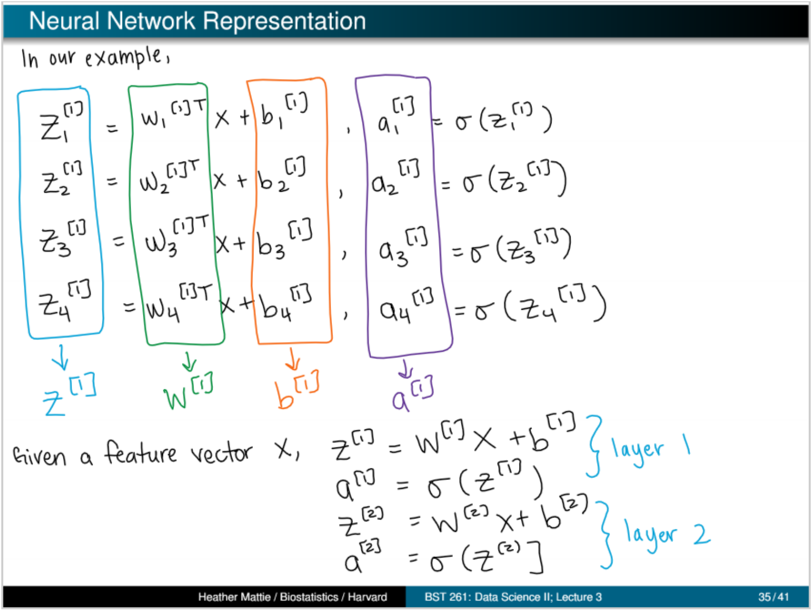
- For our example,

$\ell \in \{1, 2\}$ (2 layers)

$i \in \{1, \ldots, m\}$ (we only saw 1 example)

$j \in \{1, 2, 3\}$ (x_1, x_2, x_3)

$k \in \{1, 2, 3, 4\}$ (4 hidden units)



Neural Network Representation

In our example,

$$
\begin{aligned}
z_1^{[1]} &= w_1^{[1]T} x + b_1^{[1]}, & a_1^{[1]} &= \sigma(z_1^{[1]}) \\
z_2^{[1]} &= w_2^{[1]T} x + b_2^{[1]}, & a_2^{[1]} &= \sigma(z_2^{[1]}) \\
z_3^{[1]} &= w_3^{[1]T} x + b_3^{[1]}, & a_3^{[1]} &= \sigma(z_3^{[1]}) \\
z_4^{[1]} &= w_4^{[1]T} x + b_4^{[1]}, & a_4^{[1]} &= \sigma(z_4^{[1]})
\end{aligned}
$$

$$\downarrow z^{[1]} \quad \downarrow W^{[1]} \quad \downarrow b^{[1]} \quad \downarrow a^{[1]}$$

Given a feature vector x,

$$
\left.\begin{aligned}
z^{[1]} &= W^{[1]} x + b^{[1]} \\
a^{[1]} &= \sigma(z^{[1]})
\end{aligned}\right\} \text{layer 1}
$$

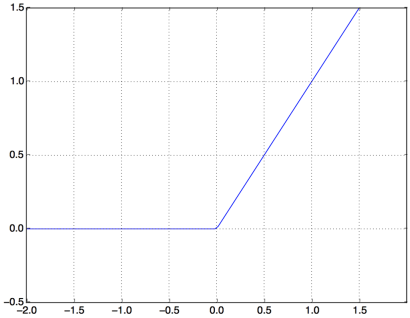
$$
\left.\begin{aligned}
z^{[2]} &= W^{[2]} x + b^{[2]} \\
a^{[2]} &= \sigma(z^{(2)}]
\end{aligned}\right\} \text{layer 2}
$$

ReLU

- A common activation function for hidden layers is the **Re**ctified **L**inear **U**nit function
- It zeros out negative values

$$g(z) = max(0, z) \tag{14}$$

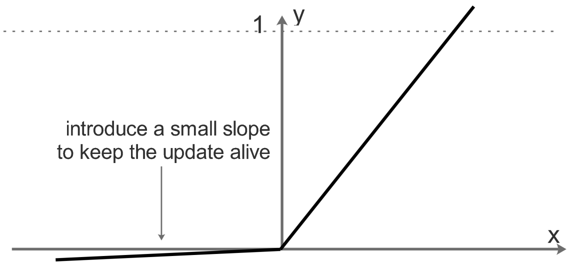
- Derivative is 0 for $x < 0$ and 1 for $x > 0$
- Derivative isn't defined for $x = 0$, but not a problem because of the limited precision of computers



Leaky ReLU

- Usually performs better in practice, but less commonly used

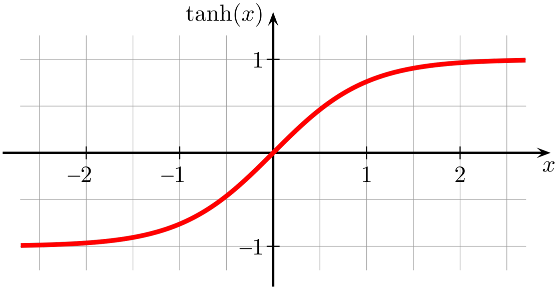
$$g(z) = max(0.01z, z) \tag{15}$$



Hyperbolic Tangent

- Performs better than the logistic sigmoid
- Not as commonly used as the ReLU function

$$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{16}$$



Activation Functions

- The ReLU function is by far the most popular and commonly used
- If you're not sure which one to use, try them all and see which works best

Optimization Algirithms

- Gradient Descent
- Stochastic Gradient Descent
- RMS prop
- Adam
- Adagrad
- Momentum