Supplementary material for Geographic ratemaking with spatial embeddings

Christopher Blier-Wong[1,3], Hélène Cossette[1,3,4], Luc Lamontagne[2,3], and Etienne Marceau[1,3,4]

[1] *École d'actuariat, Université Laval, Québec, Canada*
[2] *Département d'informatique et de génie logiciel, Université Laval, Québec, Canada*
[3] *Centre de recherche en données massives, Université Laval, Québec, Canada*
[4] *Centre interdisciplinaire en modelisation mathématique, Université Laval, Québec, Canada*

July 19, 2021

1 Canadian census details

In this section, we present the categories of variables that are available for every FSA in the Canadian census data. The data is aggregated into polygons called forward sortation areas, which correspond to the first three characters of a Canadian postal code; see Figure 1 for the decomposition of a postal code. Statistics Canada aggregates the public release of census data to avoid revealing confidential and individual information. The data is also available at the dissemination area polygon level, which is more granular than FSA. We work with FSAs because they are simpler to explain.

The first issue with using census data for insurance pricing is the use of protected attributes, i.e., variables that should legally or ethically not influence the model prediction. Territories may exhibit a high correlation with protected attributes like ethnic origin. For this reason, we retain only variables that a Canadian insurance company could use for ratemaking. Below, we provide a complete list of the categories of variables within the census dataset, and we denote with an asterisk (*) the categories of variables that we omit. What remains is information about age, education, commute, income and others, and comprises 512 variables that we denote $\boldsymbol{\gamma}$. Removing protected attributes from a model is a technique called anti-classification [Corbett-Davies and Goel, 2018], or fairness through unawareness [Kusner et al., 2018], which does not eliminate discrimination entirely and in some cases may increase it [Kusner et al., 2018]. Studying discrimination-free methods to construct geographic embeddings is kept as future work. For analysis and discussion of discrimination in actuarial ratemaking, see [Lindholm et al., 2020].

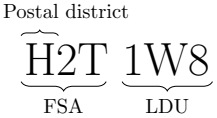


Figure 1: Deconstruction of a Canadian postal code

1. Population and dwellings
2. Age characteristics
3. Household and dwelling characteristics
4. Marital status
5. Family characteristics
6. Household type
7. Knowledge of official languages*
8. First official language spoken*
9. Mother tongue*
10. Language spoken most often at home*
11. Other language spoken regularly at home*
12. Income of individuals in 2015
13. Income of households in 2015
14. Income of economic families in 2015
15. Low income in 2015
16. Knowledge of languages*
17. Citizenship*
18. Immigrant status and period of immigration*
19. Age at immigration*
20. Immigrants by selected place of birth*
21. Recent immigrants by selected places of birth*
22. Generation status*
23. Admission category and applicant type*
24. Aboriginal population*
25. Visible minority population*
26. Ethnic origin population*
27. Household characteristics
28. Highest certificate, diploma or degree
29. Major field of study - Classification of Instructional Programs (CIP) 2016
30. Location of study compared with province or territory of residence with countries outside Canada*
31. Labour force status
32. Work activity during the reference year
33. Class of worker
34. Occupation - National Occupational Classification (NOC) 2016
35. Industry - North American Industry Classification System (NAICS) 2012
36. Place of work status
37. Commuting destination
38. Main mode of commuting
39. Commuting duration
40. Time leaving for work
41. Language used most often at work
42. Other language used regularly at work
43. Mobility status - Place of residence 1 year ago
44. Mobility status - Place of residence 5 years ago

2 Model code

We provide the code for the CBOW-CRAE model in the following listing. In PyTorch, one defines a class, where trainable parameters appear in the `__init__` method, and forward operations appear in the `forward` function.

```
from torch import nn

class CBOWConvolutionalAE(nn.Module):
  def __init__(self, census_values, bottleneck, first_descent, second_descent,
    fc_descent):
    super().__init__()
    self.census_values = nn.Embedding.from_pretrained(census_values)
    self.conv_encoder_size = 16 * second_descent
    self.bottleneck = bottleneck
    self.second_descent = second_descent

    self.encode_conv_1 = nn.Conv2d(512, first_descent, 3, 1, 1)
    self.encode_relu = nn.LeakyReLU()
    self.encode_max_pool_1 = nn.MaxPool2d(2, stride=2, return_indices=True)
    self.conv_bn_1 = nn.BatchNorm2d(first_descent)
    self.encode_conv_2 = nn.Conv2d(first_descent, second_descent, 3, 1, 1)
```

```
    self.conv_bn_2 = nn.BatchNorm2d(second_descent)
    self.encode_max_pool_2 = nn.MaxPool2d(2, stride=2, return_indices=True)
    self.encode_fc_1 = nn.Linear(self.conv_encoder_size, fc_descent)
    self.encode_fc_2 = nn.Linear(fc_descent, bottleneck)
    self.encode_tanh_fc = nn.Tanh()

    self.decode_fc_1 = nn.Linear(bottleneck, fc_descent)
    self.decode_fc_2 = nn.Linear(fc_descent, 512)
    self.decode_sigmoid_fc = nn.Sigmoid()

    nn.init.kaiming_uniform_(self.encode_conv_1.weight, mode = 'fan_in', a = 1)
    nn.init.kaiming_uniform_(self.encode_conv_2.weight, mode = 'fan_in', a = 1)
    nn.init.kaiming_uniform_(self.encode_fc_1.weight, mode = 'fan_in', a = 10)
    nn.init.kaiming_uniform_(self.encode_fc_2.weight, mode = 'fan_in', a = 10)
    nn.init.kaiming_uniform_(self.decode_fc_1.weight, mode = 'fan_in', a = 1)
    nn.init.kaiming_uniform_(self.decode_fc_2.weight, mode = 'fan_in', a = 0.1)

  def forward(self, x):
    x = self.encode(x)
    x = self.decode(x)
    return x

  def encode(self, x):
    x = x.permute(0, 3, 1, 2)
    x = self.encode_conv_1(x)
    x = self.conv_bn_1(x)
    x = self.encode_relu(x)
    x, pool_index_1 = self.encode_max_pool_1(x)

    x = self.encode_conv_2(x)
    x = self.conv_bn_2(x)
    x = self.encode_relu(x)
    x, pool_index_2 = self.encode_max_pool_2(x)

    x = x.reshape(-1, self.conv_encoder_size)

    x = self.encode_fc_1(x)
    x = self.encode_relu(x)
    x = self.encode_fc_2(x)
    x = self.encode_tanh_fc(x)
    return x

  def decode(self, x):
    x = self.decode_fc_1(x)
    x = self.encode_relu(x)
    x = self.decode_fc_2(x)
    x = self.decode_sigmoid_fc(x)
    return x

  def get_embeddings(self, x):
    x = self.encode(x)
    return x
```

One then instantiates the class for the Large CBOW-CRAE model with

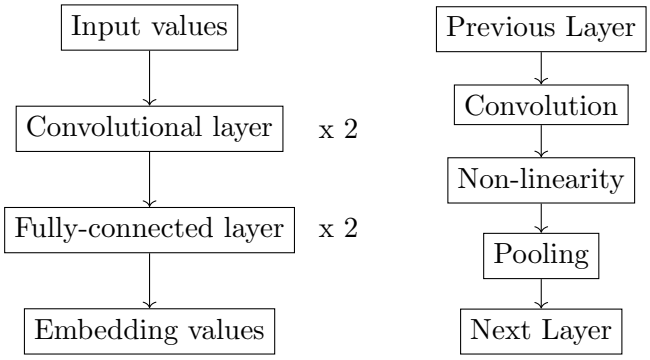
```
net = CBOWConvolutionalAE(census_values, 16, 1024, 2048, 128)
```

where the first argument is a table containing the raw census values. Let us now explain the code.

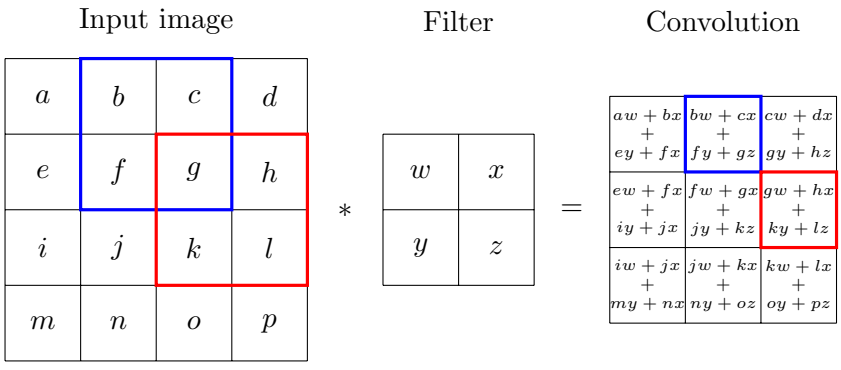
- Line 6, we store the census values for FSAs as pretrained embeddings, which lets us construct the geographic data square cuboid every time and avoid storing all GDSCs.
- Line 7 defines the size of the unroll vector.
- Lines 11 to 24 define the operations required for the encoders and the decoders.
- Lines 26 to 31 apply weight initialization to weights of the convolution and fully-connected layers. The weights are sampled from $Unif(-b, b)$ with

$$b = \sqrt{\frac{2}{1+a^2}}\sqrt{\frac{3}{fan}},$$

  where fan is a constant that preserves the magnitude of the variance of the weights in the forward pass, see [He et al., 2015] for details. Note that the value of a is typically the slope of the Leaky ReLU, but we used it as a hyperparameter (along with trial and error) to avoid saturated embedding values.
- Lines 33 to 36 define the forward function, the complete model during training time. It is simply the encoder followed by the decoder.
- Lines 38 to 56 define the encoder.
  - The architecture is presented below. Left: general blocks, right: a single convolution block.



  - Line 39 changes the dimensions of the data to use in PyTorch's convolution function.
  - Lines 40 to 43 consists of Conv1 and apply the convolution, the batch norm, the leaky ReLU, and the max-pooling.
  - An example of convolution of 2×2 convolution with stride 1:



  - Batch norm applies the following transformation to each feature:

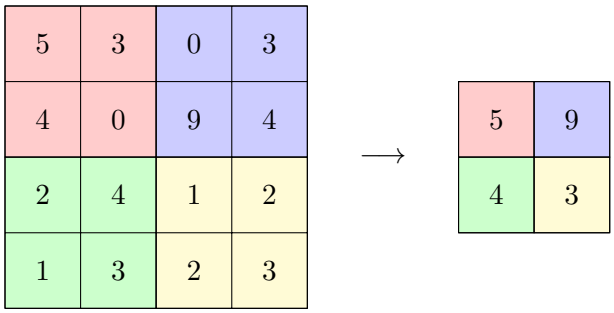
$$a\frac{x - \overline{x}}{sd(x)} + b,$$

where $\overline{x}$ and $sd(x)$ are the empirical mean and standard deviation of the feature, and a, b are trainable weights.

- Leaky ReLU applies the following transformation to each feature:

$$\text{LeakyReLU}(x) = \max(x, 0.01x).$$

- An example of 2×2 max-pooling with stride 2:



- Lines 45 to 48 consists of Conv2 and apply the same steps.
- Line 50 unrolls the vector.
- Lines 52 and 53 apply the first fully-connected layer (FC1), followed by leaky ReLU activation.
- Lines 54 and 55 apply the second fully-connected layer (FC2), followed by tanh.
- The tanh activation is

$$\tanh(x) = \frac{\sinh(x)}{\cosh(x)} = \frac{e^{2x} - 1}{e^{2x} + 1}.$$

- Lines 58 to 63 define the decoder.
  - Lines 59 and 60 apply the third fully-connected layer (FC3), followed by leaky ReLU activation.
  - Lines 61 and 62 apply the fourth fully-connected layer (FC4). Notice that the output dimension of FC4 is 512, the dimension of the census data for one location. We follow FC4 by the sigmoid activation function. The sigmoid activation is

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

  and has an image of $(0, 1)$, the same as the input variables (because of min-max normalization of input variables).
- The function `get_embeddings` is a short function that only applies the encoder, which lets us extract the embedding vector.

3 Comparison of architectures

In the main paper, we limit the number of embedding models to the one actually used within our empirical study, in order to present the model in detail. Here, we present CRAE, the original idea for spatial embeddings from [Blier-Wong et al., 2020]. Then, we compare small and large versions of CRAE and CBOW-CRAE for encoding Canadian census data.

3.1 CRAE decoder

The original model for geographic embeddings we explored was the convolutional regional autoencoder (CRAE), presented in [Blier-Wong et al., 2020]. The input of CRAE is the geographic data cuboid, and the output is also the geographic data cuboid. The neural network architecture is called a convolutional autoencoder since the model's objective is to reconstruct the input data after going through a bottleneck of layers. The decoder in CRAE is a function $\mathbb{R}^{\ell} \to \mathbb{R}^{p \times p \times d}$. One can interpret CRAE as *using contextual variables to predict the same contextual variables*. The loss function for CRAE is the average reconstruction error. For a dataset of N coordinates, the loss function is

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} \left|\left| g_{CRAE}\left(f\left(\underline{\gamma}_{\delta_i}\right)\right) - \underline{\gamma}_{\delta_i} \right|\right|^2, \tag{1}$$

where f is the encoder, g_{CRAE} is the CRAE decoder, $\underline{\gamma}_{\delta_i} \in \mathbb{R}^{p \times p \times d}$ is the geographic data cuboid for the coordinate of location i and $||\cdot||$ is the euclidean norm.

CRAE and CBOW-CRAE share the same encoder, the difference lies in the decoders. CRAE starts from the embedding vector, includes fully-conencted layers to gradually increase the dimension to the Unroll dimension, then rolls the large vector into a matrix. Then, deconvolution blocks (the opposite operation as convolutions) return the feature map to the original GDSC size. A great disadvantage of CRAE is that attribute *coordinate* only requires information from the grid's central location, not the entire grid. Therefore, much of the information contained within CRAE embeddings captures irrelevant or redundant information. Also, the CRAE decoder contains many more parameters (over 26 000 000) than the CBOW-CRAE decoder (under 100 000).

3.2 Encoders

This subsection will detail specific architecture choices for encoders with Canadian census data, presenting two strategies to determine the optimal architecture. Each contains two convolution layers with batch normalization [Ioffe and Szegedy, 2015] and two fully-connected layers. Strategy 1 reduces the feature size between the last convolution layer and the first fully-connected layer, while strategy 2 reduces the feature size between convolution layers. Each encoder uses a hyperbolic tangent (tanh) activation function after the last fully-connected layer to constrain the embedding values between -1 and 1. After testing convolutional kernels of size $k = \{3, 5, 7\}$, the value $k = 3$ resulted in the lowest reconstruction errors.

3.2.1 Strategy 1

A popular strategy for CNN architectures is to reduce the width and height but increase the depth of intermediate features as we go deeper into the network, see [Simonyan and Zisserman, 2014, He et al., 2016]. The first strategy follows three heuristics:

1. Apply half padding, such that the output dimension of intermediate convolution features remains the same.
2. Apply max-pooling after each convolution step with a stride and kernel size of 2, reducing the feature size by a factor of 4.
3. Double the square cuboid depth after each convolution step.

The result of this strategy is that the size (the number of features in the intermediate representations) is reduced by two after every convolution operation. We present the square cuboid depth and dimension at all stages of the models in Table 1. The feature size (row 3) is the product of square cuboid depth (number of channels) and the dimension of the intermediate features. In strategy 1,

	Input	Conv1	Conv2	Unroll	FC1	FC2
Square cuboid depth	512	1 024	2 048	32 768	128	16
Square cuboid width × height	16×16	8×8	4×4	1	1	1
Feature size	131 072	65 536	32 768	32 768	128	16
% of parameters	NA	17	68	NA	15	0

Table 1: Large encoder model with 27 798 672 parameters

the convolution step accounts for most parameters. The steepest decrease in feature size occurs between the second convolution block and the first fully-connected layer (from 32 768 to 128).

3.2.2 Strategy 2

For the second strategy, we follow a trial and error approach and attempt to restrict the number of parameters in the model. We retain heuristics 1 and 2 from strategy 1, but the depth of features decrease between each convolution block.

	Input	Conv1	Conv2	Unroll	FC1	FC2
Square cuboid depth	512	48	16	256	16	8
Square cuboid width × height	16×16	8×8	4×4	1	1	1
Feature size	131 072	3 072	256	256	16	8
% of parameters	NA	95	3	NA	2	0

Table 2: Small encoder model with 232 514 parameters

In strategy 2, 95% of the parameters are in the first convolution step. The feature size decreases steadily between each operation.

3.3 CRAE & CBOW-CRAE decoders

The output for the CRAE model is the reconstructed geographic data cuboid. The decoder in this model is the inverse operations of the encoder (deconvolutions and max-unpooling). The final activation function is sigmoid because the original inputs are between 0 and 1. We present the decoder operations for the large and the small decoders in Tables 3 and 4. Recall that the input to the decoder is the embedding layer from the encoder.

The CBOW-CRAE is a context to location model, so we select a fully-connected decoder, increasing from the embedding ($\boldsymbol{\gamma}^*$) size ℓ to the geographic variable ($\boldsymbol{\gamma}$) size d. In our experience, the decoder's exact dimensions did not significantly impact the reconstruction error, so we select the ascent dimensions (FC3 and FC4) to be the same as the fully-connected descent dimensions (FC1 and FC2). When there is no hidden layer from the embedding to the output (if there is only one fully-connected layer), the model is too linear to reconstruct the input data. When there is one hidden layer, the model is mainly able to reconstruct the data. Additional hidden layers did

	Input	FC3	FC4	Roll	Deconv1	Deconv2
Square cuboid depth	16	128	32 768	2 048	1 024	512
Square cuboid width × height	1	1	1	4×4	8×8	16×16
Feature size	16	128	32 768	32 768	65 536	131 072
% of parameters	NA	0	15	NA	68	17

Table 3: Large CRAE decoder model

	Input	FC3	FC4	Roll	Deconv1	Deconv2
Square cuboid depth	8	16	256	16	48	512
Square cuboid width × height	1	1	1	4×4	8×8	16×16
Feature size	8	16	256	256	3 072	131 072
% of parameters	NA	91	3	NA	2	0

Table 4: Small CRAE decoder model

not significantly reduce the reconstruction error, so we select only one hidden layer in the decoder. Table 5 presents the CBOW CRAE decoders in our implementation.

	Input	FC3	FC4
Small model	8	16	512
Large model	16	128	512

Table 5: CBOW-CRAE decoders

3.4 Training results

In this section, we provide results on the implementations of the four geographic embedding architectures, along with observations. Table 6 presents the training and validation reconstruction errors, along with the training time, the number of parameters and the mean embedding values.

One cannot directly compare the reconstruction errors for the classic CRAE and CBOW-CRAE since classic CRAE reconstructs p^2 as many values as CBOW-CRAE. The average reconstruction error for CRAE is smaller than for CBOW-CRAE, which could be because the output of CBOW-CRAE does not have a determined equivalent vector in the input data. The CRAE model attempts to construct a one-to-one identity function for every neighbor because the input is identical to the output. On the other hand, CBOW-CRAE cannot exactly predict the values for a specific neighbor in the grid since there is no guarantee that the specific neighbor belongs to the same polygon as the central coordinate. One also notices that the validation data's reconstruction error is smaller than the training data, which is atypical in machine learning. However, changing the initial seed for training and validation data changes this relationship, so one attributes this effect to the specific data split. This also means that the model does not overfit on the training data: if it did, then the training error would be much smaller than the validation error. The lack of overfit is a result of the bottleneck dimension being small (8 or 16 dimensions) with respect to the dimension of the input data (131 072).

We do not find that one set of embeddings always performs better than the others, but find that the Large CBOW-CRAE behaves more appropriately (based on implicit and explicit evaluations,

	Training MSE	Validation MSE	Time	Parameters	Mean value
Small CRAE	0.21299473	0.21207126	5 hours	465 688	0.2051166
Large CRAE	0.21088413	0.20995240	3 days	55 622 416	0.6909323
Small CBOW-CRAE	0.21833613	0.21715157	2 hours	241 384	0.3463651
Large CBOW-CRAE	0.21731463	0.21609975	2 days	27 866 896	0.1174563

Table 6: Reconstruction errors from architectures

see the main text), even if the reconstruction error is worse than for CRAE model. First, the average embeddings values for the Large CBOW-CRAE models are closer to 0, which is desirable to increase the representation flexibility (especially within a GLM, because the range of embeddings is [-1, 1]). Attempts to manually correct these issues (normalization of embedding values after training) do not improve the quality of embeddings. In addition, the Large CBOW-CRAE had less saturated embedding dimensions, as we discuss in the implicit evaluations. For these reasons, we continue our evaluation of embeddings with the Large CBOW-CRAE model, but we encourage researchers to experiment with other configurations. In particular, the small model achieves small training MSE in less time and much less parameters and lets data scientists validate the model quickly.

References

[Blier-Wong et al., 2020] Blier-Wong, C., Baillargeon, J.-T., Cossette, H., Lamontagne, L., and Marceau, E. (2020). Encoding neighbor information into geographical embeddings using convolutional neural networks. In *The Thirty-Third International Flairs Conference*.

[Corbett-Davies and Goel, 2018] Corbett-Davies, S. and Goel, S. (2018). The Measure and Mismeasure of Fairness: A Critical Review of Fair Machine Learning. *arXiv:1808.00023 [cs]*.

[He et al., 2015] He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034.

[He et al., 2016] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778.

[Ioffe and Szegedy, 2015] Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456.

[Kusner et al., 2018] Kusner, M. J., Loftus, J. R., Russell, C., and Silva, R. (2018). Counterfactual Fairness. *arXiv:1703.06856 [cs, stat]*.

[Lindholm et al., 2020] Lindholm, M., Richman, R., Tsanakas, A., and Wüthrich, M. V. (2020). Discrimination-free insurance pricing. *Available at SSRN : 3520676*.

[Simonyan and Zisserman, 2014] Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.