Guidebook to Exercise 5

Prof. Dr. Ivan Habernal
Yassine Thlija

2025-11-11

1 The Softmax Function

1.1 Concept and Motivation

In the lecture, the Softmax function was introduced as a mechanism to convert an unbounded vector of real-valued scores (often called *logits*) into a valid probability distribution over discrete classes.

Formally, for a vector $\mathbf{x} \in \mathbb{R}^K$, Softmax is defined as:

$$\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_{k=1}^{K} e^{x_k}}, \quad i = 1, \ldots, K \tag{1}$$

For its derivative, see *derivatives_guidebook*.

The function maps arbitrary real numbers into the interval $(0, 1)$ and ensures:

$$\sum_{i=1}^{K} \text{softmax}(x_i) = 1$$

Thus, Softmax converts the raw outputs of a linear model $f(\mathbf{x}) = \mathbf{x}\mathbf{W} + \mathbf{b}$ into interpretable probabilities over K classes.

1.2 Why Softmax is Important

In classification tasks, especially in multi-class language tasks (e.g., predicting the language of a document as in the lecture example), Softmax serves two critical purposes:

a) It provides a probabilistic interpretation of scores.

b) It allows differentiable training through gradient-based optimization.

The probabilistic nature enables the use of the **cross-entropy loss**, defined as:

$$L_{\text{cross-entropy}} = -\sum_{k=1}^{K} y[k] \log(\hat{y}[k]), \tag{2}$$

where $\mathbf{y}$ is the true (one-hot) label vector and $\hat{\mathbf{y}}$ is the output of Softmax.

1.3 Illustrative Example

Consider this example: classifying a document into one of six languages (English, French, German, Italian, Spanish, Other). Let the linear layer produce raw outputs:

$$\mathbf{xW} + \mathbf{b} = [3.2, 1.1, -0.5, 2.0, 0.4, -1.2]$$

After applying Softmax:

$$\hat{\mathbf{y}} = [0.65, 0.09, 0.02, 0.18, 0.05, 0.01]$$

The model assigns the highest probability to English.

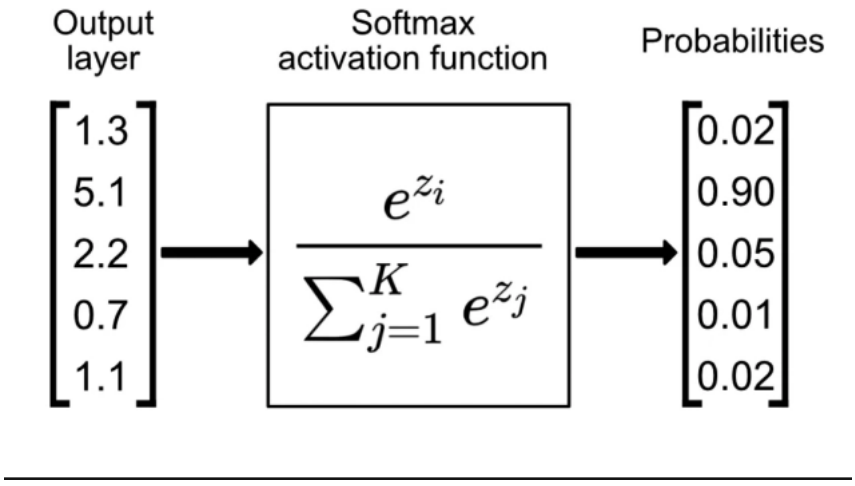

Figure 1: (Theoretical Figure unrelated to our example) Visualization of the Softmax mapping from raw logits to normalized probabilities.
Source: Singlestore Guide

1.4 Exercises

1. Compute the Softmax probabilities for $\mathbf{x} = [2, 1, 0]$.
2. Explain why Softmax is sensitive to large values of its input (hint: consider the exponential term).
3. Using the temperature parameter T, derive how $\text{softmax}(x_i; T) = \frac{e^{x_i/T}}{\sum_j e^{x_j/T}}$ affects the sharpness of the output distribution. (hint: check Lecture 5!)

2 Online Gradient Descent (OGD)

2.1 Definition and Context

Online Gradient Descent (OGD) is a variant of the gradient descent algorithm where parameters are updated after processing each training example, rather than after the entire dataset (as in batch gradient descent).

Given a parameter vector $\boldsymbol{\theta}$ and loss function $L(\boldsymbol{\theta}; \mathbf{x}_t)$ at time step t, the update rule is:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_t; \mathbf{x}_t) \tag{3}$$

where η is the learning rate.

2.2 Why OGD is Important

Online updates make it possible to:

- Train models efficiently on very large corpora.
- Adapt to non-stationary data distributions (as often encountered in language modeling).
- Enable real-time learning where data arrives sequentially.

In the context of neural language models, OGD (or its stochastic equivalent, SGD) allows the network to adjust parameters as it observes each n-gram or context window.

2.3 Practical Considerations

OGD introduces noise in updates due to single-sample gradients, but this noise often helps escape local minima and improves generalization. Modern optimizers like Adam or RMSProp are built on the same fundamental principle of incremental parameter updates.

2.4 Exercises

1. Derive the update rule for a parameter w in a one-dimensional quadratic loss function $L = \frac{1}{2}(wx - y)^2$ under OGD.
2. Implement a simple OGD update loop for a toy linear regression problem. (This will help you tackle Task 2 (Exercise 5))

3 ReLU Revisited

3.1 Definition

The **Rectified Linear Unit (ReLU)** is a non-linear activation function defined as:

$$\text{ReLU}(z) = \max(0, z) \tag{4}$$

For derivative details, see *derivatives_guidebook*.

3.2 Why ReLU Matters

ReLU plays a critical role in enabling neural networks to model non-linear relationships. Unlike sigmoid or tanh activations, ReLU does not saturate for large positive inputs, which helps maintain gradient flow during backpropagation.

Key advantages:

- Prevents vanishing gradients (as in deep sigmoid networks).
- Computationally efficient (simple thresholding).
- Encourages sparse activations, improving generalization.

3.3 Example

In Lecture 5, ReLU was introduced in the context of the XOR problem, a simple demonstration of how adding non-linearity allows a network to separate data that a purely linear model cannot.

3.4 Exercises

1. Plot $\text{ReLU}(z)$ and its derivative for $z \in [-5, 5]$.
2. Try to understand why ReLU helps networks learn faster compared to sigmoid activations.

4 Optional: Visualizing Data and Learned Representations

Understanding the structure of your data and the representations learned by neural models is an essential step in model development. Visualization helps diagnose model behavior, detect outliers, and build intuition about how features or embeddings relate to one another.

This section introduces 2 common methods for visualizing data in NLP:

1. Scatter plots for low-dimensional projections.
2. Dimensionality reduction using PCA.

Each method is illustrated with short Python code examples and the output graph.

4.1 1. Scatter Plots for Two-Dimensional Data

Scatter plots are the simplest way to visualize features or model outputs when the data is already two-dimensional or can be projected into two dimensions.

Example: Visualizing two features from document vectors

```
import matplotlib.pyplot as plt
import numpy as np

# Example: document representations for two features (e.g., language indicators)
x = np.random.rand(100)
y = 0.5 * x + np.random.normal(0, 0.05, 100)
labels = np.random.choice(['English', 'German'], size=100)

for l in np.unique(labels):
    plt.scatter(x[labels == l], y[labels == l], label=l, alpha=0.7)

plt.title("Scatter plot of two document features")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.legend()
plt.show()
```

Scatter plots are particularly useful when examining correlations between two learned features, such as activations of two hidden neurons in a feed-forward layer.

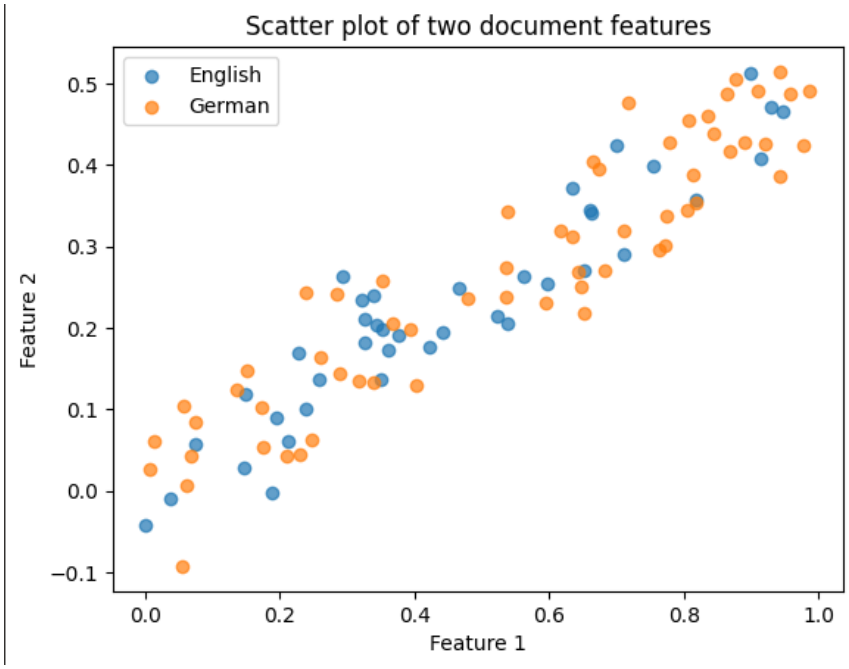

Figure 2: Example scatter plot showing two features of document representations (Using the code above)

4.2 2. Visualizing High-Dimensional Data with PCA

When working with embeddings or hidden representations, we often deal with hundreds of dimensions. **Principal Component Analysis (PCA)** is a linear technique that projects high-dimensional data onto a lower-dimensional subspace while retaining maximal variance.

This can be used to visualize, for example, the word embeddings learned in a neural language model.

Example: Visualizing word embeddings with PCA

```
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
import numpy as np

# Suppose E is the embedding matrix from your model
E = np.random.randn(200, 50)   # 200 words, 50 dimensions
words = [f"word_{i}" for i in range(200)]

pca = PCA(n_components=2)
proj = pca.fit_transform(E)

plt.scatter(proj[:,0], proj[:,1], alpha=0.6)
for i, word in enumerate(words[:10]):
    plt.text(proj[i,0], proj[i,1], word)
plt.title("PCA projection of word embeddings")
plt.show()
```

PCA reveals global structure in embeddings, how words, documents, or class representations cluster in space, helping verify whether similar words (e.g., “dog”, “cat”) lie close together.

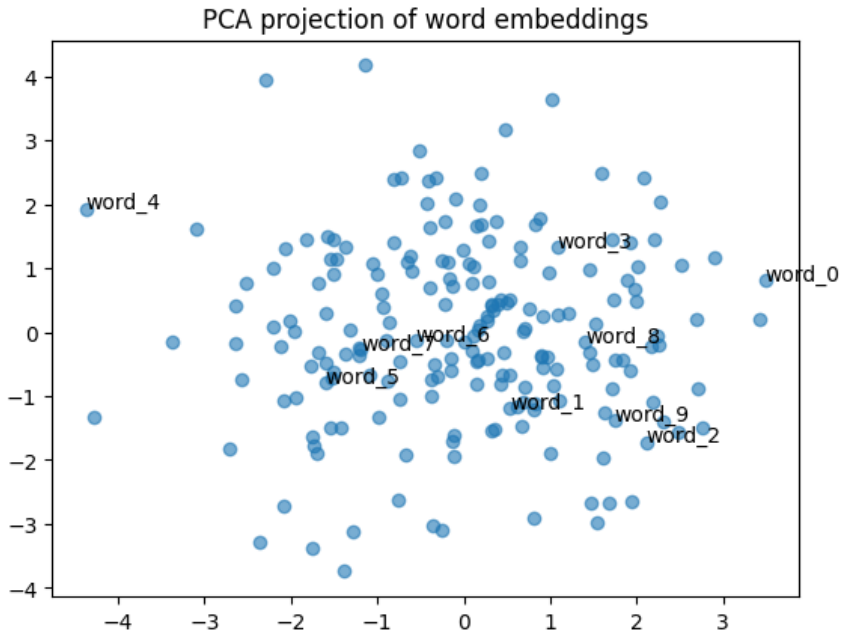

Figure 3: Example PCA projection of word embeddings learned by a neural language model (Using the code above)