Lecture Notes

Part 2 of 2

Mathematical Foundations of Computational Linguistics

Manfred Klenner, Jannis Vamvas

Spring Semester 2024

University of Zurich

Contents

Chapter 10

Derivatives

Motivation

The steepness or *slope* of a linear function can be calculated using the first derivative:

$$\begin{aligned} f(x) &= -2x + 4 \\ f'(x) &= -2 \end{aligned}$$

An alternative notation for the derivative of a function $f(x)$ is $\frac{df}{dx}$:

$$f'(x) = \frac{df}{dx} = -2$$

In machine learning, we calculate derivatives for finding the optimal weights of classifier functions, i.e., the weights that produce as little classification error as possible.

Rules for Calculating Derivatives

Constant:

$$f(x) = c \quad \Rightarrow \quad f'(x) = 0$$

Multiplication:

$$f(x) = cx \quad \Rightarrow \quad f'(x) = c$$

Exponent:

$$f(x) = x^n \quad \Rightarrow \quad f'(x) = nx^{n-1}$$

Logarithm:

$$f(x) = \log_a(x) \quad \Rightarrow \quad f'(x) = \frac{1}{x\ln(a)}$$

Sum rule:

$$f(x) = g(x) + h(x) \quad \Rightarrow \quad f'(x) = g'(x) + h'(x)$$

Chain rule:

$$f(x) = g(h(x)) \quad \Rightarrow \quad f'(x) = g'(h(x)) \cdot h'(x)$$

Partial Derivatives

When a function takes multiple input variables $x_1, x_2, \ldots$, we calculate a separate derivative for each variable. This is called a partial derivative and denoted with the symbol ∂.

To calculate the partial derivative of a function f with respect to a variable x_i, we treat all other variables as constants:

$$f(x_1, x_2) = 2x_1 + x_2 - 4$$

$$\frac{\partial f}{\partial x_1} = 2 \quad \text{and} \quad \frac{\partial f}{\partial x_2} = 1$$

Reading example: The partial derivative of f with respect to x_1 is 2 and the partial derivative of f with respect to x_2 is 1. x_2 is treated as a constant when calculating the partial derivative with respect to x_1, and vice versa.

The vector of all the partial derivatives of f is called the gradient:

$$\nabla f(x_1, x_2) = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$$

Chapter 11

Linear Regression

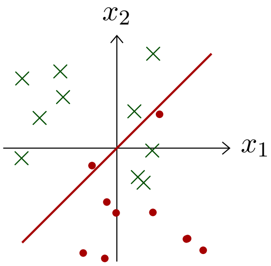
Motivation

By linear regression, we mean the process of finding a linear function that best fits a set of data points.

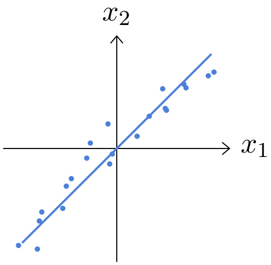
Each data point corresponds to an object, which is represented by a vector of features $\mathbf{x}$. The input to the linear function is the feature vector, and the output is the predicted value.

There are two use cases of linear regression:

Classification: The data points are labeled with a class (e.g., positive or negative). The goal is to find a linear function that separates the classes as neatly as possible. We call the data points *linearly separable* if there exists a linear function that perfectly separates the classes



Regression: The data points are labeled with a real number (e.g., the price of a house). The goal is to find a linear function that can predict the label of a new data point as closely as possible.



Loss Function

In both cases, we need a way to measure how well a linear function fits the data points. This is the loss function.

A typical loss function for regression is the mean squared error (MSE):

$$\mathrm{MSE}(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2,$$

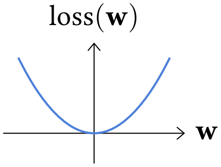
where n is the number of data points, y_i is the label of the ith data point, and $\hat{y}_i$ is the prediction of the linear function for the ith data point. With $\mathbf{w}$ we denote the weights of the linear function.

The MSE loss quantifies the overall difference between the actual labels and the labels predicted by a linear function parameterized by $\mathbf{w}$.

Our goal is to find weights that minimize the loss function.

Minimum of the Loss Function

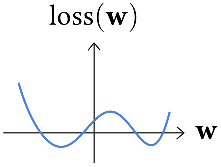
A *convex* function is a function that has a single minimum:



An example of such a function is the MSE loss.

The *global minimum* of a function is the minimum over the entire domain of the function. A *local minimum* of a function is the minimum within a small region of the function.

Non-convex functions can have multiple local minima:



Convex functions are easier to optimize than non-convex functions because it is easier to find their global minimum.

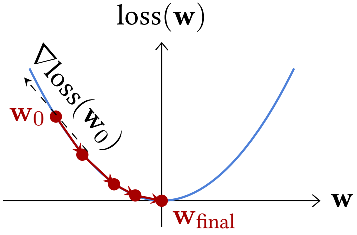
Gradient Descent

Gradient descent is an iterative optimization algorithm that can be used for finding the weights that minimize a loss function.

The idea is to start with an initial guess for the weights and to iteratively update the weights.

To know how to update the weights, we need to calculate the gradient of the loss function with respect to the weights. If the loss function is convex, the gradient leads us to the global minimum:

- If the partial derivative w.r.t. a weight is positive, the weight needs to be decreased.
- If the partial derivative w.r.t. a weight is negative, the weight needs to be increased.
- When all partial derivatives are zero, we have reached the minimum (point of *convergence*).


$\text{loss}(\mathbf{w})$
$\nabla\text{loss}(\mathbf{w}_0)$
$\mathbf{w}_0$
$\mathbf{w}$
$\mathbf{w}_{\text{final}}$

Learning Rate

The gradient tells us in which direction we need to update the weights – but by how much? We need to select a learning rate η that determines the size of the update steps.

The update rule for a weight w_i is:

$$\Delta w_i = -\eta \frac{\partial \text{loss}}{\partial w_i} \quad \text{and} \quad w_i \leftarrow w_i + \Delta w_i.$$

The learning rate is an example for a hyperparameter of the machine learning model. Hyperparameters are parameters that are not automatically learned from the data but that need to be set during model development.

On the one hand, if the chosen learning rate is too small, the optimization will take a long time.

On the other hand, if the learning rate is too large, the optimization does not converge to the minimum but oscillates around it.

Perceptron

The perceptron is an early form of a linear classifier. It has been proposed by Frank Rosenblatt in 1957, inspired by the mathematical modeling of neurons in the brain.

The perceptron combines a linear function with an activation function that maps the output of the linear function to a binary output:

$$\hat{y} = \phi(\mathbf{w} \cdot \mathbf{x}),$$

where ϕ is the activation function, $\mathbf{w}$ are the weights of the linear function, and $\mathbf{x}$ are the input features.

As activation function, we can use a function similar to the sign function:

$$\phi(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ -1 & \text{if } x < 0 \end{cases}$$

If the two classes are linearly separable, a perceptron with well-chosen weights can find a linear function that separates the classes. The output will be 1 for points above the line and -1 for points below.

Chapter 12

Logistic Regression

Motivation

Previously, we introduced linear classifiers as a simple classification algorithm. In contrast to linear classifiers, logistic regression provides an estimate of the *probability that an input belongs to a certain class*, and not just a binary classification decision.

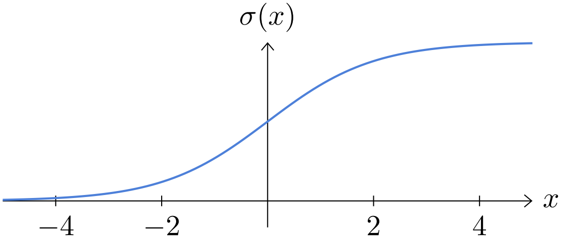
Sigmoid Activation Function

A linear function is not suitable for predicting probabilities because it can produce any real number as an output. Probabilities, however, are always between 0 and 1.

Likewise, the activation function that we used in the perceptron produces a binary output (-1 or 1) and not a probability.

The sigmoid function helps us to transform any real number into a number between 0 and 1:

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



Small negative numbers are mapped to values close to 0, large positive numbers are mapped to values close to 1, and 0 is mapped to 0.5.

Cross-Entropy Loss

In the context of linear regression, we introduced the idea of a *loss function* that measures the difference between the actual labels and the labels predicted by the classifier. We used MSE as the loss function.

In the context of logistic regression, we often use a different loss function called cross-entropy, which quantifies the difference between the predicted probabilities and the actual labels.

Let $\hat{y}$ be the predicted probability that a data point belongs to a certain class. The true label y is 1 if the data point actually belongs to the class, and 0 otherwise.

The cross-entropy loss function is defined as follows:

$$\mathrm{CE}(\hat{y}, y) = -y \log(\hat{y}) - (1 - y) \log(1 - \hat{y})$$

The value of the cross-entropy loss function is close to 0 if the predicted probability is close to the true label, and it increases as the predicted probability diverges from the true label.

Chapter 13

Word Embeddings

Motivation

Generally, a <u>word embedding</u> is a mapping of words to vectors:

$$\text{embedding}(\text{`dog'}) = \begin{pmatrix} 0.2 \\ -0.4 \\ 0.7 \end{pmatrix}$$

$$\text{embedding}(\text{`cat'}) = \begin{pmatrix} -0.1 \\ 0.3 \\ 0.9 \end{pmatrix}$$

$$\dots$$

Word embeddings are a fundamental concept in computational linguistics because they allow us to directly use words as input features for machine learning models.

One-Hot Encoding

The simplest way to represent words as vectors is a <u>one-hot encoding</u>:

$$\text{one-hot}(\text{`hello'}) = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Such a vector has as many dimensions as there are words in the vocabulary. All the elements of the vector are zero, except for the element that corresponds to the index of the word in the vocabulary.

A one-hot encoded vector is a <u>sparse</u> vector because almost all of its elements are zero.

In the following, we will focus on <u>dense</u> word embeddings, where most or all of the elements of the vector are real-valued, non-zero numbers.

Cosine Similarity

One application of word embeddings is to measure the similarity between the vectors of two words. We would expect that a good word embedding assigns similar vectors to similar words.

A standard measure of similarity between two vectors is the cosine similarity:

$$\cos(\mathbf{v},\mathbf{w}) = \frac{\mathbf{v}\cdot\mathbf{w}}{\|\mathbf{v}\|\|\mathbf{w}\|},$$

where $\mathbf{v}$ and $\mathbf{w}$ are the vectors, and $\|\mathbf{v}\|$ and $\|\mathbf{w}\|$ are their norms, i.e., the square root of the sum of the squares of their elements:

$$\|\mathbf{v}\| = \sqrt{v_1^2 + \ldots + v_n^2}$$

Intuitively, the cosine similarity measures the angle between normalized versions of the vectors. Normalization means that the length of the vectors is set to 1.

The cosine similarity is 1 if the vectors point in the same direction, 0 if they are orthogonal, and -1 if they point in opposite directions.

Word2Vec

Word2Vec is an approach introduced by Mikolov et al. in 2013 for learning dense word embeddings from a text corpus. Two alternative ideas for learning word embeddings are introduced in the Word2Vec paper:

- *Continuous Bag of Words (CBOW)*: word embeddings are learned by training a classifier to predict words based on their surrounding context.

- *Skip-gram*: conversely, learn to predict the context words based on a target word.

The process of learning word embeddings using Word2Vec can be summarized as follows:

1. Identify the target word and a neighboring context word as positive examples.
2. Randomly sample other words from the vocabulary to serve as negative samples.
3. Employ logistic regression to train a classifier capable of distinguishing between these positive and negative cases.
4. Utilize the learned weights from the classifier as the embeddings for the words.

Chapter 14

Entropy

Entropy is a measure of the uncertainty associated with a probability distribution.

For example, a fair coin toss has a high entropy because the two outcomes (heads or tails) are equally likely. In contrast, a heavily weighted coin that almost always lands on tails has low entropy, as the outcome is almost certain.

Given a random variable X with possible outcomes $x \in X$ and probabilities $p(x)$, the entropy $H(X)$ is defined as:

$$H(X) = -\sum_{x \in X} p(x) \log_2(p(x))$$

Entropy can be interpreted as the expected number of bits needed to encode the outcome of X.

Chapter 15 *Vectorization*

Most machine learning algorithms require numerical data as input. Vectorization is the process of converting text or categorical labels into (vectors of) numbers.

Vectorizing Documents

- *Count Vectorization*: This method counts the frequency of each word in a document and represents a document as a vector of these counts.

- *tf-idf Vectorization*: This method is based on the intuition that words appearing frequently in some documents but rarely in other documents are more important than those that appear frequently in all documents. It uses the *Term Frequency–Inverse Document Frequency* metric.

tf–idf in Detail

The term frequency $\mathrm{tf}(t, d)$ is the relative frequency of term t in document d:

$$\mathrm{tf}(t,d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}}$$

The inverse document frequency $\mathrm{idf}(t, D)$ is defined as:

$$\mathrm{idf}(t,D) = \log\left(\frac{|D|}{|\{d \in D : t \in d\}|}\right),$$

where D is the set of all documents and $|\{d \in D : t \in d\}|$ is the number of documents that contain term t.

Finally, the tf–idf score for term t in document d is given by:

$$\text{tf-idf}(t,d,D) = \mathrm{tf}(t,d) \cdot \mathrm{idf}(t,D)$$

Encoding Categorical Labels

Two approaches for encoding categorical labels (e.g., the class labels in a classification task) are discussed:

- *Label Encoding*: Assigning a unique integer to each category. While simple, this can be problematic for algorithms with geometric interpretations as it implies an ordinal relationship between categories.
- *One-Hot Encoding*: Representing each category as a one-hot vector. Each instance is then represented by a vector where only the feature corresponding to its category has a value of 1, and all other features are 0. This avoids imposing artificial relationships between categories.

Chapter 16

Support Vector Machines

Support Vector Machines (SVMs) are a classification approach that is similar to linear regression, but with a different training objective:

- In linear regression, we aim to minimize the mean squared error between the predicted labels and the actual labels.

- In SVMs, we aim to find a linear function that separates the classes as clearly as possible. The hyperplane is chosen to maximize the *margin* between the closest points of the classes, which are referred to as *support vectors.*

An advantage of SVMs is that they are robust to outliers because they only consider the support vectors when fitting the model.

SVMs can also be applied to non-linearly separable data. For this, SVMs make use of *kernel functions* to transform the input features into a higher-dimensional space where the classes become linearly separable.

The so-called *kernel trick* allows us to avoid the computational cost of explicitly transforming the features. Instead, we can calculate the dot product of the transformed features directly.

Relevant Sklearn Functionality

(not needed for exam)

```
from sklearn import ...
```

- `naive_bayes.MultinomialNB`
- `stats.tttest_ind`
- `metrics.confusion_matrix`
- `metrics.accuracy_score`
- `metrics.precision_score`
- `metrics.recall_score`
- `metrics.f1_score`
- `metrics.roc_curve`
- `metrics.roc_auc_score`
- `linear_model.LinearRegression`
- `linear_model.Perceptron`
- `linear_model.LogisticRegression`
- `svm.SVC` (Support Vector Classification)

Index