TF-Ranking

Neural Learning to Rank using TensorFlow

ICTIR 2019

Rama Kumar Pasumarthi
Sebastian Bruch
Michael Bendersky
Xuanhui Wang

Google Research
Talk Outline

1. Motivation

2. Neural Networks for Learning-to-Rank

3. Introduction to Deep Learning and TensorFlow

4. TF-Ranking Library Overview

5. Empirical Results

6. Hands-on Tutorial
Motivation
Learning to Rank
Applications

- Search
- Recommendation
- Dialogue systems
- Question Answering
General Problem Statement

Problem Learning a scoring function $f^*$ to sort a list of examples

- Input: List of examples (with Context)
- Output: Scoring function $f^*$ that produces the most optimal example ordering
  - Can be parameterized by linear functions, SVM, GBDTs, Neural Networks

Formally

$$
\psi = (x, y) \in \mathcal{X}^n \times \mathbb{R}^n
$$

Training sample with relevance labels

$$
\mathcal{L}(f) = \frac{1}{|\Psi|} \sum_{(x, y) \in \Psi} \ell(y, f(x))
$$

Choose $f^*$ to minimize empirical loss
Ranking Metric Optimization

- Ranking metrics are *piecewise constant*
- Cannot be directly optimized with gradient descent
- Therefore, various proxy losses were proposed
Pointwise LTR methods

- Documents are considered independently of each other
- Some examples: \textit{ordinal regression}, \textit{classification}, \textit{GBRTs}

\[
\begin{align*}
& f(A) \Rightarrow P(A \text{ is Relevant}) \\
& f(B) \Rightarrow P(B \text{ is Relevant}) \\
& f(C) \Rightarrow P(C \text{ is Relevant})
\end{align*}
\]
Pairwise LTR methods

- Document pairs are considered
- Some examples: \textit{RankNet}, \textit{RankSVM}, \textit{RankBoost}
Listwise LTR methods

- Consider the ordering of the entire list
- Some examples: LambdaMART, ApproxNDCG, List{Net, MLE}
Standard LTR setting

- **Handcrafted** features based on query, document and their match scores
  - Web30K has 136 features per document
    - tf-idf scores
    - BM25 scores
    - Inlink counts
    - URL length
    - Page quality
    - ...

- **Human** relevance judgments
  - The largest datasets have tens of thousands of labeled examples
    - Web30K, Istella, Yahoo! ~30K queries
Current State-of-the-Art in LTR

The best LambdaMART implementation is still the most competitive on public LTR datasets

"Revisiting Approximate Metric Optimization in the Age of Deep Neural Networks"
Bruch et al., SIGIR 2019
Neural Networks for Learning-to-Rank
Why Neural Networks for Ranking?

- Are complementary to standard LTR methods, *not a direct replacement*
  - Can be ensembled with GBDTs for further performance gains

"Combining Decision Trees and Neural Networks for Learning-to-Rank in Personal Search"
Pan et al., KDD 2019
Why Neural Networks for Ranking?

- Allow learning feature representations **directly from the data**
  - Directly employ query and document text instead of relying on handcrafted features
  - NNs are clearly outperforming standard LTR on short text ranking tasks
Neural models for IR

- Neural IR is increasingly popular
- Major focus is on neural matching models
- Less research on neural ranking models

Figure 1.1: The percentage of neural IR papers at the ACM SIGIR conference—as determined by a manual inspection of the papers—shows a clear trend in the growing popularity of the field.

Figure source: "An Introduction to Neural Information Retrieval" Bhaskar et al., FnTIR (2018)
DSSM model

| Posterior probability computed by softmax |
| Relevance measured by cosine similarity |

<table>
<thead>
<tr>
<th>Semantic feature</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>( { W_4, b_4 } )</td>
<td>128</td>
</tr>
<tr>
<td>( { W_3, b_3 } )</td>
<td>128</td>
</tr>
<tr>
<td>( { W_2, b_2 } )</td>
<td>128</td>
</tr>
</tbody>
</table>

| Multi-layer non-linear projection |

<table>
<thead>
<tr>
<th>Word Hashing</th>
<th>( l_1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( W_1 )</td>
<td>30k</td>
</tr>
<tr>
<td>( W_2 )</td>
<td>300</td>
</tr>
<tr>
<td>( W_3 )</td>
<td>300</td>
</tr>
<tr>
<td>( W_4 )</td>
<td>300</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Term Vector</th>
<th>( x )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( W_1 )</td>
<td>500k</td>
</tr>
<tr>
<td>( W_2 )</td>
<td>500k</td>
</tr>
<tr>
<td>( W_3 )</td>
<td>500k</td>
</tr>
<tr>
<td>( W_4 )</td>
<td>500k</td>
</tr>
</tbody>
</table>

\[ R(Q, D_i) \]

\[ P(D_1|Q) \]

\[ P(D_2|Q) \]

\[ P(D_n|Q) \]

"Learning Deep Structured Semantic Models for Web Search using Clickthrough Data"
Huang et al., CIKM 2013
Deep Listwise Context Model (DLCM)

"Learning a Deep Listwise Context Model for Ranking Refinement"
Ai et al., SIGIR 2018
Neural Ranking with Weak Supervision

(a) Score model
(b) Rank model
(c) RankProb model

"Neural Ranking Models with Weak Supervision"
Dehghani et al., SIGIR 2017
Groupwise Multivariate Scoring Functions

\[ f(x) \]
\[ \sum \quad \sum \quad \sum \]
\[ g([x_3, x_2]) \]

\[ [x_1, x_2], [x_1, x_3], [x_2, x_1], [x_2, x_3], [x_3, x_1], [x_3, x_2] \]

\[ x \{ x_1, x_2, x_3 \} \]

"Learning Groupwise Multivariate Scoring Functions Using Deep Neural Networks"

Ai et al., ICTIR 2019
Introduction to Deep Learning and TensorFlow

Many materials are from Lex Friedman’s MIT Deep Learning Course
https://www.dropbox.com/s/c0g3sc1shi63x3q/deep_learningBasics.pdf
Deep Neural Network

Simple Neural Network

Deep Learning Neural Network

- Red: Input Layer
- Yellow: Hidden Layer
- Blue: Output Layer
Neuron

1. weigh
2. sum up
3. activate
Activation Function $\rightarrow$ Non-Linearity

**Sigmoid**
- Vanishing gradients
- Not zero centered

**Tanh**
- Vanishing gradients

**ReLU**
- Not zero centered
Loss Function

Mean Squared Error

\[ MSE = \frac{1}{N} \sum (t_i - s_i)^2 \]

Cross Entropy Loss

\[ CE = - \sum_i^C t_i \log(s_i) \]
Backpropagation

\[
\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial w_{ij}} = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial net_j} \frac{\partial net_j}{\partial w_{ij}}
\]

Task: Update the **weights** and **biases** to decrease loss function
TensorFlow: A Deep Learning Framework

- Computation is a dataflow graph
  - Node: tf.Operations / ops
  - Edge: tf.Tensors
- Declarative language to build a graph
- Symbolic differentiation
Computation is a dataflow graph

Graph of Nodes, also called Operations or ops.

- biases
- weights
- examples
- labels

MatMul → Add → Relu → Xent
Computation is a dataflow graph

Edges are N-dimensional arrays: Tensors
Declarative Language to Build a Graph

```python
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data

mnist = input_data.read_data_sets('MNIST_data', one_hot=True)
x = tf.placeholder("float", shape=[None, 784])
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))
y = tf.nn.softmax(tf.matmul(x, W) + b)
```
Computation is a dataflow graph with state

'Biases' is a variable

Some ops compute gradients

-= updates biases

biases

... Add ...

learning rate

Mul

-=

...
Symbolic Differentiation

- Automatically add ops to calculate symbolic gradients of variables w.r.t. loss function.
- Apply these gradients with an optimization algorithm

```python
y_ = tf.placeholder(tf.float32, [None, 10])
cross_entropy = -tf.reduce_sum(y_ * tf.log(y))
opt = tf.train.GradientDescentOptimizer(0.01)
train_op = opt.minimize(cross_entropy)
```
Define graph and then execute it repeatedly

- Launch the graph and run the training ops in a loop

```python
init = tf.initialize_all_variables()
sess = tf.Session()
sess.run(init)
for i in range(1000):
    batch_xs, batch_ys = mnist.train.next_batch(100)
    sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})
```
TensorFlow Estimator API

High-Level TensorFlow APIs
- Estimators

Mid-Level TensorFlow APIs
- Layers
- Datasets
- Metrics

Low-level TensorFlow APIs
- Python

TensorFlow Kernel
- TensorFlow Distributed Execution Engine

Languages:
- C++
- Java
- Go
TF-Ranking Library Overview
Challenges for LTR in TensorFlow

- **Data representation**
  - How to represent a ranked list of varying size
  - tf.Example is not suitable for a ranked list
  - tf.Tensor is not friendly for varying size

- **Losses & Metrics**
  - No built-in ranking losses/metrics in TensorFlow
  - Implemented based on Tensors/Ops

- **Serving**
  - For some training modes (e.g., with ranked lists of varying size), there may be a training/serving discrepancy
ExampleInExample Format

Each q, d is a tf.Example and serialized as a string
- EIE is tf.Example with 2 features:
  - “serialized_context”: q
  - “serialized_examples”: [d₁, d₂, …]
Internal Representation: Tensor

- Tensor: multi-dim array for a batch of queries
  - [batch_size, list_size, ...]
  - [num_query, max_num_doc, ...]
- Padding is used but ignored in TF-Ranking computation

```
[relevant, not-relevant, very-relevant],
[not-relevant, relevant, padding-doc],
```

Shape: [2, 3]
**Supported Components**

- Supports pointwise/pairwise/listwise losses
- Supports popular ranking metrics
  - Mean Reciprocal Rank (MRR)
  - Normalized Discounted Cumulative Gain (NDCG)
- Supports multivariate scoring functions
- Supports unbiased learning-to-rank
- Supports sparse/embedding features
Supported Metrics

Mean Reciprocal Rank

$$MRR(\pi, y) = \mathbb{E}\left[ \frac{1}{\min_j \{y_{\pi^{-1}(j)} > 0\}} \right]$$

Average Relevance Position

$$ARP(\pi, y) = \mathbb{E}\left[ \frac{\sum_{j=1}^{n} y_j \pi(j)}{\sum_{j=1}^{n} y_j} \right]$$

Discounted Cumulative Gain

$$DCG(\pi, y) = \mathbb{E}\left[ \sum_{j=1}^{n} \frac{2y_j - 1}{\log_2(1 + \pi(j))} \right]$$
Supported Scoring Functions

- **Univariate** - scoring function $f(x)$ scores each document separately (most existing LTR methods)

- **Bivariate** - scoring function $f(x_1, x_2)$ scores a pair of documents

- **Multivariate** - scoring functions $f(x_1, ..., x_m)$ jointly scores a group of $m$ documents
Supported Loss Examples (Binary Labels)

(Pointwise) Sigmoid Cross Entropy

\[
\hat{\ell}(\mathbf{y}, \mathbf{\hat{y}}) = - \sum_{j=1}^{n} y_j \log(p_j) + (1 - y_j) \log(1 - p_j)
\]

(Pairwise) Logistic Loss

\[
\hat{\ell}(\mathbf{y}, \mathbf{\hat{y}}) = \sum_{j=1}^{n} \sum_{k=1}^{n} \mathbb{I}(y_j > y_k) \log(1 + \exp(\hat{y}_k - \hat{y}_j))
\]

(Listwise) Softmax Loss (aka ListNET)

\[
\hat{\ell}(\mathbf{y}, \mathbf{\hat{y}}) = - \sum_{j=1}^{n} y_j \log\left(\frac{\exp(\hat{y}_j)}{\sum_{j=1}^{n} \exp(\hat{y}_j)}\right)
\]

"An Analysis of the Softmax Cross Entropy Loss for Learning-to-Rank with Binary Relevance"
Bruch et al., ICTIR 2019
ApproxNDCG - Ranking Metric Approximation

\[ DCG(\pi_f, y) = \sum_{j=1}^{n} \frac{2y_j - 1}{\log_2(1 + \pi_f(j))} \]

\[ \pi_f(i) \equiv 1 + \sum_{j \neq i} \mathbb{I}_{f(x)|i < f(x)|j} \]

\[ \mathbb{I}_{s < t} = \mathbb{I}_{t - s > 0} \approx \sigma(t - s) \equiv \frac{1}{1 + e^{-\alpha(t-s)}} \]

"A general approximation framework for direct optimization of information retrieval measures"
Qin et al., Information Retrieval, 2010

"Revisiting Approximate Metric Optimization in the Age of Deep Neural Networks"
Bruch et al., SIGIR 2019
TF-Ranking Ecosystem

- Feature Transforms
- Scoring Function
- Ranking Head
- datasets
- losses
- metrics
- tf.data
- Layers
- Feature Columns
- Model Builder
- Ranking Building Blocks
- TensorFlow Core
- Python Ops
- C++ Ops
- TensorFlow Distributed Execution Engine
- CPU
- GPU
- Android
- iOS
- ...
TF-Ranking Architecture

Training

Data

Input Reader
input_fn

Feature Transformation
transform_fn

Scoring Function
score_fn

Scores

Ranking Head

Loss Metrics

Loss Builder
make_loss_fn

Metrics Builder
make_metrics_fn

metric keys

loss keys

Labels

Serving

raw data

Serving receiver
input_fn

Feature Transformation
transform_fn

Features

Scores

Scoring Function
score_fn
Empirical Results
## Datasets

<table>
<thead>
<tr>
<th>Dataset</th>
<th># queries</th>
<th>Access</th>
<th>Type</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSLR-Web30k</td>
<td>~30K</td>
<td>Public</td>
<td>Search</td>
<td>dense</td>
</tr>
<tr>
<td>MS-Marco</td>
<td>~800K</td>
<td>Public</td>
<td>Q&amp;A</td>
<td>sparse</td>
</tr>
<tr>
<td>Quick Access</td>
<td>~30M</td>
<td>Internal</td>
<td>Recommendation</td>
<td>dense</td>
</tr>
<tr>
<td>Gmail Instant Search</td>
<td>~300M</td>
<td>Internal</td>
<td>Search</td>
<td>sparse dense features</td>
</tr>
</tbody>
</table>
Quick Access: Recommendation in Google Drive
## MSLR-Web30k

(a) Comparison w/ other LTR models

<table>
<thead>
<tr>
<th>Model</th>
<th>NDCG@5</th>
</tr>
</thead>
<tbody>
<tr>
<td>RankNet_{RankLib}</td>
<td>32.28</td>
</tr>
<tr>
<td>RankSVM_{RankLib}</td>
<td>33.74</td>
</tr>
<tr>
<td>MART_{RankLib}</td>
<td>43.54</td>
</tr>
<tr>
<td>λMART_{RankLib}</td>
<td>44.50</td>
</tr>
<tr>
<td>λMART_{LightGBM}</td>
<td>49.20</td>
</tr>
<tr>
<td>Softmax CE w/ GSF(m=32)</td>
<td>44.42</td>
</tr>
<tr>
<td>ApproxNDCG</td>
<td>45.38</td>
</tr>
</tbody>
</table>

(b) The effect of the group size

![Graph showing the effect of the group size](image)
Preliminary Results on MS-Marco

- TF-Ranking enables faster iterations over ideas to build ranking-appropriate modules
- An early attempt is illustrated to the right
  - Trained with Softmax Cross Entropy (ListNet) loss, it achieves MRR of .244 on the (held-out) “dev” set.
    - [Official Baseline] BM25 -- .167
    - [Official Baseline] Duet V2 -- .243
    - Best non-BERT result -- .318
Quick Access

Model performance with **various loss functions**

<table>
<thead>
<tr>
<th>Quick Access</th>
<th>ΔMRR</th>
<th>ΔARP</th>
<th>ΔNDCG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sigmoid Cross Entropy (Pointwise)</td>
<td>−</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td>Logistic Loss (Pairwise)</td>
<td>+0.70</td>
<td>+1.86</td>
<td>+0.35</td>
</tr>
<tr>
<td>Softmax Cross Entropy (Listwise)</td>
<td>+1.08</td>
<td>+1.88</td>
<td>+1.05</td>
</tr>
</tbody>
</table>
Gmail Search

Model performance with various loss functions

<table>
<thead>
<tr>
<th>Model</th>
<th>ΔMRR</th>
<th>ΔARP</th>
<th>ΔNDCG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gmail Search</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Sigmoid Cross Entropy</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>(Pointwise)</td>
<td>+1.52</td>
<td>+1.64</td>
<td>+1.00</td>
</tr>
<tr>
<td>Logistic Loss (Pairwise)</td>
<td>+1.52</td>
<td>+1.64</td>
<td>+1.00</td>
</tr>
<tr>
<td>Softmax Cross Entropy</td>
<td>+1.80</td>
<td>+1.88</td>
<td>+1.57</td>
</tr>
<tr>
<td>(Listwise)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

"TF-Ranking: Scalable TensorFlow Library for Learning-to-Rank"
Pasumarthi et al., KDD 2019
Gmail Search: Incorporating Sparse Features

Model performance as compared to LambdaMART

<table>
<thead>
<tr>
<th>Gmail Search</th>
<th>Dense Features ($\Delta$MRR)</th>
<th>Dense + Sparse Features ($\Delta$MRR)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\lambda$MART</td>
<td>0.0</td>
<td>--</td>
</tr>
<tr>
<td>Softmax CE w/ GSF(m=2)</td>
<td>+0.3</td>
<td>+2.4</td>
</tr>
<tr>
<td>$\lambda$MART + Softmax CE w/ GSF(m=2)</td>
<td>+0.95</td>
<td>+3.42</td>
</tr>
</tbody>
</table>
Hands-on Tutorial
Steps to get started

- Go to [git.io/tf-ranking-demo](https://git.io/tf-ranking-demo)
- Open the notebook in colaboratory
  - Make sure the URL starts with “colab.research.google.com”
- Click “Connect” to connect to a hosted runtime.
  - This is where the code runs, and the files reside.
- Open “Runtime” and select “Run All”
- Scroll down to the section on “Train and evaluate the ranker”, to see the training in execution
git.io/tf-ranking-demo
TF-Ranking Architecture

Training

- Data
- Input Reader
  - `input_fn`
- Feature Transformation
  - `transform_fn`
- Scoring Function
  - `score_fn`
- Scores
- Ranking Head
  - `make_metrics_fn`
  - `make_loss_fn`
  - Loss Builder
  - `loss_keys`
  - Metric keys
- Model function
  - `model_fn`
- Loss Metrics

Serving

- Raw data
- Serving receiver
  - `input_fn`
- Feature Transformation
  - `transform_fn`
- Features
- Scoring Function
  - `score_fn`
- Scores
"Course Homework"

- **Try running the colab with a different loss function**
  - Use one of the losses listed at: [git.io/tfr-losses](git.io/tfr-losses)
  - Advanced: Implement your own custom loss function

- **Try running with an additional metric**
  - You can use Average Relevance Position, listed at: [git.io/tfr-metrics](git.io/tfr-metrics)
  - Advanced: Implement a metric that is a linear combination of two existing metrics

- **Explore different neural networks for scoring function**
  - Increase the number of layers: when does it start to overfit?

- **Try running TF-Ranking on your ranking problem**
  - Let us know your experience by filing an [issue](https://github.com) on github!