Optimized Conversion of Categorical and Numerical Features in Machine Learning Models

Tom Butler, Emily Liang, Wren Paris-Moe,
Andrea Stine

PICMath

Problem:

- Ritwik Sinha, Ph.D., Adobe Research
- Given Problem: Using different encoding methods for categorical features in supervised learning models
- Research Question: What encoding method is best for categorical features?

Datasets

Name of Data Set	Size	Train Size	Test Size	Features
Criteo Conversion	15,898,883	70%	30%	9 numeric + 9 categorical
Amazon Employee Access	32,769	70%	30%	9 categorical
Avazu Click Through Rate Prediction	40,428,968	50%	50%	20 categorical
KDD 2009	50,000	70%	30%	189 categorical + 20 continuous
US Census 1990	2,458,285	70%	30%	67 categorical
Adult	48,842	67%	33%	8 categorical

Methods Explored:

Encoding Methods Used in Supervised/Semi-Supervised ML algorithms:

- One-hot Encoding and other category encoders
- Learned Embedding

Unsupervised Learning Models

- Wide and Deep

Results

- With high-cardinality features, it is better to used learned embedding and WDL (e.g. US census)
- One-hot encoding and other category encoders are useful when your categorical features are not of a high-cardinality (e.g. adult dataset)

One-Hot Encoding

Methodology:

- Converts categorical features into binary vectors
- Prevents the model from weighting the variables/treating them as ordinal variables

Challenges:

- High cardinality in features

Other Category Encoders

Ordinal:

- Assigns an integer value to every distinct object within a feature column
- Same dimensions

Backwards Difference

- Ordinally encodes the data
- The difference is calculated between the mean of the feature values for the current level and the mean for the prior level
- Iterates through the levels creating a new column for each value within a feature

Binary

- Ordinally encodes data
- Integers are converted to binary
- Each digit is placed in a separate column

Base N

- Ordinally encodes data
- Encodes columns into arrays of their base-N representation
- Base-1 = one-hot encoding (not really base-1)
- Base-2 = binary encoding
- N = # of values in a feature → ordinal encoding

Metrics for Category Encoders:

- **Adult Dataset**
- **Decision tree with optimized hyperparameters**

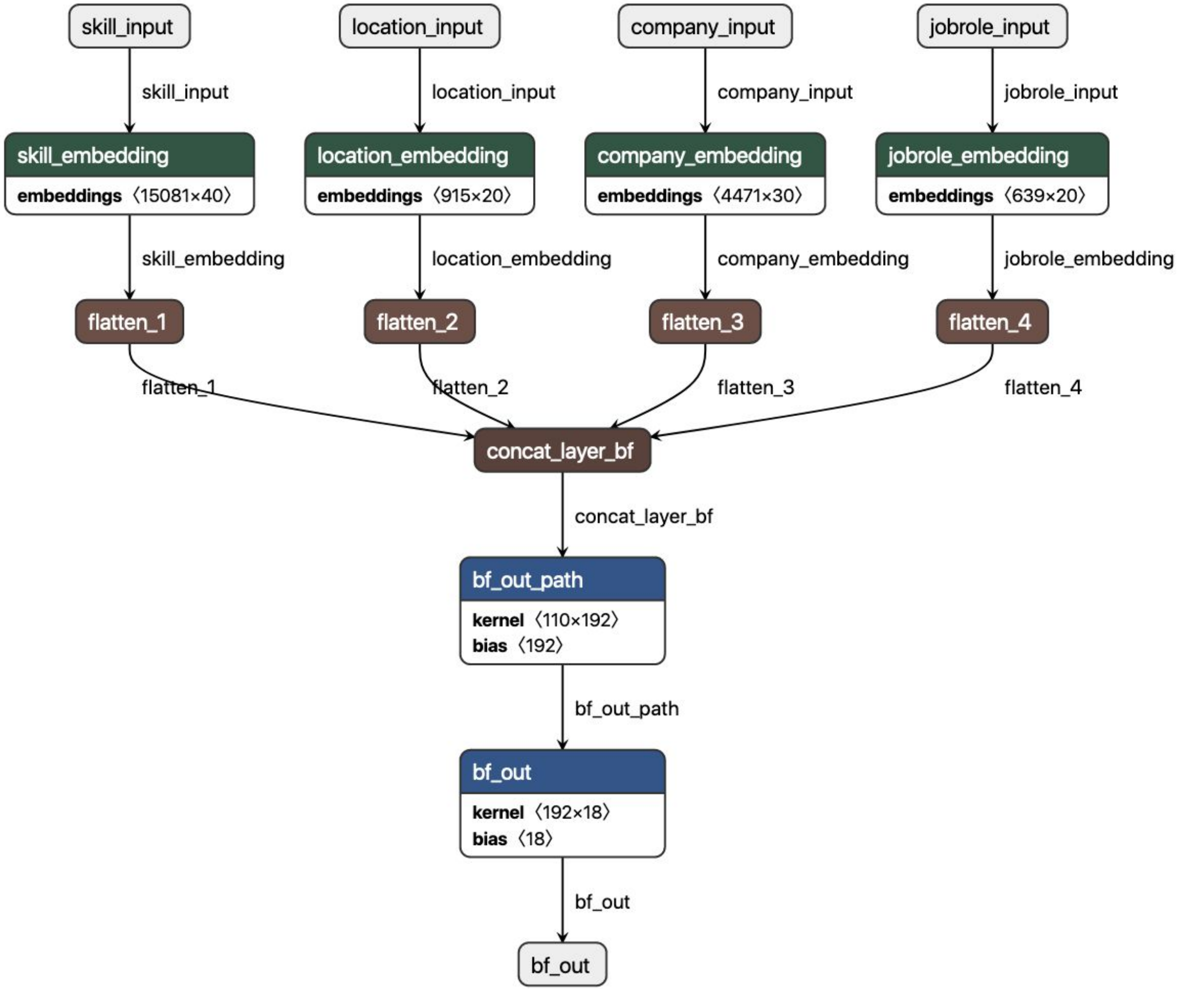
Metric	One-hot	Ordinal	Backwards Difference	Base 1 (similar to one-hot)	Base 2	Base 5	Base 10	Binary
Accuracy	**0.8571**	0.8567	0.8567	**0.8571**	0.8533	0.8566	**0.8587**	0.8533
Precision	0.7301	0.7407	0.7407	0.7302	0.7356	0.7489	0.7585	0.7356
Recall	0.6266	0.6053	0.6053	0.6269	0.5918	0.5897	0.5897	0.5918

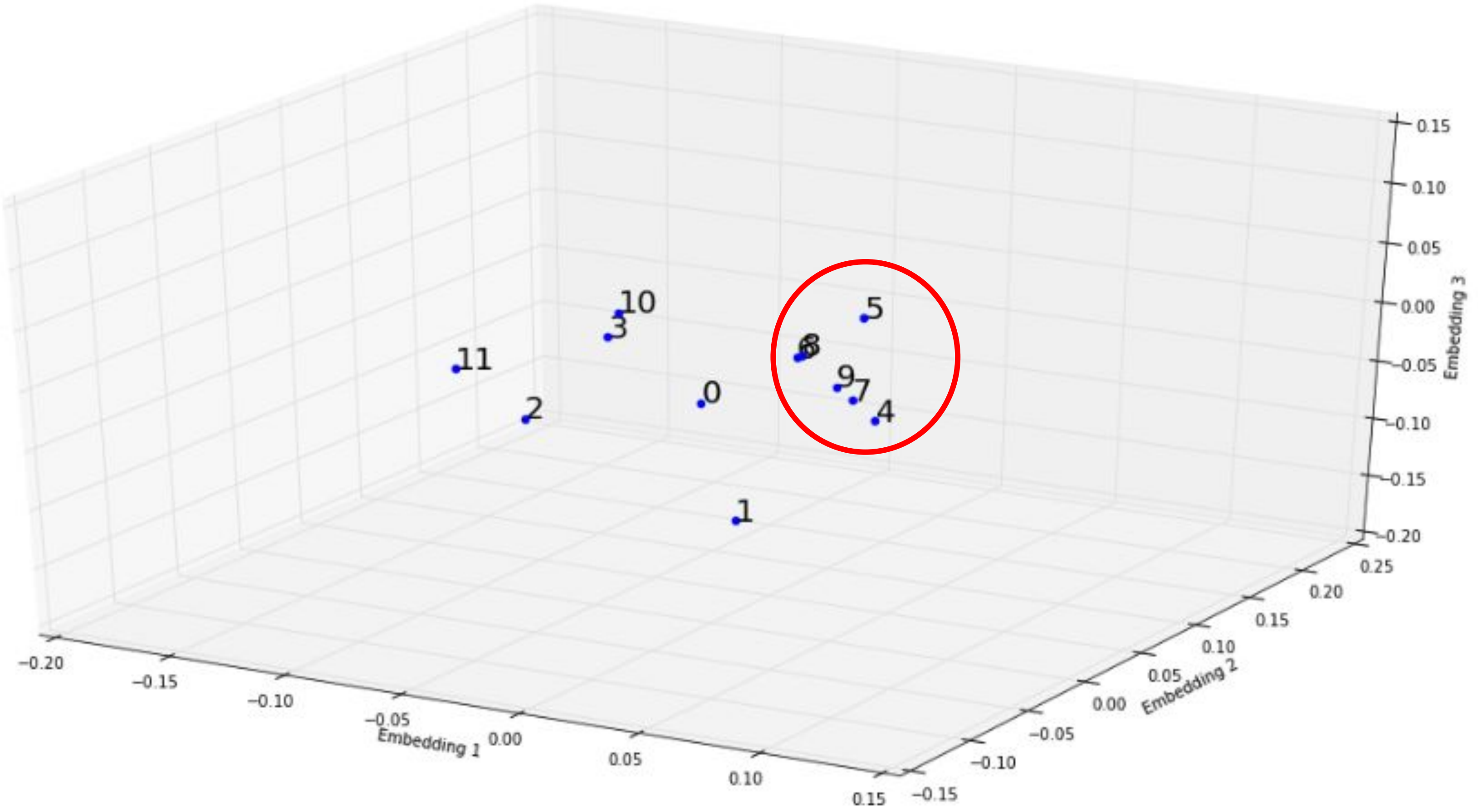
Metrics for OHE

			Accuracy	Precision	Recall	F-beta
Decision Tree	Amazon	Test	0.9408	0.9408	1	0.9695
		Train	0.9456	0.9479	0.9971	0.9718
	US Census	Test	1	1	1	1
		Train	1	1	1	1
Random Forest	Amazon	Test	0.9408	0.9408	1	0.9695
		Train	0.9426	0.9462	1	0.9704
	US Census	Test	0.999	0.999	1	0.999
		Train	0.999	0.999	1	0.999

Learned Embedding

- Reduce dimensionality of large datasets to lower dimension space
  - Reduces the memory/computing demands
- One embedding layer is required for each categorical variable
  - Must be ordinally encoded
- Maps similar inputs closer together
- Does not rely on memorization of input values


skill_input
location_input
company_input
jobrole_input
skill_input
location_input
company_input
jobrole_input
skill_embedding
embeddings ⟨15081×40⟩
location_embedding
embeddings ⟨915×20⟩
company_embedding
embeddings ⟨4471×30⟩
jobrole_embedding
embeddings ⟨639×20⟩
skill_embedding
location_embedding
company_embedding
jobrole_embedding
flatten_1
flatten_2
flatten_3
flatten_4
flatten_1
flatten_2
flatten_3
flatten_4
concat_layer_bf
concat_layer_bf
bf_out_path
kernel ⟨110×192⟩
bias ⟨192⟩
bf_out_path
bf_out
kernel ⟨192×18⟩
bias ⟨18⟩
bf_out
bf_out



Embedding 1
Embedding 2
Embedding 3
0
1
2
3
4
5
7
9
10
11

Our Approach

- Embed categorical features of each data set
- Keras Neural Network API
- Mapping our datasets to 10 dimension space
- For optimization, we presented embedded data multiple times to neural network
    - Epoch: One complete presentation of data set to neural network model

Embedding Results

Data Set	Accuracy (%)
Adult	83.31
US Census	99.99
KDD	98.44
Amazon	98.55

Wide and Deep Learning for Prediction Models

Method that bypasses the need for complex feature extraction and conversion in preprocessing
→ Combines feature encoding and prediction model into one system
→ Structure of feature space is passed to neural network prediction model
→ Feature encoding is learned as the prediction model trains

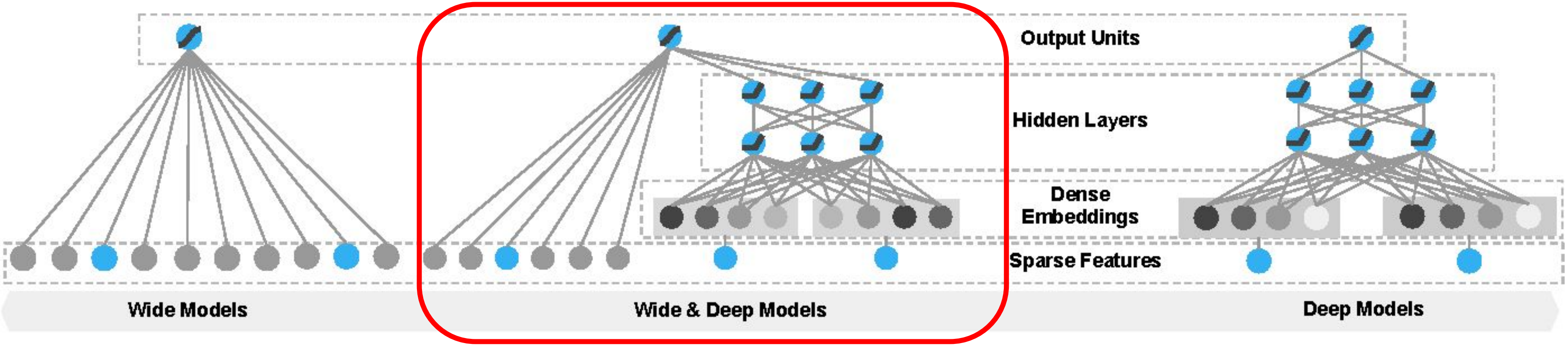
The Model:

Wide Component:

- Cross-product feature transformations
- Memorization

Deep Component:

- Feed-forward neural network
- Generalization



Implementation of Wide and Deep Learning

Data Preprocessing & Initial Feature Extraction:

1) Initially, preprocesses the input data by ordinally encoding the categorical features and using a standard scalar transformation on the numerical features
2) Applies polynomial transformation with degree 2 to previously labelly encoded categorical features (used in wide component of algorithm)
   a) Ex: Converts lower dimensional feature space to higher dimensional space [a, b] → [1, a, b, a^2, ab, b^2]

The preprocessed feature spaces are used in the wide and deep components of the algorithm

Implementation of Wide and Deep Cont.

Defines the feature space to be passed to the neural network in two components: **wide_component** & **deep_component**

deep_component:

1) Maps each categorical column into a dimension of 0.25*cardinality
   a) Uses keras package ‘Embedding’ and ‘Flatten’ to decrease dimensionality of categorical features
2) Each hidden layer of the neural network halves the dimensions of the feature space
   a) New generalizations are learned by extracting new feature sets from the old ones
   b) Final layer produces a dimensionality of 128 which forms the deep_component output layer

wide_component:

1) Uses the previous polynomial transformation as the output layer for the wide component

Implementation of Wide and Deep Cont.

Create_model method:

1) Concatenates the output layers from the deep_component & wide_component
2) Uses a standard sigmoid activation function
3) Creates the prediction model with initial input data and extracted output layer
4) Python Code:
   a) output = Dense(1, activation='sigmoid')(out_layer)
   b) model = Model(inputs=inputs, outputs=output)
5) Then, fit and train the model to make new predictions

Results

Using fitted model based on training data, prediction accuracy is calculated through evaluating model with the test data

Premise of prediction: (adult dataset)

- uses age, working class, education, marital status, race, gender, hours per week, native country, etc... (14 features in total)
- determines if an individual makes more or less than $50k annually

Prediction accuracy: 82.58%

Conclusion

- Low cardinality feature space:
  - OHE produces sufficient metrics when the cardinality of the feature space is low
  - Computationally inexpensive / easy to implement
- High-cardinality feature space:
  - OHE runs into memory issues - drastically increases dimensions of feature space
  - Learned embedding + WDL - don’t have memory issues
    - Learn and extract new features through a DL neural network framework
    - Allows for generalizations and new dependencies to be discovered within the data

Future Implementation

“The wide and deep model truly shines on larger data sets with high-cardinality features”

- One-Hot encoding is the opposite
  - Memory issues - creates a new feature column for every distinct value within a column

Future implementation idea:

- Use method to calculate cardinalities of feature space during preprocessing
- If cardinality is low, use OHE or other categorical encoders since dimensionality is not an issue
- If cardinality is high, use a DL based framework

Additional Ideas

Instead of a fully integrated system like Wide & Deep → Independently use a DL model for feature extraction and conversion

- Hard to compare Wide & Deep to OHE or other basic category encoders (Different classification models used)
- Fix: use DL models as a preprocessing method
  - Easy to compare when the same classifier can be used on separately encoded data

Stacked Autoencoders:

- Neural networks used for feature extraction
- Transform input to a new representation with lower dimensions
- Learning new features by transforming the old data
- Several autoencoders are stacked to complete the process
- Goal: Newly extracted features are more informative than original feature space
- Often used for noise reduction in time series
- Can easily be implemented in a hybrid system → Often done in financial prediction systems

Acknowledgements

PIC Math is a program of the Mathematical Association of America (MAA) and the Society for Industrial and Applied Mathematics (SIAM). Support is provided by the National Science Foundation (NSF grant DMS-1345499).

Professor Treena Basu, Occidental College

Professor Ron Buckmire, Occidental College

Dr. Ritwik Sinha, Adobe Research

Resources

https://twimlai.com/googles-wide-deep-learning-models/
https://ai.googleblog.com/2016/06/wide-deep-learning-better-together-with.html
https://ai.googleblog.com/2016/06/wide-deep-learning-better-together-with.html
https://machinelearningmastery.com/how-to-prepare-categorical-data-for-deep-learning-in-python/
https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/
https://towardsdatascience.com/categorical-embedding-and-transfer-learning-dd3c4af6345d
https://towardsdatascience.com/deep-embeddings-for-categorical-variables-cat2vec-b05c8ab63ac0