Exercise Sheet 8: Self Supervised Learning

Due on 11.07.2025, 10:00

Nick Stracke (nick.stracke@lmu.de)

Task 1: Temperature in Self Supervised Learning (3P)

Goal: In this exercise, you will develop an intuitive understanding of how the *softmax* function works, and how the *temperature* parameter influences its behavior. The softmax function is widely used in machine learning to convert raw scores (e.g., similarities or logits) into probabilities. In self-supervised learning, temperature plays a key role in shaping how confident the model is in its predictions.

1.1 Visualizing the Effect of Temperature

Let us start with a simple vector of similarity scores:

$$\texttt{similarities} = [2.0,\ 1.0,\ 0.2,\ -0.5]$$

Implement the temperature-scaled softmax function:

$$\texttt{softmax}_\tau(x_i) = \frac{\exp(x_i/\tau)}{\sum_j \exp(x_j/\tau)}$$

- Plot the output of the softmax function for different temperature values: $\tau \in \{0.1,\ 0.5,\ 1.0,\ 2.0, 5.0\}$.
- Use a bar plot to visualize the resulting probability distribution for each τ.

1.2 Intuition: Confidence vs. Uncertainty

Think of the softmax output as a way to distribute *attention* or *confidence* over a set of candidates (e.g., nearest neighbors in feature space). Answer the following:

- What happens to the distribution when the temperature is very low (e.g., $\tau = 0.1$)? What does this mean in terms of model confidence?
- What happens when τ is very high (e.g., $\tau = 10.0$)? What does this mean for the model's ability to focus?

1.3 Why Temperature Matters in Contrastive Learning

In contrastive learning, the model must identify the correct (positive) example among many negative ones using similarity scores. Consider the following example:

$$\texttt{similarities} = [1.0\ (\text{positive}),\ 0.2,\ 0.1,\ -0.1,\ -0.3]$$

- Compute the softmax probabilities for $\tau = 10.0$ and $\tau = 0.1$.
- Which temperature gives the positive example the highest probability?
- Discuss: Why might a sharper distribution (lower τ) help the model learn more effectively?

Task 2: SimCLR (14P)

Goal: In this exercise, you will implement a simplified version of SimCLR[1] for contrastive self-supervised learning. You will train an encoder to distinguish between differently augmented views of the same image and then evaluate the learned representations using a linear classifier and a t-SNE visualization.

2.1 Implement and Train a SimCLR Model

- Take a look at `ex2.py`, which already includes a large part of the model and the training loop. This is your starting point.
- Finish the ToDo's in `SimCLRDataset`. Check the paper for what augmentations you need, but leave out Gaussian blur.
- Implement the contrastive loss. For a positive pair (i, j), the loss is defined as
$$\ell_{i,j} = -\log \frac{\exp(\text{sim}(\mathbf{z}_i, \mathbf{z}_j)/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{[k \neq i]} \exp(\text{sim}(\mathbf{z}_i, \mathbf{z}_k)/\tau)}$$
with $\mathbb{1}$ being the indicator function, $\mathbf{z}$ the embedding and τ the temperature. You can also have a look at Algorithm 1 in the paper for additional help.
- Train the model by calling the file directly. This is a standalone Python file because using `num_workers`> 0 in a Jupyter notebook can cause issues. If you prefer working in a notebook, you might have to set `num_workers`$= 0$ if you encounter errors, but this often degrades performance. On an M1 Pro, training one epoch took roughly one minute when using mps.

[1] https://arxiv.org/abs/2002.05709

2.2 Visualize with t-SNE

- We will use t-SNE to reduce the high-dimensional encoder features of the test set to 2D and visualize it.
- You have to finish the ToDo for extracting features.
- Don't worry if the result is not perfect. This is expected after only training for 10 epochs.

2.3 Quantitative Evaluation using Linear Probing

- To get a better feeling of the usefulness of the learned representation, we will perform classification on the frozen encoder features using a linear probe.
- Similar to the previous exercise, first extract the features for the entire train set.
- Then, define the linear probe, the optimizer, and the classification loss. Train the linear probe while keeping everything else frozen.
- If you trained the SimCLR model for 10 epochs and the linear probe for 50 epochs, you should achieve ca. 60% classification accuracy. For better performance, you have to train the base model for more epochs.

Task 3: CLIP for Zero-Shot Classification (3P)

Goal: In this exercise, you will explore the CLIP model, which uses contrastive learning to align images and natural language. You will implement a simple zero-shot classifier by comparing the cosine similarity between image and text embeddings.

3.1 Understanding how CLIP Works

CLIP (Contrastive Language–Image Pretraining) is trained similarly to SimCLR: it maximizes the similarity between paired inputs. However, instead of comparing two augmented views of an image, CLIP learns to align **images and text descriptions** in a joint embedding space.

- Image and text encoders are trained to produce embeddings such that matching pairs (image, caption) have high cosine similarity.
- This enables CLIP to perform classification by comparing an image embedding to the embeddings of class name prompts, without needing any training on the target task.

3.2 Implement a Zero-Shot Classifier with CLIP

- Load the pretrained CLIP model from Hugging Face.
- Define a small set of unlabeled test images (e.g. 5–10).
- For each image, write down a set of possible text descriptions (e.g., “a photo of a dog”, “an image of a cat”, ...).
- Compute cosine similarities between the image embedding and each candidate text embedding.
- Assign the label of the text with the highest similarity.

3.3 Discussion

- Try varying your text prompts. Does the classification change?
- What kind of errors does CLIP make? Are they semantically reasonable?
- How does this compare to traditional classification models?

Important Note: Submit exactly one ZIP file via Moodle before the deadline. Please submit your solution as a Jupyter Notebook. The ZIP file should contain your notebook and the images/videos you are loading. Make sure that it runs on different operating systems and uses relative paths. Non-trivial sections of your code should be explained with short comments, and variables should have self-explanatory names. The notebook file should contain your written code, all figures, explanations, and answers to questions.