Composable Machine Learning

Eric Xing
Petuum & Carnegie Mellon
Acknowledgements

Zhiting Hu
Scientist @Petuum Inc.
PhD Candidate @CMU

Qirong Ho
Co-founder, CTO @Petuum Inc.

R&D team @
Petuum

Sailing Lab @
Carnegie Mellon University
An AI era?
Real-world Machine Learning Problems
The Cement Industry

**PRODUCTION PROCESS OF CEMENT**

**STAGE 1**
**RAW MATERIALS PREPARATION**
Raw materials needed to produce cement (e.g., calcium carbonate, silica, alumina and iron ore) are produced from limestone, rock, dust, coke or clay and ferrous containing material. These raw materials are crushed through a milling process.

**STAGE 2**
**PYRO PROCESSING**
Grinding produces a fine powder, known as raw meal, which is preheated and then sent to the kiln. The raw meal is heated to around 1,450°C, where chemical reactions take place to form cement clinker.

**STAGE 3**
**CEMENT GRINDING AND DISTRIBUTION**
A small amount of gypsum is added to the clinker to regulate cement setting time. The mixture is then sent to the cement mill where it is ground to a fine powder known as cement. The cement is stored in silos or storage tanks before being distributed in bulk or in bags to the sites where it will be used.

**Operational Excellence Initiatives**

- **Emissions (NOx, SOx)**
- **Fuel-Mix (Traditional vs. Renewables vs. Alternatives)**
- **Benchmarking fleet Performance**
- **Predictive/Preventive Maintenance**
Assess, Process, Factory, Corporate Optimization

Kiln Optimization

20,000 Variables

Clinker Cooler Optimization

~2,000 Variables
Building an ready-to-use AI solution for this is extremely complex. Raw data far from ideal and simultaneous prediction and control systems/infra require inter-operation between diverse systems. Can’t solve with “algo marketplace” or Kaggle competition.
This is where evidence and information start.
Building an ready-to-use AI solution for this is

Extremely complex

Task: Automatic Medical Report Generation

Requires inter-operation between diverse systems

User interface for Doctors

Can't solve with "algo marketplace" or Kaggle competition
AI as of now: Not Built, but Crafted
Outline

- Composable ML
  A first-principle view of ML components and assemblage

- Texar: A Modularized ML toolkit
  Compose your ML applications like playing building blocks

- Scalable AI Infrastructure
  Composable ML in production
Composable ML

One-off design and programming

Modular and standardized

Complex “rhythms” and “chords” systematically built out of simpler “notes”
Single Models with Increasing Size and Performance

- Increasingly large black-box neural networks
- Good, even super-human performance on some tasks

<table>
<thead>
<tr>
<th>Model</th>
<th>Year</th>
<th>Size (Approx.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlexNet</td>
<td>2012</td>
<td>~60M</td>
</tr>
<tr>
<td>ResNet</td>
<td>2015</td>
<td>~25M</td>
</tr>
<tr>
<td>AlphaGo</td>
<td>2016</td>
<td>~5M</td>
</tr>
<tr>
<td>Transformer</td>
<td>2017</td>
<td>~210M</td>
</tr>
<tr>
<td>BERT</td>
<td>2018</td>
<td>~340M</td>
</tr>
<tr>
<td>GPT2</td>
<td>2019</td>
<td>1.5B</td>
</tr>
</tbody>
</table>
Single Giant Models Enough?
Difficulties of Single Giant Models

- Explainability
- Expertise
- Debugging
- Maintenance
- Upgrade
- Scalability
- ...

Good work -- but I think we might need a little more detail right here.
Far from Solving Real Complex Problems

Extremely complex
Raw data far from ideal
Simultaneous Prediction and Control

Requires inter-operation between diverse systems
Can’t solve with “algo marketplace” or Kaggle competition

Task: Automatic Medical Report Generation

Extremely complex

V.S.

User interface for Doctors

Systems/Infra

Proprietary & Confidential | 16
Running Example: Machine Translation (MT)

We choose MT as our running example to keep this talk to 90min.

**raw data**
- I like this movie.
- Lovely and poignant
- Insanely hilarious!
- …

**source.dat**
- Ich mag diesen film.
- Schön und ergreifend
- Wahnsinnig witzig!
- …

**target.dat**
- clean data
- clean data
- cleaning
- tokenizing
- vocabulary
- truncation
- …

**model**
- training
- maximum likelihood training
- reinforcement learning
- adversarial learning
- finetuning

**evaluation**
- post-processing
- …

Proprietary & Confidential | 17
Running Example: Machine Translation (MT)

raw data
evaluation
post-processing
Solution: Composable ML

A modularized way to build complex applications
Solution: Composable ML

● Build AI solutions more easily, via pick-and-choose:
  ○ Data: text, speech, image, video, time series, …
  ○ Models: recurrent, transformer, convolutional, …
  ○ Tasks: classification, regression, generation, discovery, …

● Stop writing same one-off code again and again
  ○ More reliable and easier to debug
  ○ Easier to onboard new developers
First Principles: Decomposing Machine Learning

Loss functions
(likelihood, reconstruction, margin, ...)

Model architectures
(RNNs, Transformers, Graphical, ...)

Constraints
(normality, sparsity, logical, KL, sum, ...)

Algorithms
MC (MCMC, Importance), Opt (gradient, IP), ...

Data
-processing, augmentation, weighting, ...
Decomposing Machine Learning

Machine Learning:
Computational methods that enable machines to learn concepts and improve performance from experience

$$\min_\theta \mathcal{L}(\theta, D) + \Omega(\theta)$$
Decomposing Machine Learning

Machine Learning:
Computational methods that enable machines to learn concepts and improve performance from experience

$$\min_{\theta} \mathcal{L}(\theta, D) + \Omega(\theta)$$

$$y \sim p_{\theta}(y|x)$$

model architecture/inference procedure
Decomposing Machine Learning

Machine Learning:
Computational methods that enable machines to learn concepts and improve performance from experience

\[ \min_{\theta} \mathcal{L}(\theta, D) + \Omega(\theta) \]

experience (data)

model architecture/inference procedure

\((x^*, y^*)\)
Decomposing Machine Learning

Machine Learning:
Computational methods that enable machines to learn concepts and improve performance from experience

$$\min_{\theta} \mathcal{L}(\theta, D) + \Omega(\theta)$$

- loss
- model architecture/inference procedure
- experience (data)
Decomposing Machine Learning

Machine Learning:
Computational methods that enable machines to learn concepts and improve performance from experience

\[
\min_{\theta} \mathcal{L}(\theta, D) + \Omega(\theta)
\]

- **loss**
- **model architecture/inference procedure**
- **experience (data)**
- **constraint**
Decomposing Machine Learning

Machine Learning:
Computational methods that enable machines to learn concepts and improve performance from experience.
Running Example: Machine Translation

I like this movie.  
Lovely and poignant  
Insanely hilarious!  

Ich mag diesen film.  
Schön und ergreifend  
Wahnsinnig witzig!  

\[
\min_{\theta} \mathcal{L}(\theta, D) + \Omega(\theta)
\]
ML Components

- Constraint
- Loss
- Learning
- Inference
- Architecture

\[ \min_{\theta} \mathcal{L}(\theta, D) + \Omega(\theta) \]
ML Components

- Constraint
- Loss
- Learning
- Inference
- Architecture

\[ \min_{\theta} \mathcal{L}(\theta, D) + \Omega(\theta) \]
Architecture (1): Language Model

- Calculates the probability of a sentence:
  - Sentence: $y = (y_1, y_2, \ldots, y_T)$
  - Example: $(I, like, this, \ldots)$

$$\min_{\theta} \mathcal{L}(\theta, D) + \Omega(\theta)$$
Architecture (1): Language Model

- Calculates the probability of a sentence:
  - Sentence:
    \[ y = (y_1, y_2, ..., y_T) \]
    \[ p_\theta(y) = \prod_{t=1}^{T} p_\theta(y_t \mid y_{1:t-1}) \]
  - Example:
    \[(I, like, this, ...)
    \[ \cdots p_\theta(\text{like} \mid I) \cdot p_\theta(\text{this} \mid I, like) \cdots \]
Architecture (1): Language Model

- Calculates the probability of a sentence:
  - Sentence:
    \[ y = (y_1, y_2, \ldots, y_T) \]
    \[ p_\theta(y) = \prod_{t=1}^{T} p_\theta(y_t \mid y_{1:t-1}) \]
  - Example:
    \[ (I, \ like, \ this, \ \ldots) \]
    \[ \ldots p_\theta(\text{like} \mid I) p_\theta(\text{this} \mid I, \text{like}) \ldots \]
Architecture (1): Language Model

- Calculates the probability of a sentence:
  - Sentence:
    \[ y = (y_1, y_2, ..., y_T) \]
    \[ p_\theta(y) = \prod_{t=1}^{T} p_\theta(y_t | y_{1:t-1}) \]
  - Example:
    \[(I, like, this, ...)\]
    \[... p_\theta (like | I) p_\theta (this | I, like) ...\]
Architecture (2): **Conditional** Language Model

- Conditions on additional task-dependent context $x$
  - Machine translation: source sentence
    
    \[ I \text{ like this movie.} \quad \rightarrow \quad \text{Ich mag diesen film.} \]
  
  - Medical image report generation: medical image
    
    \[ \ldots \text{There is chronic pleural-parenchymal scarring within the lung bases. No lobar consolidation is seen. } \ldots \]
Architecture (2): **Conditional Language Model**

- Conditions on additional task-dependent context $x$

$$p_\theta(y \mid x) = \prod_{t=1}^{T} p_\theta(y_t \mid y_{1:t-1}, x)$$

$$\min_\theta L(\theta, D) + \Omega(\theta)$$
Architecture (2): **Conditional Language Model**

- Conditions on additional task-dependent context $\mathbf{x}$

$$p_\theta(\mathbf{y} | \mathbf{x}) = \prod_{t=1}^{T} p_\theta(y_t | \mathbf{y}_{1:t-1}, \mathbf{x})$$
Architecture (2): **Conditional Language Model**

- Conditions on additional task-dependent context $x$

$$p_\theta(y \mid x) = \prod_{t=1}^{T} p_\theta(y_t \mid y_{1:t-1}, x)$$

- Language model as a **decoder**
Architecture (2): **Conditional Language Model**

- Conditions on additional task-dependent context $x$

The conditional probability of the output sequence $y$ given the input context $x$ is given by:

$$p_{θ}(y | x) = \prod_{t=1}^{T} p_{θ}(y_t | y_{1:t-1}, x)$$

- Language model as a decoder
- Encodes context with an encoder
Architecture Graph

- Classifier
- Encoder-Decoder

\[
\min_{\theta} L(\theta, D) + \Omega(\theta)
\]
Architecture Graph

- FFNetwork
- Encoder
- Decoder
- Classifier
- Encoder-Decoder
- Embeder

Mathematical expression:
\[ \min_\theta \mathcal{L}(\theta, D) + \Omega(\theta) \]
Architecture Graph

- FFNetwork
- Classifier
- Encoder
- Encoder-Decoder
- Decoder
- Embeder

Layers: Conv, Dense ...

$$\min_{\theta} \mathcal{L}(\theta, D) + \Omega(\theta)$$
Architecture Graph

- Classifier
- Encoder-Decoder
- Encoder
- Decoder
- Embeder
- FFNetwork
- RNN
- Transformer
- Multi-head Attention
- Recur-Attention: Bah, Luo …
- Layers: Conv, Dense …
- Cell: LSTM, GRU …

\[ \min_{\theta} \mathcal{L}(\theta, D) + \Omega(\theta) \]
Architecture Graph

- Classifier
- Encoder-Decoder
- Encoder
- Decoder
- Embeder
- FFNetwork
- RNN
- Transformer
- WordEmbeder
- PositionEmbeder
- Recur-Attention: Bah, Luo ...
- Multi-head Attention
- Cell: LSTM, GRU ...
- Layers: Conv, Dense ...

\[ \min_{\theta} \mathcal{L}(\theta, D) + \Omega(\theta) \]
Complex Composite Architectures

E refers to encoder, D to decoder, C to Classifier, A to attention, Prior to prior distribution, and M to memory.
ML Components

Constraint  Loss  Learning  Inference

Architecture

decoder
LSTM RNN
Attention RNN
Transformer
...
encoder
classifier
...
ML Components

Constraint  Loss  Learning  Inference  Architecture

decoder
LSTM RNN
Attention RNN
Transformer

\[ \min_\theta \mathcal{L}(\theta, D) + \Omega(\theta) \]
Learning, Inference & Loss (1): Maximum Likelihood Estimation

- Given data example \((x^*, y^*)\)
- Maximizes log-likelihood of the data

Learning

\[
\min_\theta \mathcal{L}_{\text{MLE}} = -\log p_\theta(y^* | x^*)
\]

\[
= - \prod_{t=1}^{T} p_\theta(y_t^* | y_{1:t-1}^*, x^*)
\]

Inference

Teacher-forcing decoding:

For every step \(t\), feeds in the previous ground-truth tokens \(y_{1:t-1}^*\) to decode next step

Loss

Cross-entropy loss
Learning, Inference & Loss (2): Adversarial Learning

Learning

- A discriminator is trained to distinguish between real data examples and fake generated samples.
- The model is trained to fool the discriminator.

Loss

- Binary adversarial loss.
- Feature-matching adversarial loss.

Inference

Gumbel-softmax decoding:

Uses a differentiable approximation of sample $\hat{y}$ for gradient backpropagation:

$$\frac{\partial L(\hat{y})}{\partial \theta} = \frac{\partial L(\hat{y})}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \theta}$$
Learning, Inference & Loss (3): Reinforcement Learning

Learning

- Optimizes *expected* task reward
  - $\mathbb{E}_{\hat{y} \sim p_{\theta}(y \mid x)} \left[ \text{BLEU}(\hat{y}, y^*) \right]$ for machine translation

Loss

- Policy gradient loss
- Policy gradient loss w/ baseline
- ...

Inference

- Greedy decoding
- Sampling decoding
- Beam search decoding
- Top-$k$ / Top-$p$ decoding
- ...

Policy Gradient Agent

Decoder

$\min_{\theta} L(\theta, D) + \Omega(\theta)$
ML Components

Constraint
- Cross-entropy
- Binary Adv loss
- Matching Adv loss
- PG loss
- PG loss + baseline
- ...

Loss
- Cross-entropy
- Binary Adv loss
- Matching Adv loss
- PG loss
- PG loss + baseline
- ...

Learning
- MLE
- Adversarial
- Reinforcement
- Adv + RL
- Structured super.
- Reward-aug.
- ...

Inference
- Teacher-forcing
- Gumbel-softmax
- Sample
- Greedy
- Beam-search
- Top-k sample
- ...

Architecture
- decoder
- LSTM RNN
- Attention RNN
- Transformer
- encoder
- classifier
- ...

\[
\min_{\theta} \mathcal{L}(\theta, D) + \Omega(\theta)
\]
## ML Components

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Loss</th>
<th>Learning</th>
<th>Inference</th>
<th>Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cross-entropy</td>
<td>Binary Adv loss</td>
<td>MLE</td>
<td>Teacher-forcing</td>
<td>LSTM RNN</td>
</tr>
<tr>
<td>Matching Adv loss</td>
<td>Adv + RL</td>
<td>Adversarial</td>
<td>Gumbel-softmax</td>
<td>Attention RNN</td>
</tr>
<tr>
<td>PG loss</td>
<td>Structured super.</td>
<td>Reinforcement</td>
<td>Sample</td>
<td>Transformer</td>
</tr>
<tr>
<td>PG loss + baseline</td>
<td>Reward-aug.</td>
<td>Adv + RL</td>
<td>Greedy</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>encoder</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>classifier</td>
</tr>
</tbody>
</table>

\[ \min_\theta \mathcal{L}(\theta, D) + \Omega(\theta) \]
Constraint (1): Conventional Constraints

- Many choices for get different statistical properties:
  - Normality, Sparsity, KL, sum, ...
Constraint (2): Structured Knowledge

- Structured knowledge as constraints

Sentiment classification:
- ``Food was good, but the service was very disappointing.``

Logic rule: \[ f(x = \text{sentence}, y = \text{sentiment}) = \text{truth value} \]
- Sentence \( x \) with structure A-but-B \( \Rightarrow \) sentiment of B dominates
Constraint (2): Structured Knowledge

- Structured knowledge as constraints

Human Image Generation

\[ f(y = \text{generated}, o = \text{ground truth}) = \text{match score} \]

**Diagram:**
- Source image \(p\)
- Pose
- Target pose
- True target
- Generated image
- Human part parser
- Structured consistency
- Minimize \( L(\theta, D) + \Omega(\theta) \)
Constraint (2): Structured Knowledge

- Structured knowledge as constraints
  - Constraint function: \( f(x, y, o) \in \mathbb{R} \)
  - Model: \( p_\theta(y|x) \)
  - Variational Knowledge Regularization [Hu et al., 2016, 2018; Ganchev et al. 2010]

\[
\mathcal{L}_{VKR}(q, \theta) = \mathbb{E}_{q(y|x)}[ f(x, y, o) ] - \text{KL}(q(y|x) \| p_\theta(y|x))
\]

- Learning procedure: at iteration \( n \)
  \[
  q^{n+1}(y|x) \propto p_\theta^n(y|x) \exp\{f(x, y, o)\}
  \]
  \[
  \theta^{n+1} = \arg\max_\theta \mathbb{E}_{q^{n+1}(y|x)}[\log p_\theta(y|x)]
  \]
### ML Components

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Loss</th>
<th>Learning</th>
<th>Inference</th>
<th>Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1 / L2</td>
<td>Cross-entropy</td>
<td>MLE</td>
<td>Teacher-forcing</td>
<td>decoder</td>
</tr>
<tr>
<td>Logical</td>
<td>Binary Adv loss</td>
<td>Adversarial</td>
<td>Gumbel-softmax</td>
<td>LSTM RNN</td>
</tr>
<tr>
<td>Structured</td>
<td>Matching Adv loss</td>
<td>Reinforcement</td>
<td>Sample</td>
<td>Attention RNN</td>
</tr>
<tr>
<td></td>
<td>PG loss</td>
<td>Adv + RL</td>
<td>Greedy</td>
<td>Transformer</td>
</tr>
<tr>
<td></td>
<td>PG loss + baseline</td>
<td>Structured super.</td>
<td>Beam-search</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Reward-aug.</td>
<td>Top-k sample</td>
<td></td>
</tr>
</tbody>
</table>

\[
\min_{\theta} \mathcal{L}(\theta, D) + \Omega(\theta)
\]
Holistic View

**Constraint**
- L1 / L2
- Logical
- Structured
  - ...

**Loss**
- Cross-entropy
- Binary Adv loss
- Matching Adv loss
- PG loss
- PG loss + baseline
  - ...

**Learning**
- MLE
- Adversarial
- Reinforcement
- Adv + RL
- Structured super.
  - Reward-aug.
  - ...

**Inference**
- Teacher-forcing
- Gumbel-softmax
- Sample
- Greedy
- Beam-search
- Top-k sample
  - ...

**Architecture**
- decoder
- LSTM RNN
- Attention RNN
- Transformer
  - ...
- encoder
- classifier
  - ...

\[
\min_{\theta} \mathcal{L}(\theta, D) + \Omega(\theta)
\]
Holistic View

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Loss</th>
<th>Learning</th>
<th>Inference</th>
<th>Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1 / L2</td>
<td>Cross-entropy</td>
<td>MLE</td>
<td>Teacher-forcing</td>
<td>decoder</td>
</tr>
<tr>
<td>Logical</td>
<td>Binary Adv loss</td>
<td>Adversarial</td>
<td>Gumbel-softmax</td>
<td>LSTM RNN</td>
</tr>
<tr>
<td>Structured</td>
<td>Matching Adv loss</td>
<td>Reinforcement</td>
<td>Sample</td>
<td>Attention RNN</td>
</tr>
<tr>
<td></td>
<td>PG loss</td>
<td>Adv + RL</td>
<td>Greedy</td>
<td>Transformer</td>
</tr>
<tr>
<td></td>
<td>PG loss + baseline</td>
<td>Structured super.</td>
<td>Beam-search</td>
<td>encoder</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Reward-aug.</td>
<td>Top-k sample</td>
<td>classifier</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Holistic View

Constraint
- L1 / L2
- Logical
- Structured
  ...

Loss
- Cross-entropy
- Binary Adv loss
- Matching Adv loss
- PG loss
- PG loss + baseline
  ...

Learning
- MLE
- Adversarial
- Reinforcement
- Adv + RL
- Structured super.
  ...

Inference
- Teacher-forcing
- Gumbel-softmax
- Sample
- Greedy
- Beam-search
- Top-k sample
  ...

Architecture
- decoder
- LSTM RNN
- Attention RNN
- Transformer
  ...
- encoder
- classifier
  ...

min$_\theta$ $\mathcal{L}(\theta, D) + \Omega(\theta)$

Petuum
Holistic View

Constraint
- L1 / L2
- Logical
- Structured

Loss
- Cross-entropy
- Binary Adv loss
- Matching Adv loss
- PG loss
- PG loss + baseline

Learning
- MLE
- Adversarial
- Reinforcement
- Structured super.
- Reward-aug.

Inference
- Teacher-forcing
- Gumbel-softmax
- Sample
- Greedy
- Beam-search
- Top-k sample

Architecture
- decoder
- LSTM RNN
- Attention RNN
- Transformer
- encoder
- classifier

Min $L(\theta, D) + \Omega(\theta)$

-called as subroutines
-uniform interfaces
Composable ML – Take-Home Message

- **Composable ML**
  - Basic “musical notes” for complex ML systems
  - Structure/rules for combining notes into “chords”/“rhythms” …
  - … and chords/rhythms into compositions
  - (or just think of it as Lego for ML)

- **Today** – Symbolic programming via Petuum Texar open source

- **Future** – Drag and Play graphical UI
Compose your ML applications like playing building blocks
Decomposing Machine Learning

\[ \min_{\theta} \mathcal{L}(\theta, D) + \Omega(\theta) \]

- learning procedure
- loss
- model architecture/inference procedure
- experience (data)
- constraint
Expert’s Intellectual “View” of Composable ML
Texar Stack – Operationalized “View” of Composable ML
Module Catalog in Texar

**Model architecture**

- **Encoder**
  - UnidirectionalRNNEncoder
  - BidirectionalRNNEncoder
  - HierarchicalRNNEncoder
  - ConvEncoder
  - TransformerEncoder
- **Decoder**
  - BasicRNNDetector
  - AttentionRNNDetector
    - BahdanauAttn
    - LuongAttn
    - MonotonicAttn
  - TransformerDecoder
  - Greedy/Sample/BeamSearch/GumbelSoftmax/... Decoding
- **Embedder**
  - WordEmbedder (one-hot / soft)
  - PositionEmbedder
    - Parametrized Sinusoids
- **Classifier/Discriminator**
  - RNNClassifier
  - ConvClassifier
  - HierarchicalClassifier
- **Connector**
  - MLPTransformer
  - Stochastic
  - ReparameterizedStochastic
  - Concat
  - Forward

**Model loss**

- **Loss**
  - MLE Loss
  - (Sequence) Cross-entropy
  - Adversarial Loss
  - Binary Adversarial Loss
  - Rewards

- **RL Agent**
  - Seq RL Agent
    - Seq Policy Gradient Agent
  - Episodic RL Agent
    - Policy Gradient Agent
    - DQN Agent
    - Actor-critic Agent
  - GumbelSoftmax/... Decoding

- **Optimization**
  - Optimizer
    - Adam/SGD/...
  - Learning Rate Decay
    - Piecewise/Exp/...

- **Data**
  - MonoText
  - PairedText
  - MultiAligned
  - Dialog
  - Numerical
  - ...
Texar Highlights

Modularized
Assembles any complex model like playing building blocks

Versatile
Supports a large variety of models/algorithms/applications ...

Extensible
Allows to plug in any customized or external modules
Running Example: Machine Translation

I like this movie.
Lovely and poignant
Insanely hilarious!

Ich mag diesen film.
Schön und ergreifend
Wahnsinnig witzig!

source.dat

target.dat

cleaning
tokenizing
vocabulary
truncation

raw data
clean data
Running Example: Machine Translation - Data Preparation

- Input: a source sentence:
  
  I like this movie.

- Output: a target sentence:
  
  Ich mag diesen film.

- Dataset:

<table>
<thead>
<tr>
<th>source.txt</th>
<th>target.txt</th>
<th>vocab.txt</th>
</tr>
</thead>
<tbody>
<tr>
<td>I like this movie.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lovely and poignant</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Insanely hilarious!</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ich mag diesen film.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Schön und ergreifend</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wahnsinnig witzig!</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>I</td>
<td></td>
<td></td>
</tr>
<tr>
<td>like</td>
<td></td>
<td></td>
</tr>
<tr>
<td>this</td>
<td></td>
<td></td>
</tr>
<tr>
<td>movie</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ich</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mag</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Running Example: Machine Translation - Programming
# Read data

```python
dataset = PairedTextData(data_hparams)
```

```python
batch = DataIterator(dataset).get_next()
```

# Encode

```python
embedder = WordEmbedder(dataset.vocab.size, hparams=embedder_hparams)
encoder = TransformerEncoder(hparams=encoder_hparams)
```

```python
c_enc_outputs = encoder(embedder(batch['source_text_ids']),
batch['source_length'])
```

# Build decoder

```python
decoder = AttentionRNNDecoder(memory=c_enc_outputs,
hparams=decoder_hparams)
```

# Maximum Likelihood Estimation

```python
outputs, length = decoder(decoding_strategy='teacher-forcing',
inputs=embedder(batch['target_text_ids']),
seq_length=batch['target_length']-1)
```

```python
loss = sequence_sparse_softmax_cross_entropy(labels=batch['target_text_ids'][:, 1:],
logits=outputs.logits, seq_length=length)
```

---

**Running Example: Machine Translation - Programming**

**Data**

```python
# Read data
dataset = PairedTextData(data_hparams)
batch = DataIterator(dataset).get_next()
```

```python
{  
  'batch_size': 64,
  'num_epochs': 10,
  'shuffle': True,
  'source_dataset': {'files': 'source.txt',
                    'vocab_file': 'vocabulary.txt',
                    'max_seq_length': 100,
                    'bos_token': '<BOS>',
                    'eos_token': '<EOS>',
                    'embedding_init': {...}
                     },
  'target_dataset': {'files': 'target.txt',
                     }
}
```
# Read data

dataset = PairedTextData(data_hparams)

# Batch
batch = DataIterator(dataset).get_next()

# Encode
embedder = WordEmbedder(dataset.vocab.size, hparams=embedder_hparams)

encoder = TransformerEncoder(hparams=encoder_hparams)

enc_outputs = encoder(embedder(batch['source_text_ids']), batch['source_length'])

# Decode
decoder = AttentionRNNDecoder(memory=enc_outputs, hparams=decoder_hparams)

# Maximum Likelihood Estimation
outputs, length = decoder(decoding_strategy='teacher-forcing', inputs=embedder(batch['target_text_ids']), seq_length=batch['target_length'] - 1)

# Loss
loss = sequence_sparse_softmax_cross_entropy(labels=batch['target_text_ids'][:, 1:], logits=outputs.logits, seq_length=length)
# Running Example: Machine Translation - Programming

```python
1 # Read data
2 dataset = PairedTextData(data_hparams)
3 batch = DataIterator(dataset).get_next()
4 # Encode
5 embedder = WordEmbedder(dataset.vocab.size, hparams=embedder_hparams)
6 encoder = TransformerEncoder(hparams=encoder_hparams)
7 enc_outputs = encoder(embedder(batch['source_text_ids']), batch['source_length'])
8 # Build decoder
9 decoder = AttentionRNNDecoder(memory=enc_outputs, hparams=decoder_hparams)
10 # Teacher-forcing decoding
11 outputs, length, _ = decoder(decoding_strategy='teacher-forcing', inputs=embedder(batch['target_text_ids']), seq_length=batch['target_length'] - 1)
12 # Cross-entropy loss
13 loss = sequence_sparse_softmax_cross_entropy(labels=batch['target_text_ids'][:,1:], logits=outputs.logits, seq_length=length)
```

```
{ embedding_dim
dropout
initialization
...}
```
Running Example: Machine Translation - Programming

### Data
1. # Read data
2. dataset = PairedTextData(data_hparams)
3. batch = DataIterator(dataset).get_next()

### Architecture & Inference
4. # Encode
5. embedder = WordEmbedder(dataset.vocab.size, hparams=embedder_hparams)
6. encoder = TransformerEncoder(hparams=encoder_hparams)

```
{  #blocks
  #heads
  hidden dim
  output dim
  dropout
  initialization
  }
```
Running Example: Machine Translation - Programming

Data

1. # Read data
2. dataset = PairedTextData(data_hparams)
3. batch = DataIterator(dataset).get_next()

Architecture & Inference

4. # Encode
5. embedder = WordEmbedder(dataset.vocab.size, hparams=embedder_hparams)
6. encoder = TransformerEncoder(hparams=encoder_hparams)
7. enc_outputs = encoder(embedder(batch['source_text_ids']),
   batch['source_length'])
Running Example: Machine Translation - Programming

1. # Read data
2. dataset = PairedTextData(data_hparams)
3. batch = DataIterator(dataset).get_next()
4. # Encode
5. embedder = WordEmbedder(dataset.vocab.size, hparams=embedder_hparams)
6. encoder = TransformerEncoder(hparams=encoder_hparams)
7. enc_outputs = encoder(embedder(batch['source_text_ids']),
                        batch['source_length'])
8. # Build decoder
9. decoder = AttentionRNNDecoder(memory=enc_outputs,
                                hparams=decoder_hparams)
10. # Maximum Likelihood Estimation
11. ## Teacher-forcing decoding
12. outputs, length, _ = decoder(decoding_strategy='teacher-forcing',
                               inputs=embedder(batch['target_text_ids']),
                               seq_length=batch['target_length']-1)
13. # Running Example: Machine Translation - Programming
14. Data
15. Architecture & Inference
16. Learning
17. Running Example: Machine Translation - Programming
# Read data
```python
dataset = PairedTextData(data_hparams)
batches = DataIterator(dataset).get_next()
```

# Encode
```python
eMBEDDER = WordEmbedder(dataset.vocab.size, hparams=embedder_hparams)
encoder = TransformerEncoder(hparams=encoder_hparams)
enc_outputs = encoder(embedder(batch['source_text_ids']),
batch['source_length'])
```

# Build decoder
```python
decoder = AttentionRNNDecoder(memory=enc_outputs,
hparams=decoder_hparams)
```

# Maximum Likelihood Estimation
```python
## Teacher-forcing decoding
outputs, length, _ = decoder(decoding_strategy='teacher-forcing',
inputs=embedder(batch['target_text_ids']),
seq_length=batch['target_length']-1)
## Cross-entropy loss
loss = sequence_sparse_softmax_cross_entropy(
  labels=batch['target_text_ids'][:,1:],
  logits=outputs.logits,
  seq_length=length)
```
DEMO
Example (cont’d): Machine Translation

```
# Read data
dataset = PairedTextData(data_hparams)
batch = DataIterator(dataset).get_next()

# Encode
embedder = WordEmbedder(dataset.vocab.size, hparams=embedder_hparams)
encoder = TransformerEncoder(hparams=encoder_hparams)
enc_outputs = encoder(embedder(batch['source_text_ids']),
                      batch['source_length'])

# Build decoder
decoder = AttentionRNNDegoder(memory=enc_outputs,
                               hparams=decoder_hparams)

# Maximum Likelihood Estimation
## Teacher-forcing decoding
outputs, length, _ = decoder(decoding_strategy='teacher-forcing',
                            inputs=embedder(batch['target_text_ids']),
                            seq_length=batch['target_length']-1)

## Cross-entropy loss
loss = sequence_sparse_softmax_cross_entropy(
       labels=batch['target_text_ids'][:,-1:],
       logits=outputs.logits, seq_length=length)
```
Different Learning Algorithms

- Maximum Likelihood Estimation
- Adversarial Learning
- Reinforcement Learning
Switching between Learning Algorithms

# Read data
dataset = PairedTextData(data_hparams)
batch = DataIterator(dataset).get_next()

# Encode
embedder = WordEmbedder(dataset.vocab.size, hparams=embedder_hparams)
encoder = TransformerEncoder(hparams=encoder_hparams)
enc_outputs = encoder(embedder(batch['source_text_ids']),
batch['source_length'])

# Build decoder
decoder = AttentionRNNDecoder(memory=enc_outputs,
hparams=decoder_hparams)

# Maximum Likelihood Estimation
### Teacher-forcing decoding
outputs, length, _ = decoder(decoding_strategy='teacher-forcing',
inputs=embedder(batch['target_text_ids']),
seq_length=batch['target_length']-1)

### Cross-entropy loss
loss = sequence_sparse_softmax_cross_entropy(
lables=batch['target_text_ids'][:,-1], logits=outputs.logits, seq_length=length)
Switching from MLE to Adversarial Learning

- Maximum likelihood

Cross entropy loss

# Teacher-forcing decoding
outputs, length, _ = decoder(decoding_strategy='teacher-forcing',
inputs=embedder(batch['target_text_ids']),
seq_length=batch['target_length']-1)

# Cross-entropy loss
loss = sequence_sparse_softmax_cross_entropy(
    labels=batch['target_text_ids'][:,1:],
    logits=outputs.logits,
    seq_length=length)
Switching from MLE to Adversarial Learning

- Maximum likelihood
  
  Cross entropy loss

- Adversarial learning

# Teacher-forcing decoding
outputs, length, _ = decoder(decoding_strategy='teacher-forcing',
inputs=embedder(batch['target_text_ids']),
seq_length=batch['target_length']-1)

# Cross-entropy loss
loss = sequence_sparse_softmax_cross_entropy(
  labels=batch['target_text_ids'][:,1:],
  logits=outputs.logits, seq_length=length)

# Gumbel-softmax decoding
helper = GumbelSoftmaxTrainingHelper(
  start_tokens=[BOS]*batch_size, end_token=EOS, embedding=embedder)
outputs, _, _ = decoder(helper=helper)

discriminator = Conv1DClassifier(hparams=conv_hparams)
# Binary adversarial loss
G_loss, D_loss = binary_adversarial_losses(
  embedder(batch['target_text_ids'][:,1:]),
  embedder(soft_ids=softmax(outputs.logits)),
  discriminator)

(a) Maximum likelihood learning
(b) Adversarial learning
(c) Reinforcement learning
Switching from MLE to Reinforcement Learning

- **Maximum likelihood**
  
  \[ \text{Cross entropy loss} \]

  \[
  y_1 \quad y_2 \quad y_3
  \]

  \[
  \text{Decoder}
  \]

- **Reinforcement learning**

  ![Diagram](image)

  
  \[
  \text{Policy Gradient Agent} \quad \text{data example} \quad y^* \quad \text{sample} \quad \text{BLEU}
  \]

  \[
  \text{Decoder} \quad \text{rewards}
  \]

  \[
  \text{Cross entropy loss}
  \]

  
  ```python
  # Teacher-forcing decoding
  outputs, length, _ = decoder(decoding_strategy='teacher-forcing',
  inputs=embedder(batch['target_text_ids']),
  seq_length=batch['target_length']-1)

  # Cross-entropy loss
  loss = sequence_sparse_softmax_cross_entropy(
  labels=batch['target_text_ids'][1:], logis=outputs.logits, seq_length=length)

  # Gumbel-softmax decoding
  helper = GumbelSoftmaxTrainingHelper(
  start_tokens=[BOS]*batch_size, end_token=EOS, embedding=embedder)
  outputs, _, _ = decoder(helper=helper)

  discriminator = Conv1DClassifier(hparams=conv_hparams)
  # Binary adversarial loss
  G_loss, D_loss = binary_adversarial_losses(
  embedder(batch['target_text_ids'][1:]),
  embedder(soft_ids=softmax(outputs.logits)),
  discriminator)

  # Random sample decoding
  outputs, length, _ = decoder(decoding_strategy='random_sample',
  start_tokens=[BOS]*batch_size, end_token=EOS,
  embedding=embedder)

  # Policy gradient agent for learning
  agent = SeqPGAgent(
  samples=outputs.sample_id, logis=outputs.logits, seq_length=length)

  for _ in range(STEPS):
    samples = agent.get_samples()
    rewards = BLEU(batch['target_text_ids'], samples)  # Reward
    agent.observe(rewards)
  ```
# Read data

dataset = PairedTextData(data_hparams)
batch = DataIterator(dataset).get_next()

# Encode

embedder = WordEmbedder(dataset.vocab.size, hparams=embedder_hparams)
encoder = TransformerEncoder(hparams=encoder_hparams)
enc_outputs = encoder(embedder(batch['source_text_ids']),
                      batch['source_length'])

# Build decoder

decoder = AttentionRNNDecoder(memory=enc_outputs,
                               hparams=decoder_hparams)

# Maximum Likelihood Estimation

## Teacher-forcing decoding

outputs, length, _ = decoder(decoding_strategy='teacher-forcing',
                             inputs=embedder(batch['target_text_ids']),
                             seq_length=batch['target_length']-1)

## Cross-entropy loss

loss = sequence_sparse_softmax_cross_entropy(
       labels=batch['target_text_ids'][:,1:],
       logits=outputs.logits,
       seq_length=length)

Summary of MT in Texar

- Highly modularized programming
  - Data, architecture, loss, inference, learning, ...
  - Intuitive conceptual-level APIs

- Easy switch between learning algorithms
  - Plug in & out modules
  - No changes to irrelevant parts
Other Features (I): Support of TensorFlow and PyTorch

- Texar is built upon TF and PyTorch
  - Texar-TF & Texar-PyTorch: mostly the same interfaces!
  - Higher-level intuitive APIs without loss of flexibility
  - Lots of ML components ready to use

- Combine the best design of TF and PyTorch
  - TF:
    - Easy and efficient data processing APIs
    - Excellent factorization of ML modules
    - Turnkey model training processor
  - PyTorch:
    - Intuitive programming interfaces
    - Transparent variable scope and sharing to users
Other Features (I): Support of TensorFlow and PyTorch

- Texar-TF
  - Texar provides the same comprehensive supports of ML functionalities
  - Fully compatible with native TF & PyTorch APIs
    - Get all other useful features of TF or PyTorch with Texar-TF & Texar-PyTorch

- Texar-PyTorch
  - Comprehensive Machine Learning functionalities
  - Static programming
  - TF Serving
  - Mobile support
  - Dynamic programming
  - Easier editing & debugging
Other Features (II): SOTA Pretrained Models

Bert [Devlin et al., 2018]
Other Features (II): SOTA Pretrained Models

**Bert** [Devlin et al., 2018]

```python
model = BertEncoder()
features = model(input_ids, input_length, segment_ids)
```

**contextual embedding**

**input sentence**

```
[CLS] Help Prince Mayuko
```
Other Features (II): SOTA Pretrained Models

**Bert** [Devlin et al., 2018]

```python
model = BertClassifier(hparams={'clas_strategy': 'cls_time'})
logits, preds = model(input_ids, input_length, segment_ids)
```

- **sentence class**
  - 0/1
  - sequence classification

- **input sentence**
  - [CLS] Help Prince Mayuko
Bert \cite{devlin2018bert}

```python
model = BertClassifier(hparams={'clas_strategy': 'all_time'})
logits, preds = model(input_ids, input_length, segment_ids)
```

**Input sentence:**

```
[CLS] Help Prince Mayuko
```

**Sequence labeling:**

```
O V N N A
```

**Label per step:**

- `O`: Others
- `V`: Various
- `N`: Name
- `A`: Action

**Diagram:**

- Encoder layers:
  - Layer 1
  - Layer 2
  - Layer 12

- Input sentence:
  - [CLS]
  - Help
  - Prince
  - Mayuko

**Proprietary & Confidential | 92**
Spectrum of Existing Tools

Interfaces at multiple abstraction levels
- Simplified APIs for common functionalities
- Advanced APIs for advanced functionalities and customizability

Fixed Structure
- Limited composability

Modularized, Composable
Spectrum of Existing Tools

- **Texar**
  - Composable ML system – create ML models of complex design from ML first principles

- TensorFlow, PyTorch
  - Symbolic languages for low-level development of ML models

- BERT, GPT2, …
  - ML models used to model human languages; also can refer to their downloadable pre-trained weights (ready-to-run)

- Amazon SageMaker, Google AutoML, Microsoft Azure ML
  - Commercial products for training, inference and management of pre-built ML models; may perform algorithm-driven model parameter or architecture tuning
Applications of Texar

Many products built on Texar

- FORTE – templates for larger complex NLP applications
- Chest X-Ray report writer
- Medical Registry report writer
- ICD coding system
- Financial knowledge base builder
- Financial summary/report writer
- Multi-Lingual Cognitive Chat Bots
  - For Call Center Support
  - For Retail In-Store Assistance
Example: Chest X-Ray Report Writer

Visual Feature Extraction

Lesion Label Classification

normal
granulama
opacity
calciﬁed
...

Semantic Feature Extraction

Co-attention Feature Aggregation

Hidden Topic Generation

Hierarchical Text Generation

Medical Report

Raw Data Enrichment

Post-processing

Text Analysis

Machine Translation

Report Generation
Example: Chest X-Ray Report Writer + Translation
Texar Resources

- Website: https://asyml.io
- GitHub (TF version): https://github.com/asyml/texar
- GitHub (PyTorch version): https://github.com/asyml/texar-pytorch
- Examples: https://github.com/asyml/texar/blob/master/examples
- Documentation: https://texar.readthedocs.io/
- Blog: https://medium.com/@texar
Scalable AI Infrastructure

Scaling out AI applications – easier, faster, cheaper
What is needed to scale AI?

**Inter-Task Interfacing**
Application Templates
Assemble many tasks with uniform interfaces to form larger, complex applications

**High Speed & Scalability**
Made-to-order, just-in-time distributed training strategies

**Save Cost & Time**
Dynamically reassign CPU & GPU resources for fastest/cheapest workload completion time

**FORTE**
- Abstract representation
- All NLP tasks support
- Module sharing/reuse

**ARION**
- Resource Optimizer
- Optimal Resource Mapping
- Dynamic Re-Allocator

**ESPER**
- Perf Model 1: 1 GPU
- Perf Model 2: 3 GPU
- Perf Model 3: 4 GPU
- Re-balance Resources
Forte – Flexible Scaffold for Larger NLP Applications

Built upon Texar

Tag remover, tokenizer, parsing, etc.

I like this movie.
Lovely and poignant
Insanely hilarious!

Ich mag diesen film.
Schön und ergreifend
Wahnsinnig witzig!

More NLP analysis: sentiment, parsing, discourse, etc.
Can be done on both languages.

Writer and Consumer
Arion – High Speed, Scalability for ML Model Training
Tailor-made distributed training strategies for hard-to-scale ML models

Pain Points

- Existing systems are mostly specialized and implemented based on 1 (probably at most 2) of those system architectures.

- Distribution performance/scalability is only achieved under certain condition, for a narrow family of models where the system is specialized.

Solution

Arion expresses models and training algorithms as an intermediate representation (IR) with distributed execution semantics

Arion rewrites IR and generates tailor-made distributed execution strategy using multiple system architectures (e.g. PS + SFB)

Value: Model training – from un-scalable (speed $\propto 1$) to highly-scalable (speed $\propto n$ machines)

Arion is 1.1-1.5x faster vs Parallax (SOTA) | 2-5x faster vs TF | 4-9x faster vs Horovod

Proprietary & Confidential | 102
Esper – Save Compute Costs and Time

*Dynamically reassign CPU/GPU to the most impactful work*

**Pain Points**

Teams of users

Limited resources

Many tasks and trials

How to best share limited resources between many ML jobs?
Can a software infrastructure automate resource allocation for ML?

Does fairness-based resource allocation work well for ML? Consider:

Scalability of job 1 (E.g. LSTM)

Scalability of job 2 (E.g. CNN)

Is it best to give the same amount of resources to these two jobs? NO.

**Solution**

Esper learns the performance model for each model training job

Esper uses performance models to optimize (in near-real-time) CPUs/GPUs needed by each running job on the cluster/cloud

**Value:** Finish training *multiple* models faster (and cheaper!)

Esper is 2-4x faster vs Google Kubeflow
What is needed to productize AI?

**Drag & Play UI**
- Manage, Experiment, Version
- Systems, Tools, Infra, Visualizations, Dashboards

**Programming Language InterOp**
- Integrate & Manage different code (C++, Python, Scala, TF, PyT, Spark, …)
  - all into the same ML app

**Infra Management**
- Authentication/Security
- Containers
- High-Availability
- Distributed & Cloud Storage
- Cloud & On-Premise Deployments

---

**COMPOSER**

**FUGUE**

**CHIRON**

---

[Petuum Logo]

---
AI (Civil) Engineering with Petuum

Industry Agnostic

- Robo-radiologist
- Insurance Auto-Report
- Virtual EA
- Smart Expense Reports
- Smart Catalog
- Robot Store Staff

Building AI Like Lego

- Completed software
- 10% White-Glove Assembly
- 90% Completed building blocks

AI With No Tears

Petuum OS

Proprietary & Confidential | 105
Petuum’s Mission

**Industrialize AI technology**
– turning it from black-box artisanship into standardized engineering process

**Transform enterprises across industries**
– turning them into owners, builders, and informed users of AI

Sustainable & Standardized Building Blocks

One foundation for your current and future AI-building needs

A 2018 WEF Technology Pioneer Winner
WE ARE HARRYING:

ML engineer/manager
Software engineer/manager
System engineer
UI engineer
...

Proprietary & Confidential | 107
Thank You