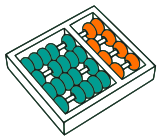

UNICAMP

Self-supervised learning: non-generative representation learning

Lesson No. 07

Gustavo de J. Merli - 262948

1 Denoising Autoencoders [1]

1.1 Algorithm

A denoising autoencoder (DAE) is trained to reconstruct a clean "repaired" input from a corrupted version of it (the specific types of corruptions we consider will be discussed below). This is done by first corrupting the initial input x into $\tilde{x}$ by means of a stochastic mapping $\tilde{x} \sim q_D(\tilde{x}|x)$.

Corrupted input $\tilde{x}$ is then mapped, as with the basic autoencoder, to a hidden representation $y = f_\theta(\tilde{x}) = s(W\tilde{x} + b)$ from which we reconstruct a $z = g_{\theta'}(y)$. Parameters θ and θ' are trained to minimize the average reconstruction error over a training set, that is, to have z as close as possible to the uncorrupted input x. The key difference is that z is now a deterministic function of $\tilde{x}$ rather than x. As previously, the considered reconstruction error is either the cross-entropy loss $L_{IH}(x,z) = IH(B(x)||B(z))$, with an affine+sigmoid decoder, or the squared error loss $L_2(x,z) = ||x-z||^2$, with an affine decoder. Parameters are initialized at random and then optimized by stochastic gradient descent. Note that each time a training example x is presented, a different corrupted version $\tilde{x}$ of it is generated according to $q_D(\tilde{x}|x)$.

1.2 Types of Corruption Considered

There are 3 types of corruptions processes that were considered. They are

- Additive isotropic Gaussian noise (GS): $\tilde{x}|x \sim \mathcal{N}(x, \sigma^2 I)$;
- Masking noise (MN): a fraction v of the elements of x (chosen at random for each example) is forced to 0;
- Salt-and-pepper noise (SP): a fraction v of the elements of x (chosen at random for each example) is set to their minimum or maximum possible value (typically 0 or 1) according to a fair coin flip.

Additive Gaussian noise is a very common noise model, and is a natural choice for real valued inputs. The salt-and-pepper noise is a natural choice for input domains which are interpretable as binary or near binary such as black and white images or the representations produced at the hidden layer after a sigmoid squashing function.

Masking noise can be viewed as turning off components considered missing or replacing their value by a default value—that is, a common technique for handling missing values. All information about these masked components is thus removed from that particular input pattern, and we can view the denoising autoencoder as trained to fill-in these artificially introduced "blanks". Also, numerically, forcing components to zero means that they are totally ignored in the computations of downstream neurons.

1.3 Stacking Denoising Autoencoders to Build Deep Architectures

Input corruption is only used for the initial denoising-training of each individual layer, so it may learn useful feature extractors. Once the mapping f_θ has thus been learnt, it will henceforth be used on uncorrupted inputs. In particular no corruption is applied to produce the representation that will serve as clean input for training the next layer. Once a stack of encoders has thus been built, its highest level output representation can be used as input to a stand-alone supervised learning algorithm, for example a Support Vector Machine classifier or a (multi-class) logistic regression. Alternatively, a logistic regression layer can be added on top of the encoders, yielding a deep neural network amenable to supervised learning. The parameters of all layers can then be simultaneously fine-tuned using a gradient-based procedure such as stochastic gradient descent.

2 Context Encoders [2]

The objective is predicting the relative position of patches within an image. It is employed Convolutional Neural Networks (ConvNets), which are well known to learn complex image representations with minimal human feature design. Building a ConvNet that can predict a relative offset for a pair of patches is, in principle, straightforward: the network must feed the two input patches through several convolution layers, and produce an output that assigns a probability to each of the eight spatial configurations (like a jigsaw puzzle) that might have been sampled (i.e. a softmax output). Note, however, that it is ultimately wished to learn a feature embedding for individual patches, such that patches which are visually similar (across different images) would be close in the embedding space.

To achieve this, late-fusion architecture is used: a pair of architectures that process each patch separately, until a depth analogous, after which point the representations are fused. For the layers that process only one of the patches, weights are tied between both sides of the network, such that the same embedding function is computed for both patches.

To obtain training examples given an image, the first patch is uniformly sampled, without any reference to image content. Given the position of the first patch, the second patch randomly sampled from the eight possible neighboring locations.

3 Tracking Emerges from Colorization [3]

It is first described how to train the model for video colorization, then how to use it for tracking is discussed.

3.1 Model

Let $c_i \in \mathbb{R}^d$ be the true color for pixel i in the reference frame, and let $c_j \in \mathbb{R}^d$ be the true color for a pixel j in the target frame. We denote $y_j \in \mathbb{R}^d$ as the model's prediction for c_j. The model predicts y_j as a linear combination of colors in the reference frame:

$$y_j = \sum_i A_{ij} c_i \tag{1}$$

where A is a similarity matrix between the target and reference frame such that the rows sum to one. Several similarity metrics are possible. We use inner product similarity normalized by softmax:

$$A_{ij} = \frac{exp(f_i^T f_j)}{\sum_k exp(f_k^T f_j)} \tag{2}$$

where $f_i \in \mathbb{R}^D$ is a low-dimensional embedding for pixel i that is estimated by a convolutional neural network. Since we are computing distances between all pairs, the similarity matrix is potentially large. However, because color is fairly low spatial frequency, we can operate with lower resolution video frames allowing us to calculate and store all pairs on commodity hardware.

3.2 Learning

It is used a large dataset of unlabeled videos for learning. We train the parameters of the model θ such that the predicted colors y_j are close to the target colors c_j across the training set:

$$\min_\theta \sum_j \mathscr{L}(y_j, c_j) \tag{3}$$

where $\mathscr{L}$ is the loss function. Since video colorization is a multi-modal problem, it is used the cross-entropy categorical loss after quantizing the color-space into discrete categories.

Quantized by clustering the color channels across the dataset using k-means (using 16 clusters). Optimized Equation 3 using stochastic gradient descent.

3.3 Inference

After learning, the model can compute a similarity matrix A for a pair of target and reference frames. Given an initially labeled frame from a heldout video, this pointer is used to propagate labels throughout the video. To do this, the property that the model is non-parametric in the label space is exploited. Simply re-using Equation 1 to propagate, but instead of propagating colors, distributions of categories are propagate. Since the rows of A sum to one, Equation 1 can be interpreted as a mixture model where A is the mixing coefficients. The model will be described for two different types of tasks: segment tracking and key-point tracking.

Segment Tracking: To track segments, $c_i \in \mathbb{R}^d$ is re-iterpreted as a vector indicating probabilities for d categories. The initial frame labels c_i will be one-hot vectors (since the ground truth for the first frame is known), but the predictions c_j in subsequent frames will be soft, indicating the confidence of the model.

Keypoints Tracking: Unlike colors and segmentation, keypoints are often sparse, but the model can still track them. The keypoints are converted into a dense representation where $c_i \in \mathbb{R}^d$ is a binary vector indicating whether a keypoint is located at pixel i, if any. In this case, d corresponds to the number of keypoints in the initial frame.

4 Contrastive Predictive Coding (CPC) [4]

The architecture of Contrastive Predictive Coding models is described as follows. First, a non-linear encoder g_{enc} maps the input sequence of observations x_t to a sequence of latent representations $z_t = g_{enc}(x_t)$, potentially with a lower temporal resolution. Next, an autoregressive model g_{ar} summarizes all $z \leq t$ in the latent space and produces a context latent representation $c_t = g_{ar}(z \leq t)$.

The model doesn't predict future observations x_{t+k} directly with a generative model $p_k(x_{t+k}|c_t)$. Instead we model a density ratio which preserves the mutual information between x_{t+k} and c_t as follows:

$$f_k(x_{t+k}, c_t) \propto \frac{p(x_{t+k}|c_t)}{p(x_{t+k})} \tag{4}$$

Note that the density ratio f can be unnormalized (does not have to integrate to 1). Although any positive real score can be used here, a simple log-bilinear model is used:

$$f_k(x_{t+k}, c_t) = exp(z_{t+k}^T, W_k c_t), \tag{5}$$

By using a density ratio $f(x_{t+k}, c_t)$ and inferring z_{t+k} with an encoder, the model is relieved from modeling the high dimensional distribution x_{t_k}. Although $p(x)$ or $p(x|c)$ cannot be evaluated directly, samples from these distributions can be used, allowing to use techniques such as Noise-Contrastive Estimation and Importance Sampling that are based on comparing the target value with randomly sampled negative values.

5 BERT [5]

There are two steps in the framework: pre-training and fine-tuning. During pre-training, the model is trained on unlabeled data over different pre-training tasks. For finetuning, the BERT model is first initialized with the pre-trained parameters, and all of the parameters are fine-tuned using labeled data from the downstream tasks. Each downstream task has separate fine-tuned models, even though they are initialized with the same pre-trained parameters.

5.1 Pre-training BERT

BERT is pre-trained using two unsupervised tasks, described in this section.

Task 1: Masked LM In order to train a deep bidirectional representation, some percentage of the input tokens are masked at random, and then those masked tokens are predicted. In all of the experiments, 15% of all Word-Piece tokens are masked at random. In contrast to denoising auto-encoders, only the masked words are predicted rather than reconstructing the entire input.

Task 2: Next Sentence Prediction (NSP) In order to train a model that understands sentence relationships, the model is pre-trained for a binarized next sentence prediction task that can be trivially generated from any monolingual corpus. Specifically, when choosing the sentences A and B for each pretraining example, 50% of the

time B is the actual next sentence that follows A (labeled as *IsNext*), and 50% of the time it is a random sentence from the corpus (labeled as *NotNext*).

5.2 Fine-tuning BERT

For each task, the taskspecific inputs and outputs are plugged into BERT and finetune all the parameters end-to-end. At the input, sentence A and sentence B from pre-training are analogous to (1) sentence pairs in paraphrasing, (2) hypothesis-premise pairs in entailment, (3) question-passage pairs in question answering, and (4) a degenerate text pair in text classification or sequence tagging. At the output, the token representations are fed into an output layer for tokenlevel tasks, such as sequence tagging or question answering, and the [*CLS*] representation is fed into an output layer for classification, such as entailment or sentiment analysis.

References

[1] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *J. Mach. Learn. Res.*, vol. 11, pp. 3371–3408, Dec. 2010, ISSN: 1532-4435.

[2] C. Doersch, A. Gupta, and A. A. Efros, *Unsupervised visual representation learning by context prediction*, 2015. arXiv: 1505.05192 [cs.CV].

[3] C. Vondrick, A. Shrivastava, A. Fathi, S. Guadarrama, and K. Murphy, *Tracking emerges by colorizing videos*, 2018. arXiv: 1806.09594 [cs.CV].

[4] A. van den Oord, Y. Li, and O. Vinyals, *Representation learning with contrastive predictive coding*, 2018. arXiv: 1807.03748 [cs.LG].

[5] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, *Bert: Pre-training of deep bidirectional transformers for language understanding*, 2018. arXiv: 1810.04805 [cs.CL].