Announcing Angel 3.0: A Major Milestone Achieving a Full Stack ML Platform

Fitz Wang, Ph.D.
Senior Researcher, Tencent

In this paper, Angel’s maintainer and core developer, Fitz Wang introduces the new features of Angel as part of the major 3.0 release makes Angel a full-stack machine learning platform and is a major milestone in the evolution of the project.
Contents

What is Angel? 3
Angel’s Architecture 5
Angel’s Adoption Inside and Outside Tencent 6
Open Source Statistics 8
New Features in Angel 3.0 9
  Auto Feature Engineering (AFE) 10
  Spark on Angel (SONA) 12
  PyTorch on Angel (PyTONA) 15
  Hyperparameter Tuning (HPT) 18
  Angel Serving 20
Use Case: Short Video Recommendation 22
Use Case: Financial Anti-Fraud 23
Conclusion 24
Get Involved 26
References 27
Acknowledgments 28
Feedback 28
About the Author 29
Angel: A Brief Introduction

Angel, an LF AI Foundation incubation project, is a large-scale distributed machine learning platform designated for sparse data and high dimensional models. What distinguishes Angel from existing machine learning platforms in the industry such as TensorFlow, PyTorch, MxNet, PaddlePaddle and Spark is that:

1. Angel is built on a high-performance Parameter Server (PS). The most distinguishing feature of Angel PS is that it supports parameter server function (PSF), which allows involving the computing power of PS by pushing specific distributed computation tasks to it. Furthermore, compared to other parallel machine learning approaches such as Allreduce, Angel PS does not require all parameters to be stored on a single node. Both the horizontal scalability and the flexibility of parameter server empower Angel PS to deal with high dimensional models with trillions of parameters.

2. Angel has its own math lib designated for computation on high dimensional sparse data, performing 10X faster than breeze on sparse calculation. Both the parameter server and the built-in MLcore of Angel are based on the math lib.

3. Angel’s machine learning algorithms are mainly focused on recommendations and graph learning. The radar chart in Figure 1 illustrates the differences between Angel and other machine learning and deep learning platforms such as TensorFlow, PyTorch and spark. Both TensorFlow and PyTorch are outstanding in terms of deep learning and integrated ecosystem, but are lacking efficient support for sparse data and high dimensional models. Angel can serve as a supplement.
Figure 1: Comparison between Angel and Other Machine Learning Platforms
Angel’s Architecture

The component architecture of Angel 3.0 is demonstrated in Figure 2. The high-performance math library lies at the bottom of the entire system. Both Angel PS and the built-in MLcore are built upon the math library.

As mentioned in the previous section, Angel PS provides efficient, high available and flexible parameter services. In the 3.0 version, Angel supports storing not only matrices but also non-Euclidean graph data on parameter server.

The MLcore is Angel’s built-in computation engine, which supports running machine learning and deep learning algorithms as computing graphs by simply defining the algorithms through JSON. In Angel 3.0, we also supported plugging other computation engines into Angel and introduced PyTorch for graph learning. Above the computation engines stand several distributed execution frameworks, including Angel native, Spark on Angel (SONA) and PyTorch on Angel (PyTONA). Such wrapped execution frameworks allow users to switch to Angel from Spark or PyTorch smoothly with nearly zero cost. At the top of the architecture are two shared service components: AutoML and model serving.
Angel’s Adoption Inside and Outside Tencent

As shown in Figure 3, the number of computing tasks on Angel has increased by 1.5 times over the past 12 months. The number of tasks on Spark on Angel (SONA) has increased by almost 10 times. Given the quickly increasing usage of SONA, we have upgraded SONA in the 3.0 version to make it more user-friendly. Moreover, users of Angel include the most popular mobile and web applications across China, such as Tencent Video, Tencent News and WeChat.

![Task stats](image)

Figure 3: Usage Statistic of Angel since June 2018

We manage an official social media account for Angel to communicate with users outside Tencent. According to user statistics on the account:

- Most of the users are from China, especially from Beijing, Shanghai, Hangzhou, Chengdu and Shenzhen. (Figure 4a)
- More than 100 companies or institutions, including some top IT companies in China such as Sina Weibo, Huawei, Baidu, etc. use Angel in their products or inner systems. (Figure 4b)
Figure 4: Angel’s users outside Tencent. The map (Figure 4a) shows the geometric distribution of users, and the bar plot (Figure 4b) shows Angel’s top 9 organization users.
Open Source Statistics

Figure 5: Angel’s GitHub Statistics and Published Papers

Angel has received more than 4200 stars and 1000 forks on GitHub. There are in total 7 sub projects with more than 500,000 lines of code under Angel-ML project (https://github.com/Angel-ML).

- Sub projects: math2, format, mlcore, Angel, Spark on Angel (SONA), PyTorch on Angel (PyTONA), Angel-serving
- Commits: more than 2000 commits by an active contributor community
- Contributors and committers: 38 contributors in total, including 8 committers

Based on the research and development of Angel, we have published three papers in top conferences including SIGMOD, VLDB and ICDE.
New Features in Angel 3.0

Figure 6 provides an overview of the new features in Angel 3.0. Angel 3.0 implements all components in a full-stack machine learning pipeline, including feature engineering, model training, evaluation, tuning and serving.

Angel’s feature engineering is based on Spark. We add or enhance the feature combination/cross, selection, and re-indexing, and put them into a pipeline to enable Angel’s feature engineering to perform as an automatic workflow.

Angel currently supports two model training engines, Spark on Angel and PyTorch on Angel. The former is based on Angel’s “mlcore” and supports recommendation algorithms, while the latter is based on PyTorch and supports graph learning algorithms.

Angel provides a hyperparameter tuning module with three tuning methods (grid search, random search and Bayesian optimization). At the end of the full-stack machine learning pipeline, users can deploy the trained models for prediction in production environment by Angel serving, a cross-platform serving framework.
AUTO FEATURE ENGINEERING (AFE)

Feature engineering, such as feature cross and selection, has significant importance in industry-level applications of machine learning. Although Spark has provided a bunch of excellent feature selection operators, there are still some limitations that need to be solved. We enhance the feature selection operators in Spark by adding two categories of operators: 1) statistic-based operators, including VarianceSelector and FtestSelector, and 2) model-based operators, including LassoSelector and RandomForestSelector.

A majority of online recommendation systems choose Logistic Regression as their machine learning model for its high throughput and low latency, yet Logistic Regression requires complex feature engineering to achieve high accuracy, which makes automatic feature synthesis essential. However, existing automatic high-order features synthesis methods incur a common drawback of dimension curse. Therefore, we propose Auto Feature Engineering (AFE), an iterative approach to generate high-order features.

In AFE, each iteration is composed of two stages:

- Amplification stage: Cartesian product of arbitrary features
- Reduction stage: feature selection and feature re-indexing

Here is an example of an AFE iteration:

- The features are first amplified through Cartesian product. The number of features will increase quadratically after this step.
- Next, the most important features are selected from the previous step by a specific selector (e.g. VarianceSelector and RandomForestSelector)
- Then, the selected features are re-indexed to reduce the feature space.
- Finally, the generated features are concatenated to the existing features at the end of the iteration.
As can be seen from Figure 7, AFE will increase the feature dimension linearly. Experiment on Higgs dataset demonstrates that our approach is superior to other feature engineering methods in terms of AUC (Table 1).

![Auto Feature Engineering Pipeline Diagram]

**Table 1: Evaluation of feature synthesis**

<table>
<thead>
<tr>
<th></th>
<th>LR</th>
<th>FM</th>
<th>Ours</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUC</td>
<td>0.68</td>
<td>0.69</td>
<td>0.70</td>
</tr>
</tbody>
</table>
SPARK ON ANGEL (SONA)

Similar to Spark MLlib, Spark on Angel is a standalone machine learning library built on Spark (yet it does not rely on Spark MLlib). SONA was based on RDD APIs and, in previous versions, only included a model training step. In Angel 3.0, we introduce various new features to SONA:

- Integration of feature engineering into SONA. Instead of simply borrowing Spark’s feature engineering operators, we add support for long index vector to all the operators to enable training of high dimensional sparse models.
- Seamless connection with automatic hyperparameter tuning.
- Spark-fashion APIs that introduce no cost for Spark users to switch to Angel.
- Support for two new data formats: LibFFM and Dummy.

We also develop a variety of new algorithms on SONA, such as Deep & Cross Network (DCN) and Attention Factorization Machines (AFM). As can be seen from Figure 9, there are significant differences between algorithms on SONA and those on Spark: algorithms on SONA are mainly designated...
for recommendations and graph embedding, while algorithms on Spark tend to be more general-purpose.

Figure 9: Algorithms comparison of Spark and Angel

Figure 10: Programming Example of SONA

Figure 10 provides an example of running distributed machine learning algorithms on SONA, including the following steps:
• Start parameter server at the beginning and stop it at the end.
• Load training and test data as Spark DataFrame.
• Define an Angel model and set parameters in Spark fashion. In this example, the algorithm is defined as a computing graph via JSON.
• Use “fit” method to train the model.
• Use “evaluate” method to evaluate the trained model.

After the training process, SONA will display various model metrics such as accuracy, ROC curve, AUC, etc. Users can save the trained model to a file system for further use.

![Performance Comparison of Angel and TensorFlow](image)

**Figure 11: Performance Comparison of Angel and TensorFlow**

We compare the performances of Angel and TensorFlow on two popular recommendation algorithms, Deep & Wide and DeepFM, using the same resources and Criteo dataset. According to Figure 11, SONA outperforms TensorFlow by 3x on Deep & Wide, while TensorFlow computes slightly faster on DeepFM.
PyTONA is a new component of Angel 3.0 developed for learning node representations over large graphs. Graph Neural Networks have been rapidly emerging over the past two years, with a large number of papers proposing related algorithms, such as GCN, GraphSAGE and GAT, and demonstrating their prominent performances. Algorithm engineers and data analysts in Tencent, the company that owns the largest social networks in China (QQ and WeChat), seek to incorporate network information via Graph Convolutional Networks into various industrial applications. This demand motivated us to develop PyTorch on Angel.

Learning representation on large graphs incurs two challenges: 1) the sparse structure and the gigantic size of networks, and 2) the complex computation logic of graph convolutional networks. The first challenge requires the ability to store the network data and provide easy access to at least two-hop neighbors of any given node. The second challenge requires an automatic differentiation module to realize various graph convolution network algorithms. By examining existing systems, we found that:

- TensorFlow and PyTorch are both excellent as automatic differentiation solutions but lack the ability to deal with sparse and high-dimensional data.
- Angel can tackle sparse and high-dimensional data, but the built-in autograd module of Angel is insufficient for graph convolution networks.

<table>
<thead>
<tr>
<th>System</th>
<th>TensorFlow</th>
<th>PyTorch</th>
<th>Angel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sparse Data &amp; Huge models</td>
<td>x</td>
<td>x</td>
<td>✓</td>
</tr>
<tr>
<td>Auto Differentiation</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Table 2: Comparison of TensorFlow, PyTorch and Angel
Therefore, we combined the advantages of Angel and PyTorch, and launched a new project called “PyTorch on Angel (PyTONA)”.

Figure 12: The architecture of PyTONA

The architecture of PyTONA is demonstrated in Figure 12. The execution pipeline of PyTorch on Angel includes just two steps:

1. Define the deep learning model in Python using TorchScript and generate a model file, for example, gcn.pt.
2. Submit the model file to PyTorch on Angel through a script and do distributed training.

There are three major components in PyTONA:

1. Angel PS: stores model parameters, network structures and node features, providing access to two-hop neighbors of nodes.
2. Spark Driver: creates, initializes and saves model parameters.
3. Spark Worker: Pulls model parameters, network structures, node features or sampled neighbors from PS, executes mini-batch training through PyTorch C++ APIs and pushes the derived gradients to PS.
There is no need for end users (e.g. algorithm engineers) to know the details of distributed training on PyTONA. The only thing they need to focus on is how to implement the logic of deep learning algorithms through the interface of TorchScript. The following picture provides an example of implementing a two-layer GCN through PyTorch on Angel.

```python
class GCN2(torch.jit.ScriptModule):
    def __init__(self, input_dim, hidden_dim, num_classes):
        super(GCN2, self).__init__()
        self.conv1 = GCNConv2(input_dim, hidden_dim)
        self.conv2 = GCNConv2(hidden_dim, num_classes)

    @torch.jit.script_method
    def forward(self, x, first_edge_index, second_edge_index):
        # type: (Tensor, Tensor, Tensor) -> Tensor
        x = F.relu(self.conv1(x, second_edge_index))
        x = F.dropout(x, training=self.training)
        x = self.conv2(x, first_edge_index)
        return F.log_softmax(x, dim=1)

    @torch.jit.script_method
    def loss(self, outputs, targets):
        return F.nll_loss(outputs, targets)

    def get_training(self):
        return self.training

if __name__ == '__main__':
    gcn = GCN2(233, 128, 2)  # huge
    gcn.save('gcn2_huge.pt')
```

*Figure 13: An Example Implementation of GCN on PyTONA*
HYPERPARAMETER TUNING (HPT)

There are basically two types of traditional hyperparameter tuning approaches (Figure 14):

- **Grid search**: grid search separates the search space into grids with a fundamental assumption that the distributions of hyperparameters are uniform. Though intuitive, grid search has two significant drawbacks: 1) the computing cost increases exponentially with respect to the number of parameters, and 2) the distributions of hyperparameters are usually not uniform. Thus, in many cases, grid search might spend too much effort on optimizing less important hyperparameters.

- **Random Search**: random search randomly samples a sequence of combinations of hyperparameters from the configuration space, and evaluates the sampled combination. Although this approach will more likely pay attention to more critical hyperparameters, there is still no guarantee of finding the optimal combination.

![Grid Layout](image1.png) ![Random Layout](image2.png)

*Figure 14: Grid search and random search*

Different from the traditional modeless methods, Bayesian optimization uses a less computationally-expensive surrogate function to approximate the original target function. In Bayesian optimization, the surrogate function
generates the probabilistic mean and variance of a given hyperparameter combination. Then an acquisition function will evaluate the expected loss or improvement and scores the generated combination. Such a probabilistic interpretation approach enables Bayesian optimization to find the optima with much fewer evaluations on target function.

Angel 3.0 includes both the traditional approaches and Bayesian optimization. To be specific, we have implemented the following Bayesian optimization functions:

- Surrogate function: besides the two major surrogate functions (Gaussian process and random forest), we also implement the EM+LBFGS to optimize the hyperparameters in kernel functions of the Gaussian process.
- Acquisition function: we have implemented several popular acquisition functions, including Probability of Improvement (PI), Expected Improvement (EI) and Upper Confidence Bound (UCB).

Since evaluating the target function is computationally expensive, evaluation on a hyperparameter combination can be stopped early if it is observed to perform poorly in several beginning iterations. This early stopping strategy is also implemented in the 3.0 release.

<table>
<thead>
<tr>
<th></th>
<th>Random</th>
<th>Grid</th>
<th>GP</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUC</td>
<td>0.926</td>
<td>0.924</td>
<td>0.933</td>
</tr>
</tbody>
</table>

*Table 3: Performance of Different Hyperparameter Tuning Methods*

Consistent with the previously reported works, our experiments on Logistic Regression shows that Bayesian optimization outperforms random search, and random search performs slightly better than grid search.
ANGEL SERVING

In order to meet the emerging needs of an efficient model service that supports application in production environments, in Angel 3.0 we introduce Angel Serving, an efficient model serving system standing on the downstream side of Angel’s full-stack machine learning pipeline, making Angel’s ecosystem a closed loop. Figure 15 demonstrates the architecture of Angel Serving.

![Figure 15: The Architecture of Angel Serving](image)

Angel Serving can be accessed via both gRPC and Restful API. Angel Serving is a general machine learning serving framework, which means that models from other third-party machine learning platforms can also be served on Angel Serving through a pluggable mechanism. Angel Serving currently supports models from three types of platforms: Angel, PyTorch, and platforms with PMML format. Through the PMML Servable, Angel can provide serving of models from Spark and XGBoost. Inspired by TensorFlow Serving, we also provide fine-grained version control strategies: earliest, latest and specified versions. Apart from version control, Angel Serving also provides fine-grained service monitoring metrics, including:
- QPS: Query per second
- Success/Total requests
- Response time distribution
- Average response Time

<table>
<thead>
<tr>
<th></th>
<th>Angel Serving</th>
<th>TensorFlow Serving</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total time(s)</td>
<td>56</td>
<td>59</td>
</tr>
<tr>
<td>Total time(s)</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Avg(ms)</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>99% consume(ms)</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>QPS</td>
<td>1900</td>
<td>1800</td>
</tr>
</tbody>
</table>

*Table 4: Performance Comparison of Angel Serving and TensorFlow Serving*

Table 4 exhibits the performances of Angel Serving and TensorFlow Serving on 100,000 prediction requests for a DeepFM model with 1 million features. The total runtimes of Angel Serving and TensorFlow Serving are 56 and 59 seconds respectively. Both serving systems have an average response time of 2 milliseconds. The QPS of Angel Serving is 1,900, while that of TensorFlow Serving is 1,800. The result above provides strong evidence that Angel Serving is comparable with TensorFlow Serving, even slightly better.
This is a use case from Tencent’s short video department. The user’s video play log and context information are forwarded to Kafka in real-time, and the stream computing engine, Storm, subscribes to the data from Kafka. Storm is a real-time feature generator that queries user and video profiles from an offline key-value store and concatenates them to generate features. The generated features are transmitted to the online training system to update the online model; At the same time, those features are dumped into HDFS as an input of offline training. The offline model is used to initialize the online system and reset the online system when an abnormality occurs in the online system. The recommendation algorithm used is FM with dimension of 63,611 and 2.4 billion samples. The Spark-based training would cost 10+ hours before adopting Angel, while the runtime is reduced to 1 hour using Angel.
Use Case: Financial Anti-Fraud

Financial fraud detection is a common use case of large-scale graph learning. The network data is heterogeneous, including several types of edges:

- **Payment edges**: a payment edge between user A and B denotes a payment/transaction relationship between them.
- **Device edges**: a device edge indicates that the two users on the edge once shared the same device.
- **Wi-Fi edges**: a Wi-Fi edge between user A and B demonstrates that they once accessed the Internet via the same Wi-Fi.

Financial fraudsters usually share devices and Wi-Fi, generating communities via corresponding edges. The fast unfolding algorithm on Angel is adopted to discover such communities. Downstream fraud risk models can then use the discovered communities, user profiles and network features as input to learn and derive anti-fraud strategies. The graph data includes 1.5 billion nodes and 20 billion edges. The previous implementation based on Spark GraphX takes about 20 hours to finish computing, while Angel only takes 5 hours.
Conclusion

In this paper, we introduce the distinguishing features of Angel: 1) efficient support for sparse data and high-dimensional models, 2) recommendations and graph learning algorithms. We also report what we have achieved since the release of Angel 2.0:

- Inside of Tencent: use and task increase 2.5x times
- Outside of Tencent: more than 100 companies or institutions use Angel in their products or inner systems
- Open Source: 4200+ stars, 7 sub-projects, 1100+ forks, 2000+ commits

This paper is announcing Angel 3.0. This new release makes Angel a full stack machine learning platform. New features of Angel 3.0 include:

- Auto feature engineering: New feature selection and combination operators are introduced. What’s more, we put feature synthesis, selection and re-index in a pipeline, and generate High Order features in an iterative fashion. In this way, we can do feature engineering automatically.

- New or enhanced computation engines:
  - SONA (enhanced): Feature engineering is enhanced by introducing long index vector. All the algorithms are Spark style APIs. The algorithms in SONA are different from Spark, so SONA can serve as a supplement to Spark.
  - PyTONA (new): This engine is introduced for graph learning. Two GNN algorithms, GCN and GraphSAGE, are available currently. Recommendation algorithms are also available. PyTorch adopts a Python interface, hence it is user friendly.
  - AutoML: Angel 3.0 includes three types of hyperparameter tuning algorithms, grid search, random search and Bayesian Optimization. Those methods show excellent performance in practice.
• Angel Serving: Angel provide a cross-platform server framework. Models from Angel, PyTorch and Spark can serve on it. The performance of Angel serving is comparable with TensorFlow serving.

• Kubernetes support: to enable it to run in the cloud, Angel 3.0 supports Kubernetes.

In summary, Angel is designed for sparse data and huge model scenarios. If you have a big high dimensional data set, Angel is a better choice.
Get Involved

Angel is an open source project incubated by the LF AI Foundation. We appreciate all contributions, e.g., bug reporting, code changes, reviewing, testing and documentation. Please feel free to join us and engage in our community.

Github: https://github.com/Angel-ML/angel
Web: https://github.com/Angel-ML/angel/wiki
Charter: https://lists.deeplearningfoundation.org/g/angel-tsc/wiki/Project-Documentation
Mail Lists: https://lists.lfai.foundation/g/angel-announce
Wiki: https://wiki.lfai.foundation/display/ANGEL/Angel+Home
References

PS2: Parameter Server on Spark, Zhipeng Zhang, Bin Cui, Yingxia Shao, Lele Yu, Jiawei Jiang, Xupeng Miao, SIGMOD 2019

An Experimental Evaluation of Large Scale GBDT Systems, Fangcheng Fu, Jiawei Jiang, Yingxia Shao, Bin Cui, VLDB 2019

MLlib*: Fast Training of GLMs Using Spark Mllib, Zhipeng Zhang, Jiawei Jiang, Wentao Wu, Ce Zhang, Lele Yu, Bin Cui, ICDE 2019


Acknowledgments

The author would like to express his sincere appreciation to Jeremy Jiang, Lele Yu, Gdp Ouyang, Geoff Cheng, Junping Du, Paynie Xiao, Peng Chen and Ehome Liu for their valuable reviews and feedback. This paper has benefited immensely from the experiences and contributions of all reviewers.

Feedback

Suggestions for improvement will be appreciated. Please send comments to the author directly via fitzwang@tencent.com.
ABOUT THE AUTHOR

Dr. Fitz Wang is a Senior Researcher at Tencent focusing on Machine Learning platforms where he leads the development of math lib, machine learning library (mlcore), and Angel serving. He is a core developer on the Angel open source project, an LF AI Foundation incubation project, and leads its Technical Steering Committee. Prior to Tencent, Dr. Wang worked at Huawei as a Big Data and Database development engineer, mainly engaged in the development of Spark SQL. He graduated in 2005 from Wuhan University with a Ph.D. in BioInformatics.

GitHub: https://github.com/wangcaihua
LinkedIn: https://www.linkedin.com/in/fitzwang