Swift for TensorFlow

First-Class Machine Learning in Swift

莲叔(aaaron7)

About me

- 莲叔 / aaaron7
- SwiftGG 成员，目前没有参与翻译工作，主要负责在群里灌水；
- 目前供职于 UC 短视频组；
- 兴趣：Swift，函数式编程，机器学习；
- 自我要求：争取不坑

Roadmap

Intro

- Why
- Swift for TensorFlow

How

Magic behind

- Graph Program Extraction

Example

- Linear Classification
- Collaborative Filtering

Why

Why Swift

As we know

Swift has a few problems

swift 1.2 到swift 2.0 升级之路上的坑

原 Swift3.0的坑

升级到Swift 4.0可能遇到的坑总结

An internal error occurred. Source editor functionality is limited. Attempting to restore...

Why Swift

但对我们来说这是问题吗?

不是。

Why Swift for Machine Learning

Server-side Swift get increasing attention.

Swift do a better job.

- High performance
- Open source with active community
- Productive & Safety

Next Big Thing on Server

ML based server apps booms

Recommender System | Text Analysis | Audio/Video Understanding | Super Resolution | Chatbot

Main-stream Language

Swift

C++

- Safe & Performanc
- Less productive

Python

- Easy to write
- Also easy to write rubbish code
- No safety

If Swift has built-in support for machine learning

Swift for TensorFlow

Also known as TFiwS for brief. (from official Youtube Channel)

Intro

- First released on TFDevSummit 2018
- Open source
- TensorFlow Ecosystem
- First-class Machine Learning, Not just a TensorFlow API wrapper
- Support CPU / GPU / Cloud TPU
- Still in early stage, active developing
- Combine usability & performance
- Work nicely with Xcode Playground, support macOS & Ubuntu currently

Playground

```
import TensorFlow

let m : Tensor<Float> = [[1,2,3],[4,5,6]]
m + 1
m * 2
m * m
m • m.transposed()

let bm = Tensor<Float>(ones : [1024,1024])
bm • bm
```

```
"[[1.0, 2.0, 3.0], [4.0, 5.0, 6.0]]"
"[[2.0, 3.0, 4.0], [5.0, 6.0, 7.0]]"
"[[2.0, 4.0, 6.0], [8.0, 10.0, 12...
"[[1.0, 4.0, 9.0], [16.0, 25.0, 36.
"[[14.0, 32.0], [32.0, 77.0]]"

"[[1.0, 1.0, 1.0, 1.0, 1.0, 1.0,...
"[[1024.0, 1024.0, 1024.0, 1...
```

"Script Language Feeling" **is very important for usability.**

WTF it is ???

Core Ideas

Two TF Modes

Graph

```python
import tensorflow as tf

x = tf.constant([[1,2,3], [4,5,6]])
xt = tf.transpose(x)
y = tf.matmul(x, xt)

with tf.Session() as sess:
    print sess.run(y)
```

Operations just build the graph structure, will not be executed until session.run.

Pros:

- **High performance.**
- **Strongly optimized.**

Cons:

- **Hardly debug. (can't print intermediate value)**
- **Poor usability.**

Eager Execution

```python
import tensorflow as tf
tf.enable_eager_execution()

x = tf.constant([[1,2,3],[4,5,6]])
y = tf.matmul(x, tf.transpose(x))
print(y)
```

Each operation will be executed intermediately

Pros

- **Define-by-run, capable of natrual control flow**
- **Debug friendly, capable of value printing step by step.**

Cons:

- **No optimization, poor performance**

Now, we have the Third mode

TFiwS Mode

```
let x : Tensor<Float> = [[1,2,3],
                          [4,5,6]]
var y = Tensor<Float>(zeros:
                        x.shape)
print(y)
if x.sum() > 100{
    y = x
}else{
    y = x * x
}
print(y)
```

- Eager-style code, but run as graph.
- With full optimization from Graph mode.
- Generate graph in compiling stage, then intelligently coordinate it with program during execution.
- Transparent graph logic to engineer.
- Capable of putting the ***`print`*** to anywhere you like. TFiwS will take care of everything.

“Swift 的东西不都这样吗？ ”

“意思是这玩意儿又快又好用呗？ ”

Magic Behind

Magic Behind

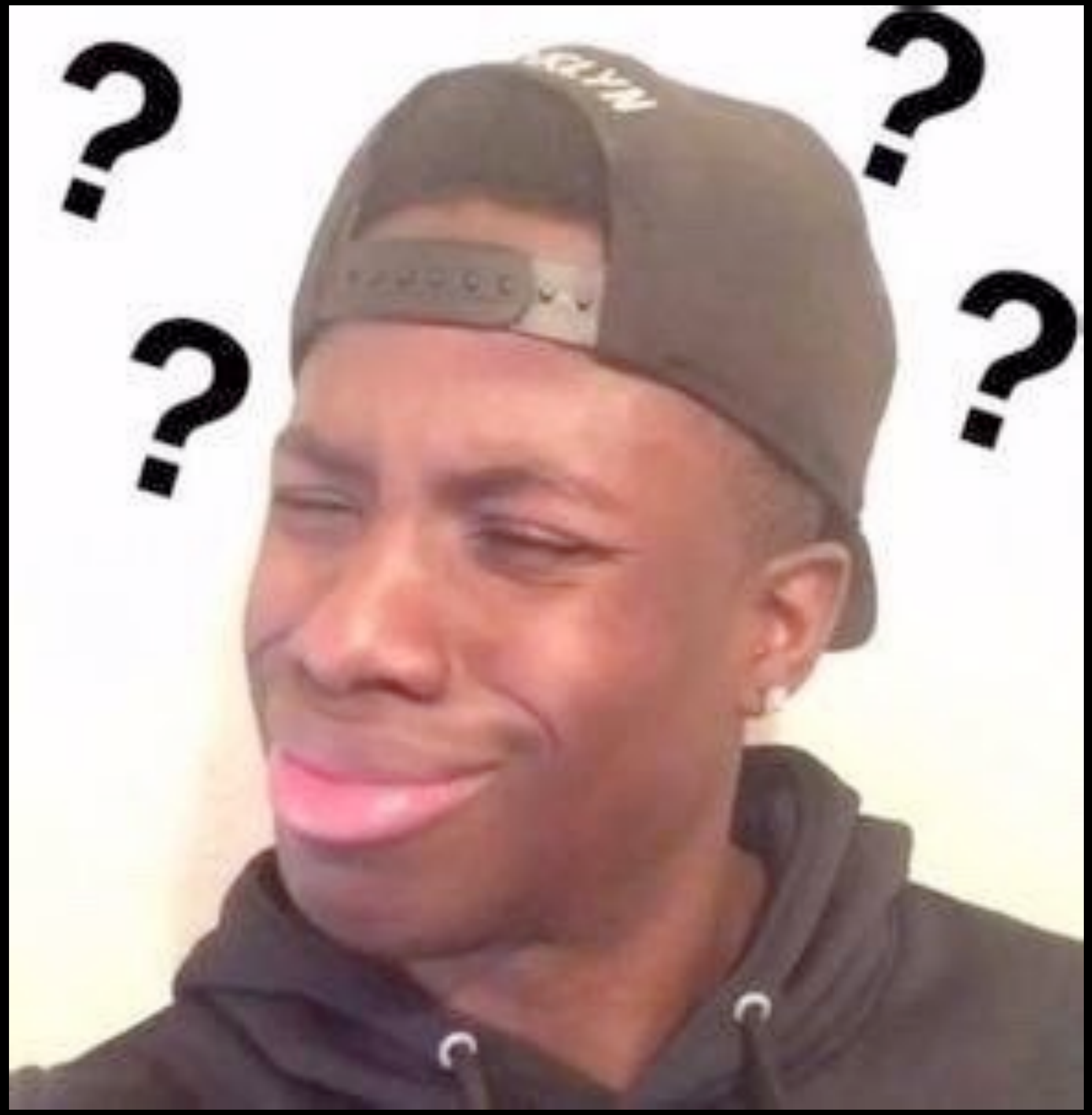
Build graph from code

- Can we directly bridge Swift Tensor class to TensorFlow graph node?

Co-executing code & graph, feeling like eager execution.

- This is the key to enable natural control flow and debug usability
- Therefore, we can’t. Direct bridging is exactly traditional graph mode. Not TFiwS mode.

TFiwS use program slicing technology to extract Tensor operation from code by compiler transformation



WTF are you talking about?

Example

```
typealias FloatTensor = Tensor<Float>

func linear(x : FloatTensor, w : FloatTensor, b : FloatTensor) -> FloatTensor
{
    let tmp = w • x
    let tmp2 = tmp + b
    return tmp2
}
```

How to build the corresponding graph for above code automatically?

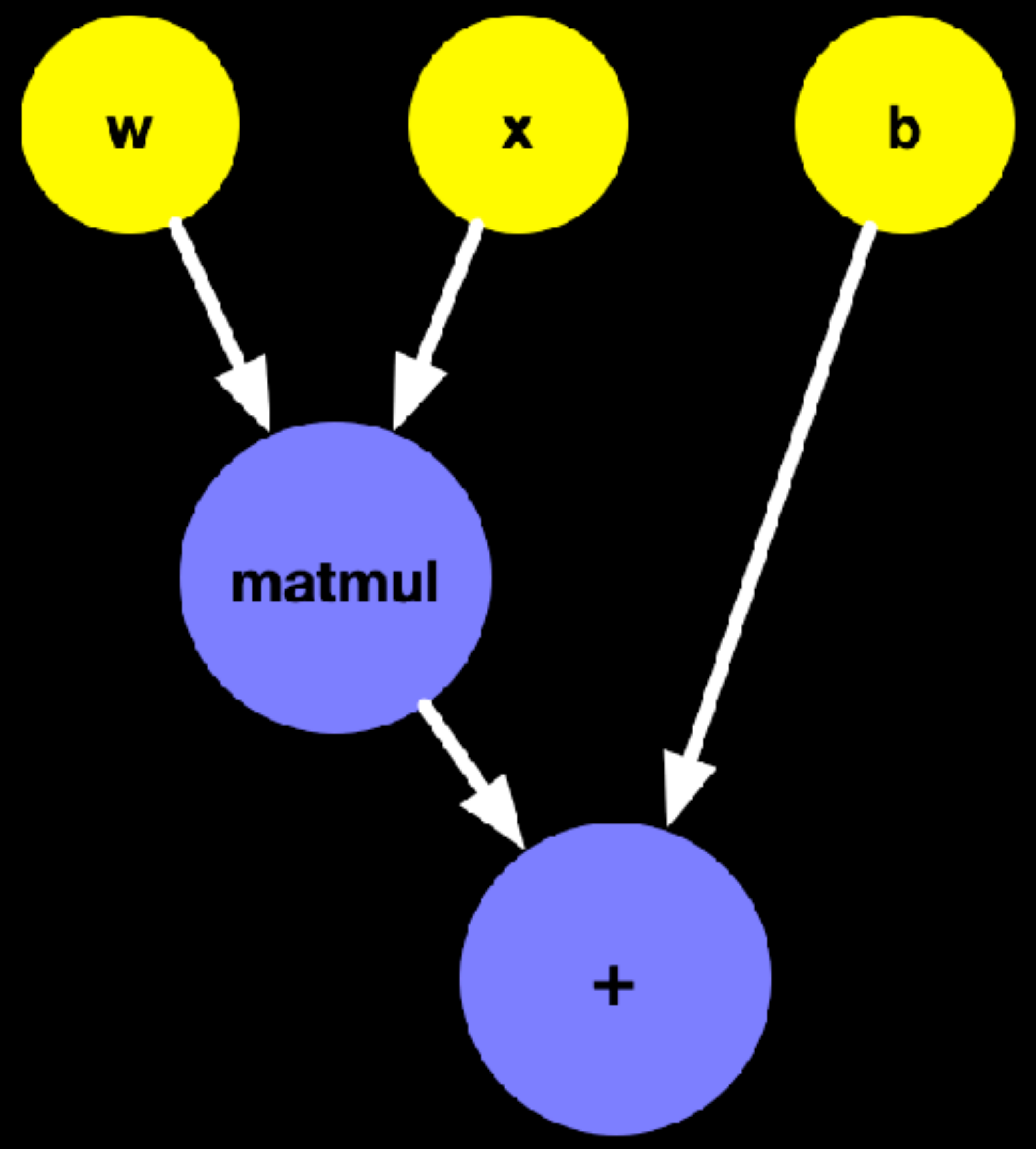
Since there is no runtime shit in Swift

Can directly deduct by AST traverse

```
(import_decl range=[atswift.swift:8:1 - line:8:8] 'Foundation')
(import_decl range=[atswift.swift:9:1 - line:9:8] 'TensorFlow')
(typealias range=[atswift.swift:11:1 - line:11:37] "FloatTensor" interface type='FloatTensor.Type' access=internal type='Tensor<Fl
(func_decl range=[atswift.swift:13:1 - line:18:1] "linear(x:w:b:)" interface type='(FloatTensor, FloatTensor, FloatTensor) -> Floa
  (parameter_list
    (parameter "x" apiName=x type='FloatTensor' interface type='FloatTensor')
    (parameter "w" apiName=w type='FloatTensor' interface type='FloatTensor')
    (parameter "b" apiName=b type='FloatTensor' interface type='FloatTensor') range=[atswift.swift:13:12 - line:13:62])
  (result
    (type_ident
      (component id='FloatTensor' bind=atswift.(file).FloatTensor@atswift.swift:11:11)))
  (brace_stmt range=[atswift.swift:14:1 - line:18:1]
    (pattern_binding_decl range=[atswift.swift:15:5 - line:15:26]
      (pattern_named type='Tensor<Float>' 'tmp')
      (call_expr type='FloatTensor' location=atswift.swift:15:15 range=[atswift.swift:15:15 - line:15:26] nothrow arg_labels=_:_:
        (declref_expr type='(Tensor<Float>, Tensor<Float>) -> Tensor<Float>' location=atswift.swift:15:15 range=[atswift.swift:15:
 map generic_signature=<Scalar where Scalar : Numeric, Scalar : AccelerableByTensorFlow> (substitution Scalar -> Float))] function_re
        (tuple_expr type='(FloatTensor, FloatTensor)' location=atswift.swift:15:21 range=[atswift.swift:15:21 - line:15:26] names=
          (declref_expr type='FloatTensor' location=atswift.swift:15:22 range=[atswift.swift:15:22 - line:15:22] decl=atswift.(fil
          (declref_expr type='FloatTensor' location=atswift.swift:15:25 range=[atswift.swift:15:25 - line:15:25] decl=atswift.(fil

    (var_decl range=[atswift.swift:15:9 - line:15:9] "tmp" type='Tensor<Float>' interface type='Tensor<Float>' access=private let

    (pattern_binding_decl range=[atswift.swift:16:5 - line:16:22]
      (pattern_named type='Tensor<Float>' 'tmp2')
      (binary_expr type='Tensor<Float>' location=atswift.swift:16:20 range=[atswift.swift:16:16 - line:16:22] nothrow
```

Generate graph in compiling stage

- Compiler will generate TF graph based on static analysis result.
- **Static type system of Swift** enable the compiler knowing every Tensor operation.
- This is why Swift is a good choice for the approach rather than Python.



Reference: Swift for TensorFlow white paper document

After Compiling

```
func linear(x : FloatTensor, w : FloatTensor, b : FloatTensor) -> FloatTensor
{
    let tmp = w • x
    let tmp2 = tmp + b
    return tmp2
}
```

```
func linear(x : FloatTensor, w : FloatTensor, b : FloatTensor) -> FloatTensor
{
    let result = execTensorFlowGraph("graph_generate_before")
    return result
}
```

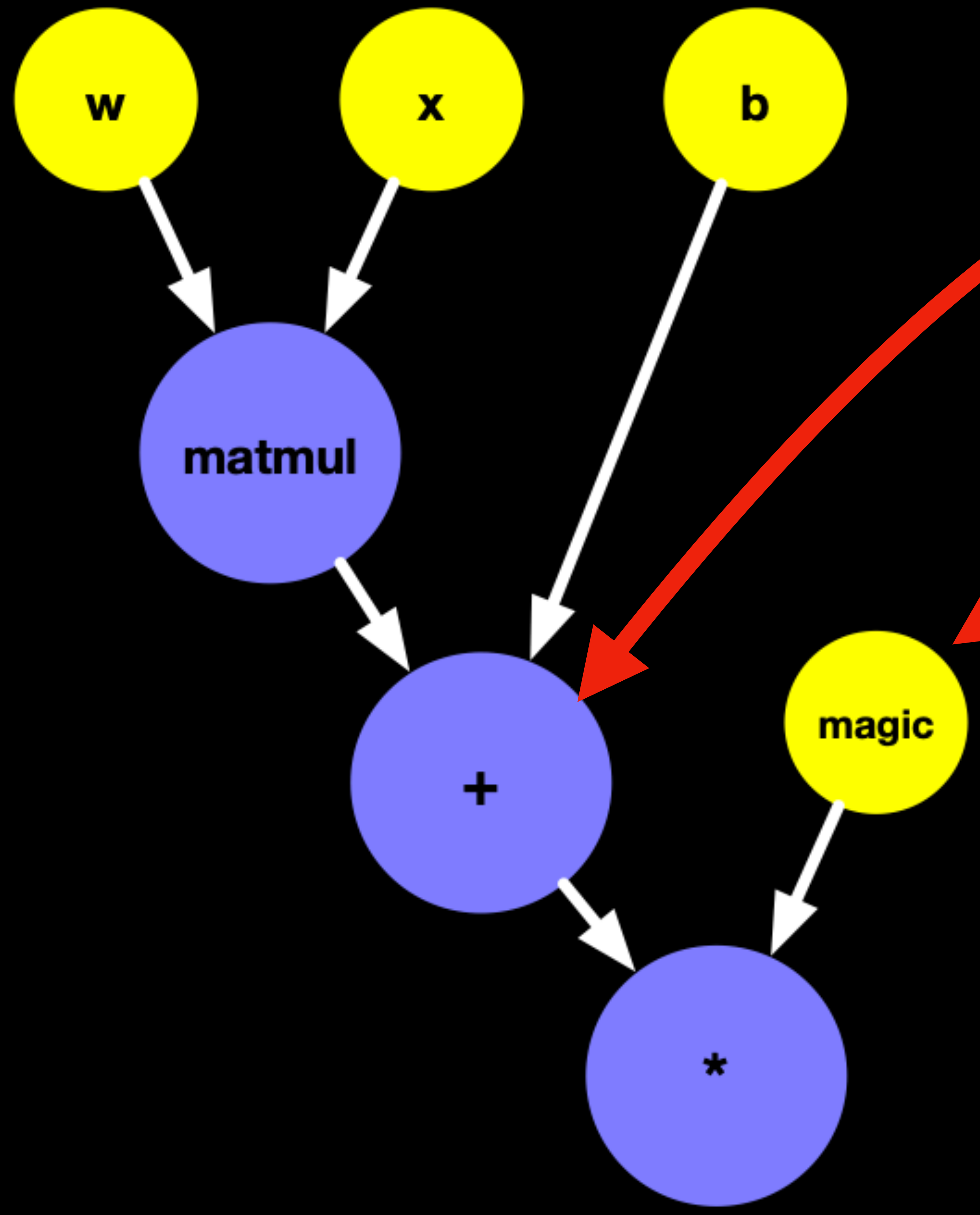
A little more complicated

```
func linear(x : FloatTensor, w : FloatTensor, b : FloatTensor) -> FloatTensor
{
    let tmp = matmul(x, w)
    let tmp2 = tmp + b
    print(tmp2)
    let tmp3 = tmp2 * magicNumberGenerateFromTensor(x: tmp2)
    return tmp3
}

func magicNumberGenerateFromTensor(x : FloatTensor) -> Float
{
    return 3.0
}
```

Debug tensor

Mix tensor operation with host logic.
Host & graph is replying on each other.

The Problem



How & **When** to automatically extract the intermediate value from graph for printing ?

During compiling, how do we know **WTF** it is ?

Program Slice Stage 1

Original

```
func linear(x : FloatTensor, w :
FloatTensor, b : FloatTensor) -> FloatTensor
{
    let tmp = matmul(x, w)
    let tmp2 = tmp + b
    print(tmp2)
    let tmp3 = tmp2 *
magicNumberGenerateFromTensor(x: tmp2)
    return tmp3
}
```

Graph Partition(remove host code)

```
func linear(x : FloatTensor, w :
FloatTensor, b : FloatTensor) ->
FloatTensor
{
    let tmp = matmul(x, w)
    let tmp2 = tmp + b
    //REMOVED: print(tmp2)
    tfop("send", tmp2)
    let result = tfop("receive")
    // REMOVED:
magicNumberGenerateFromTensor(x: tmp2)
    let tmp3 = tmp2 * result
    return tmp3
}
```

send/recv is standard TensorFlow node operation, being used to distributed learning

Program Slice Stage 2

Original

```
func linear(x : FloatTensor, w :
FloatTensor, b : FloatTensor) -> FloatTensor
{
    let tmp = matmul(x, w)
    let tmp2 = tmp + b
    print(tmp2)
    let tmp3 = tmp2 *
magicNumberGenerateFromTensor(x: tmp2)
    return tmp3
}
```

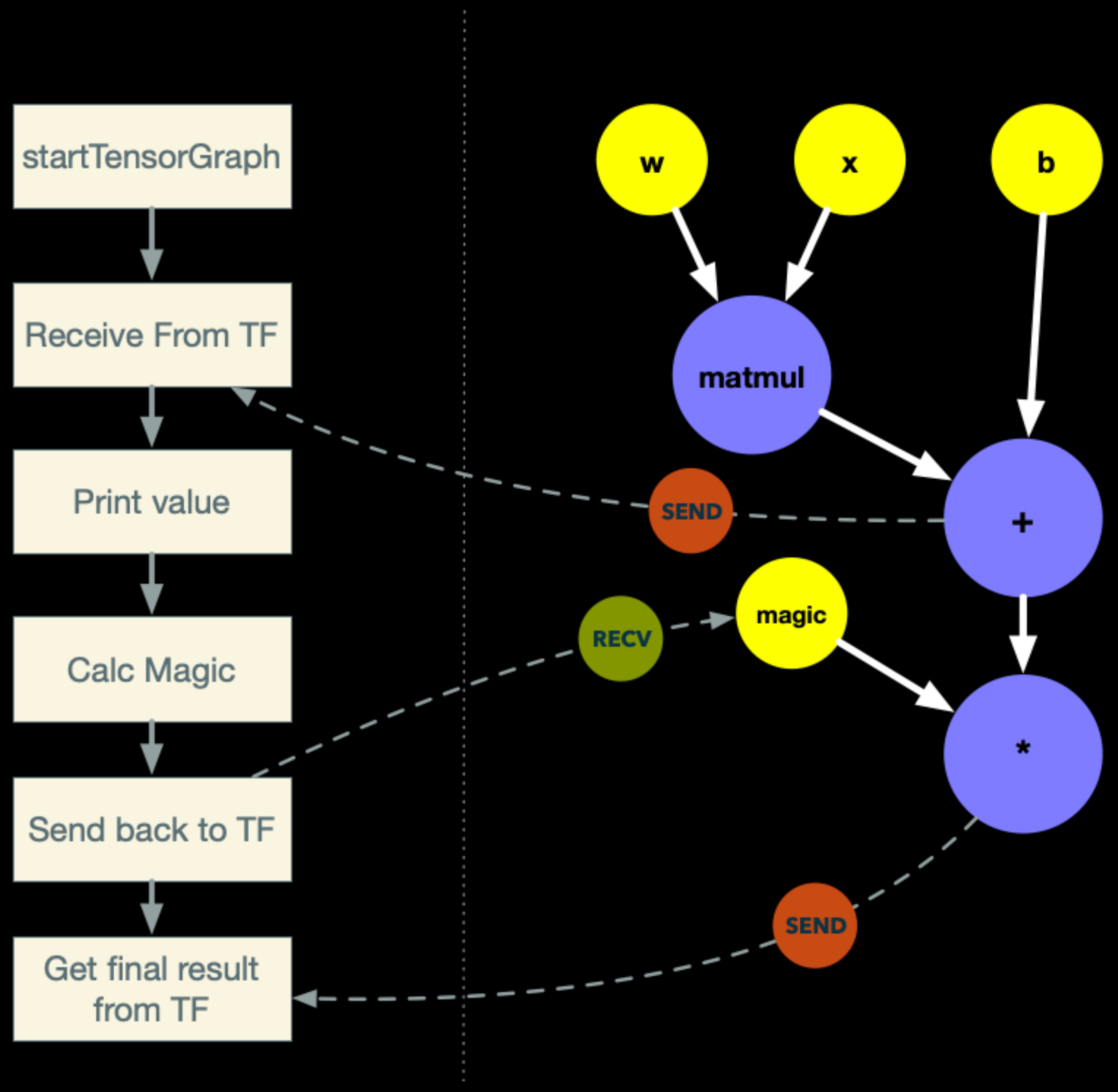
Host Partition(remove tensor code)

```
func linear(x : FloatTensor, w : FloatTensor, b :
FloatTensor) -> FloatTensor
{
    let tensorProgram = startTensorGraph(graphName:
"GeneratedGraphName")
    //REMOVED: let tmp = matmul(x, w)
    //REMOVED: let tmp2 = tmp + b
    let tmp2 = receivedFromTensorFlow(tensorProgram)
    print(tmp2)
    let result = magicNumberGenerateFromTensor(x: tmp2)
    sendToTensorFlow(tensorProgram, result)
    let tmp3 = finishTensorGraph(handle: tensorProgram)
    //REMOVED: let tmp3 = tmp2 *
magicNumberGenerateFromTensor(x: tmp2)
    return tmp3
}
```

Now we have

Trivial eager-style code

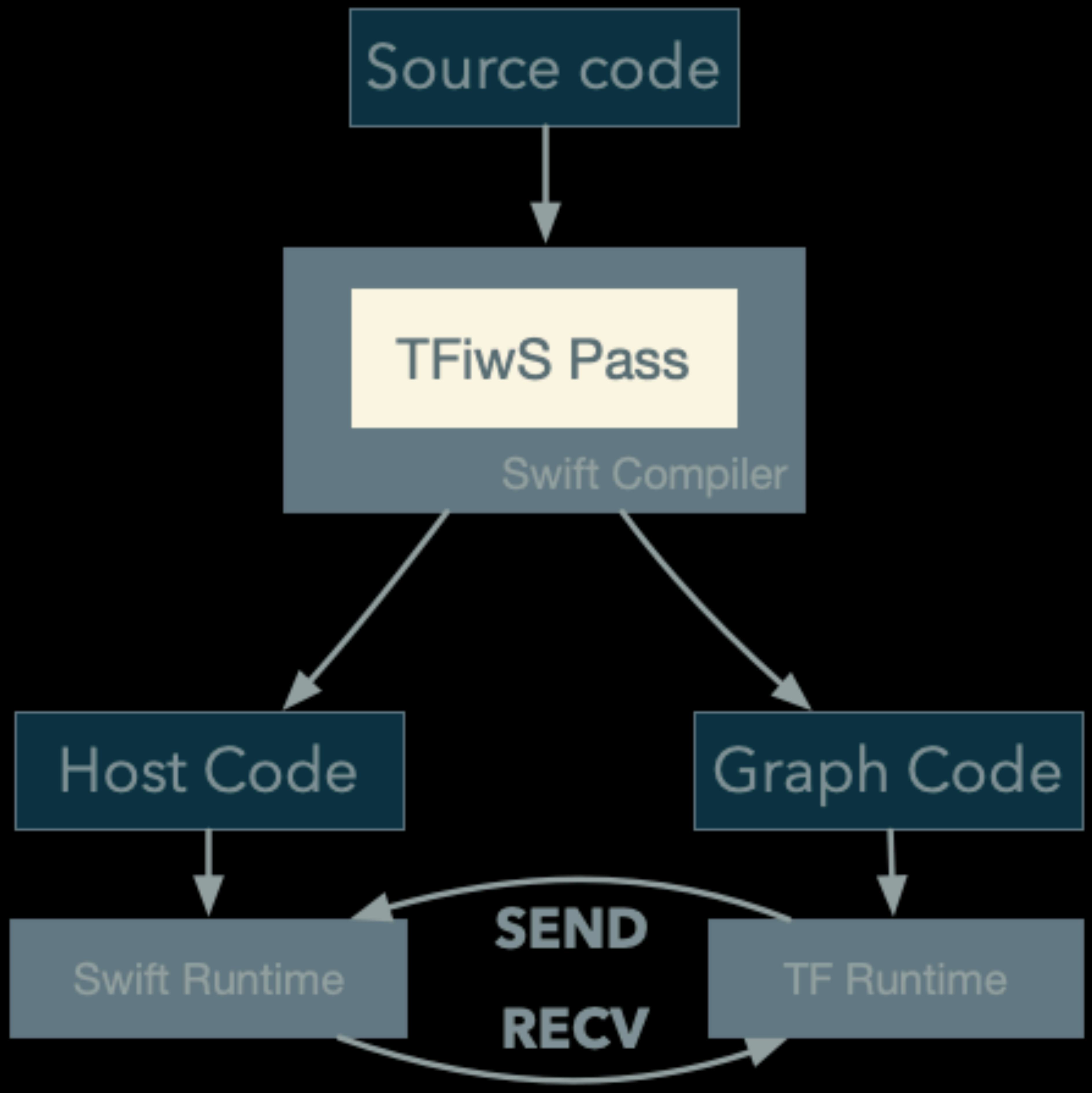
- No tensor operation



Pure graph

- Executed by TensorFlow Runtime
- No host operation
- Can be run on GPU/TPU/ Distributed Device.

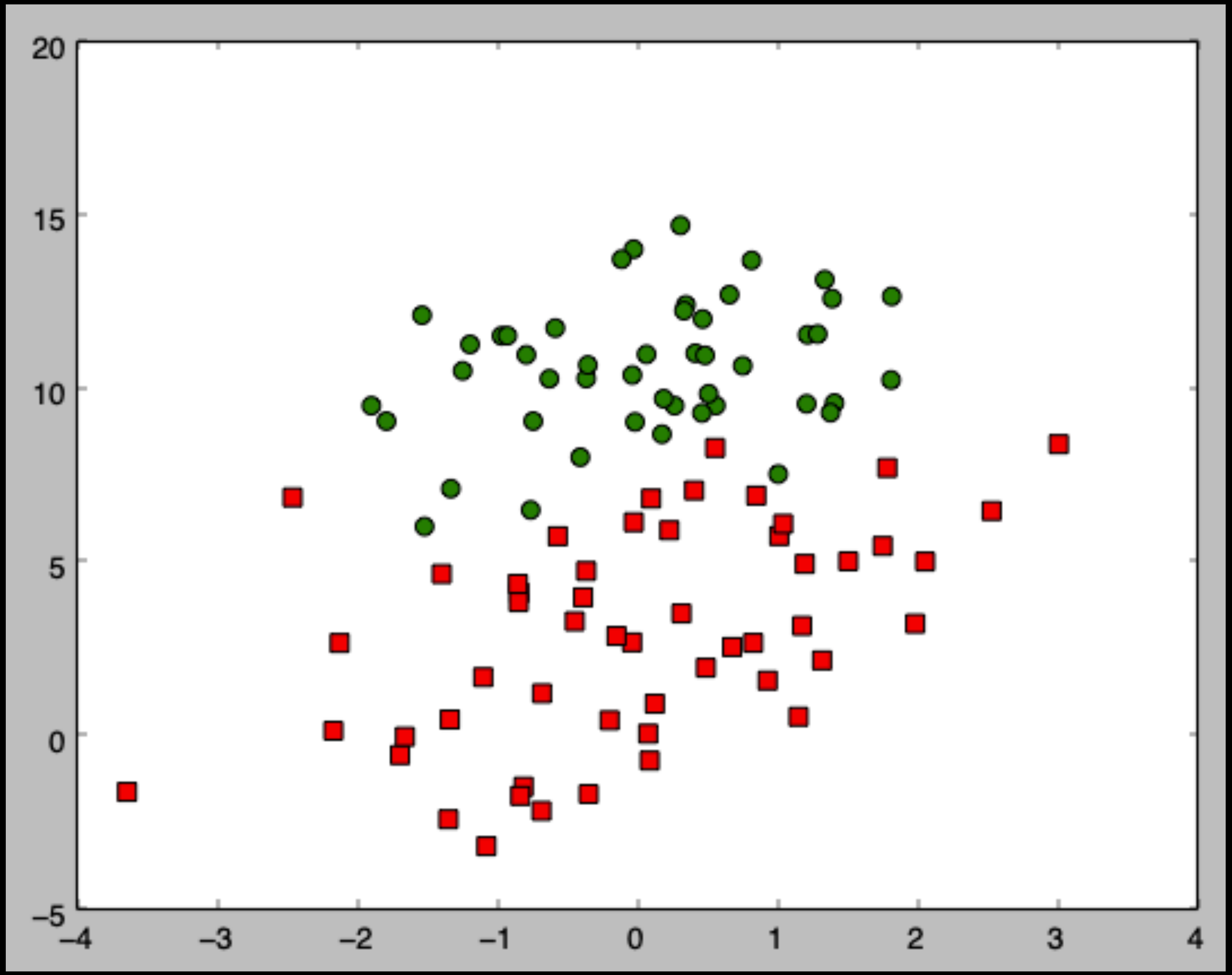
Summary



- Based on Compiler Based Program Slicing and TensorFlow Distributed Infra
- All communication & transformation are transparent to developer.
- Developer only need to write the trivial eager code, and then enjoy the benefits from both graph and eager mode.
- Swift compiler(with tf support) will take care of everything for you.

Example

Linear Classification



- A part of the ocean being found be contaminated, now we have the left data after 53 times observations
- Point - clean, Cube - contaminated
- Given new coordinates, can we predict whether it’s clean or not?

Analysis

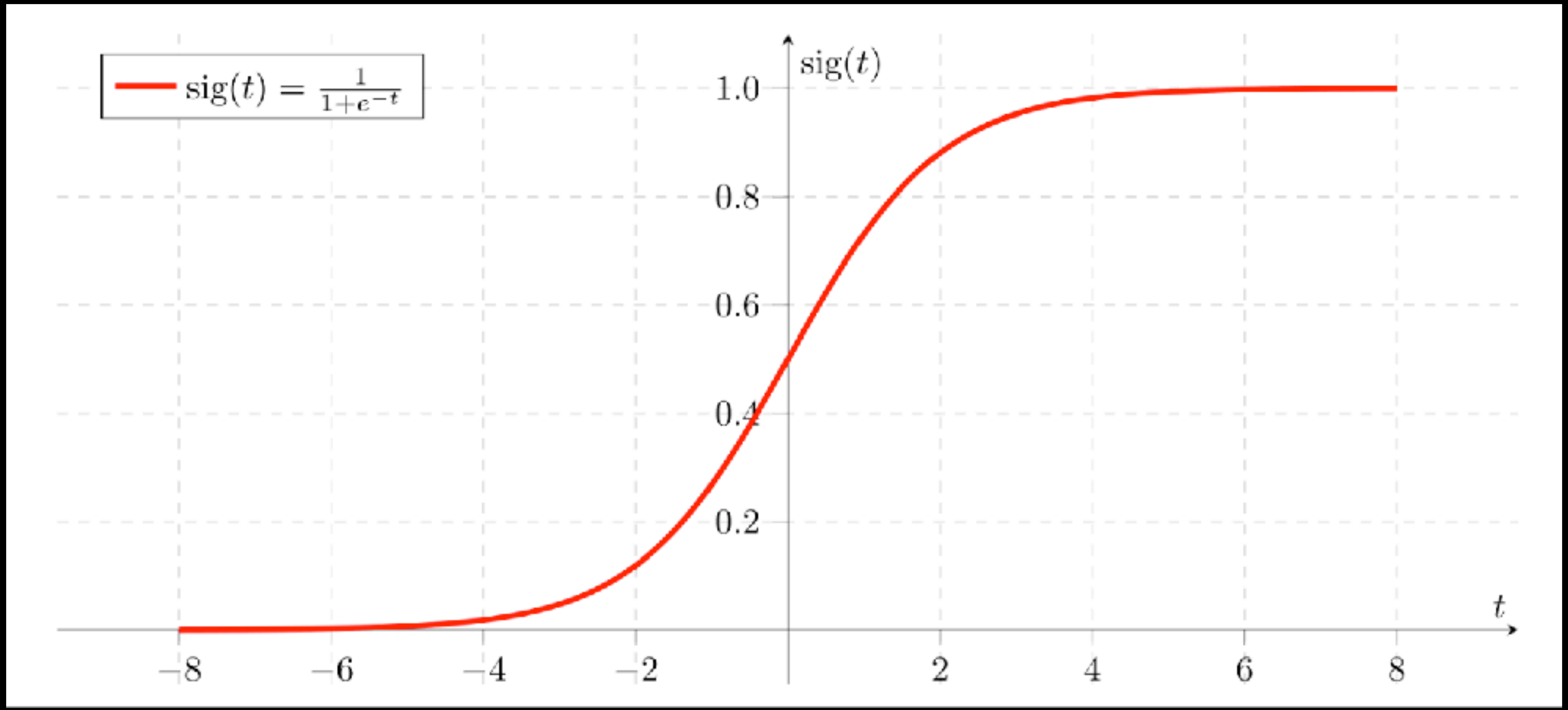
- Firstly it's linear separable.

- To begin with we need to find a line L. *Assume there is x0 which always equals 1*

$$Line\,(X) = w_0 + w_1x_1 + w_2x_2 = W^TX$$

- Assuming we already have w0 to w3, how to determine the label of given X?
  a) find a threshold of Line(X)
  b) map Line(X) to a static value range by some other function

Drafting the model



- Sigmoid function has a perfect shape for this job, it ranged [0,1].
- We can use 0 to represent one label and 1 for another label.
- Now we have a new model:

$$Classifier = sigmoid\left(Line\left(X\right)\right) = \frac{1}{1 + e^{-\left(W^T X\right)}}$$

Calc W from dataset

- Find best W

$$Classifier = sigmoid\left(Line\left(X\right)\right) = \frac{1}{1 + e^{-\left(W^T X\right)}}$$

- *Classifier(X')* is the probability of *X'* labeled to 1
- Then we can get new function for *W* based on *Maximum Likelihood Estimation*

$$l\left(W\right) = \sum_{i=1}^{m}\left(label^i \log\left(classifier\left(x^i\right)\right) + \left(1 - label^i\right)\log\left(1 - classifier\left(x^i\right)\right)\right)$$

- Then we can get the best *W* by maximize *l(W),* just applying gradient descent

$$w_j = w_j + a\sum_{i=1}^{m}\left(label^i - classifier\left(x^i\right)\right)x_j^{(i)} \quad \left(j = 0..2\right)$$

- Simplify to matrix form: $W = W + aX^T\left(label - classifier\left(X\right)\right)$

The method above is
also be called
Logistic Regression

对数几率回归

Coding Time

```swift
typealias FloatTensor = Tensor<Float>
let matplot = Python.import("matplotlib.pyplot")

enum Label : Int{
    case Green = 0
    case Red
}

struct Position{
    let x0 : Float = 1
    let x1 : Float
    let x2 : Float
}

struct ClassifierParameters : ParameterAggregate {
    var w = Tensor<Float>(randomNormal: [3,1])
}

struct Model
{
    var parameters : ClassifierParameters = ClassifierParameters()
}
```

Load Data

```
func loadTrainingSet() -> (trainingVec : FloatTensor , labelVec : FloatTensor)
{
    let lines = try! String(contentsOf: URL(fileURLWithPath:
"test.txt")).split(separator: "\r\n")
    let data = lines.map{$0.split(separator: "\t")}
    let rowCount = data.count
    let trainingScalars:[[Float]] = data.map{[1.0, Float($0[0])!, Float($0[1])!]}
    let labelScalarts:[Float] = data.map{Float($0[2])!}

    let trainingVec = Tensor<Float>(shape: [Int32(rowCount), 3], scalars:
trainingScalars.flatMap{$0})
    let labelVec = Tensor<Float>(shape: [Int32(rowCount) , 1], scalars:
labelScalarts)
    return (trainingVec, labelVec)
}
```

Train

```swift
func train(trainingVec : FloatTensor, labelVec : FloatTensor, model : inout Model){
    let learningRate:Float = 0.0005

    for epoch in 0...3000{
        let y = trainingVec • model.parameters.w
        let h = sigmoid(y)
        let e = labelVec - h
        let dw = trainingVec.transposed() • e

        let grad = ClassifierParameters(w: dw)
        model.parameters.update(withGradients: grad) { (p, g) in
            p += g ∗ learningRate
        }

        let p1 = -1 ∗ labelVec ∗ log(h)
        let p2 = (1 - labelVec)∗log(1 - h)
        let traditionalLogLoss = ((p1 - p2).sum() / batchSize)

        print("epoch: \(epoch), LogLoss v2: \(traditionalLogLoss)")
    }
}
```

1. Calculate gradient

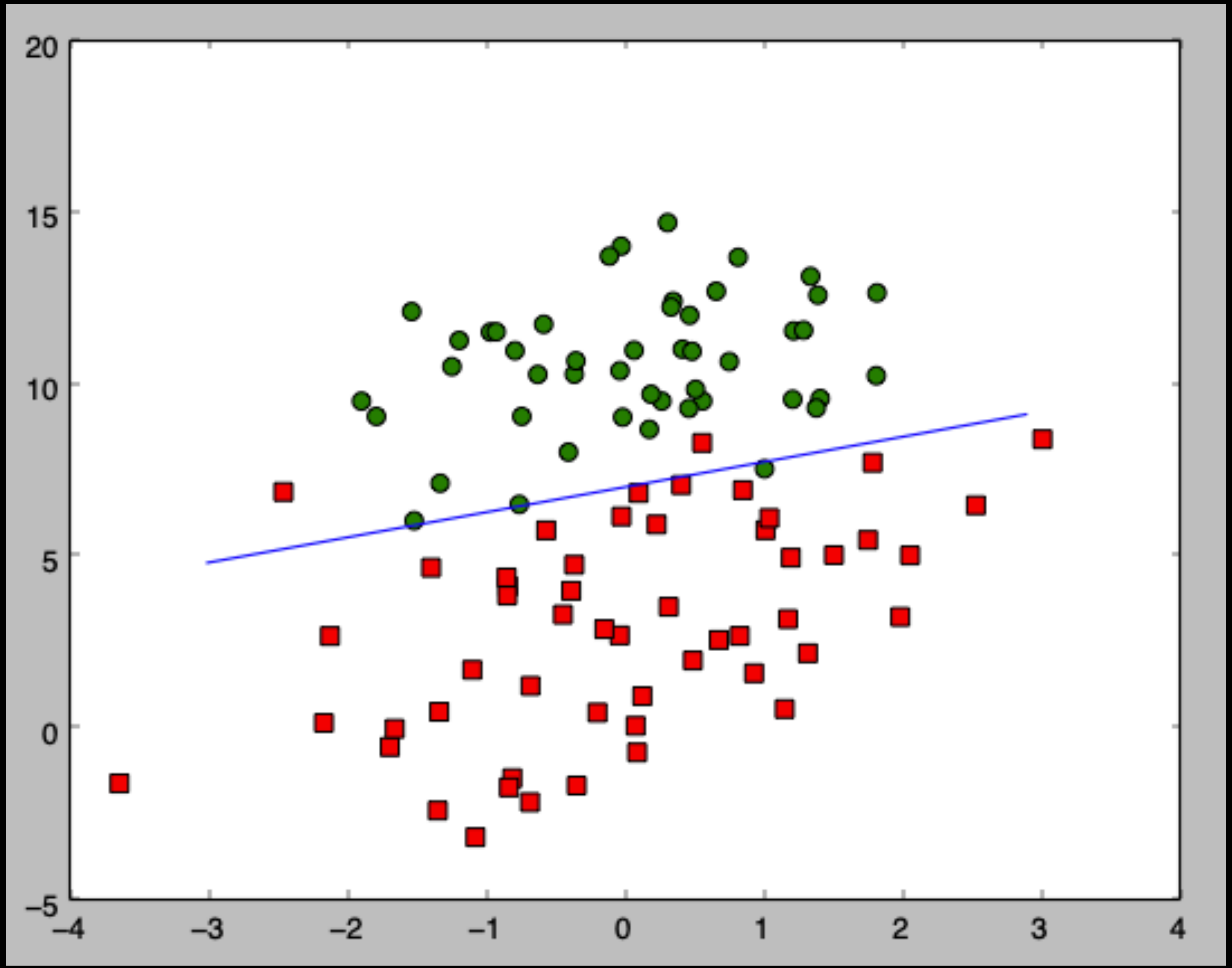
$$W = W + aX^T\left(label - classifier(X)\right)$$

2. Update weights

3. Calculate loss

Ploting the result

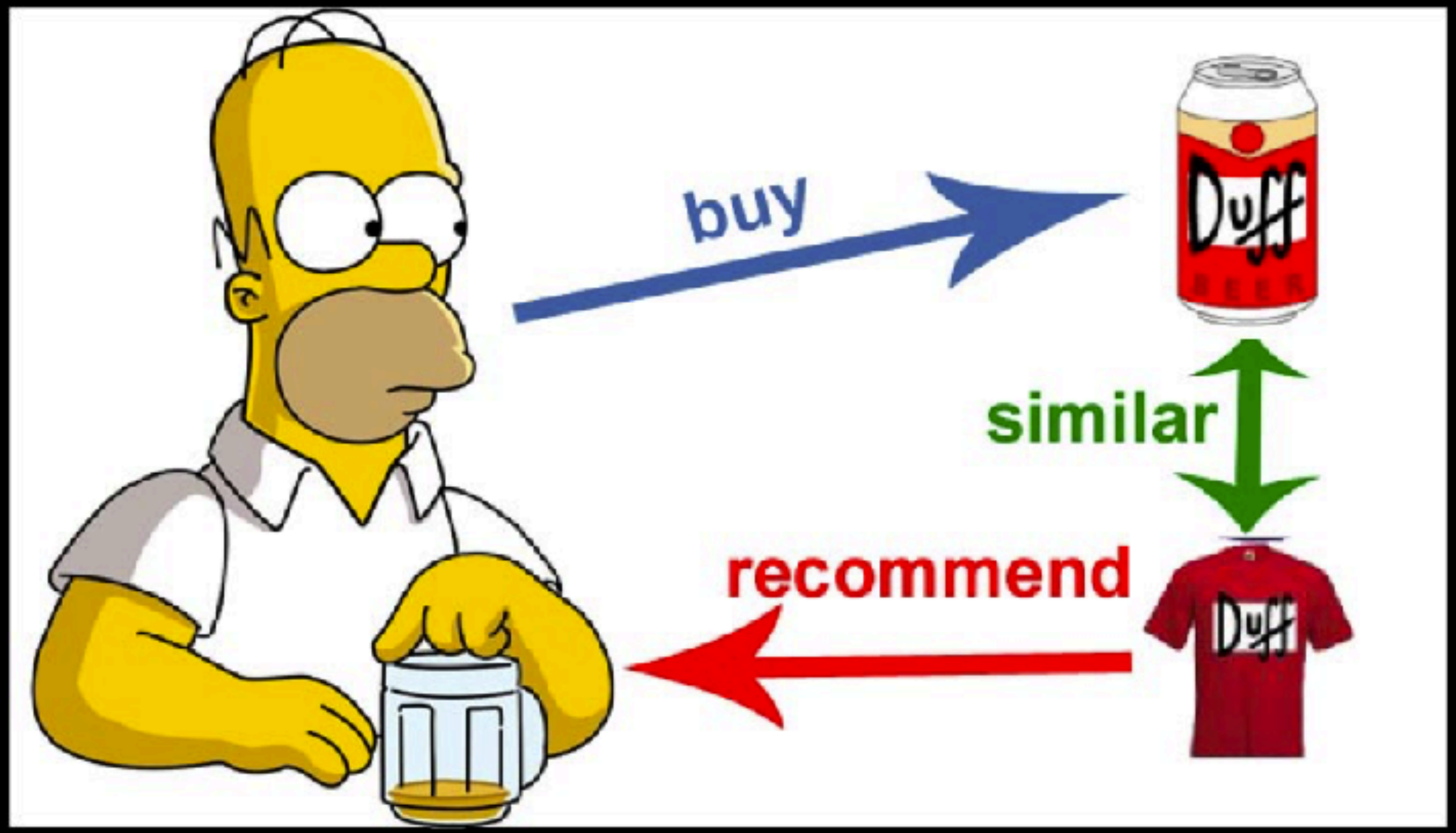
```
func plot(trainVec : FloatTensor, labelVec :
FloatTensor, parameters : ClassifierParameters)
{
    //Calculate points based parameters (w0, w1, w2)
    //…

    let matplot = Python.import("matplotlib.pyplot")
    let fig = matplot.figure()
    let ax = fig.add_subplot(111)
    ax.scatter(coord1x, coord1y, 50, "red", "s")
    ax.scatter(coord2x,coord2y,50, "green")
    ax.plot(xpts, ypts)
    matplot.show()
}
```



Collaborative Filtering

Collaborative Filtering

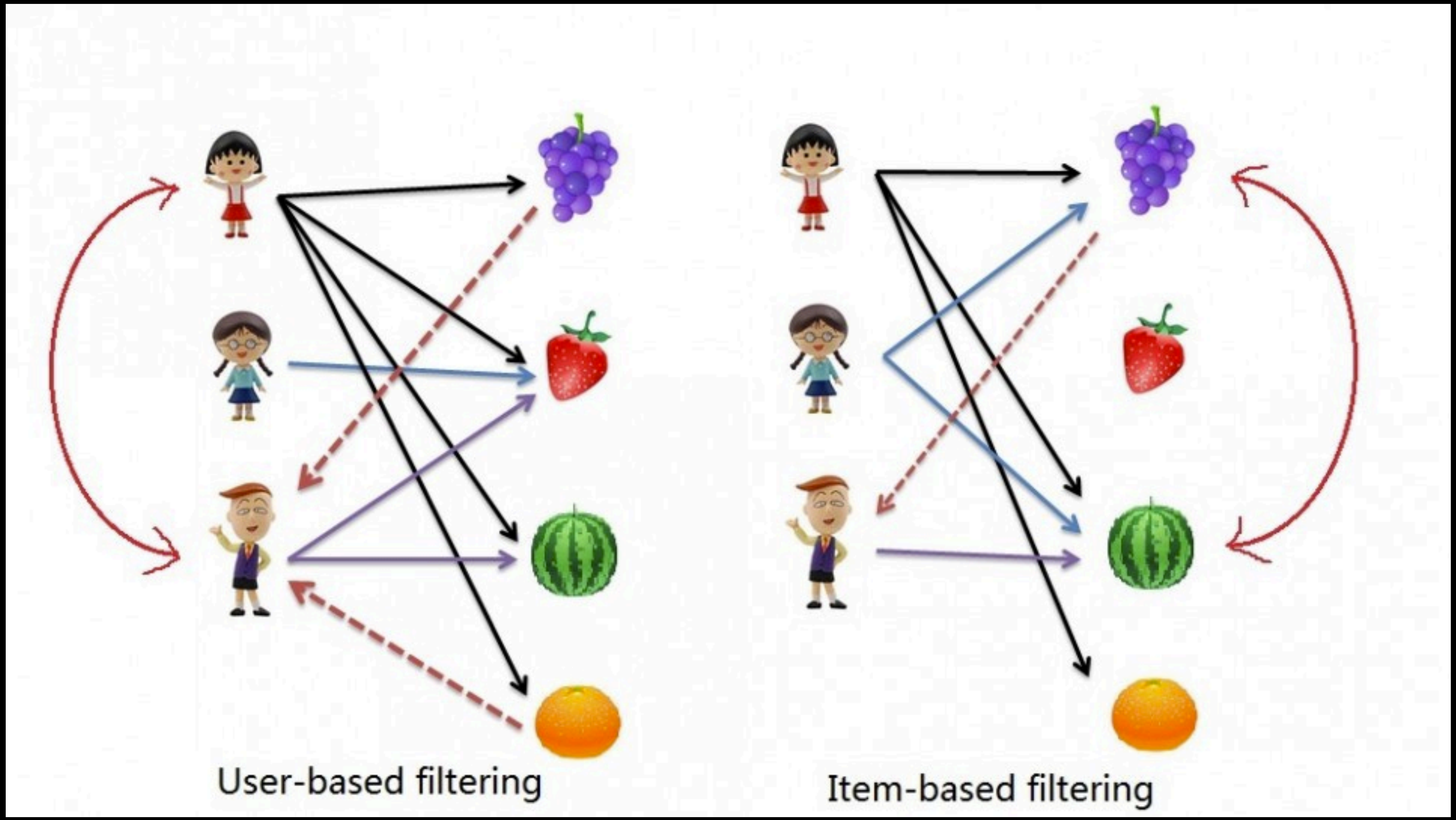
- Collaborative Filter(CF) is the foundation of recommender system.
- Predict your rating for sth.(movie/music/book) based on existing data, then provide recommend.



Method

- Item to Item:
  Users who liked this item also like …

- User to Item:
  Users who are similar to you also like …



i2i based Recommendation

Movie Recommendation

- Recommend movies based on the watch history from both you and other users

- We will use MovieLens Dataset, which is the most common dataset when implement & test recommender engine

- Including 100k movie ratings from 943 users and a selection of 1682 movies.

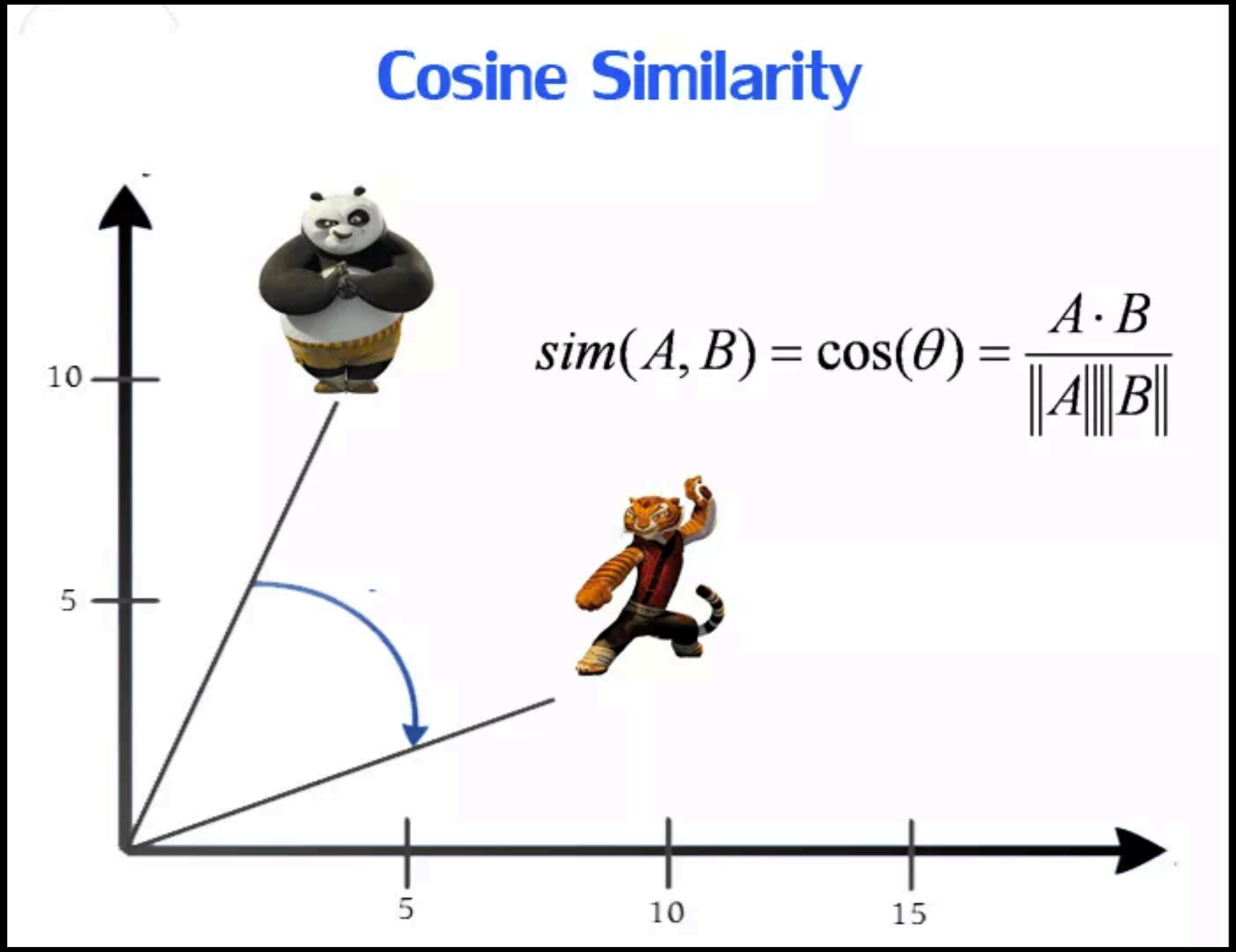
user_id, item_id, rating, timestamp

```
➜ TFPlayground tail -n 20 ml-100k/u.data
```

864	685	4	888891900
750	323	3	879445877
279	64	1	875308510
646	750	3	888528902
654	370	2	887863914
617	582	4	883789294
913	690	3	880824288
660	229	2	891406212
421	498	4	892241344
495	1091	4	888637503
806	421	4	882388897
676	538	4	892685437
721	262	3	877137285
913	209	2	881367150
378	78	3	880056976
880	476	3	880175444
716	204	5	879795543
276	1090	1	874795795
13	225	2	882399156
12	203	3	879959583

Distance Metric

- How to measure the distance between two movies?
- Cosine Similarity is a widely used metric for recommender system.
- Each movie is a vector contains 943 elements, representing ratings from all 943 users.



Modeling

- Slice train/test data
- Build movie-user rating matrix from train data

$$r_{k,n} \ (k \ movies, n \ \ users)$$

- Calc item-item similarity

$$sim\left(i_m, i_b\right) = \frac{i_m \cdot i_b}{\left|i_m\right| \left|i_b\right|} = \frac{\sum_a r_{m,a} r_{b,a}}{\sqrt{\sum_a r_{m,a}^2 r_{b,a}^2}}$$

- Predict unknown ratings

$$r_{m,k} = \frac{\sum_{i_b} sim\left(i_m, i_b\right) r_{b,k}}{\sum_{i_b} \left|sim\left(i_m, i_b\right)\right|}$$

- Compare with test data

$$mse = \frac{1}{N} \sum_i \left(x_i - x'_i\right)^2$$

Load Data

```swift
func loadData(path : String) -> (trainMatrix : Tensor<Float>, trainData : [[Float]] , testScalar : [Float])
{
    let lines = try! String(contentsOf: URL(fileURLWithPath: path)).split(separator: "\n")
    let data = lines.map{$0.split(separator: "\t")}
    let dataSet : [[Float]] = data.map{[Float($0[0])!, Float($0[1])!, Float($0[2])!]}

    let nUsers = Set(dataSet.map{$0[0]}).count
    let nMovies = Set(dataSet.map{$0[1]}).count

    print ("Total user is : \(nUsers), total movies are \(nMovies)")
    let rating = Array<Float>(repeating: 0, count: nMovies)
    var scalars:[[Float]] = Array<[Float]>(repeating: rating, count: nUsers)
    let ratingv2 = rating
    var testScalars:[[Float]] = Array<[Float]>(repeating: ratingv2, count: nUsers)

    let (trainData, testData) = sliceTrainSet(input: dataSet, ratio: 0.25)
    for item in trainData
    {
        scalars[Int(item[0]) - 1][Int(item[1]) - 1] = item[2]
    }

    for item in testData
    {
        testScalars[Int(item[0]) - 1][Int(item[1]) - 1] = item[2]
    }


    let tensor = Tensor<Float>(shape: [Int32(nUsers), Int32(nMovies)], scalars: scalars.flatMap{$0}).toAccelerator()
    return (tensor, trainData, testScalars.flatMap{$0})
}
```

1. Load data

2. Some transformation

3. Slice train/test data

4. Build tensor

Compute Similarity

$$sim\left(i_m, i_b\right) = \frac{i_m \cdot i_b}{\left|i_m\right| \left|i_b\right|} = \frac{\sum_a r_{m,a} r_{b,a}}{\sqrt{\sum_a r_{m,a}^2 r_{b,a}^2}}$$

- Can be easily implemented by a double for-loop
- But if we staring at it for …… a long time.
- It can be directly finished by matrix multiplication.

```
func pairwiseSimilarity(x : Tensor<Float>) ->
Tensor<Float> {
    let sumedX = x.squared().sum(alongAxes: 1)
    return x • x.transposed() / (sqrt(sumedX •
sumedX.transposed()) + epsilon)
}
```

Make Prediction

$$r_{m,k} = \frac{\sum_{i_b} sim\left(i_m, i_b\right) r_{b,k}}{\sum_{i_b} \left| sim\left(i_m, i_b\right) \right|}$$

- Like previous slide, we can transform double for-loop to matrix multiplication
- Then implement it by TFiwS with a few lines of code.

```
func predict(rating : Tensor<Float>,
                similarity : Tensor<Float>)
                -> Tensor<Float>
{
    let part = abs(similarity).sum(alongAxes:
1).transposed() + epsilon

    let result = (rating • similarity / part)
    return result
}
```

Validation

$$mse = \frac{1}{N}\sum_i (x_i - x'_i)^2$$

- We sliced 25% of the data into the test set in our first step.

- Comparing test data to corresponding ratings we predicted before will give us a reasonable measurement for the algorithm

```
func mse(pred : [Float], truth : [Float]) -> Float
{
    var mse:Float = 0.0
    var count = 0
    _ = zip(Range(NSMakeRange(0, pred.count))!,
truth).filter {$1 > 1e-9}
        .map { (index, truthValue) -> Float in
            mse += pow((pred[index] - truthValue),
Float(2))
            count += 1
            return 0
    }
    return mse / (Float(count) + 1e-9)
}
```

Pred is flattened matrix which contains all data, so we just compare the indexpath which test data contains

Result

swift -O CF.swift

Total user is : 943, total movies are 1682
original mse is 10.776904
➜ tfiws_snippet git:(master) ✗

We can do some quick optimize by applid top k selection in predict stage

Total user is : 943, total movies are 1682
original mse is 10.776904 topk mse is 9.9300585
➜ tfiws_snippet git:(master) ✗

Resources

- Classification & Collaborative Filter code:
  https://github.com/aaaron7/tfiws_snippet

- Documents of TFiwS:
  https://github.com/tensorflow/swift

- Tutorial & Demo:
  https://github.com/tensorflow/swift-models
  https://github.com/tensorflow/swift-tutorials

- Community: swift for tensorflow google group

Summary

- Introduce the Swift for TensorFlow, aka TFiwS.
- Brand new ML mode that TFiwS uses.
- Program Slicing based graph program extraction.
- Linear Classifier Example.
- Collaborative Filtering Example.

Thanks.