TF-Ranking: Scalable TensorFlow Library for Learning-to-Rank

Rama Kumar Pasumarthi, Xuanhui Wang, Cheng Li,
Sebastian Bruch, Michael Bendersky, Marc Najork,
Jan Pfeifer, Nadav Golbandi, Rohan Anil, Stephan Wolf
Google, Inc.
`{ramakumar,xuanhui,chgli,bruch,bemike,najork,`
`janpf,nadavg,rohananil,stephanw}@google.com`

Abstract

TensorFlow Ranking is the first open source library for solving large-scale ranking problems in a deep learning framework[1]. It is highly configurable and provides easy-to-use APIs to support different scoring mechanisms, loss functions and evaluation metrics in the learning-to-rank setting. Our library is developed on top of TensorFlow and can thus fully leverage the advantages of this platform. For example, it is highly scalable, both in training and in inference, and can be used to learn ranking models over massive amounts of user activity data. We empirically demonstrate the effectiveness of our library in learning ranking functions for large-scale search and recommendation applications in Gmail and Google Drive.

1 Introduction

With the high potential of deep learning (DL) for real-world data-intensive applications, multiple open source packages have emerged in recent years and are under active development, including TensorFlow [1], PyTorch [20], Caffe [14], MXNet [6], etc. Supervised learning is one of the main use cases of DL packages. For example, one task in the ImageNet competitions [22] is to predict image categories, which can be formulated as a multi-class classification problem. However, compared with the comprehensive support for classification or regression in open-source DL packages, there is a paucity of support for ranking problems.

A ranking problem is defined as a derivation of ordering over a list of examples that maximizes the utility of the entire list. It is widely applicable in several domains, such as Information Retrieval (IR) and Natural Language Processing (NLP). Some important practical applications include web search, recommender systems, machine translation, document summarization, question answering, etc [17].

In general, a ranking problem is different from classification or regression tasks. While the goal of classification or regression is to predict a label or a value for each individual example as accurately as possible, the goal of ranking is to optimally sort the entire example list, such that the examples of highest relevance are presented first. In practice, relevance can be obtained from human judgment (explicit feedback) or from data such as click logs (implicit feedback). Because of this difference, widely used ranking metrics such as Normalized Discounted Cumulative Gain (NDCG) [13], Mean Reciprocal Rank (MRR) [9], Mean Average Precision (MAP) and Average Relevance Position (ARP) [30] are different from those that are used in classification or regression. These metrics take the ranks of the examples into consideration and are generally designed to emphasize the examples that are ranked higher in the list.

[1]The open source library is available at: `https://github.com/tensorflow/ranking`.

Preprint. Work in progress.

Learning-to-rank (LTR) [17] is a set of supervised machine learning techniques that can be used to solve ranking problems. It attempts to learn a scoring function that maps example feature vectors to real-valued scores from labeled data. During inference, this scoring function is used to sort and rank examples. *pointwise*, *pairwise* or *listwise* loss functions are used during training with the goal to optimize the correctness of relative order among a list of examples. Using pointwise loss functions can be seen as approximating ranking problem to a classification or regression techniques. In practice, *pairwise* or *listwise* loss functions, and advanced loss functions such as LambdaLoss [27] tend to outperform the pointwise approaches [17], as they are more closely aligned with optimizing the ranking metrics over the entire list. It is worth noting that in traditional LTR settings the loss function and the ranking metrics are different, since most optimizers require differentiable loss functions whereas all ranking metrics mentioned above are non-continuous over the permutations of a ranked list.

In the past decade, several techniques unique to ranking problems were developed to directly optimize ranking metrics [4, 27]. In addition, recent work on unbiased learning-to-rank [2, 26, 15] have been proposed to handle position bias in biased data like user clicks, to produce a consistent and unbiased ranker. These approaches usually involve estimating Inverse Propensity Weights to counter position bias. This works well with pairwise or listwise losses, but not with pointwise loss [26]. Thus, a learning-to-rank library that implements advanced listwise loss functions is important for realistic ranking settings, which often involve biased click data.

Existing open source packages for learning-to-rank such as RankLib [10] and LightGBM [16] have several important drawbacks. First, they were developed for small labeled data sets such as LETOR [21] (thousands of examples), but not for massive click log datasets (hundreds of millions of examples) that are common in industrial applications. Second, since the existing learning-to-rank packages are not based on deep learning techniques, they cannot naturally handle sparse features like text, and require extensive feature engineering. In contrast, deep learning packages like TensorFlow can effectively handle raw sparse features through embeddings [18].

To address this gap, in this paper, we present our effort to build a scalable, comprehensive and configurable industry-grade learning-to-rank library in TensorFlow. Our main contributions are:

- We propose a unified library for training large scale learning-to-rank models using deep learning in TensorFlow.
- The library is flexible and highly configurable: it provides easy-to-use APIs to support different scoring mechanisms, loss functions and evaluation metrics.
- The library provides support for unbiased learning-to-rank by incorporating inverse propensity weights (IPW) in losses and metrics.
- We demonstrate the wide applicability of our framework by experiments on large-scale search and recommendation applications, where we show that listwise losses and ranking metric optimization significantly outperform standard pointwise losses.

Our current implementation of the TensorFlow Ranking library is by no means exhaustive. We envision that this library will provide a convenient open platform for hosting and advancing state-of-the-art ranking models based on deep learning techniques, and thus facilitate both academic research as well as industrial applications.

2 Learning-to-Rank

In this section, we provide a high-level overview of learning-to-rank techniques. We present our setup first and then describe scoring functions, ranking losses and ranking metrics.

2.1 Setup

Let x be the universe of examples and $\mathcal{X}$ the universe of example lists; similarly let $\mathcal{y}$ be the universe of labels and $\mathcal{Y}$ the universe of label lists. Assume that $\mathcal{y}$ is an ordered space (such as $\mathbb{Z}$ or $\mathbb{R}$), and without loss of generality that larger label values are preferable. In the LTR setting, a training data point is $(X, Y) \in \mathcal{X} \times \mathcal{Y}$, where $X = [x_j]_{j=1}^n$ is a list of n examples and $Y = [y_j]_{j=1}^n$ is a list of n labels. A label list Y whose labels are monotonically decreasing is called ideal.

Consider search as an example. In this setting, we have a list of documents for each query. Let q be the query and $[d_j]_{j=1}^n$ be the document list for this query. Each query-document pair becomes an example $x_j = (q, d_j)$. For each x_j, we have a corresponding relevance label y_j that can be a binary or graded integer value [12].

Given a list of examples X and a corresponding list of ground-truth relevance labels Y, π^* is the permutation that makes Y ideal. The goal of learning-to-rank is to learn a mapping function $M : \mathcal{X} \to \Pi_n$, where Π_n is the space of all permutations of size n, such that $\hat{\pi} = M(X)$ is close to π^* for any given X. Note that $\pi(X)$ or alternatively $\pi(Y)$ can be seen to act on a list of items to return a permuted list of items.

2.2 Scoring Function

Directly finding a mapping function M is difficult, as the space over all possible permutations is intractably large. In practice, a score-and-sort approach is used instead. Let $F : \mathcal{X} \to \mathbb{R}^n$ be a scoring function that maps a list of examples X to a list of scores $\hat{Y}$. F induces the permutation π, such that $F(\pi(X))$ is monotonically decreasing.

In the pointwise case, scoring function $F(X)$ can be decomposed into a per-example scoring function as shown in Equation 1, where $f : x \to \mathbb{R}$ maps a feature vector to a real-valued score.

$$F(X) = [f(x_1), f(x_2), ..., f(x_n)] \tag{1}$$

The scoring function F or f is typically parameterized; we use deep neural networks in this paper. The notion of scoring function can be extended to more sophisticated ones like multi-item scoring functions [3], where the scores of a set of examples are computed jointly. For the purpose of this paper, we focus on single-item scoring functions, though our library supports more advanced multi-item scoring functions as well.

2.3 Loss Functions

Learning-to-rank techniques aim to find an optimal scoring function F by minimizing a loss objective. In the machine learning setting, the *empirical risk* for a scoring function F, given a loss function L, is defined as

$$\hat{R}(F) = \frac{1}{N} \sum_{i=1}^{N} L(F(X_i), Y_i) \tag{2}$$

where $\{(X_i, Y_i)\}_{i=1}^N$ are labeled training data. We define the loss functions in learning-to-rank as follows.

A **pointwise** loss is defined over each individual example. For example, the sigmoid cross entropy for binary label $y_j \in \{0, 1\}$ is:

$$L(\hat{Y}, Y) = -\sum_{j=1}^{n} y_j \log(p_j) + (1 - y_j) \log(1 - p_j), \text{ where } p_j = \frac{1}{1 + \exp(-\hat{y}_j)} \tag{3}$$

A **pairwise** loss considers scores of a pair of examples. It includes hinge loss, logistic loss etc. For example the pairwise logistic loss is defined as:

$$L(\hat{Y}, Y) = \sum_{j=1}^{n} \sum_{k=1}^{n} \mathbb{I}(y_j > y_k) \log(1 + \exp(\hat{y}_j - \hat{y}_k))) \tag{4}$$

where $\mathbb{I}(\cdot)$ is the indicator function.

A **listwise** loss is defined over the whole list of examples [7]. Examples include ListNet [5] and ListMLE [28]. For binary labels y_j, the softmax loss defined below is a variant of ListNet loss:

$$L(\hat{Y}, Y) = -\sum_{j=1}^{n} y_j \log(\frac{\exp(\hat{y}_j)}{\sum_{j=1}^{n} \exp(\hat{y}_j)}) \tag{5}$$

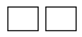

Figure 1: Building a `model_fn` using TensorFlow Ranking library.

2.4 Ranking Metrics

Several standard ranking metrics are used to evaluate ranked lists sorted by scoring functions. In ranking, it is preferable to have fewer errors at higher ranked positions than at the lower ranked positions, which is reflected in many metrics. Our library supports commonly used ranking metrics and we list the following as examples:

$$MRR = \frac{1}{N}\sum_{i=1}^{N} \frac{1}{\min_j\{r_j : y_j > 0\}},\ ARP = \frac{1}{N}\sum_{i=1}^{N} \frac{\sum_{j=1}^{n} y_j r_j}{\sum_{j=1}^{n} y_j}$$

$$DCG = \sum_{j=1}^{n} \frac{2^{y_j} - 1}{\log_2(1 + r_j)},\ NDCG = \frac{1}{N}\sum_{i=1}^{N} \frac{DCG}{DCG_{\pi^*}}$$

where r_j is the rank of x_j in X ranked according to $\hat{Y}$. MRR is the mean of the reciprocal rank of the first relevant example. ARP is the average of positions of examples weighted by their relevance values [30, 27]. DCG is the Discounted Cumulative Gain [13], and NDCG is its normalized version. The loss functions are available in the library via the factory method `tfr.losses.make_loss_fn`.

2.5 Example Weights

Unbiased learning-to-rank [15, 26] looks at dealing with bias in relevance scores arising due to position bias, where users are more likely to click on examples presented higher up the list. One approach to handle this bias is to compute Inverse Propensity Weights (IPW) and use these weights to produce a better ranking estimator. These weights can be estimated per-query or per-example [25, 26] and the losses and metrics are reweighted using the inverse propensity weights to counter this bias.

3 Implementation

Our library is based on TensorFlow. Similar to design patterns in TensorFlow, it provides functions and closures for users to construct models and also allows users to build custom functions if needed. More specifically, we use the TensorFlow Estimator framework [8] to build ranking models. This framework supports both local and distributed training. The core component of this framework is a `model_fn` function that takes features and labels as input and returns loss, prediction, metrics, and training ops, depending on the mode (`TRAIN, EVAL, PREDICT`). In the following, we first describe our input format and then show how we build a ranking model for learning-to-rank.

3.1 Input Format

A ranking problem usually has a context (e.g., a query) and a list of examples. For the sake of simplicity, we assume that all the examples have the same set of features. These features can be sparse or dense. For sparse features, we use embeddings to densify them. After transformation, for a mini-batch, we have each context feature as a 2-D tensor with shape `[batch_size, feature_size]`, where `feature_size` is the length of the vector for a feature, which varies from feature to feature. For each example feature, we have a 3-D tensor with shape `[batch_size, list_size, feature_size]`. Both `batch_size` and `list_size` are constant across features.

Our library allows users to specify features (e.g., dense or sparse) using a TensorFlow library such as `tf.feature_column`. We allow closures for customized feature transformations and provide utility functions for these transformations.

3.2 Model Building

The overall flow of building a `model_fn` is shown in Figure 1. There are two important components:

- **Scoring Function**. We focus on single-item scoring functions and multi-item scoring functions [3] in this paper. A single-item scoring function takes all context features and all features for a single example as input and outputs a score, as defined in Equation 1. A multi-item scoring function extends this to a group of examples. Conceptually, we slice the tensor for each example list into a number of tensors with shape of `[batch_size, group_size, feature_size]`, where `group_size` $= 1$ for pointwise scoring functions. These group features are combined with context features, and passed to the scoring function to generate scores. After the scoring phase, a voting layer is applied to obtain a tensor with shape `[batch_size, list_size]` for scores, as shown in Figure 1 for a mini-batch. The scoring function is a user-specified closure which is passed to the ranking `model_fn` builder.

- **Ranking Head**. The ranking head structure computes ranking metrics and ranking losses, given scores, labels and optionally example weights. The ranking head abstraction allows for the user to specify different losses and metrics, for the same scoring logic, or vice-versa. In our library, both score and labels tensors have the same shape `[batch_size, list_size]`, representing `batch_size` number of example lists. Ranking head also incorporates example weights, described in Section 2.5, which can be per-example with shape `[batch_size, list_size]` or for the entire list with shape `[batch_size]`. Ranking head is available in the library via the factory method `tfr.head.create_ranking_head`.

As shown in Figure 1, our library provides factory methods to create metrics and losses. Furthermore, our APIs allow to specify loss functions when creating a ranking head. This enables the user to switch between different loss functions or combine multiple loss functions easily. More importantly, we provide a builder function that takes a scoring function and a ranking head and returns a `model_fn` to construct a `tf.estimator.Estimator`. When `mode = PREDICT` in `model_fn`, the learned scoring function is exported for serving.

4 Use Cases

In this section, we demonstrate the effectiveness of the TensorFlow Ranking library for two real-world ranking scenarios: *Gmail search* [25, 29] and *document recommendation in Google Drive* [24]. We focus on these scenarios, since in both cases the model is trained based on large quantities of click data, which is beyond the capabilities of the existing open source packages for ranking such as RankLib [10].

Hyperparameters. For the neural network architecture, we consider a simple architecture of a 3-layer feed-forward neural network with ReLU [19] non-linear activation units and Dropout regularization [23]. We consider a pointwise, pairwise and a listwise loss function, from those defined in Section 2.3, and use `Adagrad` [11] for optimizing the parameters of the network, for a learning rate of 0.1.

Evaluation. The models are evaluated using the metrics defined in Section 2.4. Due to the proprietary nature of the models, we only report relative improvements with respect to a pointwise sigmoid cross entropy loss.

4.1 Search

We evaluate several ranking models trained on search logs from Gmail. The privacy of user information is preserved by removal of personal information and anonymization of data using k-anonymization. When a user types a query, five results are shown and user clicks are used as relevance labels for ranking. The set of features consists of dense and sparse features. Some of the sparse features considered are word and character level n-grams derived from queries and email titles. The vocabulary of n-grams is pruned to retain only n-grams that occur across more than k users. This is done to preserve user privacy, as well as to promote a common vocabulary for learning a shared representations across users. In total we collect around 250M queries and use 10% of them for evaluation. The losses and metrics are weighted by Inverse Propensity Weighting [25] computed to counter position bias.

Table 1: Ranking model performance over Gmail Search. Best performance per column is in bold.

	MRR	ARP	NDCG
Sigmoid Cross Entropy	–	–	–
Pairwise Logistic Loss	+1.52	+1.64	+1.00
Listwise Softmax Loss	**+1.80**	**+1.88**	**+1.57**

Table 2: Ranking model performance for Drive Quick Access. Best performance per column is in bold.

	MRR	ARP	NDCG
Sigmoid Cross Entropy	–	–	–
Pairwise Logistic Loss	+0.70	+1.86	+0.35
Listwise Softmax Loss	**+1.08**	**+1.88**	**+1.05**

4.2 Recommendation

Quick Access in Google Drive [24] is a zero-state recommendation engine that surfaces documents currently relevant to the user when she visits the Drive home screen. We evaluate several ranking models trained on user click data over these recommended results. The set of features consists of mostly dense features, as described in Tata et al. [24]. In total we collected around 30M instances and use 10% of them for evaluation.

4.3 Results

The results for Gmail and Google Drive are summarized in Table 1 and Table 2 respectively. From both tables, we observe that a listwise loss performs better than a pairwise loss, which is in turn better than a pointwise loss. This clearly indicates the importance of listwise losses for ranking problems, as they capture the relevancy of the entirety of example list better than their pointwise and pairwise counterparts.

5 Conclusion

In this paper we introduced TensorFlow Ranking – a learning-to-rank library that allows users to define flexible ranking models in TensorFlow. The library is highly configurable, and has easy-to-use APIs for scoring mechanisms, loss functions and evaluation metrics. Unlike the existing learning-to-rank open source packages, which are designed for small datasets, TensorFlow Ranking can be used to solve real-world large-scale ranking problems with hundreds of millions of training examples. We empirically demonstrate its performance on Gmail search and Google Drive document recommendation. We also demonstrate the improvements from using ranking estimators in these real-world applications, highlighting the need for such a library. TensorFlow Ranking is available to the open source community with the hope that it facilitates further academic research and industrial applications.

References

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. Tensorflow: a system for large-scale machine learning. In *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation*, pages 265–283, 2016.

[2] Q. Ai, J. Mao, Y. Liu, and W. B. Croft. Unbiased learning to rank: Theory and practice. In *Proceedings of the 2018 ACM SIGIR International Conference on Theory of Information Retrieval*, pages 1–2, 2018.

[3] Q. Ai, X. Wang, N. Golbandi, M. Bendersky, and M. Najork. Learning groupwise scoring functions using deep neural networks. *arXiv preprint arXiv:1811.04415*, 2018.

[4] C. J. Burges. From RankNet to LambdaRank to LambdaMART: An overview. Technical Report Technical Report MSR-TR-2010-82, Microsoft Research, 2010.

[5] Z. Cao, T. Qin, T.-Y. Liu, M.-F. Tsai, and H. Li. Learning to rank: from pairwise approach to listwise approach. In *Proceedings of the 24th International Conference on Machine Learning*, pages 129–136, 2007.

[6] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *Neural Information Processing Systems, Workshop on Machine Learning Systems*, 2015.

[7] W. Chen, T.-Y. Liu, Y. Lan, Z.-M. Ma, and H. Li. Ranking measures and loss functions in learning to rank. In *Advances in Neural Information Processing Systems*, pages 315–323, 2009.

[8] H.-T. Cheng, Z. Haque, L. Hong, M. Ispir, C. Mewald, I. Polosukhin, G. Roumpos, D. Sculley, J. Smith, D. Soergel, et al. Tensorflow estimators: Managing simplicity vs. flexibility in high-level machine learning frameworks. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1763–1771, 2017.

[9] N. Craswell. Mean reciprocal rank. In *Encyclopedia of Database Systems*, pages 1703–1703. Springer, 2009.

[10] W. B. Croft, J. Callan, J. Allan, C. Zhai, D. Fisher, T. Avrahami, T. Strohman, D. Metzler, P. Ogilvie, M. Hoy, et al. The Lemur project. *Center for Intelligent Information Retrieval, Computer Science Department, University of Massachusetts Amherst*, 140, 2013.

[11] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, 2011.

[12] K. Järvelin and J. Kekäläinen. IR evaluation methods for retrieving highly relevant documents. In *Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 41–48, 2000.

[13] K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems*, 20(4):422–446, 2002.

[14] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM International Conference on Multimedia*, pages 675–678, 2014.

[15] T. Joachims, A. Swaminathan, and T. Schnabel. Unbiased learning-to-rank with biased feedback. In *Proceedings of the 10th ACM International Conference on Web Search and Data Mining*, pages 781–789, 2017.

[16] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu. LightGBM: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems 30*, pages 3146–3154, 2017.

[17] H. Li. Learning to rank for information retrieval and natural language processing. *Synthesis Lectures on Human Language Technologies*, 4(1):1–113, 2011.

[18] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[19] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning*, pages 807–814, 2010.

[20] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in PyTorch. In *Advances in Neural Information Processing Systems, AutoDiff Workshop: The Future of Gradient-Based Machine Learning Software and Techniques*, 2017.

[21] T. Qin and T.-Y. Liu. Introducing LETOR 4.0 datasets. *arXiv preprint arXiv:1306.2597*, 2013.

[22] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[23] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[24] S. Tata, A. Popescul, M. Najork, M. Colagrosso, J. Gibbons, A. Green, A. Mah, M. Smith, D. Garg, C. Meyer, et al. Quick Access: Building a smart experience for Google Drive. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1643–1651, 2017.

[25] X. Wang, M. Bendersky, D. Metzler, and M. Najork. Learning to rank with selection bias in personal search. In *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 115–124, 2016.

[26] X. Wang, N. Golbandi, M. Bendersky, D. Metzler, and M. Najork. Position bias estimation for unbiased learning to rank in personal search. In *Proceedings of the 11th International Conference on Web Search and Data Mining*, pages 610 –618, 2018.

[27] X. Wang, C. Li, N. Golbandi, M. Bendersky, and M. Najork. The LambdaLoss framework for ranking metric optimization. In *Proceedings of The 27th ACM International Conference on Information and Knowledge Management*, 2018.

[28] F. Xia, T.-Y. Liu, J. Wang, W. Zhang, and H. Li. Listwise approach to learning to rank: Theory and algorithm. In *Proceedings of the 25th International Conference on Machine Learning*, pages 1192–1199, 2008.

[29] H. Zamani, M. Bendersky, X. Wang, and M. Zhang. Situational context for ranking in personal search. In *Proceedings of the 26th International Conference on World Wide Web*, pages 1531–1540, 2017.

[30] M. Zhu. Recall, precision and average precision. *Department of Statistics and Actuarial Science, University of Waterloo, Waterloo*, 2:30, 2004.