Categorical Variables in the Synthesized API

Amandla Mabona

August 16, 2019

Contents

1 Purpose

Columns in a table hold data of various types, for example, dates, names, currency values. These different types are represented using different data types in a programming language and are modelled with different probability distributions. The Synthesized API must therefore handle them differently. In particular, firstly, it must allow conversion between the different data representation and an appropriate represent for training models implemented in Tensorflow. Secondly, it must provide appropriate probability distributions for each data type and loss functions corresponding to these distributions. These features are handled by `Value` classes in the API.

The `CategoricalValue` class is the `Value` class that provides representation conversion, probability distributions and losses for categorical data types which do not require special treatment. For our purposes, a "categorical data type" is one which can only take one of a finite discrete set of values. Examples of categorical data types are ratings and discretized, bounded numerical data.

The next sections describe how data representation conversion and distributions are specified in the `CategoricalValue` class.

2 Background: VAEs

The `HighDimSynthesizer` class currently fits a variational autoencoder (VAE) to a dataset and uses it to generate synthetic data. The VAE defines both a model for generating data and a loss for training this model. A `Value` class's role in training is calculating the "reconstruction" term of the VAE training objective.

Briefly, a VAE is latent variable generative model that defines a probability distribution $p_X(x)$ over the observed variables X via a latent variable Z as $p_X(x) = \sum_z p_{X,Z}(x, z)$. VAEs factorise $p_{X,Z}(x, z)$ into a prior distribution $p_Z(z)$ and a likelihood distribution $p_{X|Z}(x|z)$ as $p_{X,Z}(x, z) = p_{X|Z}(x|z) \cdot p_Z(z)$. They are trained to maximise the evidence lower bound (ELBO) $\mathcal{L}(x; p, q) = \mathbb{E}_{Z\sim q(\cdot|x)}[\log p_{X|Z}(x|Z)] + KL(q(\cdot|x)|p_Z(\cdot))$ where $q(\cdot|x)$ is an "inference" model whose parameters are trained along with the generative model to maximise the ELBO. The first term, $\mathbb{E}_{Z\sim q(\cdot|x)}[\log p_{X|Z}(x|Z)]$, is called the "reconstruction" term while the second term, $KL(q(\cdot|x)|p_Z(\cdot))$ is called the "KL divergence" term.

By analogy with standard autoencoders, the inference model $q(z|x)$ is called an "encoder" and the likelihood model $p_{X|Z}(x|z)$ is called a "decoder". Typically, and in the `HighDimSynthesizer` class, Z is a real vector-valued random variable taking values in $\mathbb{R}^d$ where d is the latent dimension, $p_Z(z)$ is a standard Gaussian $\mathcal{N}(\mathbf{0}, \mathbf{I}_d)$ and $q(z|x)$ is a Gaussian whose mean and variances are functions of x parametrised by neural networks $\mathcal{N}(\boldsymbol{\mu}_\phi(x), \boldsymbol{\sigma}_\phi(x))$.

3 Parametrization

The only probability distribution on a finite discrete set is the categorical distribution. For a set $\{v_1, \cdots, v_n\}$ with n elements, this distribution is defined by n probabilities $p_1, \cdots, p_n$ where p_i is the probability of element v_i so for all i, $0 \le p_i \le 1$ and $\sum_{i=1}^{n} p_i = 1$. For categorical variables, the decoder $p_{X|Z}(x|z)$ must therefore be a categorical distribution whose parameters $\{p_i\}_{1\le i\le n}$ are a function of the latent vector $z \in \mathbb{R}^d$. In the `CategoricalValue` class, there are two settings for this parametrization, controlled by the `similarity_based` flag passed to the class constructor.

If `similarity_based` is set to false, the unnormalized logits for $\{p\}_{1\le i\le n}$ are obtained by applying a linear transformation to z. The probabilities are obtained by then passing these through a softmax layer. The distribution is then parametrized as

$$\begin{aligned} y &= \mathbf{W} \cdot z \\ p_{X|Z}(X = i|z) &= \frac{\exp(y_i)}{\sum_j \exp(y_j)} \end{aligned} \tag{1}$$

If `similarity_based` is set to true, an additional linear transformation with weight matrix $\mathbf{W}^e$ is applied to the unnormalized logits before the softmax layer. The distribution is then parametrized as

$$\begin{aligned} y &= \mathbf{W} \cdot z \\ y' &= \mathbf{W}^e \cdot y \\ p_{X|Z}(X = i|z) &= \frac{\exp(y'_i)}{\sum_j \exp(y'_j)} \end{aligned} \tag{2}$$

4 Training

The "KL divergence" term in the ELBO is shared across columns for a record so it is not calculated by value classes. The reconstruction term is column-specific, however, and is implemented in a Value class's `loss` method[1].

For a categorical random variable, X, the reconstruction term reduces to

$$\mathbb{E}_{Z \sim q(\cdot|x)}[\log p_{X|Z}(x|Z)] = \mathbb{E}_{Z \sim q(\cdot|x)}[-\mathbb{H}(\delta_x, p_{X|Z}(\cdot|Z))] \tag{3}$$

where $\mathbb{H}(\mu, \nu)$ is the cross-entropy between μ and ν, and δ_x is the one-hot encoding of x. We appromixate the expectation over $q(\cdot|x)$ by taking a single sample. Since the sampled value of Z is shared across columns, this sampling is performed outside the Value class, so the `loss` method actually only calculates $\log p_{X|Z}(x|Z)$. For the CategoricalValue class, we calculate this using the `tensorflow` function for calculating the cross-entropy `nn.softmax_cross_entropy_with_logits_v2`.

4.1 Loss Scaling and smoothing

The `moving_average` argument to the constructor controls whether *moving-averaging loss scaling* is applied during training. If the argument's value is a `tensorflow.train.MovingAverage` object, it is using for calculating the moving average. If the the value is "None", then the scaling is not applied.

By *loss-scaling*, we mean instead of $\mathbb{H}(\delta_x, p_{X|Z}(\cdot|Z))$,
$w_x \cdot \mathbb{H}(\delta_x, p_{X|Z}(\cdot|Z))$
is used in the loss, where $w_i = \frac{1}{\sqrt{\hat{p_i}}}$ with $\hat{p_i}$ the average proportion of observations where $X = i$, calculated with a moving average over minibatches.

[1]Actually, the negative of the reconstruction term is calculated here. We actually minimise the negative of the ELBO because `tensorflow` optimizer implementations accept a function to be minimised and this is equivalent to maximising the ELBO.

In addition, the `smoothing` argument to the constructor controls how much the one-hot encoding of the observed category is smoothed towards a uniform distribution, with a value of a value of zero indicating no smoothing. If the value equals w_s, then

$\mathbb{H}((1 - w_s) \cdot \delta_x + w_s \cdot U(n), p_{X|Z}(\cdot|Z))$

is used instead instead of the cross entropy, where $U(n)$ is the uniform distribution on $1, 2, \cdots, n$.

4.2 Regularizers

4.2.1 "Similarity-based" regularizer

If the `similarity_based` is set to true, then a "similarity-based" regularization term is added to the reconstruction loss. This term is equal to

$$\sum_{i,j} \mathbf{W}^e_{:,i} \cdot \mathbf{W}^e_{:,j} = \sum_i \mathbf{W}^e_{:,i} \cdot \mathbf{W}^e_{:,i} + \sum_{i \neq j} \mathbf{W}^e_{:,i} \cdot \mathbf{W}^e_{:,j}$$

$$= \sum_i \|\mathbf{W}^e_{:,i}\|^2 + \sum_{i \neq j} \mathbf{W}^e_{:,i} \cdot \mathbf{W}^e_{:,j}$$

$$= L_{2,1}(\mathbf{W}^e) + O(\mathbf{W}^e)$$

The first term is equivalent to regularizing $\mathbf{W}^e$ by applying weight decay to all of its columns. The second term is smaller when the dot product between $\mathbf{W}^e$'s columns is small and so encourages them to be orthogonal.

4.2.2 Entropy regularizer

The `entropy_regularization` argument to the constructor sets the weight of an entropy regularization term added to the reconstruction loss.

$$\mathbb{H}[p_{X|Z}(\cdot|z)] = \mathbb{E}_{X \sim p_{X|Z}(\cdot|z)}[-\log p_{X|Z}(X|z)]$$

5 Generation

Standardly, we would generate data from a VAE by first sampling a value for the latent variable from the prior $z \sim p_Z(\cdot)$, then sampling observations from the decoder conditional on z, $x \sim p_{X|Z}(\cdot|z)$.

Instead of doing this, in the current implementation, we sample a value for the latent variable from the prior $z \sim p_Z(\cdot)$ and then return the highest probability category from the decoder $x^* = \arg\max_x p_{X|Z}(x|z)$. This is implemented in the `output_tensors` method.