

# **Natural Language Processing with Pytorch**

## Build Intelligent Language Applications Using Deep Learning

Stephen CUI

April 20, 2023

# Chapter 1

## Introduction

### 1.1 The Supervised Learning Paradigm

We can break down the supervised learning paradigm, as illustrated in [Figure 1.1](#), to six main concepts:

**Observations** Observations are items about which we want to predict something. We denote observations using  $x$ . We sometimes refer to the observations as inputs.

**Targets** Targets are labels corresponding to an observation. These are usually the things being predicted. Following standard notations in machine learning/deep learning, we use  $y$  to refer to these. Sometimes, these labels known as the *ground truth*.

**Model** A model is a mathematical expression or a function that takes an observation,  $x$ , and predicts the value of its target label.

**Parameters** Sometimes also called weights, these parameterize the model. It is standard to use the notation  $w$  (for weights) or  $\hat{w}$ .

**Predictions** Predictions, also called estimates, are the values of the targets guessed by the model, given the observations. We denote these using a “hat” notation. So, the prediction of a target  $y$  is denoted as  $\hat{y}$ .

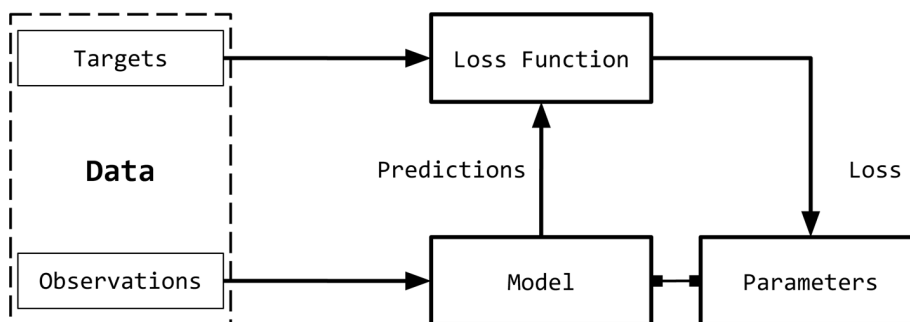


Figure 1.1: The supervised learning paradigm, a conceptual framework for learning from labeled input data.

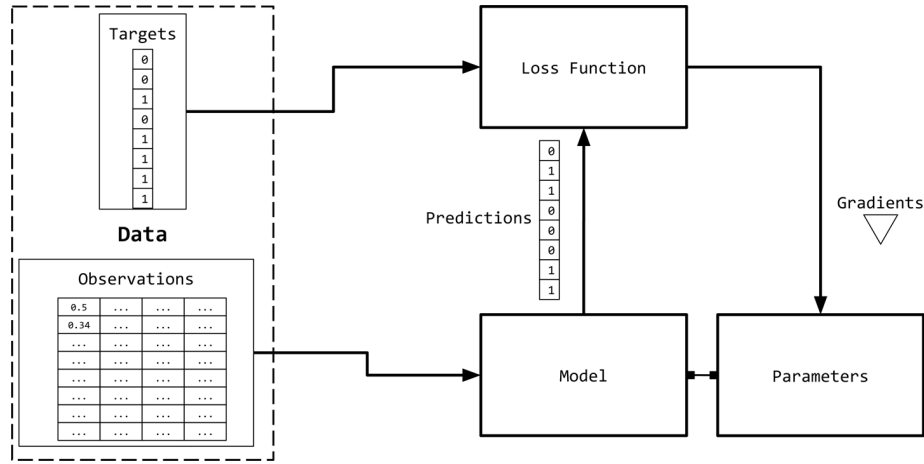


Figure 1.2: Observation and target encoding: The targets and observations from Figure 1.1 are represented numerically as vectors, or tensors. This is collectively known as input “encoding.”

**Loss function** A loss function is a function that compares how far off a prediction is from its target for observations in the training data. Given a target and its prediction, the loss function assigns a scalar real value called the loss. The lower the value of the loss, the better the model is at predicting the target. We use  $L$  to denote the loss function.

Consider a dataset  $D = \{X_i, y_i\}_{i=1}^n$  with  $n$  examples. Given this dataset, we want to learn a function (a model)  $f$  parameterized by weights  $w$ . That is, we make an assumption about the structure of  $f$ , and given that structure, the learned values of the weights  $w$  will fully characterize the model. For a given input  $X$ , the model predicts  $\hat{y}$  as the target:

$$\hat{y} = f(X, w)$$

In supervised learning, for training examples, we know the true target  $y$  for an observation. The loss for this instance will then be  $L(y, \hat{y})$ . Supervised learning then becomes a process of finding the optimal parameters/weights  $w$  that will minimize the cumulative loss for all the  $n$  examples.

## 1.2 Observation and Target Encoding

We will need to represent the observations (text) numerically to use them in conjunction with machine learning algorithms. Figure 1.2 presents a visual depiction.

### 1.2.1 One-Hot Representation

The one-hot representation, as the name suggests, starts with a zero vector, and sets as 1 the corresponding entry in the vector if the word is present in the sentence or document. we use  $1_w$  to mean one-hot representation for a token(词元)/word  $w$ .

Using the encoding shown in Figure 1-3, the one-hot representation for the phrase “like a banana” will be a  $3 \times 8$  matrix, where the columns are the eight-dimensional one-hot vectors. It is

	time	fruit	flies	like	a	an	arrow	banana
1 <sub>time</sub>	1	0	0	0	0	0	0	0
1 <sub>fruit</sub>	0	1	0	0	0	0	0	0
1 <sub>flies</sub>	0	0	1	0	0	0	0	0
1 <sub>like</sub>	0	0	0	1	0	0	0	0
1 <sub>a</sub>	0	0	0	0	1	0	0	0
1 <sub>an</sub>	0	0	0	0	0	1	0	0
1 <sub>arrow</sub>	0	0	0	0	0	0	1	0
1 <sub>banana</sub>	0	0	0	0	0	0	0	1

Figure 1.3: One-hot representation for encoding the sentences “Time flies like an arrow” and “Fruit flies like a banana.”

also common to see a “collapsed” or a binary encoding where the text/phrase is represented by a vector the length of the vocabulary, with 0s and 1s to indicate absence or presence of a word. The binary encoding for “like a banana” would then be: [0, 0, 0, 1, 1, 0, 0, 1].

### 1.2.2 Term-Frequency(TF) Representation

The TF representation of a phrase, sentence, or document is simply the sum of the one-hot representations of its constituent words. We denote the TF of a word  $w$  by  $TF(w)$ .

### 1.2.3 TF-IDF Representation

The IDF representation penalizes common tokens and rewards rare tokens in the vector representation. The  $IDF(w)$  of a token  $w$  is defined with respect to a corpus as:

$$IDF(w) = \log \frac{N}{n_w} \quad (1.1)$$

where  $n_w$  is the number of documents containing the word  $w$  and  $N$  is the total number of documents. The TF-IDF score is simply the product  $TF(w) * IDF(w)$ .

### 1.2.4 Target Encoding

Many NLP tasks actually use categorical labels, wherein the model must predict one of a fixed set of labels.

Some NLP problems involve predicting a numerical value from a given text. For example, given an English essay, we might need to assign a numeric grade or a readability score.

## 1.3 Computational Graphs

Technically, a computational graph is an abstraction that models mathematical expressions.

Inference (or prediction) is simply expression evaluation (a forward flow on a computational graph).

## 1.4 PyTorch Basics

### Dynamic Versus Static Computational Graphs

Static frameworks like Theano, Caffe, and TensorFlow require the computational graph to be first declared, compiled, and then executed.<sup>6</sup> Although this leads to extremely efficient implementations (useful in production and mobile settings), it can become quite cumbersome during research and development. Modern frameworks like Chainer, DyNet, and PyTorch implement dynamic computational graphs to allow for a more flexible, imperative style of development, without needing to compile the models before every execution. Dynamic computational graphs are especially useful in modeling NLP tasks for which each input could potentially result in a different graph structure.

At the core of the library is the tensor, which is a mathematical object holding some multidimensional data.

### 1.4.1 Creating Tensors

Any PyTorch method with an underscore (`_`) refers to an in-place operation; that is, it modifies the content in place without creating a new object.

### 1.4.2 Tensor Types and Size

Each tensor has an associated type and size. The default tensor type when you use the `torch.Tensor` constructor is `torch.FloatTensor`. There are two ways to specify the initialization type: either by directly calling the constructor of a specific tensor type, such as `FloatTensor` or `LongTensor`, or using a special method, `torch.tensor()`, and providing the dtype.

### 1.4.3 Tensors and Computational Graphs

“Bookkeeping operations”是深度学习中的一个术语，它通常用于指代在神经网络训练过程中的一系列轻量级操作，用于跟踪和记录训练期间的各种统计数据。

这些操作包括记录训练损失、计算梯度、更新模型参数等。这些操作通常被认为是“轻量级”的，因为它们不涉及任何繁重的计算，而是在训练过程中进行简单的统计计算和数据更新。

在深度学习中，由于神经网络的复杂性，训练过程通常需要进行大量的迭代和调整，而这些“轻量级”的操作可以帮助我们跟踪和调整训练过程中的各种参数和指标，从而更好地优化模型的

性能。

When you create a tensor with `requires_grad=True`, you are requiring PyTorch to manage bookkeeping information that computes gradients. First, PyTorch will keep track of the values of the forward pass. Then, at the end of the computations, a single scalar is used to compute a backward pass. The backward pass is initiated by using the `backward()` method on a tensor resulting from the evaluation of a loss function. The backward pass computes a gradient value for a tensor object that participated in the forward pass.