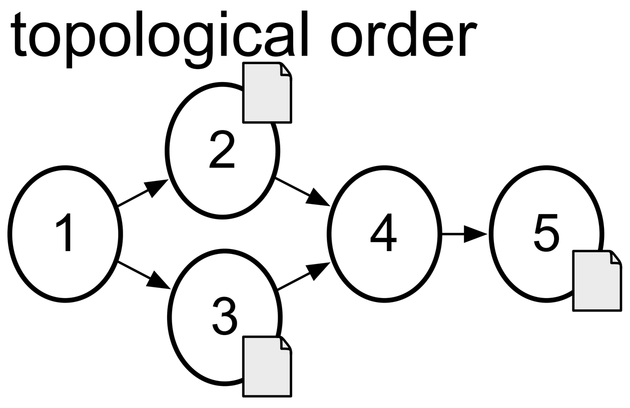
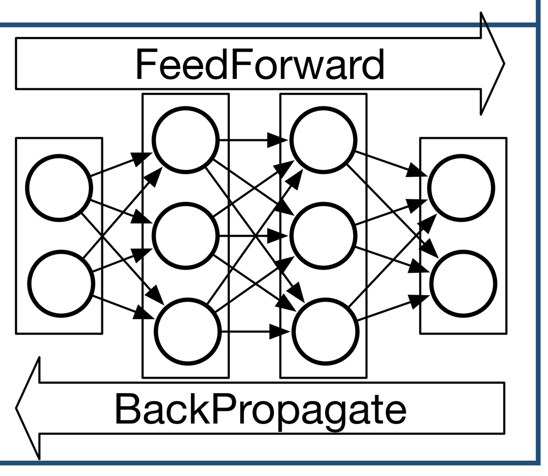
Making Classical Machine Learning Pipelines Differentiable: A Neural Translation Approach

Gyeong-In Yu[s], Saeed Amizadeh[m], Byung-Gon Chun[s], Matteo Interlandi[m], Markus Weimer[m],
[s]Seoul National University, [m]Microsoft

Motivation: Comparing Two Approaches of doing Machine Learning

	Classical Machine Learning (ML)	Deep Learning (DL)
Frameworks	ML.NET, scikit learn, H2O.ai, Spark MLlib	TensorFlow, PyTorch, Keras, mxnet
How to Use?	Compose *trainable / non-trainable ops* to form a *pipeline* ▪ Trainable ops require the estimation of parameters from input data (logistic regression, normalizer, …)	Compose multiple layers of nonlinear units ▪ Cascaded trainable operators
How to Train?	Train trainable ops **sequentially** in topological order ▪ Use previous trainable ops as a fixed feature extractor ▪ Greedy approach; parameters are not globally optimized	Train layers simultaneously ▪ Use backpropagation ▪ Parameters are **globally** estimated to reach better (local) optimum





What if we **train ML pipelines globally** as in DL training?

Neural Translation Framework

- Translate (possibly trained) ML pipelines into neural networks → **fine tuning** via backprop
  - ➢ Backpropagation instead of greedy op-wise training → improve accuracy
  - ➢ Enable GPU-acceleration without reinventing the wheel for ML pipelines

```
mlnet_pipe = Pipeline([
    OneHotVectorizer(categorical_columns),
    Expression('x: x != 0 ? 1.0 : 0.0', {'Label01': 'Label'}),
    ColumnConcatenator({'Features': categorical_columns + numerical_columns}),
    PcaTransformer('Features'),
    MinMaxScaler('Features'),
    Featurizer(SDCABinaryClassifier('Label01','Features')),
    ColumnConcatenator({'Features': ['Features', 'SDCAScore', 'SDCAProb']}),
    PoissonRegression('Label', 'Features')])
mlnet_pipe.fit(X, y) # training
pytorch_model = mlnet_pipe.to_pytorch() # translation
pytorch_model.fit(X, y) # fine-tuning
mlnet_pipe.from_pytorch(pytorch_model) # translate back
```

Translation Process

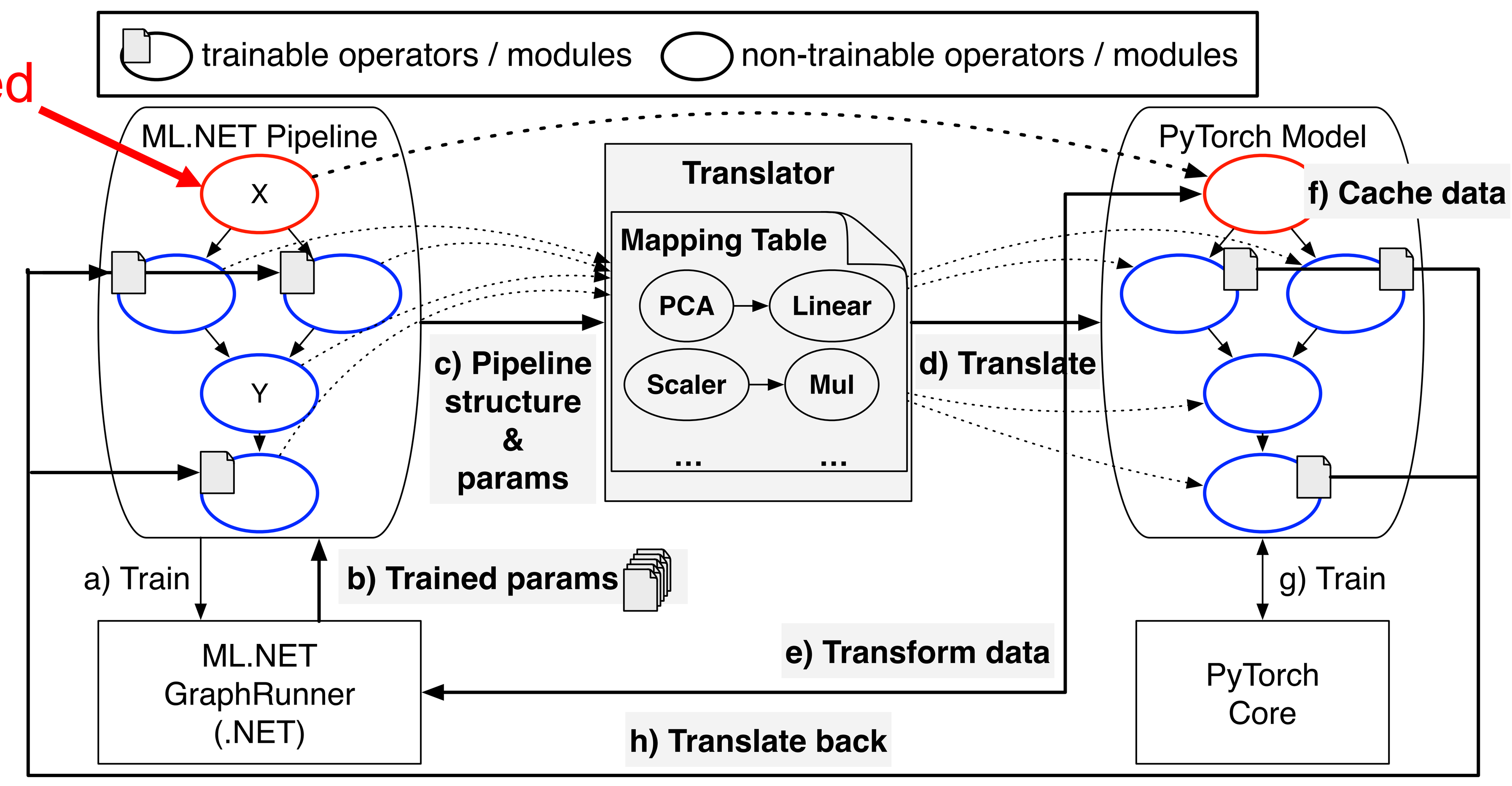
- Generate a neural network module out of each target op
  - ➢ Mirror the transformation logic & transfer parameters
- Compose all modules into a single neural network
  - ➢ Follow the dependencies in the original ML pipeline

Examples

- PCA → fully-connected linear layer
- MinMaxScaler → element-wise multiplication
- Concatenator → tensor concatenation

Prototype Implementation

- Mapping table between ML.NET op ⟺ PTH module
- Each op is either translated or referenced
- Translation targets:
  - ➢ Trainable ops
  - ➢ Ops that follow other target ops (to enable backprop)
- Cache and reuse the output of referenced ops over epochs



Preliminary Evaluation

- A regression task - 2.3K records for training
- Baselines
  - ➢ Original ML.NET pipeline
  - ➢ Translated PTH model - initialize params randomly

	ML.NET	PyTorch	Fine Tuning
Poisson Loss	13.67	15.60	**12.01**

Ongoing & Future Work

- Translate tree models into neural networks
- Hyperparameter tuning (because we train twice)
- Investigate how GPUs improve runtime performance
- System optimization (e.g., better caching)