Machine Learning Design Patterns

1/19/21 1:31 PM

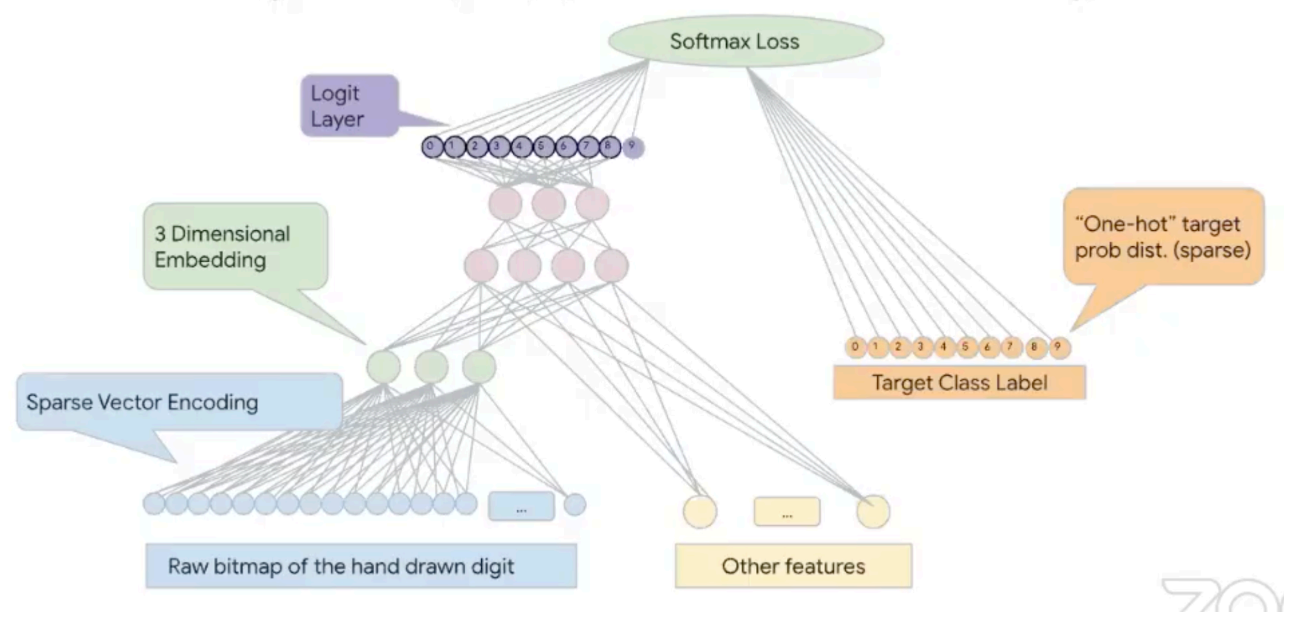
Speaker: Lak Lakshmanan, Sara Robinson, Michael Munn (Google Cloud)
Date: 1/19/2021

Design patterns are formalized best practices to solve common problems when designing a software system

Pattern Case 1: Embedding

Case: recommend movies to customers

- One way: organize movies by similarity (1D)
  - E.g. Average age of viewers
- Using a second dimension gives us more freedom in organizing
- A d-dimensional embedding assumes that user interest in movies can be approximated by d aspects
- Embeddings can be learned from data
  - The weights in the embedding layer are learned through backprop just as with other weights



Auxiliary Learning Task

- No need for labeling
- Can use much more data
- Embeddings can be more general

How embedding works

- The result of embedding is such that similar items are close to each other, and vector directions have meaning
- Then we can take advantage of this similarity property of embeddings
- Also we can cluster embedding vectors

Pattern Case 2: Multilabel

Often model prediction tasks involve applying a single classification to a given example

- Softmax to get highest probability in certain classes

In some cases, may want to let a single training example be assigned more than one label

- E.g. Toxic Comment Classification Challenge (Kaggle)
  - Each comment may involve more than more class
  - Multi-hot encode

Use sigmoid

- Multilabel pattern allows not summing to 1 for different labels

Choosing a loss function

- Categorical cross entropy (softmax + cross entropy loss)
- Binary cross entropy (sigmoid + cross entropy loss)

How to parse sigmoid results, apply a threshold

Depends on data and use case
Rule of thumb:

- # specific tags / # total examples as baseline

Dataset considerations

- E.g. on Stack Overflow, most posts with #pandas are included in #matplotlib
  - In this case we might want to massage our dataset a little, to make sure to have enough examples with only #pandas
  - Similar as handling imbalance dataset
- Ensure combinations of overlapping labels are well represented in your dataset

Hierarchical labels

- E.g. ImageNet
- Two common approaches for handling hierarchical labels:
  - Use flat approach, put every label in same output array
  - Use the Cascade Deign Pattern, leveraging multiple models based on the output of previous models

Also useful for overlapping labels

- If more than one label is correct, multilabel design allows assignment for both, while traditional multi-class only allows one label

Pattern Case 3: Model Versioning

A deployed model is only the beginning of your ML lifecycle

Many reasons to update the model:

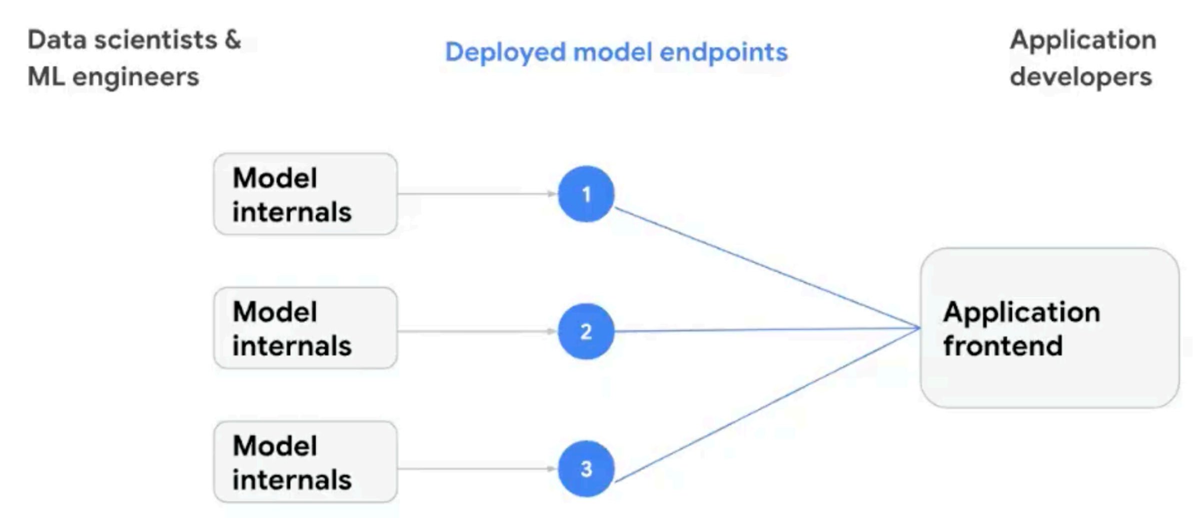
- Concept drift can cause models to become stale
  - E.g. cell (bio to phone); cloud (weather to tech)
- Data drift can cause models to become stale
- Feature requests
- Newly available data
- New data schema

Two types of model users

- Application developers
  - Model Versioning focuses mostly here
- End consumers

New versions as different REST endpoints

*REST: representational state transfer
A standard for software architecture for interactive applications that use multiple web services



Why is it useful?

- Backwards compatibility
- Fine-grained performance monitoring
- A/B testing
- Decoupling model from app frontend

Alternative: multiple serving functions

- Example in TensorFlow: signature
- Why use multiple serving functions?
  - Need for maintaining multiple versions over time
  - Support different input formats
  - Save processing time

New version or new model?

- Rule of thumb: if prediction task changes or change will break existing clients, should create a new model
  - E.g. Regression for predicting flight delays in min -> Classification for predicting delay or not
    - Should be new model
  - E.g. Model trained on 2015-2019 data -> New data 2019-2021 (but same schema)
    - Should be just a new version

Summary

Chapter	Design pattern	Problem solved	Solution
Data Representation	Embeddings	High-cardinality features where closeness relationships are important to preserve	Learn a data representation that maps high-cardinality data into a lower-dimensional space in such a way that the information relevant to the learning problem is preserved
Problem Representation	Multilabel	More than one label applies to a given training example	Encode the label using a multi-hot array, and use k sigmoids as the output layer
Reproducibility	Model Versioning	It is difficult to carry out performance monitoring and split test model changes having a single model in production or updating models without breaking existing users.	Deploy a changed model as a microservice with a different REST endpoint to achieve backward compatibility for deployed models