TF-Ranking

Neural Learning to Rank using TensorFlow

ICTIR 2019

Rama Kumar Pasumarthi
Sebastian Bruch
Michael Bendersky
Xuanhui Wang

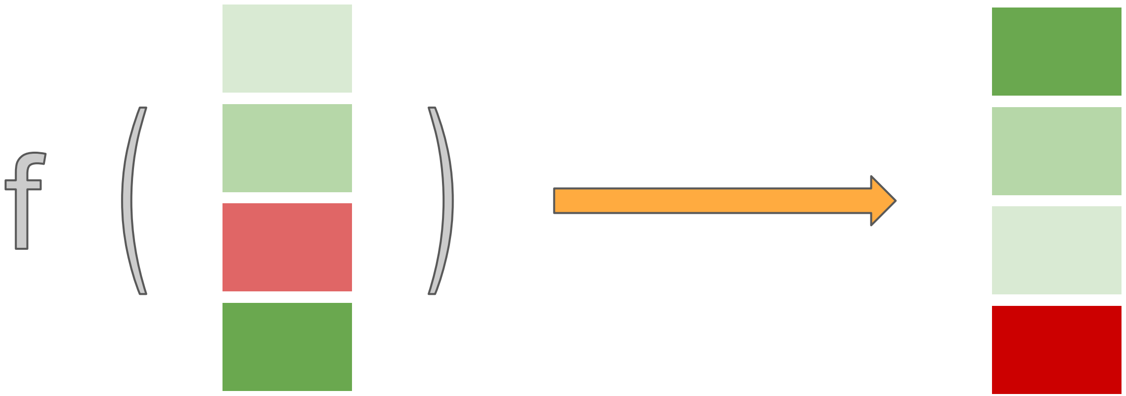
Google Research

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

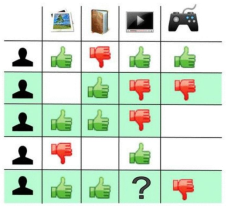
Motivation

Learning to Rank

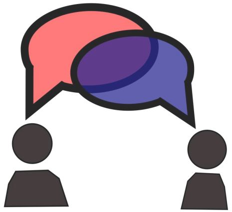


Applications

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

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

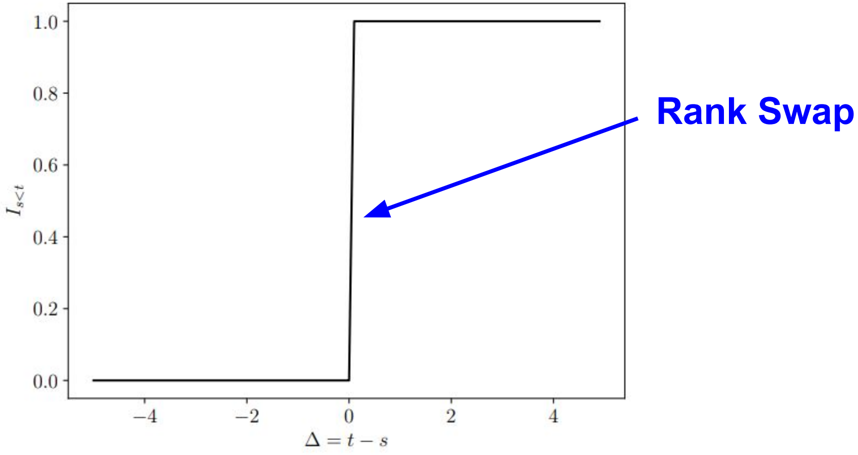
Training sample with relevance labels

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

Choose f* to minimize empirical loss

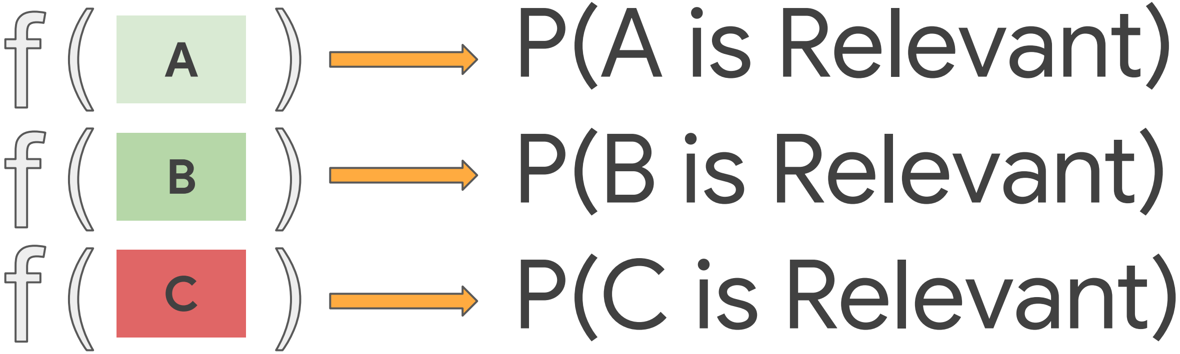
Ranking Metric Optimization

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



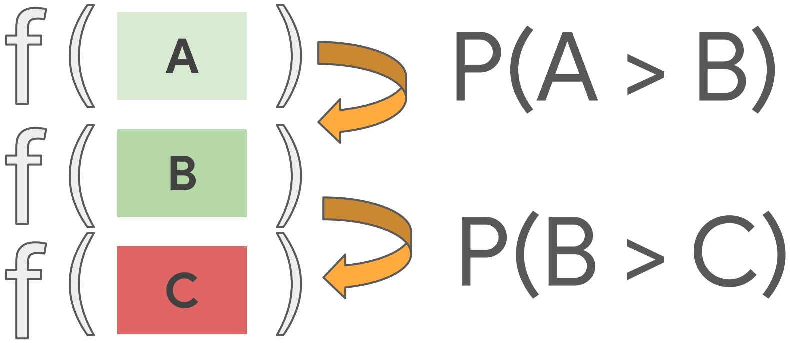
Pointwise LTR methods

- Documents are considered independently of each other
- Some examples: ***ordinal regression***, ***classification***, ***GBRTs***



Pairwise LTR methods

- Document pairs are considered
- Some examples: ***RankNet***, ***RankSVM***, ***RankBoost***



Listwise LTR methods

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

Standard LTR setting

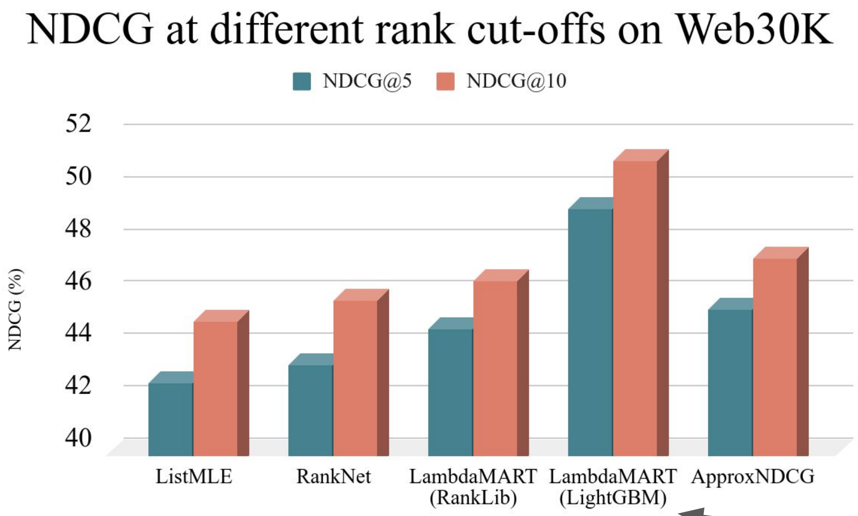
- ***Handcrafted*** features based on query, document and their match scores
  - Web30K has 136 features per document
    - tf-idf scores
    - BM25 scores
    - Inlink counts
    - URL length
    - Page quality
    - ….
- ***Human*** relevance judgments
  - The largest datasets have tens of thousands of labeled examples
    - Web30K, Istella, Yahoo! **~30K** queries

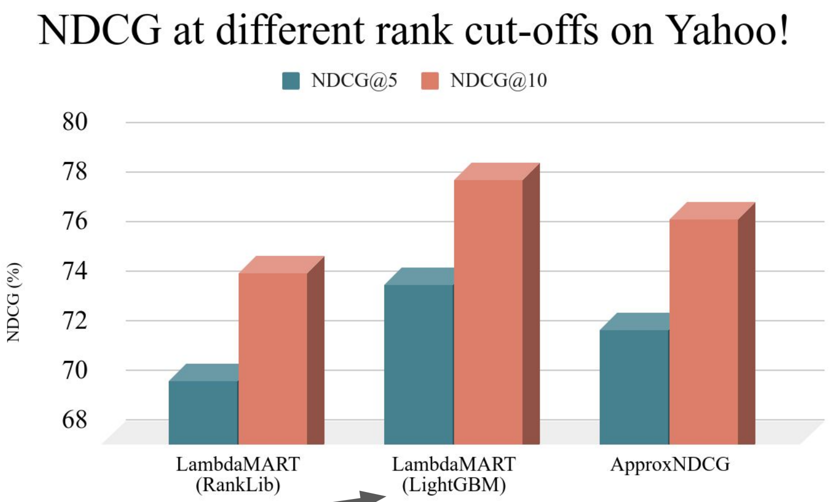
Feature List of Microsoft Learning to Rank Datasets

feature id	feature description	stream
1	covered query term number	body
2		anchor
3		title
4		url
5		whole document
6	covered query term ratio	body
7		anchor
8		title
9		url
10		whole document
11	stream length	body
12		anchor
13		title
14		url
15		whole document
16	IDF(Inverse document frequency)	body
17		anchor
18		title
19		url
20		whole document
21	sum of term frequency	body
22		anchor
23		title
24		url
25		whole document
26	min of term frequency	body
27		anchor
28		title
29		url
30		whole document
31	max of term frequency	body
32		anchor
33		title
34		url
35		whole document
36	mean of term frequency	body
37		anchor
38		title
39		url
40		whole document
41	variance of term frequency	body
42		anchor
43		title
44		url
45		whole document
46	sum of stream length normalized term frequency	body
47		anchor
48		title
49		url
50		whole document
51	min of stream length	body

Sample of features available on Web30K

Current State-of-the-Art in LTR





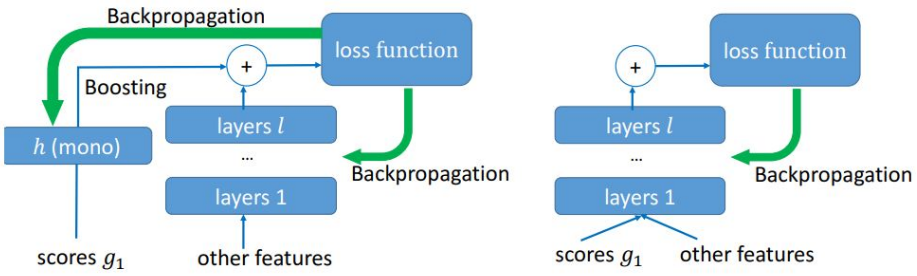
The best LambdaMART implementation is still the most competitive on public LTR datasets

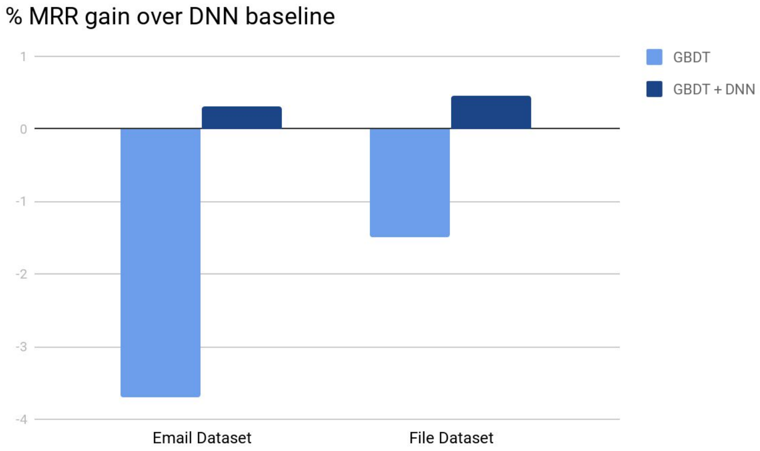
"Revisiting Approximate Metric Optimization in the Age of Deep Neural Networks"
Bruch et al., SIGIR 2019

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*

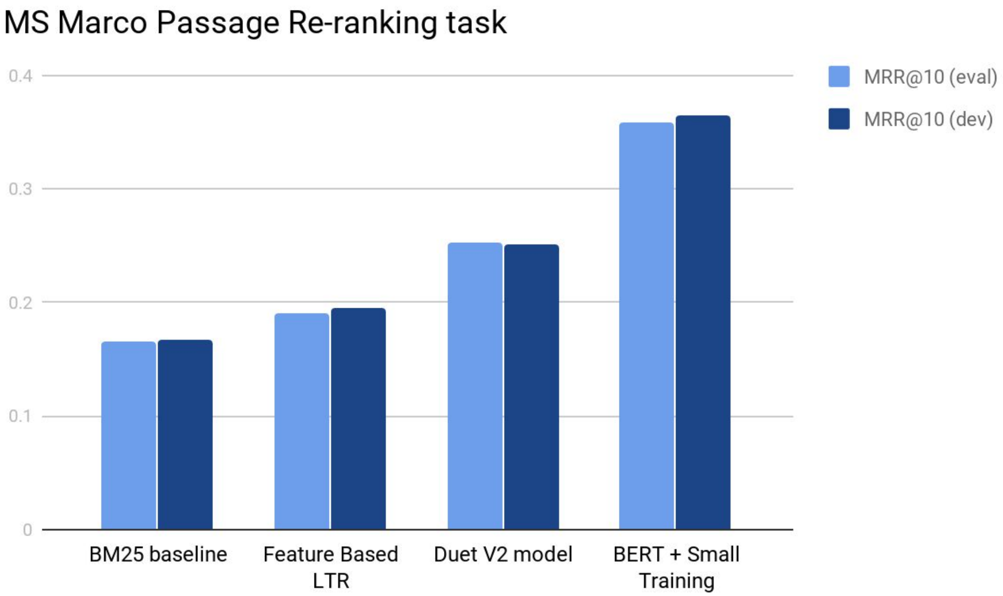




"Combining Decision Trees and Neural Networks for Learning-to-Rank in Personal Search"
Pan et al., KDD 2019

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

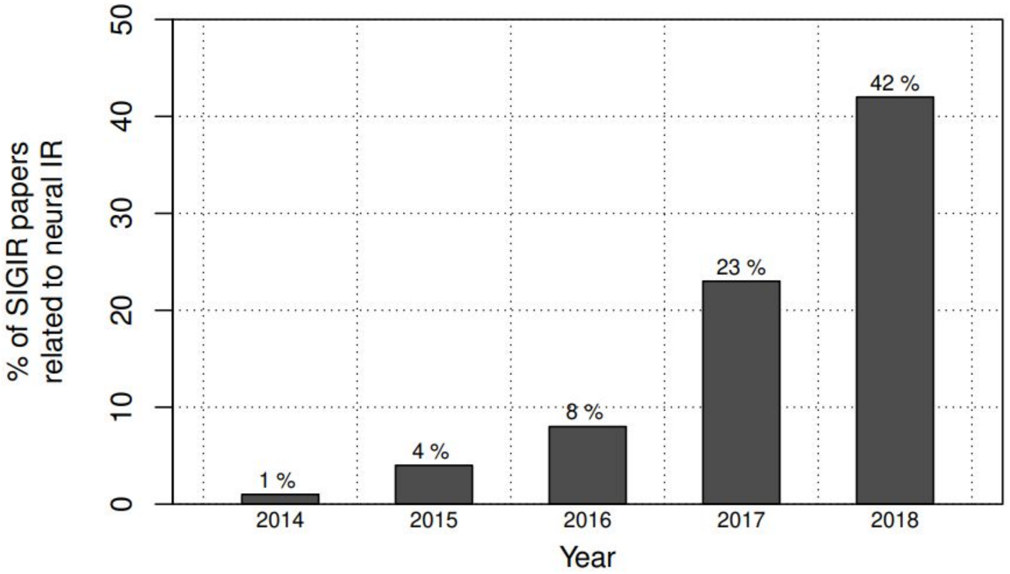
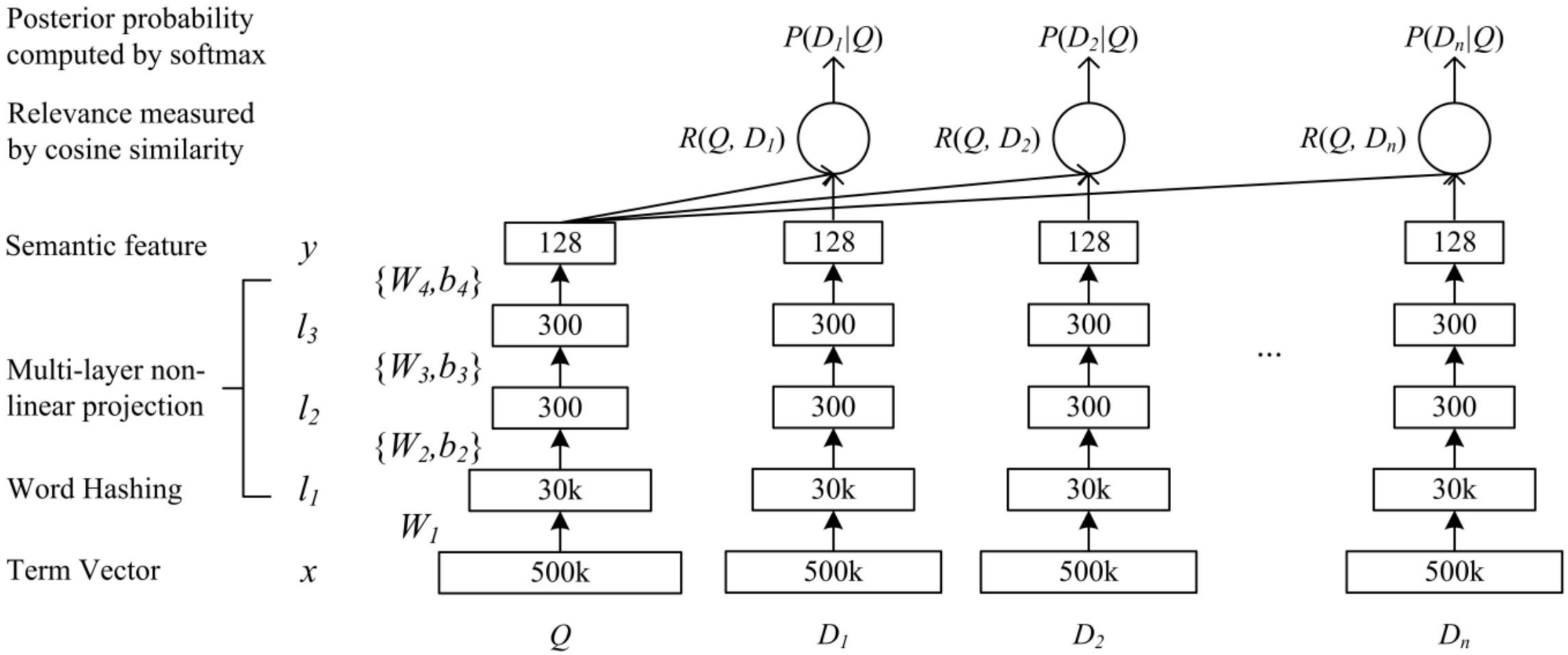


Figure 1.1: The percentage of neural IR papers at the ACM SIGIR conference—as determined by a manual inspection of the papers—shows a clear trend in the growing popularity of the field.

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

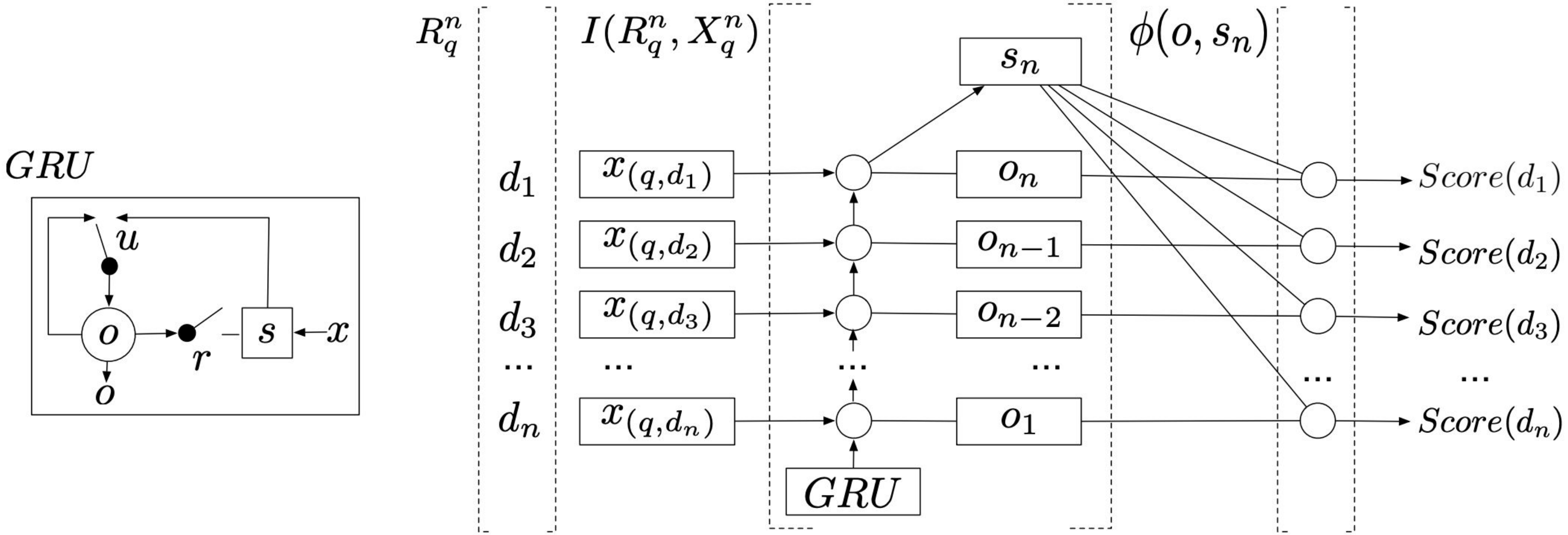
- Neural IR is increasingly popular
- Major focus is on ***neural matching models***
- Less research on ***neural ranking models***

DSSM model



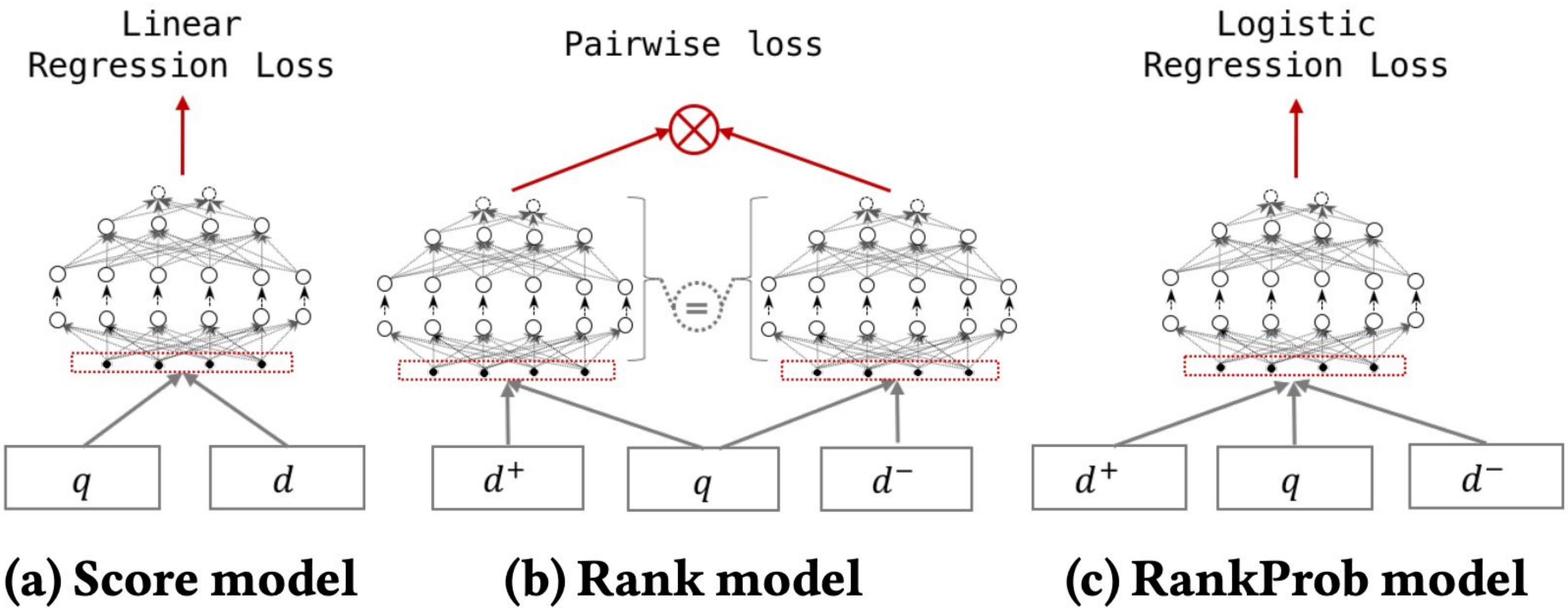
"Learning Deep Structured Semantic Models for Web Search using Clickthrough Data"
Huang et al., CIKM 2013

Deep Listwise Context Model (DLCM)



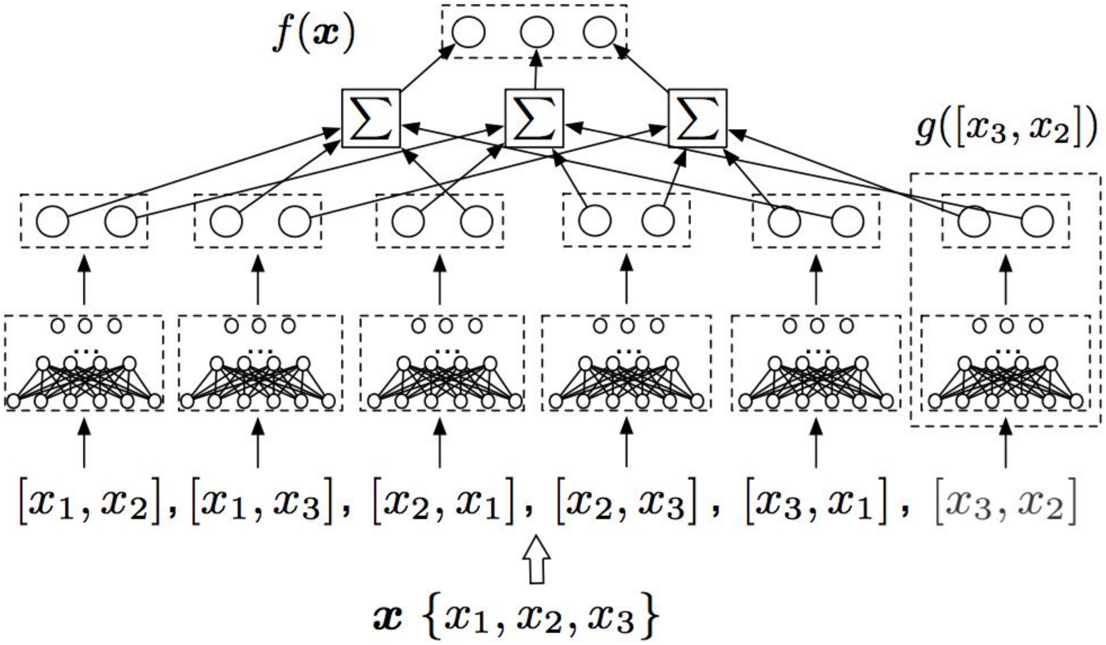
"Learning a Deep Listwise Context Model for Ranking Refinement"
Ai et al., SIGIR 2018

Neural Ranking with Weak Supervision



"Neural Ranking Models with Weak Supervision"
Dehghani et al., SIGIR 2017

Groupwise Multivariate Scoring Functions

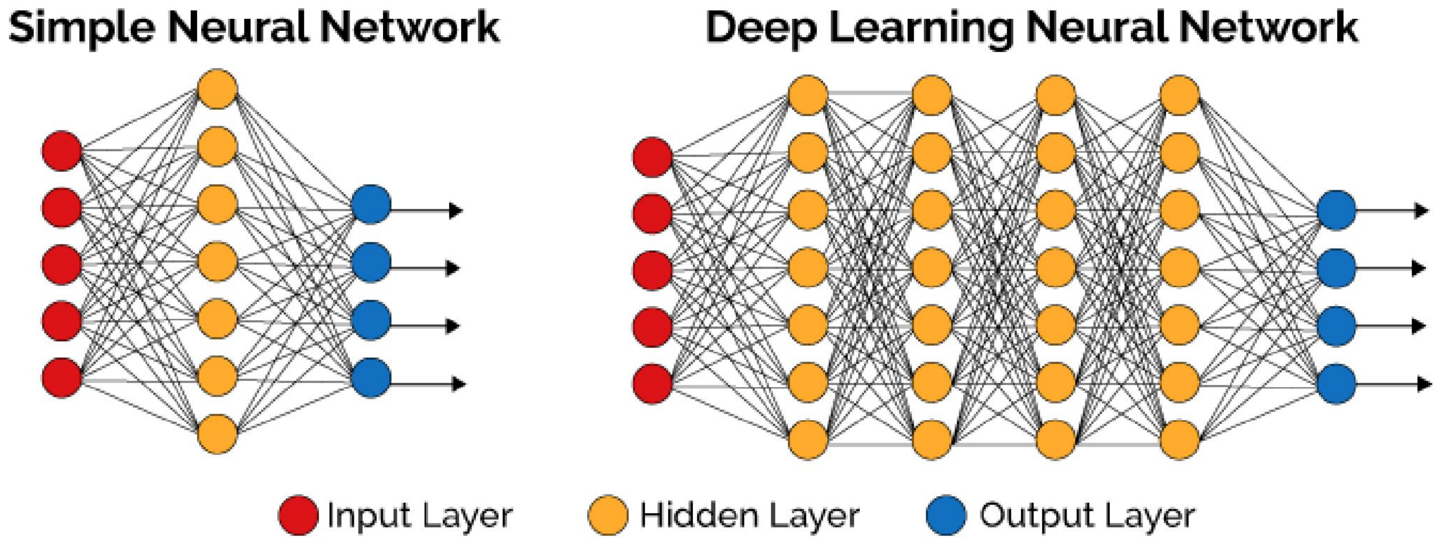


"Learning Groupwise Multivariate Scoring Functions Using Deep Neural Networks"
Ai et al., ICTIR 2019

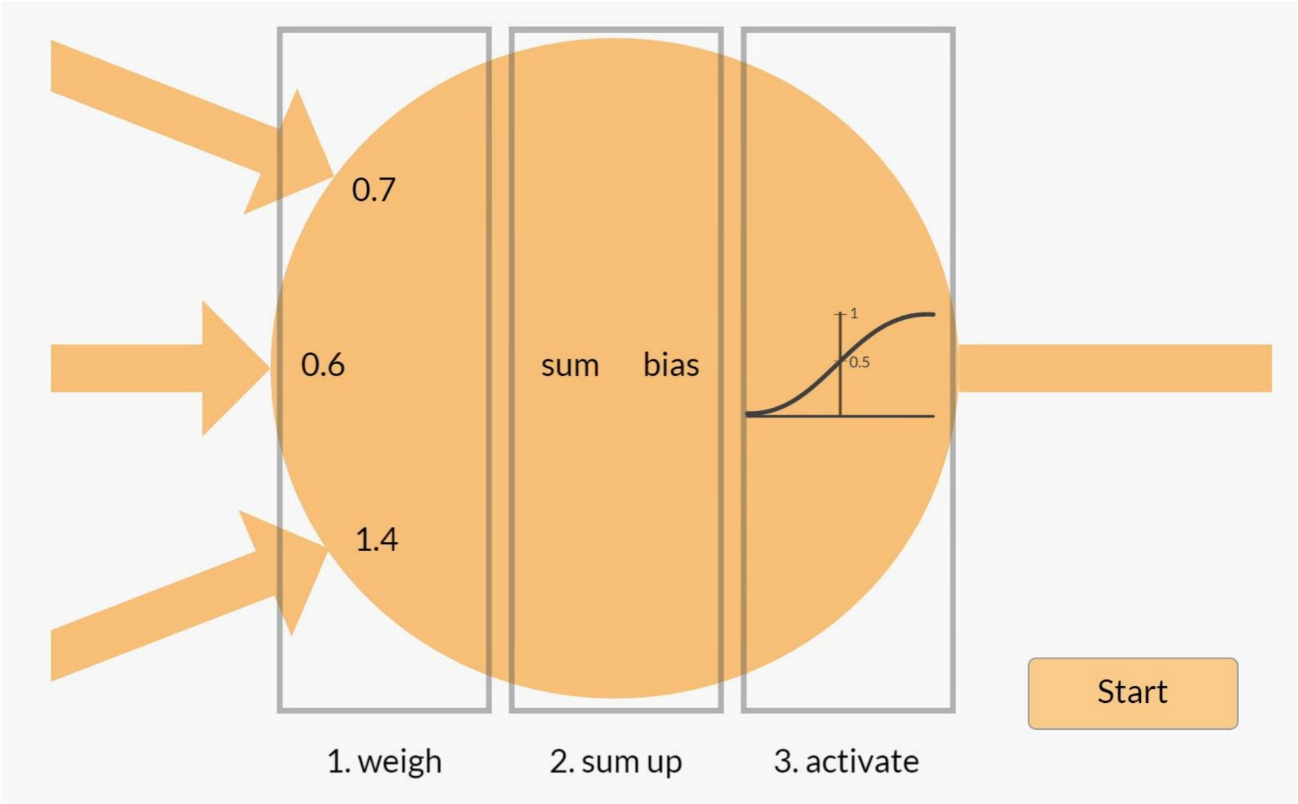
Introduction to Deep Learning and TensorFlow

Many materials are from Lex Friedman's MIT Deep Learning Course
https://www.dropbox.com/s/c0g3sc1shi63x3q/deep_learning_basics.pdf

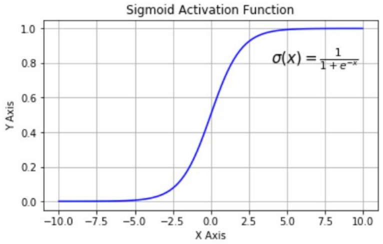
Deep Neural Network

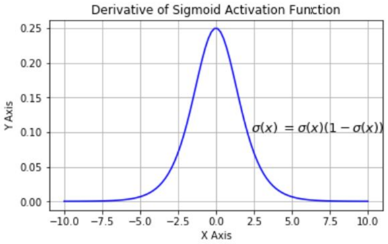


Neuron



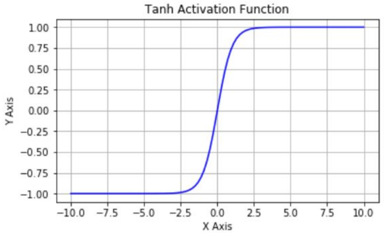
Activation Function → Non-Linearity

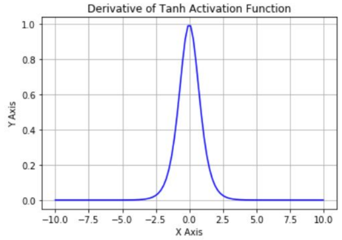




Sigmoid

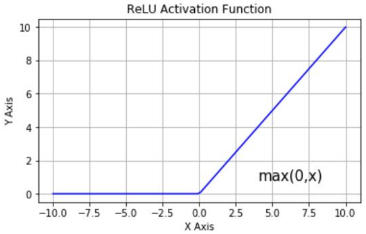
- Vanishing gradients
- Not zero centered

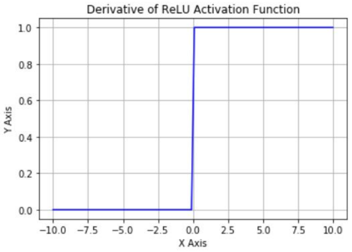




Tanh

- Vanishing gradients





ReLU

- Not zero centered

Loss Function

Mean Squared Error

Prediction

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

Ground Truth

Cross Entropy Loss

Classes

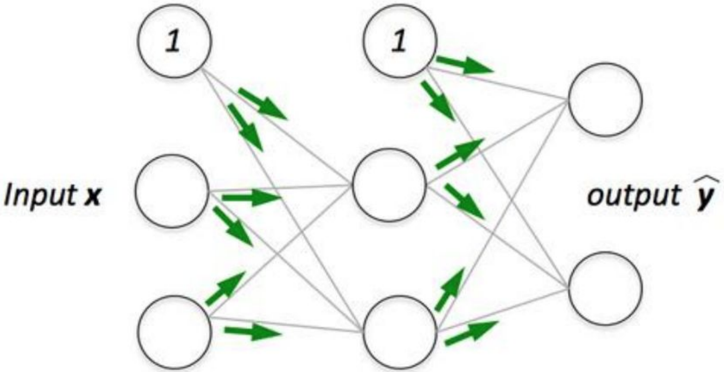
Prediction

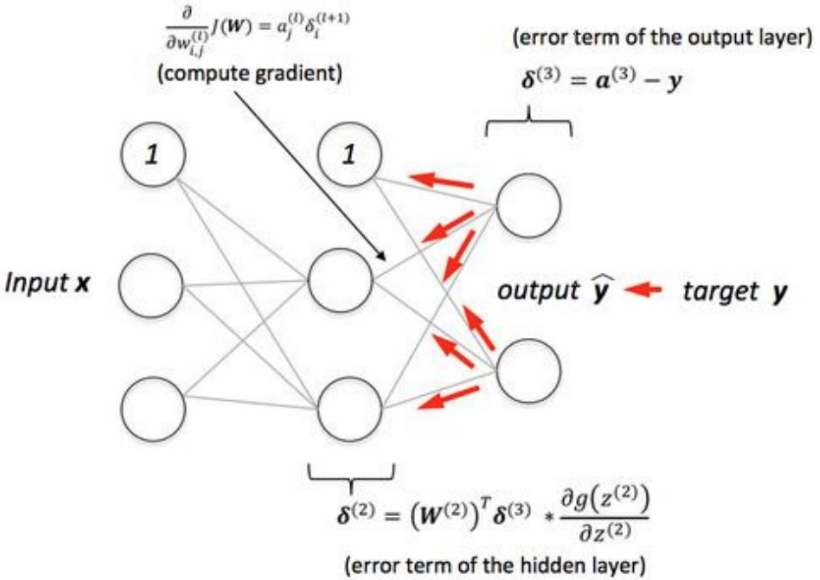
$$CE = -\sum_{i}^{C} t_i log(s_i)$$

Ground Truth {0,1}

Backpropagation

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial w_{ij}} = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial \mathrm{net}_j}\frac{\partial \mathrm{net}_j}{\partial w_{ij}}$$





Task: Update the **weights** and **biases** to decrease **loss function**

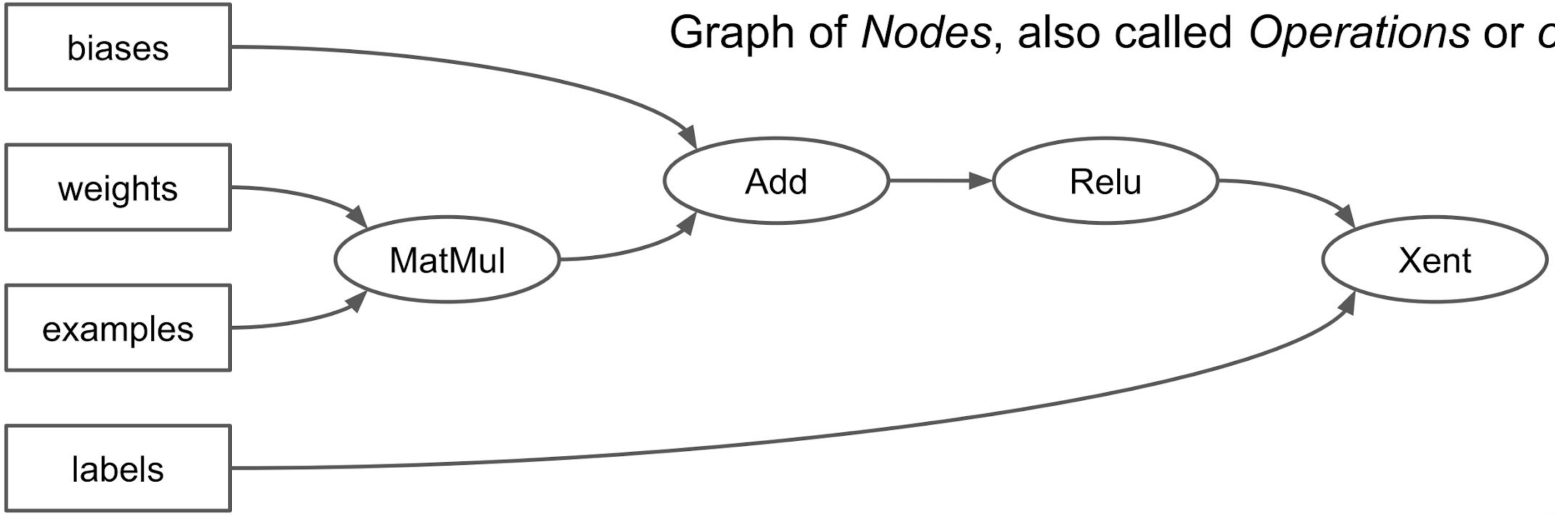
TensorFlow: A Deep Learning Framework

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

Slides are adapted from Jeff Dean's talk:
https://www.matroid.com/scaledml/slides/jeff.pdf

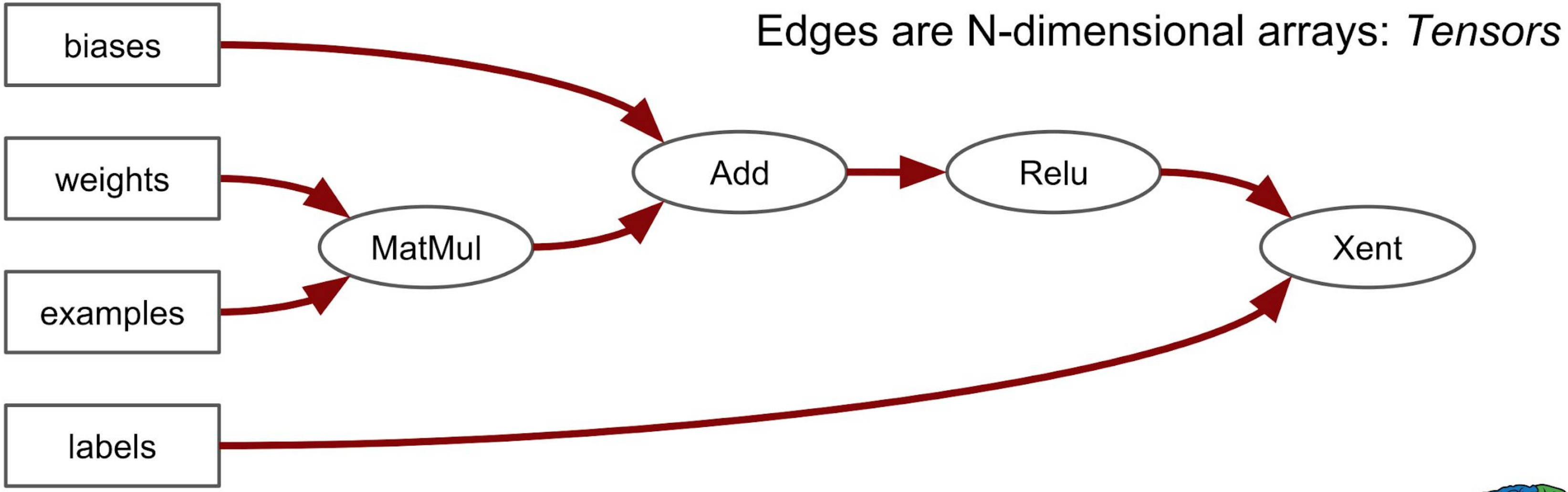
Computation is a dataflow graph

Graph of *Nodes*, also called *Operations* or *ops.*



Computation is a dataflow graph

with tensors



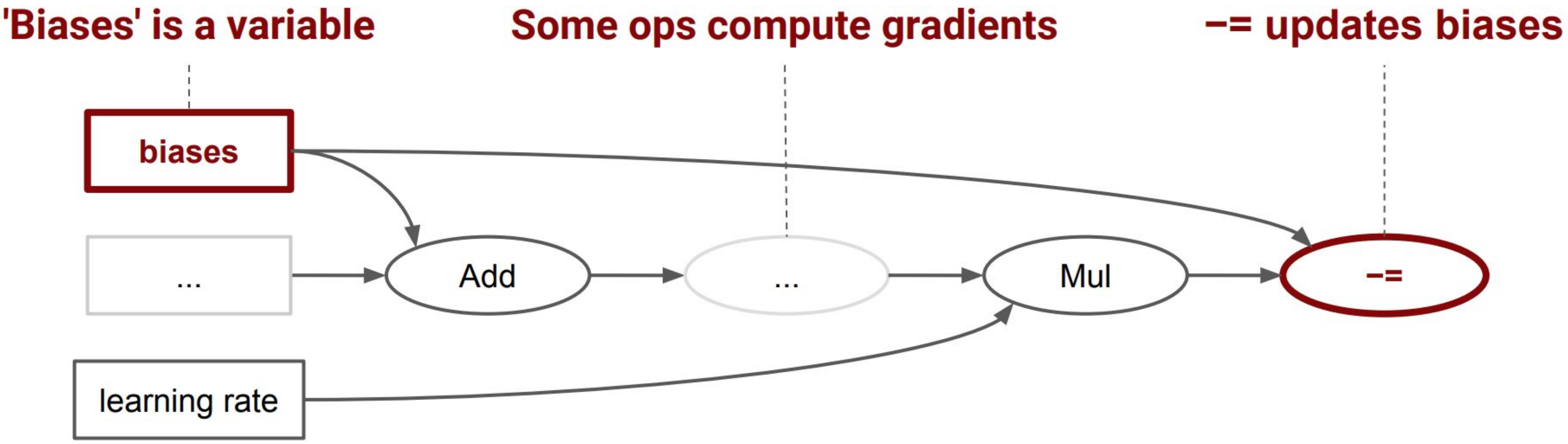
Declarative Language to Build a Graph

```
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



Symbolic Differentiation

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

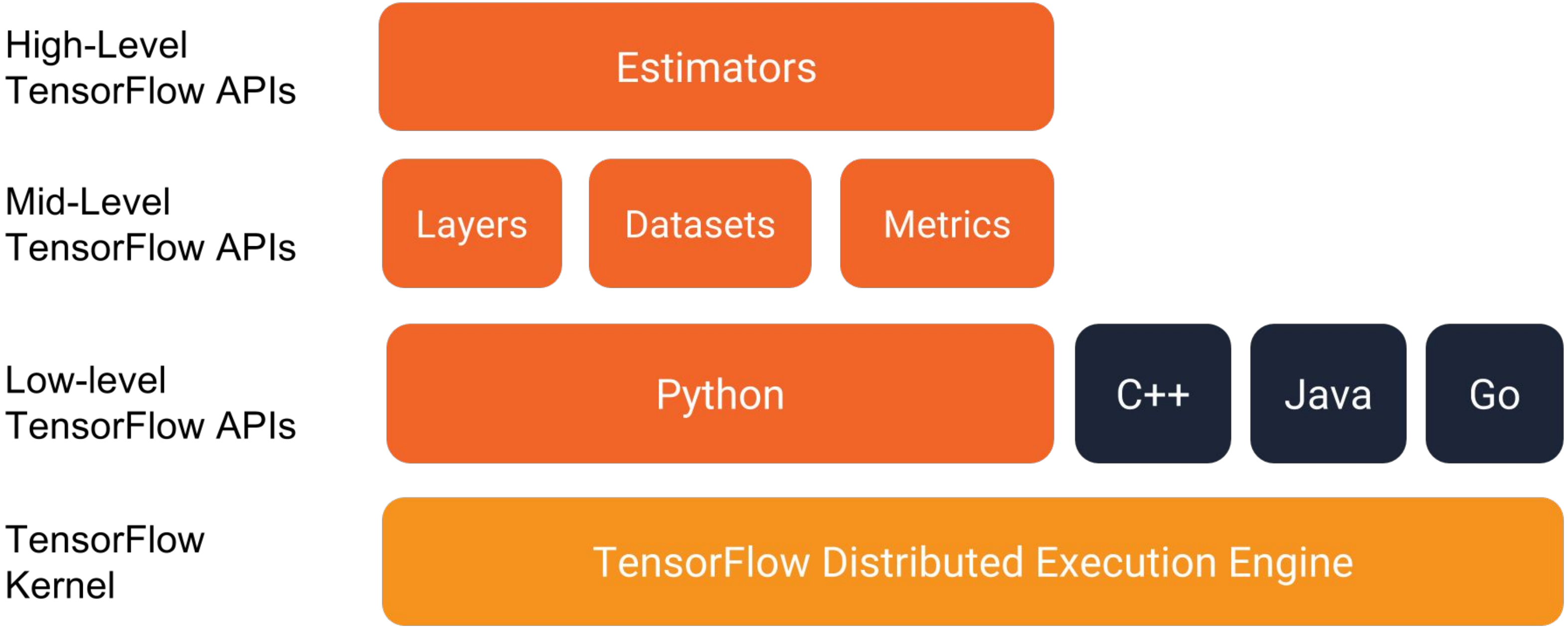
```
y_ = tf.placeholder(tf.float32, [None, 10])
cross_entropy = -tf.reduce_sum(y_ * tf.log(y))
opt = tf.train.GradientDescentOptimizer(0.01)
train_op = opt.minimize(cross_entropy)
```

Define graph and then execute it repeatedly

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

```
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



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

q_1

```
{
    key: "query_tokens"
    value: ["this", "is", "a", "query"]
}
```

doc list

$d_{1,1}$

```
{
    key: "document_tokens"
    value: ["relevant", "answer"],
    key: "relevance"
    value: 1
}
```

$d_{1,2}$

```
{
    key: "document_tokens"
    value: ["irrelevant", "data"],
    key: "relevance"
    value: 0
}
```

$d_{1,3}$

```
{
    key: "document_tokens"
    value: ["very", "relevant"],
    key: "relevance"
    value: 2
}
```

q_2

```
{
    key: "query_tokens"
    value: ["another", "query"]
}
```

doc list

$d_{2,1}$

```
{
    key: "document_tokens"
    value: ["irrelevant", "answer"],
    key: "relevance"
    value: 0
}
```

$d_{2,2}$

```
{
    key: "document_tokens"
    value: ["relevant"],
    key: "relevance"
    value: 1
}
```

- Each q, d is a tf.Example and serialized as a string
- EIE is tf.Example with 2 features:
    - “serialized_context”: q
    - “serialized_examples”: [d_1, d_2, …]

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

	$d_{1,1}$	$d_{1,2}$	$d_{1,3}$	
q_1	[relevant,	not-relevant,	very-relevant],	[1, 0, 2],
q_2	[not-relevant,	relevant,	**padding-doc**],	[0, 1, -1],
	$d_{2,1}$	$d_{2,2}$	N/A	

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

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

Average Relevance Position

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

Discounted Cumulative Gain

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

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 $f(x_1, \ldots, x_m)$ jointly scores a group of m documents

Supported Loss Examples (Binary Labels)

(*Pointwise*) Sigmoid Cross Entropy

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

(Pairwise) Logistic Loss

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

(Listwise) Softmax Loss (aka ListNET)

$$\hat{\ell}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -\sum_{j=1}^{n} y_j \log(\frac{\exp(\hat{y}_j)}{\sum_{j=1}^{n} \exp(\hat{y}_j)})$$

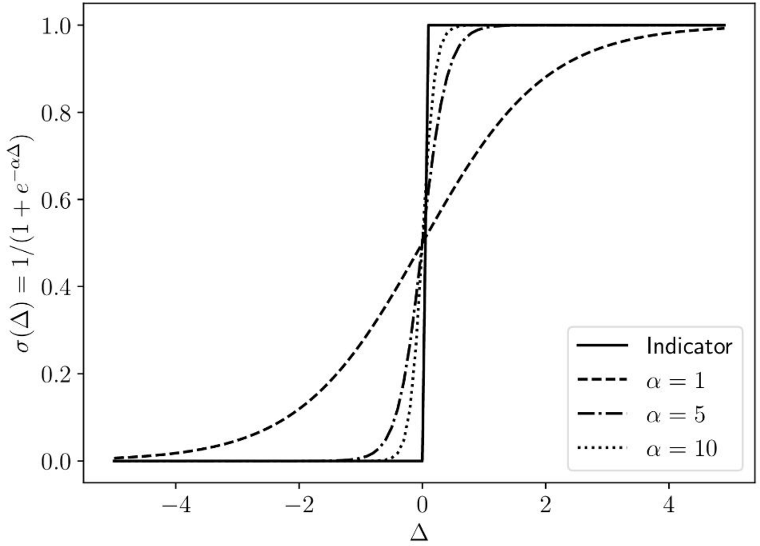
"An Analysis of the Softmax Cross Entropy Loss for Learning-to-Rank with Binary Relevance"
Bruch et al., ICTIR 2019

ApproxNDCG - Ranking Metric Approximation

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

$$\pi_f(i) \triangleq 1 + \sum_{j \neq i} \mathbb{I}_{f(\boldsymbol{x})|_i < f(\boldsymbol{x})|_j}$$

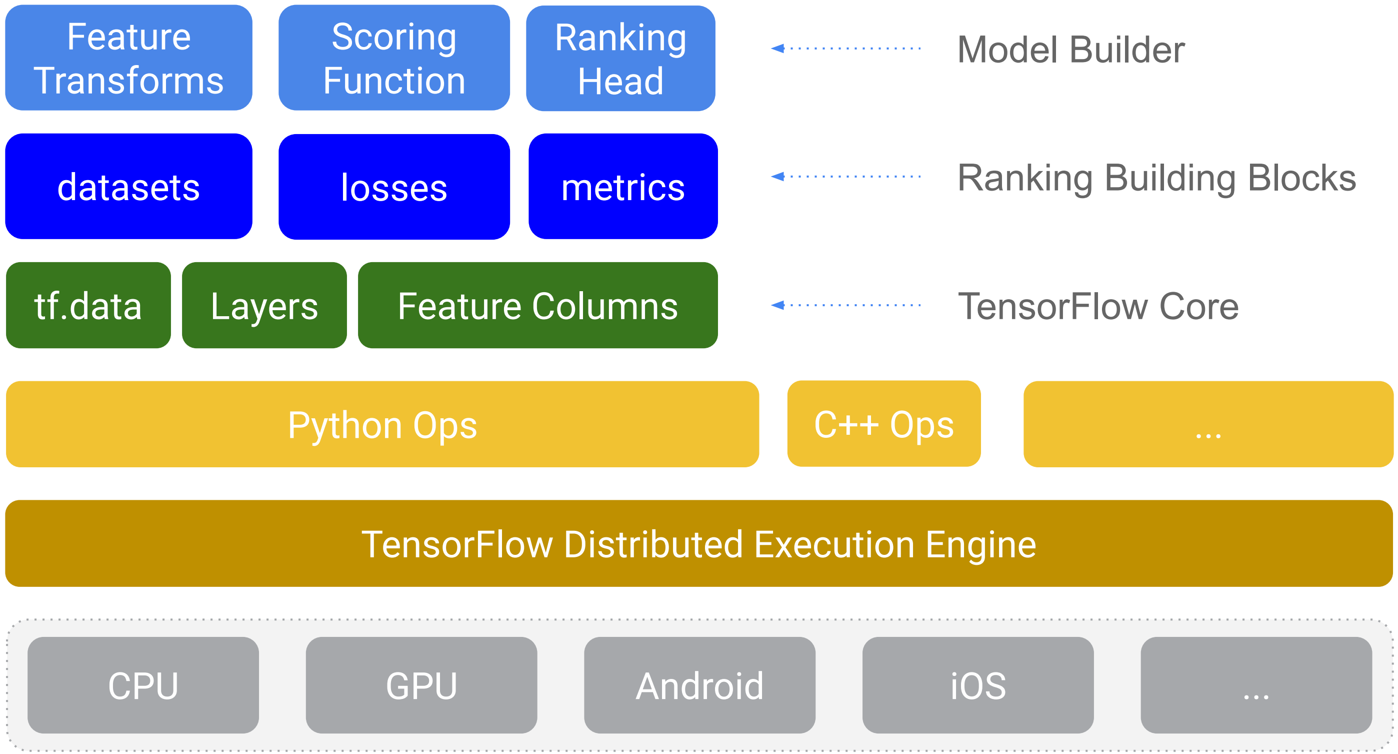
$$\mathbb{I}_{s<t} = \mathbb{I}_{t-s>0} \approx \sigma(t-s) \triangleq \frac{1}{1 + e^{-\alpha(t-s)}}$$



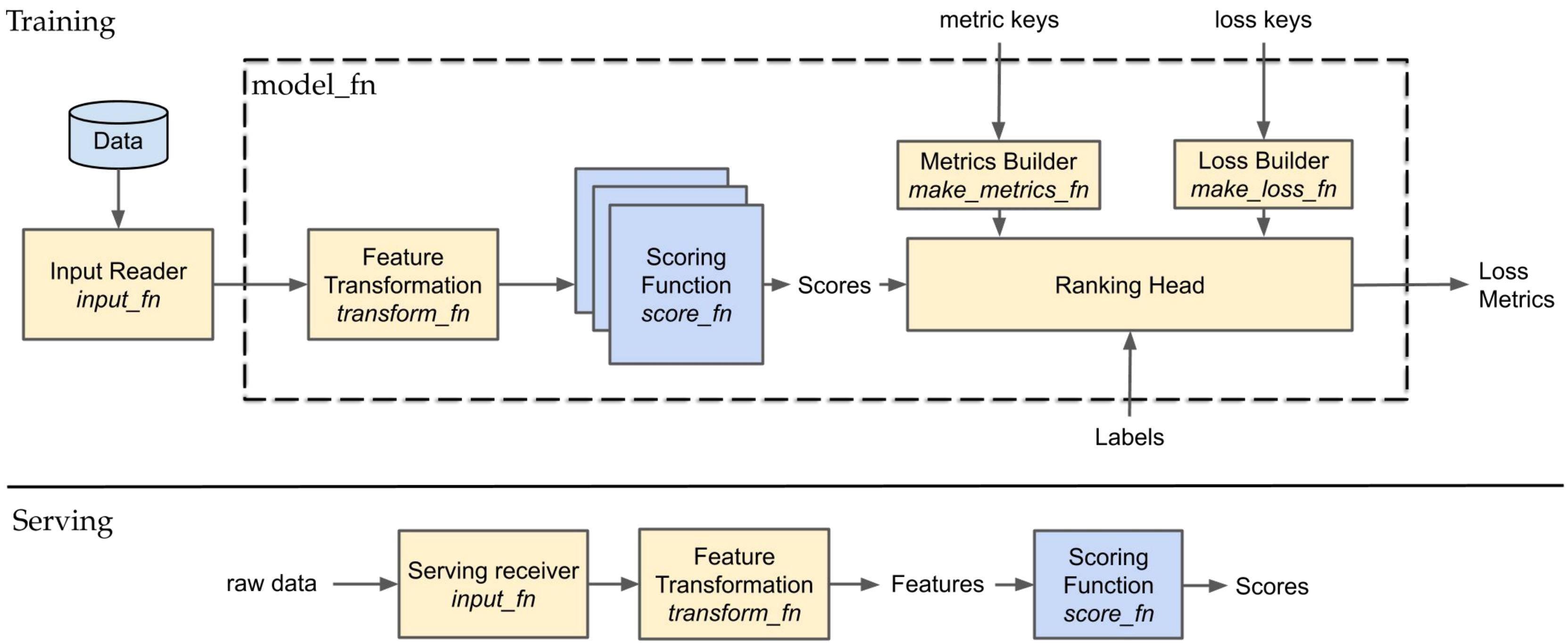
"A general approximation framework for direct optimization of information retrieval measures"
Qin et al., Information Retrieval, 2010

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

TF-Ranking Ecosystem



TF-Ranking Architecture

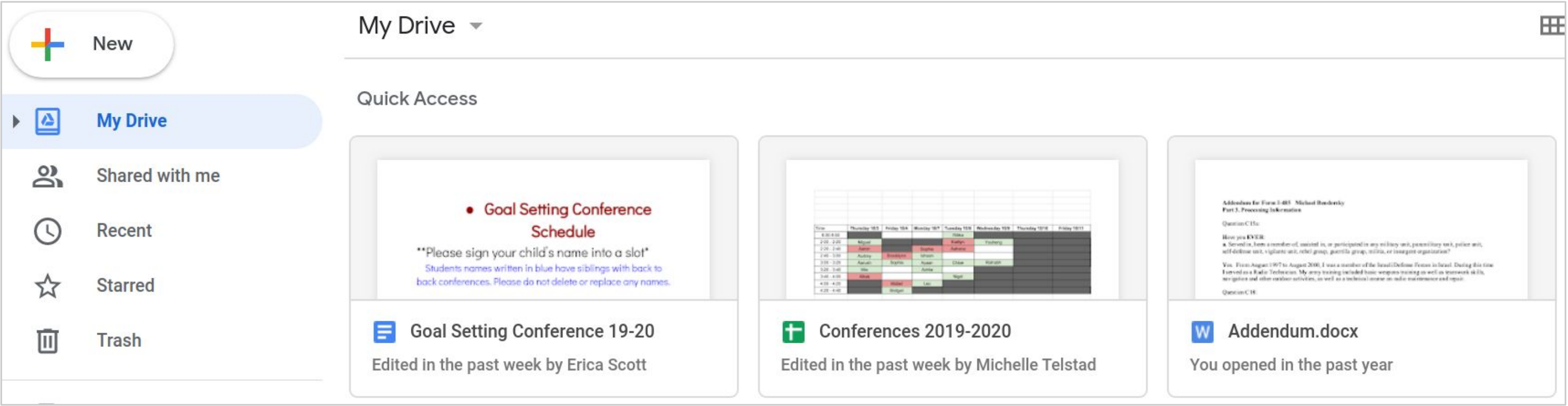


Empirical Results

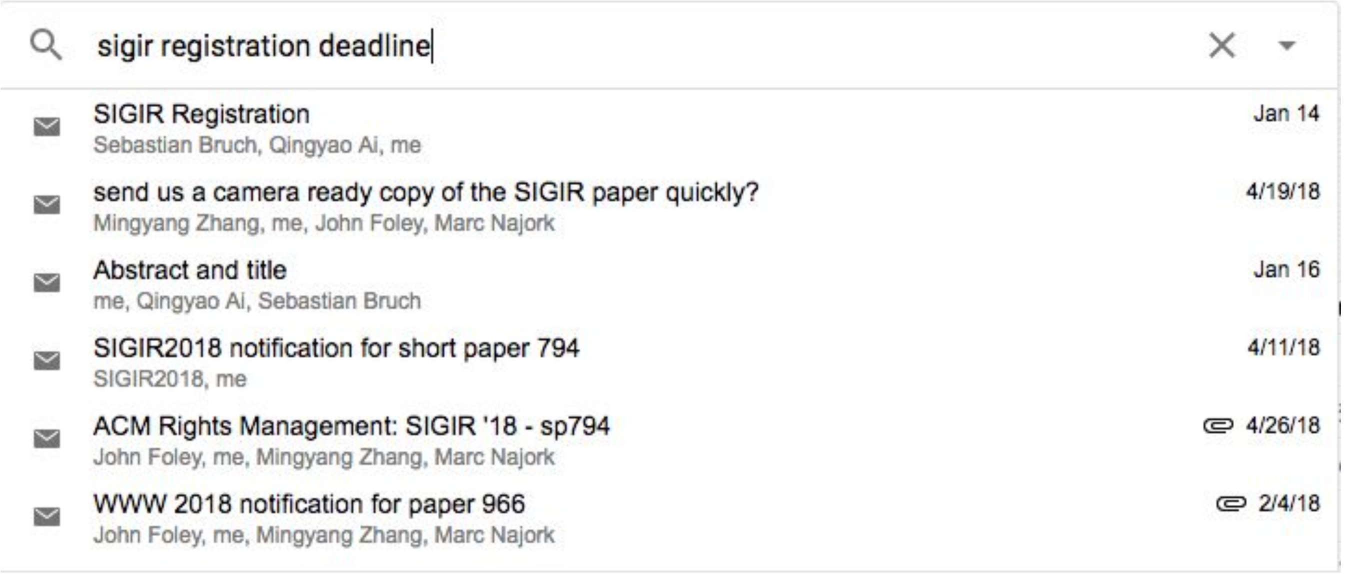
Datasets

Dataset	# queries			
MSLR-Web30k	~30K	Public	Search	dense features
MS-Marco	~800K	Public	Q&A	sparse features
Quick Access	~30M	Internal	Recommendation	dense features
Gmail Instant Search	~300M	Internal	Search	dense features sparse features

Quick Access: Recommendation in Google Drive



Gmail Instant Search

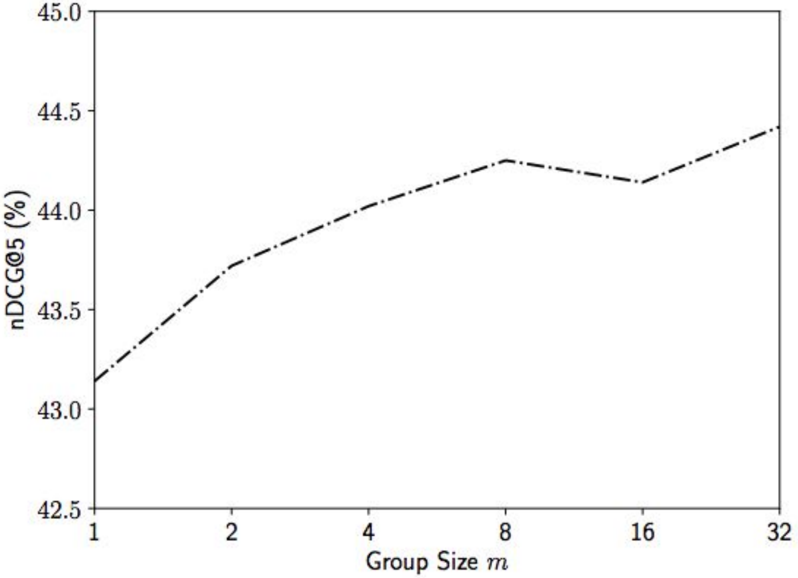


MSLR-Web30k

(a) Comparison w/ other LTR models

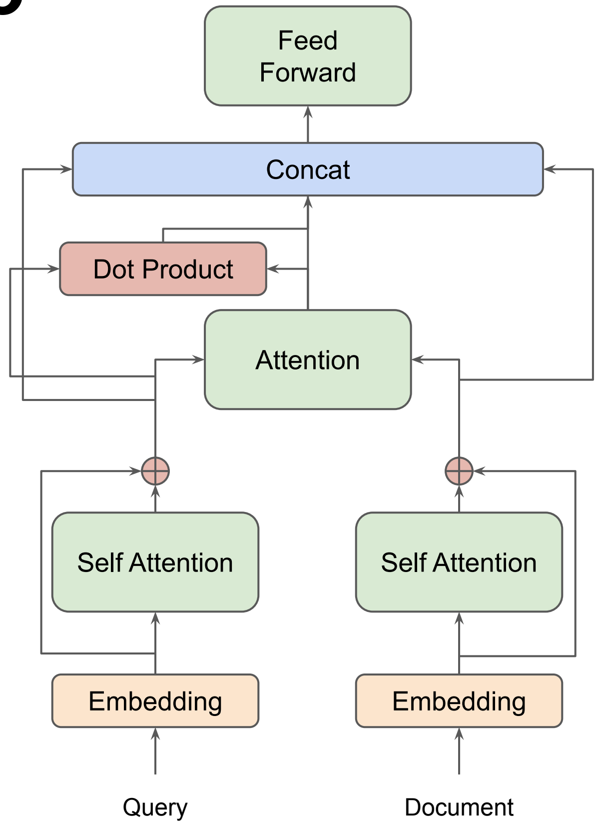
	NDCG@5
RankNet$_{\text{RankLib}}$	32.28
RankSVM$_{\text{RankLib}}$	33.74
MART$_{\text{RankLib}}$	43.54
λMART$_{\text{RankLib}}$	44.50
λMART$_{\text{LightGBM}}$	**49.20**
Softmax CE w/ GSF(m=32)	44.42
ApproxNDCG	45.38

(b) The effect of the group size



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***

Quick Access	ΔMRR	ΔARP	ΔNDCG
Sigmoid Cross Entropy (Pointwise)	–	–	–
Logistic Loss (Pairwise)	+0.70	+1.86	+0.35
Softmax Cross Entropy (Listwise)	**+1.08**	**+1.88**	**+1.05**

Gmail Search

*Model performance with **various loss functions***

Gmail Search	ΔMRR	ΔARP	ΔNDCG
Sigmoid Cross Entropy (Pointwise)	–	–	–
Logistic Loss (Pairwise)	+1.52	+1.64	+1.00
Softmax Cross Entropy (Listwise)	**+1.80**	**+1.88**	**+1.57**

Gmail Search: Incorporating Sparse Features

Model performance as compared to ***LambdaMART***

Gmail Search	Dense Features (ΔMRR)	Dense + Sparse Features (ΔMRR)
λMART	0.0	--
Softmax CE w/ GSF(m=2)	+0.3	+2.4
λMART + Softmax CE w/ GSF(m=2)	**+0.95**	**+3.42**

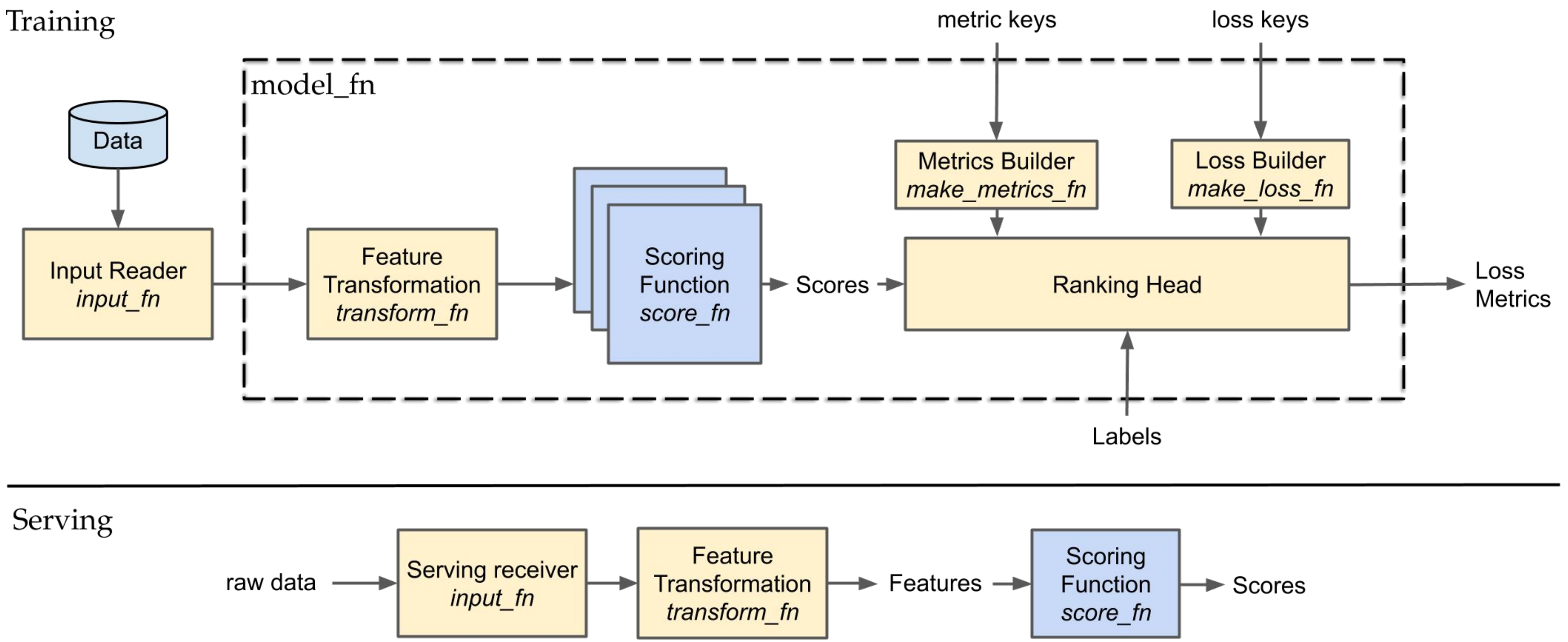
Hands-on Tutorial

Steps to get started

- Go to 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



"Course Homework"

- Try running the colab with a different loss function
  - Use one of the losses listed at: 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
  - 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 on github!