Speech and Language Processing

Tim Lawson

October 22, 2023

These notes are based on Jurafsky and Martin 2023.

Contents

1 Naive Bayes, Text Classification, and Sentiment

1.1 Naive Bayes Classifiers

- Probabilistic, generative, and discriminative classifiers:
  - A probabilistic classifier outputs the probability that an instance belongs to a class.
  - Generative classifiers model the probability that a class generated an instance.
  - Discriminative classifiers learn the features that best discriminate between classes.
- Bayes' theorem, posterior and prior probabilities, and likelihoods:
  - Naive Bayes is a probabilistic classifier.
  - For a document d and classes $c \in C$, it returns the class $\hat{c}$ that has the maximum posterior probability given the document:

$$\hat{c} = \arg\max_{c \in C} P(c \mid d) \tag{1}$$

  - Bayes' theorem states that the posterior probability is:

$$P(c \mid d) = \frac{P(d \mid c)P(c)}{P(d)} \tag{2}$$

  - $P(d)$ is the prior probability of the document, which is the same for all classes. Therefore, we can return the class:

$$\hat{c} = \arg\max_{c \in C} P(d \mid c)P(c) \tag{3}$$

- Naive Bayes classifiers' assumptions:
  - A 'bag of words' is an unordered set of words. The 'bag of words' assumption is that the order of words in a document does not matter.
  - Naive Bayes (conditional independence) assumption.

1.2 Training the Naive Bayes Classifier

- Maximum likelihood estimate (frequency counts)
- Smoothing, unknown words, and stop words
- Example training algorithm

1.4 Optimizing for Sentiment Analysis

- For sentiment classification and other tasks, whether a token occurs in a document is more important than how many times it occurs. This is called binomial or binary naive Bayes.
- A simple way to handle negation is to prepend 'not' to tokens that occur between a negation token and the next punctuation mark. Negation can be handled more accurately with parsing.
- Sentiment lexicons are lists of tokens annotated with positive or negative sentiment. When the training set is sparse or not representative of the test set, lexicon features may generalize better.

1.5 Naive Bayes for other text classification tasks

Sets of tokens and non-linguistic features may be appropriate for other tasks. E.g., character or byte n-grams are effective for language identification. Feature selection is the process of selecting the most informative features.

1.6 Naive Bayes as a Language Model

Naive Bayes classifiers can use features that depend on multiple tokens or are non-linguistic. But naive Bayes classifiers that only use single-token features are similar to language models. Specifically, they are sets of class-specific unigram language models.

1.7 Evaluation: Precision, Recall, F-measure

- 'True' (human-defined) labels are called gold or gold-standard labels.
- A confusion matrix or contingency table represents the precision, recall, and accuracy of a classifier.
- The F-measure is one way to combine precision and recall into a single measure.
  - $F_\beta = \frac{(\beta^2+1)\cdot\text{precision}\cdot\text{recall}}{\beta^2\cdot\text{precision}+\text{recall}}$
  - $\beta < 1$ favours precision and $\beta > 1$ favours recall.
  - $\beta = 1$ equally balances precision and recall.
  - $F_{\beta=1}$ or $F_1 = \frac{2\cdot\text{precision}\cdot\text{recall}}{\text{precision}+\text{recall}}$ is the most common.
  - The F-measure is a weighted harmonic mean of precision and recall.
- Multi-class classification
  - In macroaveraging, we compute the performance for each class, then average over classes. A microaverage is dominated by the most frequent class.
  - In microaveraging, we compute the performance for all classes. A microaverage better reflects the class distribution, so it is more appropriate when performance on all classes is equally important.

1.8 Test Sets and Cross-validation

- Thus far, we have described a procedure wherein we split the data into training, test, and development sets.
- We can instead use all of the data for training and testing by using k-fold cross-validation:
  - Partition the data into k disjoint subsets (folds).
  - For each fold, train the classifier on the remaining $k-1$ folds and test it on the fold.
  - Average the test-set error rate over all folds.
- But then we can't examine the data to look for features, because we would be looking at the test set.
- Therefore, it is common to split the data into training and test sets, then use k-fold cross-validation on the training set.

1.9 Statistical Significance Testing

- How do we compare the performance of two systems A and B? The effect size $\delta(x)$ is the performance difference of A relative to B on a test set x. We want to know whether $\delta(x)$ is statistically significant.
- The null hypothesis H_0 is that $\delta(x) \leq 0$. We want to reject the null hypothesis to support the hypothesis that A is better than B.
- Let X be a random variable over all test sets. The p-value is the probability $P(\delta(X) \geq \delta(x) \mid H_0)$. A result is statistically significant if the p-value is below a threshold.
- Parametric tests like t-tests or ANOVAs make assumptions about the distributions of the test statistic. In NLP, we typically use non-parametric tests based on sampling:
  - Approximate randomization
  - The bootstrap test
- Paired tests compare two sets of aligned observations, e.g., the performance of two systems on a test set.

1.9.1 The Paired Bootstrap Test

- Bootstrapping is repeatedly sampling with replacement from a set of observations. Intuitively, we can create many virtual test sets from an observed test set by sampling from it.
- The test set is biased by $\delta(x)$ towards A (the expected value of $\delta(X)$ for the bootstrapped test sets is $\delta(x)$).

$$\text{p-value}(x) = \frac{1}{b}\sum_{i=1}^{b} \mathbb{I}(\delta(x_i) \geq 2\delta(x)) \tag{4}$$

1.10 Avoiding Harms in Classification

- Representational harms affect demographic groups, e.g., by perpetuating negative sterotypes.
- Another kind of harm is censorship, which may disproportionately affect minority groups, e.g., in toxicity detection.
- A model card documents its data sources, intended use, performance across demographic groups, etc.

2 Logistic Regression

Generative and discriminative classifiers

- A *generative* model uses a likelihood term $P(d \mid c)$. It expresses how to generate the data d from the class c.
- A *discriminative* model uses a posterior term $P(c \mid d)$. It expresses how to discriminate between classes c given the data d.

Components of a probabilistic machine-learning classifier

- A *feature representation* of the input $\vec{x}$.
- A *classification function* $f(\vec{x}) = \hat{y}$, e.g., sigmoid and softmax.
- An *objective function*, e.g., the cross-entropy loss function.
- An *optimization algorithm*, e.g., stochastic gradient descent.

2.1 The sigmoid function

Logistic regression learns a vector of *weights* and a *bias* term or intercept:

$$z = \left(\sum_{i=1}^{n} w_i x_i\right) + b \equiv \vec{w} \cdot \vec{x} + b \tag{5}$$

The *sigmoid* or logistic function $\sigma : \mathbb{R} \to [0, 1]$ is:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{6}$$

It is differentiable and squashes outliers towards 0 or 1.

For binary classification, we require that $P(y = 0) + P(y = 1) = 1$:

$$P(y = 1 \mid \vec{x}) = \sigma(\vec{w} \cdot \vec{x} + b) \tag{7}$$

$$P(y = 0 \mid \vec{x}) = 1 - \sigma(\vec{w} \cdot \vec{x} + b) = \sigma(-\vec{w} \cdot \vec{x} - b) \tag{8}$$

2.2 Classification with logistic regression

For binary classification, we call 0.5 the *decision boundary* of $P(y \mid \vec{x})$:

$$\text{decision}(\vec{x}) = \begin{cases} 1 & \text{if } P(y = 1 \mid \vec{x}) > 0.5 \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

2.2.2 Other classification tasks and features

For some tasks, it helps to design combinations of features or *feature interactions*. These can be created automatically from *feature templates*. *Representation learning* tries to learn features automatically.

Scaling input features It is common to *standardize* features to have zero mean and unit variance. This transformation is called the *z-score*.

$$x'_{ij} = \frac{x_{ij} - \mu_i}{\sigma_i} \text{ where } \mu_i = \frac{1}{N}\sum_{j=1}^{N} x_{ij} \text{ and } \sigma_i = \sqrt{\frac{1}{N}\sum_{j=1}^{N}(x_{ij} - \mu_i)^2} \tag{10}$$

Alternatively, we can *normalize* features by $f : \mathbb{R} \to [0, 1]$:

$$x'_{ij} = \frac{x_{ij} - \min(x_i)}{\max(x_i) - \min(x_i)} \tag{11}$$

2.2.3 Processing many examples at once

Let $\{\vec{x}_j \mid j \in 1..N\}$ be a set of input feature vectors and $X = (x_{ij})$ a matrix where row i is the feature vector $\vec{x}_i$. Then the output vector is:

$$\vec{y} = X\vec{w} + b\vec{1} \tag{12}$$

2.2.4 Choosing a classifier

Naive Bayes assumes conditional independence, i.e., treats correlated features as independent. Logistic regression properly handles correlated features, so it generally works better on large datasets or documents. However, naive Bayes is easy to implement and fast to train, because it has no optimization step.

2.3 Multinomial logistic regression

Multinomial ($k > 2$) logistic regression is sometimes called softmax regression. It is an example of *hard* classification, i.e., it assigns a single class to each instance. A *one-hot vector* $\vec{y}$ has $y_i = 1$ and $y_j = 0 \; \forall \; i \neq j$.

2.3.1 Softmax

The *softmax* function is a generalisation of the sigmoid function to k classes:

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}} : i \in 1..k \tag{13}$$

2.3.2 Applying softmax in logistic regression

The probability that an input feature vector $\vec{x}$ belongs to class y_i is:

$$P(y_i = 1 \mid \vec{x}) = \text{softmax}(\vec{w}_i \cdot \vec{x} + b_i) = \frac{e^{\vec{w}_i \cdot \vec{x} + b_i}}{\sum_{j=1}^{k} e^{\vec{w}_j \cdot \vec{x} + b_j}} : i \in 1..k \tag{14}$$

The vector of output probabilities for the k classes is:

$$\vec{\hat{y}} = \text{softmax}(W\vec{x} + \vec{b}) \tag{15}$$

2.3.3 Features in multinomial logistic regression

In multinomial logistic regression, there is a weight vector and bias for each of the k classes. The weight vector $\vec{w}$ depends on the input feature vector and the output class, so it is sometimes written $f(x, y)$.

2.4 Learning in logistic regression

The learning task to learn the weight vector $\vec{w}$ and bias b that make the vector of output probabilities $\vec{\hat{y}}$ most similar to the true output $\vec{y}$ for the training data. Typically, the *distance* between them is measured, which is called the *loss* or cost function. The *cross-entropy* loss is commonly used for logistic regression and neural networks. The loss function is minimised by an optimisation algorithm. *Gradient descent*, e.g., stochastic gradient descent, is commonly used.

2.5 The cross-entropy loss function

A loss function that prefers that the true outputs are more likely is an example of *conditional maximum likelihood estimation*. It maximises the log probability of the true outputs given the input feature vectors.

The cross-entropy loss is:

$$L_{CE}(\vec{\hat{y}}, \vec{y}) = -\sum_{i=1}^{k} y_i \log \hat{y}_i \tag{16}$$

Derivation for binary regression For $k = 2$ classes, the probability that the output label is correct $P(y \mid x)$ is a Bernoulli distribution (it has two discrete outcomes):

$$P(y \mid x) = \hat{y}^y (1 - \hat{y})^{1-y} \tag{17}$$

The logarithm is a monotonic function. Therefore, instead of maximising the probability, we can maximise the log probability (or minimise the negative log probability):

$$-\log P(y \mid x) = -y \log \hat{y} - (1 - y) \log(1 - \hat{y}) \tag{18}$$

This is equivalent to 16 for $k = 2$.

2.6 Gradient descent

An optimisation algorithm finds the weights θ that minimise the loss function. In logistic regression, $\theta = (\vec{w}, b)$. For n instances, the weights that minimise the average loss are:

$$\vec{\hat{\theta}} = \arg\min_{\vec{\theta}} \frac{1}{n} \sum_{i=1}^{n} L(f(\vec{x}^{(i)}, \vec{\theta}), \vec{y}^{(i)}) \tag{19}$$

Gradient descent finds the minimum of a function by incrementing its parameters in the opposite direction in parameter space to the direction with the largest gradient. The increment is weighted by a *learning rate* η. In logistic regression, the loss function is convex (has at most one minimum) and gradient descent is guaranteed to find the global minimum.

The equation to update the weights is:

$$\vec{\theta'} = \vec{\theta} - \eta \nabla_{\vec{\theta}} L(f(x, \vec{\theta}), y) \tag{20}$$

2.6.1 The gradient for logistic regression

For binary logistic regression, the cross-entropy loss function is:

$$L_{\text{CE}}(\hat{y}, y) = -y \log \hat{y} - (1-y) \log(1-\hat{y}) \quad \text{where} \quad \hat{y} = \sigma(\vec{w} \cdot \vec{x} + b) \tag{21}$$

Its derivative with respect to w_i is:

$$\frac{\partial L_{\text{CE}}}{\partial w_j} = -y \frac{\partial \log \hat{y}}{\partial w_j} - (1-y) \frac{\partial \log(1-\hat{y})}{\partial w_j} \tag{22}$$

$$= -\frac{y}{\hat{y}} \frac{\partial \hat{y}}{\partial w_j} - \frac{1-y}{1-\hat{y}} \frac{\partial (1-\hat{y})}{\partial w_j} \tag{23}$$

$$= -\left(\frac{y}{\hat{y}} - \frac{1-y}{1-\hat{y}} \right) \frac{\partial \hat{y}}{\partial w_j} \tag{24}$$

The derivative of the sigmoid function is:

$$\frac{\partial \sigma(z)}{\partial z} = \sigma(z)(1-\sigma(z)) \tag{25}$$

Therefore:

$$\frac{\partial L_{\text{CE}}}{\partial w_j} = -(y-\hat{y}) \frac{\partial (\vec{w} \cdot \vec{x} + b)}{\partial w_j} \tag{26}$$

$$= (\hat{y}-y) x_j \tag{27}$$

I.e., the gradient with respect to a weight w_j is the difference between the true y and the estimated $\hat{y}$ multiplied by the input feature x_j.

2.6.2 The stochastic gradient descent algorithm

The learning rate η is a hyperparameter. It is common to start with a higher learning rate and slowly decrease it.

2.6.4 Mini-batch training

3 Vector Semantics and Embeddings

- Distributional hypothesis (quotes)
- Word embeddings, static/contextualized
- Representation learning

3.1 Lexical Semantics

- Lemmas and senses, word sense disambiguation
- Synonymy, propositional meaning, principle of contrast
- Similarity and relatnedness (association)
- Semantic fields, topic models, LDA ...
- Semantic frames
- Connotations (affective meanings) and sentiment

3.2 Vector Semantics

- Combining vector representations and distributional semantics
- Embeddings (dense, e.g., word2vec) and vectors (sparse, e.g., tf-idf)

3.3 Words and Vectors

- Cooccurrence matrix
- E.g., term-document matrix
- Dimensionality
- Information retrieval
- Row and column vectors
- E.g., term-term matrix

3.4 Cosine for measuring similarity

- Dot/inner product and length
- Normalized dot product (cosine of angle)
- Cosine similarity

TF-IDF, (P)PMI, applications.

3.5 Word2vec

- Dense, short vectors
- Skip-gram with negative sampling (SGNS)
- Self-supervision

3.5.1 The classifier

- Logistic/sigmoid functon
- Independence assumption

3.5.2 Learning skip-gram embeddings

- Random initial embedding vectors and iteration
- Negative sampling and noise words
- Maximize similarity of positive target-context pairs and minimize similarity of negative pairs
- Stochastic gradient-descent
- Proof of derivatives and update equations
- Target and context embeddings
- Context window size

3.5.3 Other kinds of static embeddings

- fasttext and subwords
- GloVe (global vectors)

3.6 Visualizing Embeddings

- Most similar words
- Clustering
- Dimensionality reduction, e.g., t-SNE

3.7 Semantic properties of embeddings

- First- and second-order cooccurrence (syntagmatic and paradigmatic association)
- Analogy and relational similarity, parallelogram model
- Historical semantics

3.8 Bias and embeddings

- Allocational and representational harm
- Tendency to amplify bias
- Debiasing

4 Neural Networks

4.1 Units

Definition 4.1 (Computational unit)**.** *A computational unit is a function that takes an input vector $\vec{x}$, a weight vector $\vec{x}$, a scalar bias b, and a non-linear function f and produces an activation y:*

$$y = f(\vec{x} \cdot \vec{x} + b) \tag{28}$$

Non-linear functions Examples of non-linear functions are:

- the sigmoid $\sigma(z) = \frac{1}{1+e^{-z}}$;
- the hyperbolic tangent $\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$; and
- the rectified linear unit (ReLU) $\max(0, z)$.

tanh is differentiable and maps outliers towards the mean, whereas ReLU is non-differentiable. For large z values, σ and tanh produce saturated y values, i.e., values that are nearly one have gradients close to zero, whereas ReLU has gradient one for positive z values. Gradients that are close to zero cannot be used for training (the *vanishing gradient* problem).

4.2 The XOR problem

A single computational unit (e.g., a *perceptron*) cannot compute simple functions of its inputs (e.g., XOR). This is because a perceptron is a linear classifier and XOR is not a linearly separable function. However, a layered network of units can compute functions like XOR.

4.3 Feedforward neural networks

A *feedforward* network is a layered network of units that are connected with no cycles. Networks with cycles are called *recurrent* networks. Feedforward networks are sometimes called *multi-layer perceptrons* but this is a misnomer unless the units are perceptrons.

A simple feedforward network has input units, hidden units, and output units. In a *fully-connected* layer of units, each unit takes as its input the outputs of all units in the previous layer.

Hidden layers A hidden layer is a set of computational units (definition 4.1) that takes an input vector $\vec{x}$, a weight matrix W, a bias vector $\vec{b}$, and a non-linear function f and produces an activation vector $\vec{h}$:

$$\vec{h} = f(W\vec{x} + \vec{b}) \tag{29}$$

The function f is applied element-wise to the vector $W\vec{x} + \vec{b}$.

Output layers In a feedforward network with a single hidden layer, the output layer is a function that takes an activation vector $\vec{h}$ and a weight matrix U and produces an intermediate output vector $\vec{z}$:

$$\vec{z} = U\vec{h} \tag{30}$$

Normalization functions A *normalization* function converts an activation vector into a vector that represents a probability distribution, e.g., softmax:

$$\vec{y} = \text{softmax}(\vec{z}) = \frac{e^{\vec{z}}}{\sum_{i=1}^{n} e^{z_i}} \tag{31}$$

Dimensionality The elements of feedforward neural network with a single fully-connected hidden layer have the following dimensions:

- $\vec{x} \in \mathbb{R}^{n_0}$ is a column vector $n_0 \times 1$;
- $\vec{h}, \vec{b}, \vec{z} \in \mathbb{R}^{n_1}$ are column vectors $n_1 \times 1$;
- $W \in \mathbb{R}^{n_1 \times n_0}$ is a matrix $n_1 \times n_0$;
- $U \in \mathbb{R}^{n_2 \times n_1}$ is a matrix $n_2 \times n_1$; and
- $\vec{y} \in \mathbb{R}^{n_2}$ is a column vector $n_2 \times 1$.

Non-linear activation functions If the activation function f is linear, then a multi-layer feedforward network is equivalent to a single-layer network (with a different weight matrix). Therefore, we use non-linear activation functions.

Replacing the bias vector Generally, we replace the bias vector with an additional unit in each layer whose activation is always one. The weights associated with this unit are the bias value b:

$$\vec{x} \in \mathbb{R}^{n_0} \rightarrow \vec{x} \in \mathbb{R}^{n_0+1},\ x_0 = 1$$
$$W \in \mathbb{R}^{n_1 \times n_0} \rightarrow W \in \mathbb{R}^{n_1 \times (n_0+1)},\ W_{i0} = b_i\ \forall\ i \in 0..n_0$$

4.4 Classification

A classifier could use hand-crafted features but most applications learn features from the data by representing words by embeddings. An input is generally represented by applying a *pooling* function to the embeddings of its words, e.g., the mean or element-wise maximum.

For a two-layer classifier, the equations to predict the output of a set of test instances are:

- n_i is the number of test instances;
- n_j is the dimensionality of the instance space;
- n_k is the number of nodes in the hidden layer;
- n_l is the number of classes;
- $X \in \mathbb{R}^{n_i \times n_j}$ is the set of test instances;
- $W \in \mathbb{R}^{n_k \times n_j}$ is the weight matrix;
- $B \in \mathbb{R}^{n_i \times n_k}$ is the bias matrix;

- $H = \sigma(XW^T + B) \in \mathbb{R}^{n_i \times n_k}$ is the hidden-layer activation matrix;
- $U \in \mathbb{R}^{n_l \times n_k}$ is the hidden-layer weight matrix;
- $Z = HU^T \in \mathbb{R}^{n_i \times n_l}$ is the intermediate output matrix; and
- $\hat{Y} = \text{softmax}(Z) \in \mathbb{R}^{n_i \times n_l}$ is the output matrix.

The use of an input representation is called *pretraining*. It is possible to train the classifier and the input representation jointly as part of an NLP task.

4.5 Language modelling

The task of *language modelling* is to predict upcoming words from prior context. Neural language modelling is an important task in itself and a precursor to many others. Generally, modern language models use architectures like recurrent neural networks or transformers.

4.5.1 Inference

Forward inference is the task of producing a probability distribution over the possible outputs, given an input. For language models, the inputs and outputs are words. A *one-hot vector* has a single element equal to 1 and the rest 0.

The equations to predict the output are:

- n_i is the context window size;
- n_j is the number of words in the vocabulary;
- n_k is the dimensionality of the embeddings;
- n_l is the number of nodes in the hidden layer;
- $\{x_{t-i} \mid i = 1..n_i\} \in \mathbb{R}^{n_j}$ are one-hot word vectors;
- $E \in \mathbb{R}^{n_k \times n_j}$ is the embedding matrix;
- $\vec{e}_i = E\vec{x}_i \in \mathbb{R}^{n_k}$ is the embedding of word x_i;
- $\vec{e} = [\vec{e}_1; \ldots; \vec{e}_{n_i}] \in \mathbb{R}^{n_i n_k}$ are the concatenated embeddings;
- $W \in \mathbb{R}^{n_l \times n_i n_k}$ is the weight matrix;
- $\vec{b} \in \mathbb{R}^{n_l}$ is the bias vector;
- $\vec{h} = \sigma(W\vec{e} + \vec{b}) \in \mathbb{R}^{n_l}$ is the hidden-layer activation vector;
- $U \in \mathbb{R}^{n_j \times n_l}$ is the hidden-layer weight matrix;
- $\vec{z} = U\vec{h} \in \mathbb{R}^{n_j}$ is the intermediate output vector; and
- $\vec{\hat{y}} = \text{softmax}(\vec{z}) \in \mathbb{R}^{n_j}$ is the output vector.

4.6 Training neural networks

A feedforward neural network is an example of a supervised machine learning model. A neural network is trained with:

- a loss function, e.g., the cross-entropy loss (section 4.6.1); and
- an optimisation algorithm, e.g., gradient descent.

For logistic regression, the derivative of the loss function can be computed directly (section 4.6.2). For neural networks, *backpropagation* (section 4.6.3) is necessary.

4.6.1 Cross-entropy loss

Definition 4.2 (Cross-entropy loss)**.** *Let $\vec{y}, \hat{\vec{y}} \in \mathbb{R}^k$ be the one-hot true output vector and predicted output vector, respectively. The cross-entropy loss of an instance with true class j is:*

$$L(\vec{y}, \hat{\vec{y}}) = -\sum_{i=1}^{k} y_i \log \hat{y}_i \tag{32}$$

$$= -\log \hat{y}_j \tag{33}$$

4.6.2 Gradient

For a neural network with a single hidden layer and sigmoid activation function, i.e., logistic regression, the derivative of the cross-entropy loss with respect to the weight w_i is:

$$\begin{aligned}\frac{\partial L(\vec{y}, \hat{\vec{y}})}{\partial w_i} &= (\hat{y} - y)\vec{x}_i \\ &= (\sigma(\vec{w} \cdot \vec{x} + b) - y)\vec{x}_i\end{aligned}$$

With a softmax activation function, i.e., multinomial logistic regression, the derivative of the cross-entropy loss with respect to the weight w_{ij} is:

$$\begin{aligned}\frac{\partial L(\vec{y}, \hat{\vec{y}})}{\partial w_{ij}} &= -(\vec{y}_i - \hat{\vec{y}}_i)\vec{x}_j \\ &= -\left(\vec{y}_i - \frac{\exp(\vec{w}_i \cdot \vec{x} + b_i)}{\sum_{k=1}^{l} \exp(\vec{w}_k \cdot \vec{x} + b_k)}\right)\vec{x}_j\end{aligned}$$

4.6.3 Backward differentiation

For a neural network with multiple hidden layers, we must compute the derivative with respect to the weights in each layer. This is achieved by backpropagation or backprop, which is a form of *backward differentiation* on the computational graph of the network.

Example TODO

4.6.4 Learning

For logistic regression, we can initialize the weights and biases to zero. For neural networks, we must initialize the weights and biases to small random numbers and normalize the inputs (zero mean and unit variance).

Regularization, e.g., *dropout*, is used to prevent overfitting. It is also important to tune *hyperparameters*, e.g., the learning rate, the mini-batch size, the model architecture, and the regularization.

References

Jurafsky, Dan and James H. Martin (2023). *Speech and Language Processing*.