ArchData

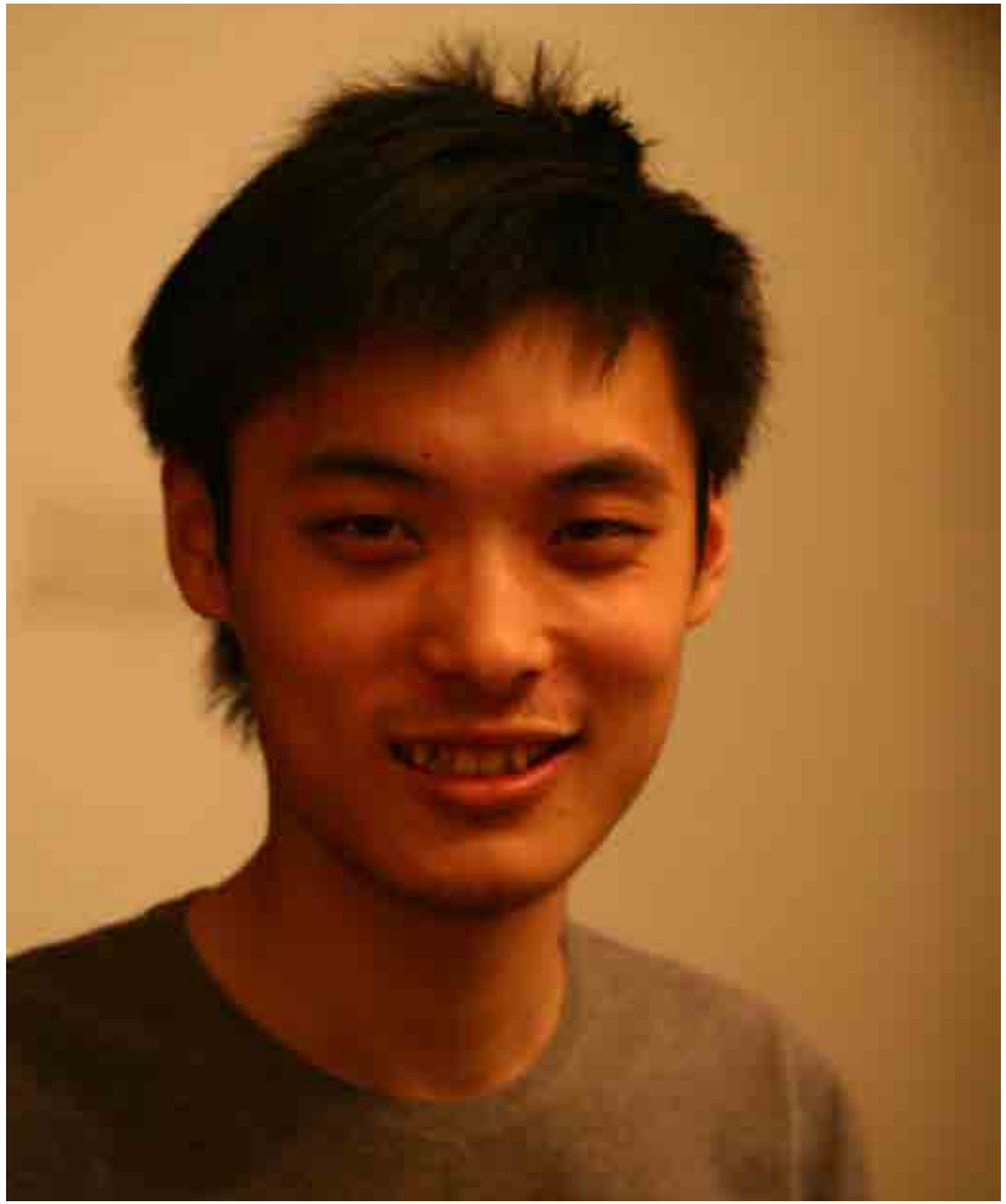
技术峰会北京站

主办方：中生代技术 FRESHMAN TECHNOLOGY 快CTO

2017年9月24日北京海淀区丹棱街5号微软亚太研发中心一号楼一层 故宫会议室

ThoughtWorks

杨博

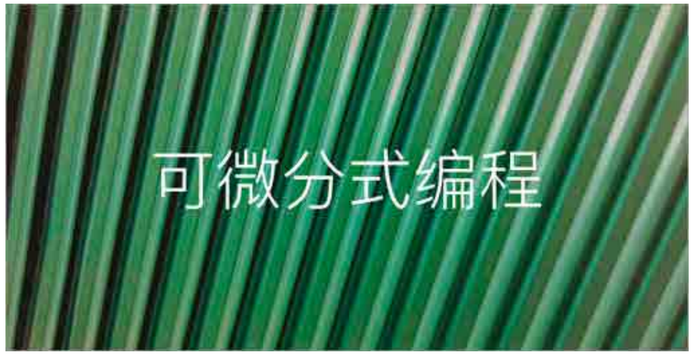


ThoughtWorks®

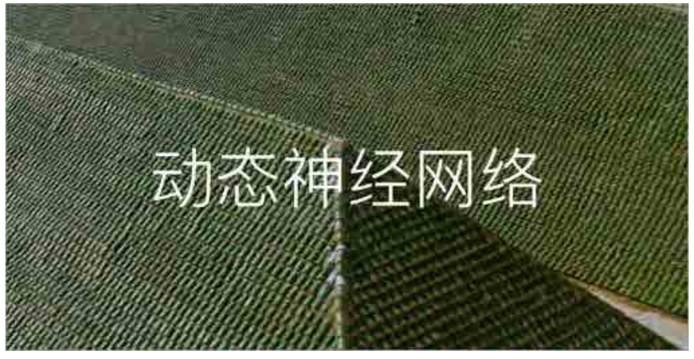
神经网络与函数式编程

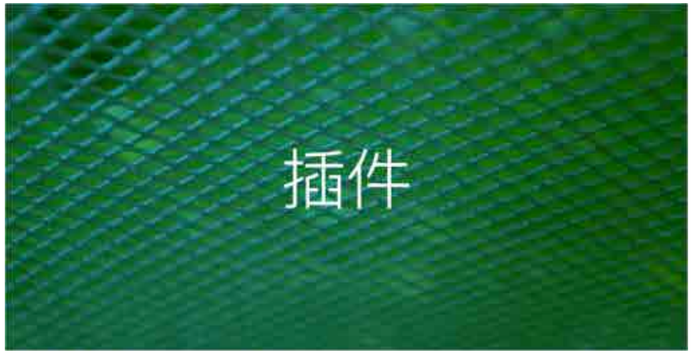
By 杨博 *at ThoughtWorks*


DeepLearning.scala


可微分式编程


函数式编程


动态神经网络

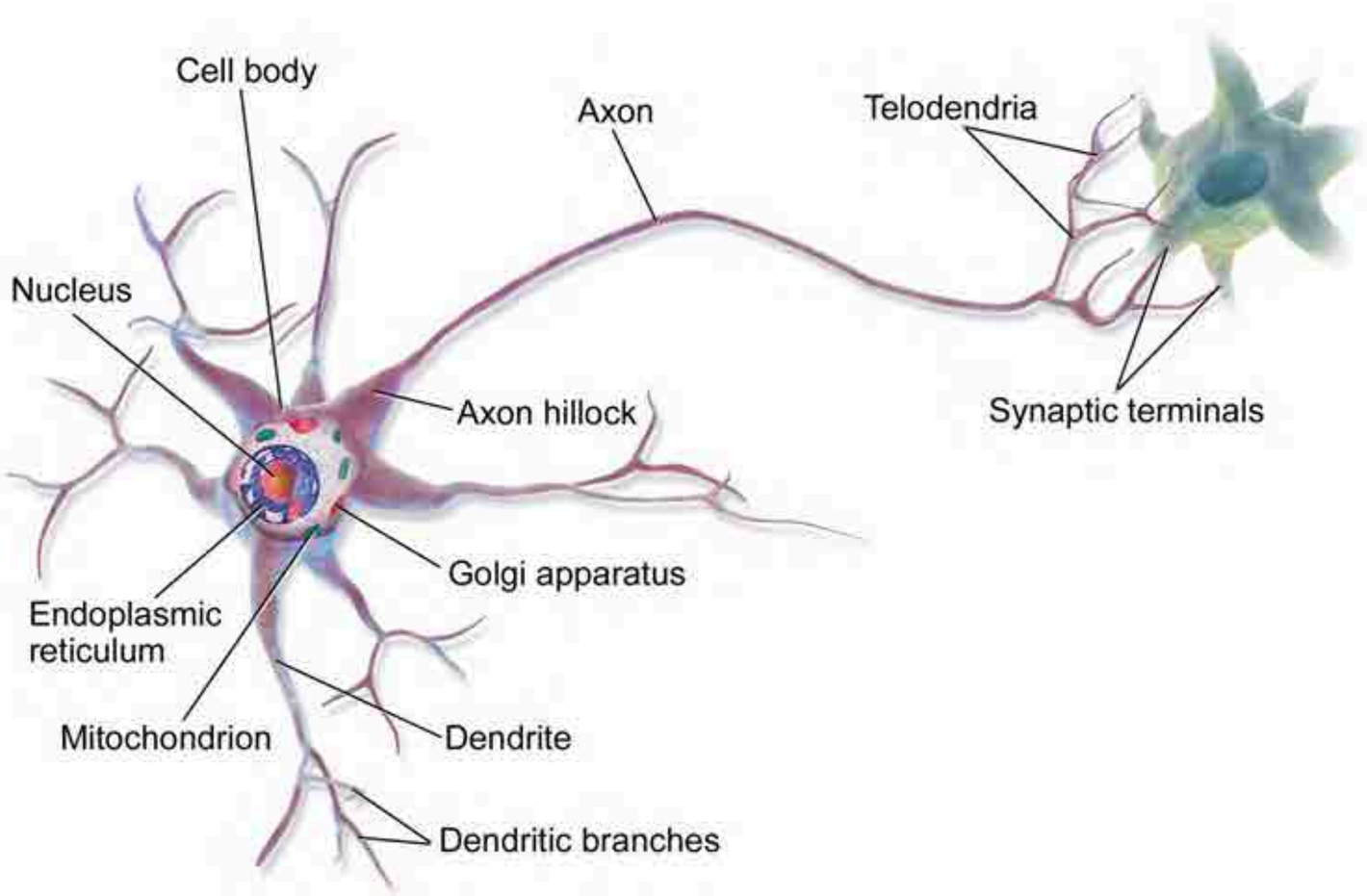

插件

ThoughtWorks®

TO DEFINE NEURO NETWORK

we have done ...

BIOLOGICAL?



https://en.wikipedia.org/wiki/Biological_neural_network#/media/File:Blausen_0657_MultipolarNeuron.png

FEATURE ENGINEERING?

http://spark.ieee.org/files/2011/12/Cartoon-for-Feature-Article.gif

ThoughtWorks®

NEURO NETWORK IS FUNCTIONAL PROGRAMMING

we have done ...

IQ TEST ROBOT

What is the next number in the sequence?

3,4,5,?

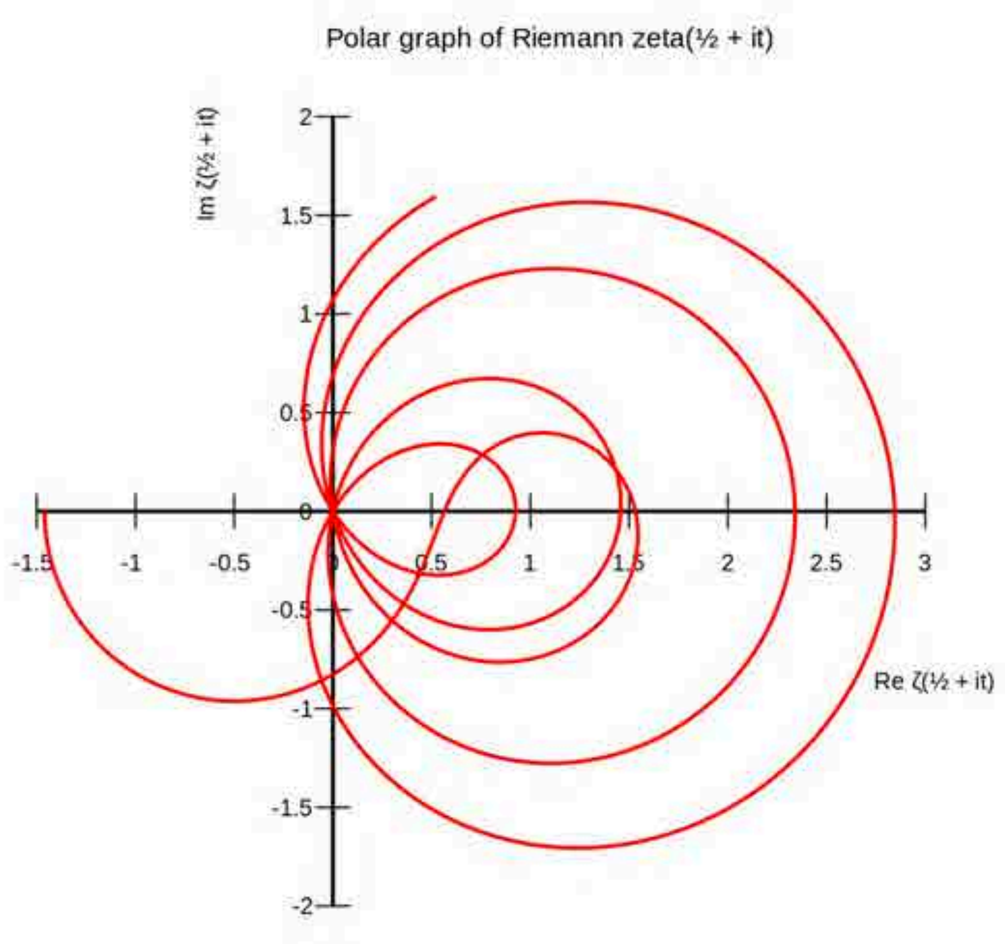
13,19,25,?

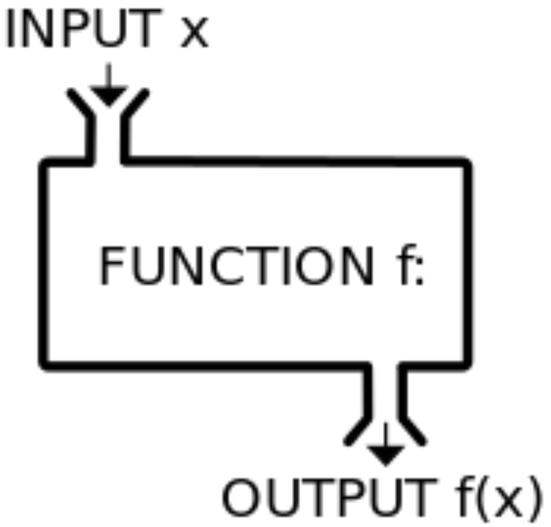
THE SIGNATURE OF IQ TEST ROBOT

```
def guessNextNumber(question: Seq[Double]): DoubleLayer = ???
```

```
println(guessNextNumber(Seq(3, 4, 5)).predict.blockingAwait)

println(guessNextNumber(Seq(13, 19, 25)).predict.blockingAwait)
```

FUNCTION?


Polar graph of Riemann zeta(½ + it)
Im ζ(½ + it)
Re ζ(½ + it)
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2
2.5
3


INPUT x
FUNCTION f:
OUTPUT f(x)

TRAINING THE IQ TEST ROBOT

```
def lossFunction(robotAnswer: DoubleLayer, expectedAnswer: Double): DoubleLayer = ???

def iqTestRobotTrainer(question: Seq[Double], expectedAnswer: Double): DoubleLayer = {
  val robotAnswer = guessNextNumber(question)
  lossFunction(robotAnswer, expectedAnswer)
}

iqTestRobotTrainer(Seq(3, 4, 5), 6).train.blockingAwait
iqTestRobotTrainer(Seq(13, 19, 25), 31).train.blockingAwait
```

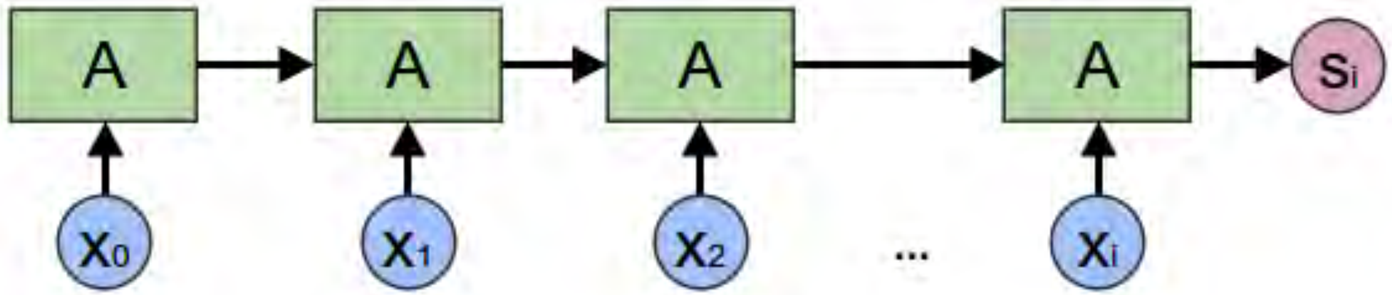
THE STRUCTURE OF THE IQ TEST ROBOT

```
// 惰性初始化的权重列表
val weights: Stream[DoubleWeight] = Stream.continually(DoubleWeight(math.random))

// 用map/reduce编写的多项式加法预测函数
def guessNextNumber(question: Seq[Double]): DoubleLayer = {
  (question zip weights).map {
    case (element, weight) => element * weight
  }.reduce(_ + _)
}

// 平方惩罚函数
def lossFunction(robotAnswer: DoubleLayer, expectedAnswer: Double): DoubleLayer = {
  val difference: DoubleLayer = robotAnswer - expectedAnswer
  difference * difference
}
```

META-PROGRAMMING

https://prateekvjoshi.files.wordpress.com/2014/04/1-main.png

ENCODING RECURRENT NEURAL NETWORKS



https://colah.github.io/posts/2015-09-NN-Types-FP/

ENCODING RECURRENT NEURAL NETWORKS

```
package scala.collection
trait Seq[A] {
  def foldLeft[B](z: B)(step: (B, A) => B): B
}
```

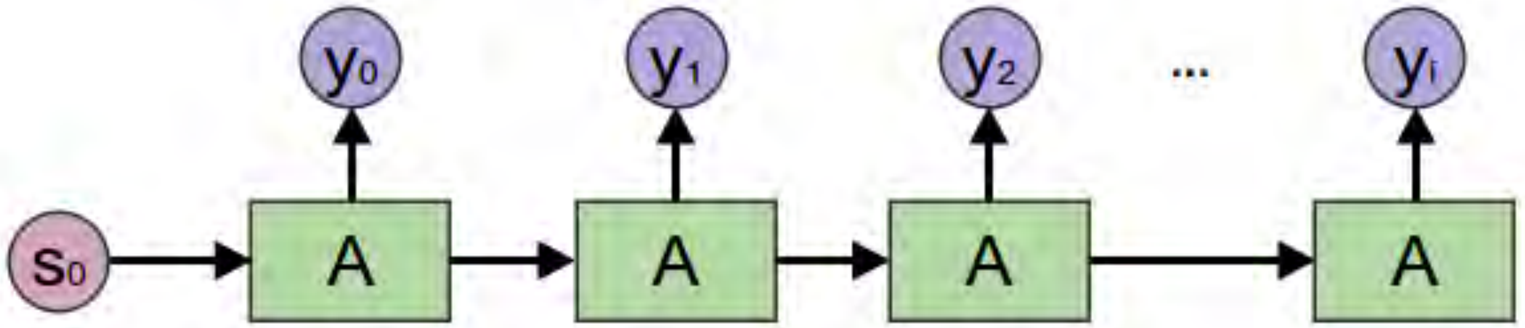
ENCODING RECURRENT NEURAL NETWORKS

```
type Input = Seq[DoubleLayer]
type State = Seq[DoubleLayer]

def step(hiddenState: State, xi: Input): State = ???

def encodingRNN(x: Seq[Input]): State = {
  val initialState: State = Seq.empty[DoubleLayer]
  x.foldLeft(initialState)(step)
}
```

ENCODING RECURRENT NEURAL NETWORKS

```
val NumberOfOutputFeatures = 10

val weight: Seq[Seq[DoubleWeight]] = Seq.fill(NumberOfOutputFeatures)(
  Stream.continually(DoubleWeight(math.random))
)

def step(hiddenState: State, xi: Input): State = {
  tanh(matrixMultiply(hiddenState ++ xi, weight))
}
```

GENERATING RECURRENT NEURAL NETWORKS

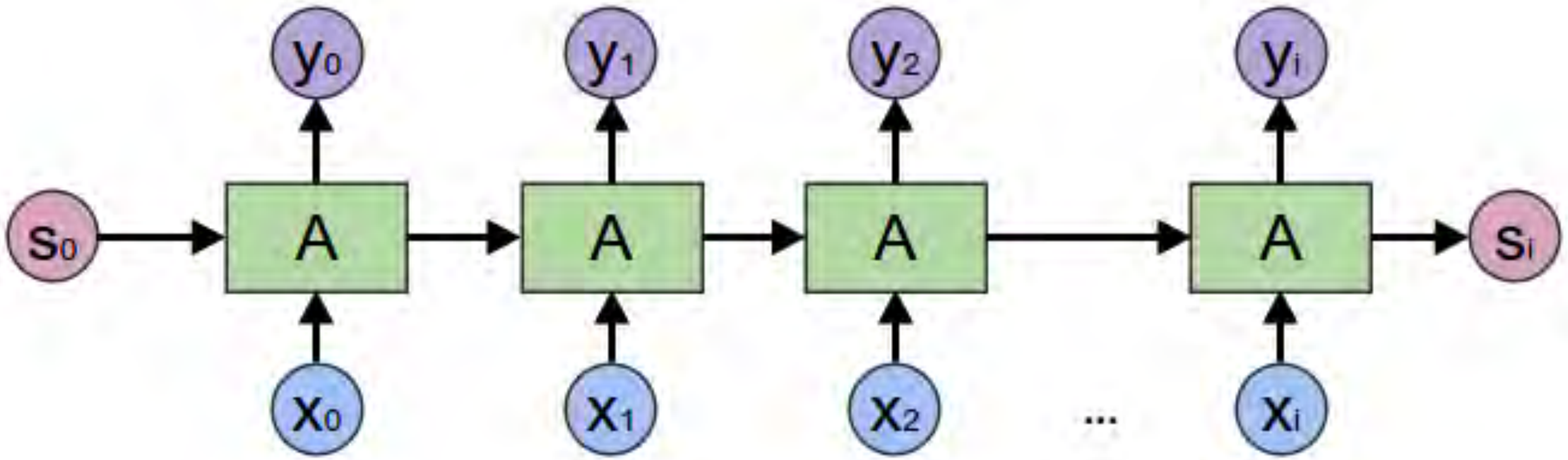



```scala
package scalaz
object DList {
  def unfoldr[A, B](b: B, f: B => Option[(A, B)]): DList[A]
}
```

```scala
type State = Seq[DoubleLayer]
type Output = Seq[DoubleLayer]

def step(hiddenState: State): Option[(State, Output)] = ???

def generatingRnn(seed: State): DList[Output] = {
  DList.unfoldr(seed)(step)
}
```

https://colah.github.io/posts/2015-09-NN-Types-FP/

GENERAL RECURRENT NEURAL NETWORKS

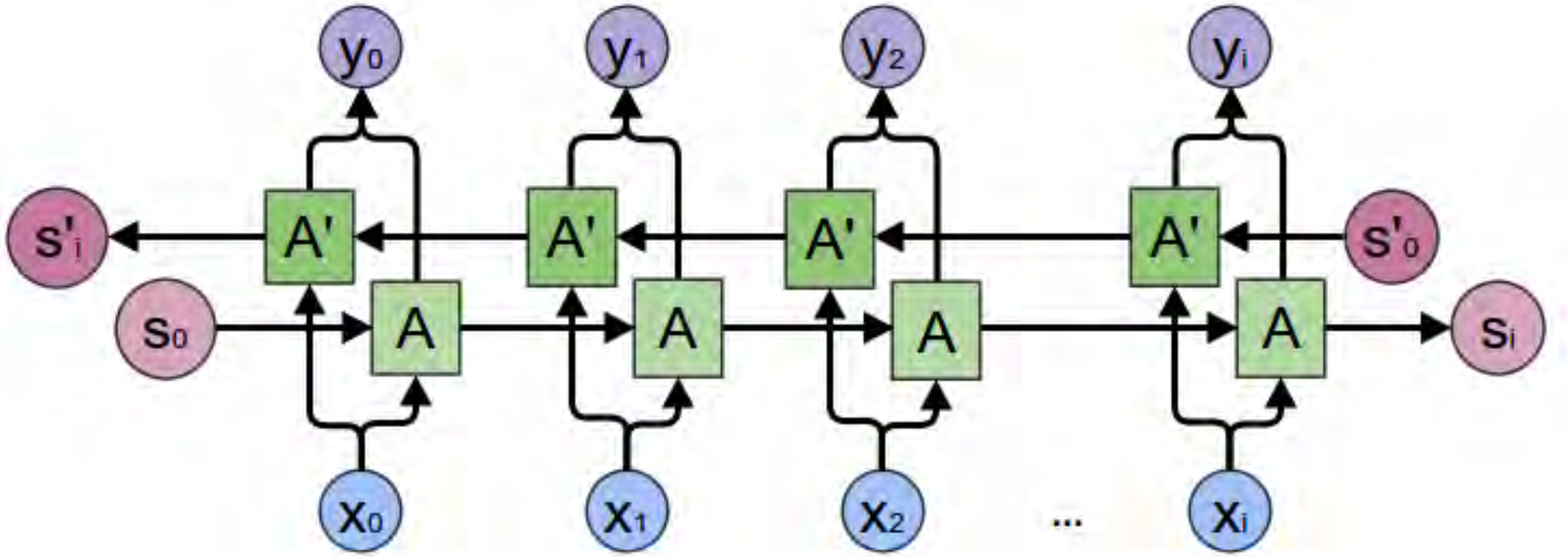



```scala
package scalaz
class IList[A] {
  def mapAccumLeft[B, C](c: C)(f: (C, A) => (C, B)): (C, IList[B])
  def mapAccumRight[B, C](c: C)(f: (C, A) => (C, B)): (C, IList[B])
}
```

```scala
type State = Seq[DoubleLayer]
type Input = Seq[DoubleLayer]
type Output = Seq[DoubleLayer]

def step(hiddenState: State, xi: Input): (State, Output) = ???

def rnn(x: IList[Input], seed: State): (State, IList[Output]) = {
  x.mapAccumLeft(seed)(step)
}
```

https://colah.github.io/posts/2015-09-NN-Types-FP/

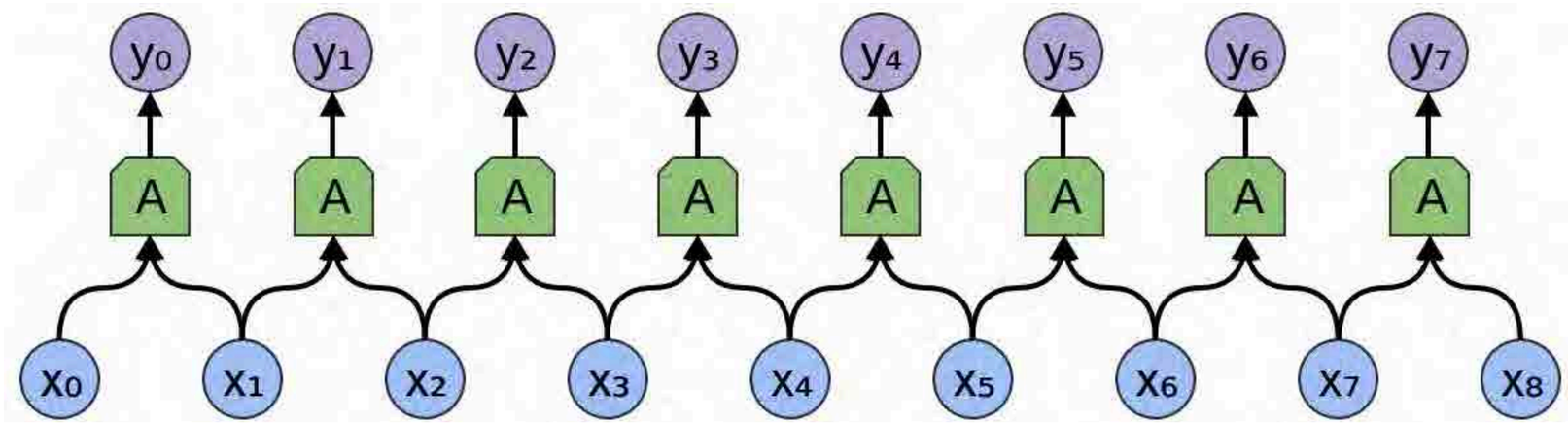
BIDIRECTIONAL RECURSIVE NEURAL NETWORKS




```
def leftStep(hiddenState: State, xi: Input): (State, Output) = ???
def rightStep(hiddenState: State, xi: Input): (State, Output) = ???

def bidirectionalRnn(x: IList[Input], leftSeed: State, rightSeed: State): IList[Output] = {
  val (_, leftToRight) = x.mapAccumLeft(leftSeed)(leftStep)
  val (_, rightToLeft) = x.mapAccumRight(rightSeed)(rightStep)
  leftToRight.zip(rightToLeft).map { pair =>
    pair._1 ++ pair._2
  }
}
```

https://colah.github.io/posts/2015-09-NN-Types-FP/

CONVOLUTIONAL NEURAL NETWORKS

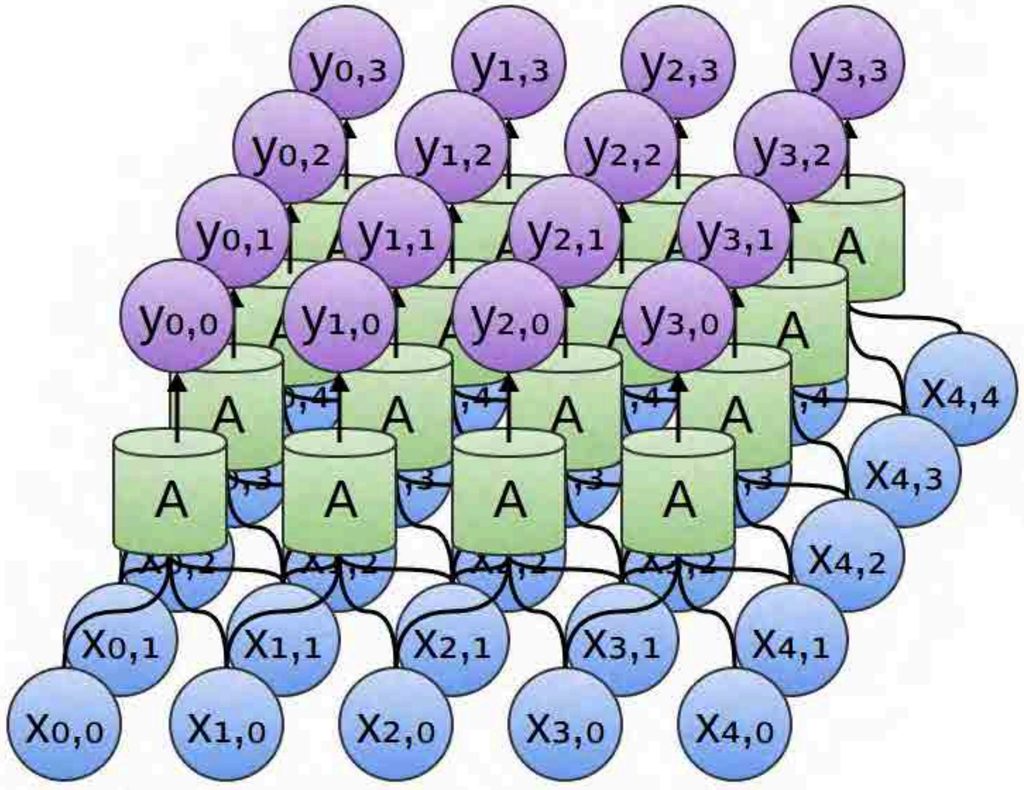



```
package scala.collection.immutable
class List {
  def map[B](f: A => B): List[B]
  def zip[B](that: List[B]): List[(A, B)]
}
```

```
type Input = Seq[DoubleLayer]
type Output = Seq[DoubleLayer]

def kernel1x2(xi: Input, xj: Input): Output

def cnn1d(x: List[Input]): List[Output]) = {
  x.zip(x.tail).map(kernel)
}
```

https://colah.github.io/posts/2015-09-NN-Types-FP/

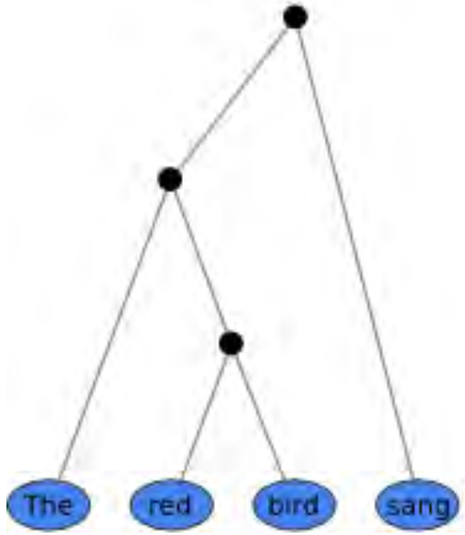
TWO DIMENSIONAL CONVOLUTIONAL NETWORK

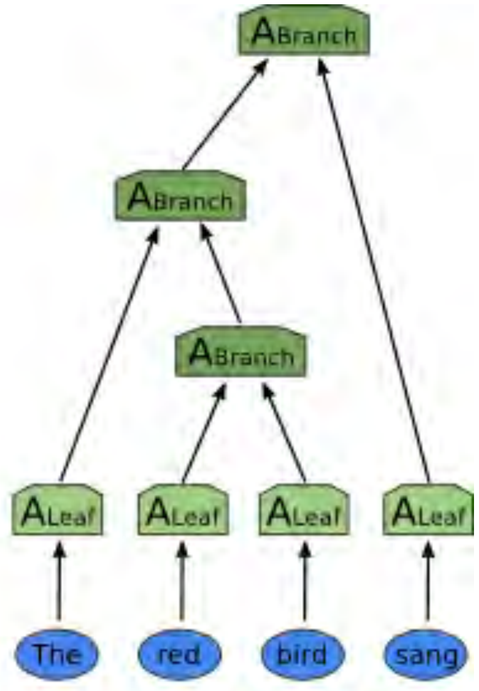



```
type Input = Seq[DoubleLayer]
type Output = Seq[DoubleLayer]

def kernel2x2(xi: Input, xj: Input, xk: Input, xl: Input): Output

def cnn2d(x: List[List[Input]]): List[Output]) = {
  val x00 = x
  val x01 = x.map(_.tail)
  val x10 = x.tail
  val x11 = x.tail.map(_.tail)
  (x zip x01 zip x10 zip x11).map {
    case (((xi, xj), xk), xl) => kernel2x2(xi, xj, xk, xl)
  }
}
```

https://colah.github.io/posts/2015-09-NN-Types-FP/

RECURSIVE NEURAL NETWORKS






```
package scalaz
class Tree {
  def scanr[B](g: (A, Stream[Tree[B]]) => B): Tree[B]
}
```

```
type Input = Seq[DoubleLayer]
type Output = Seq[DoubleLayer]

def step(x: Input, children: Stream[Output]): Output = ???

def treeNet(x: Tree[Input]): Tree[Output] = {
  x.scanr(step)
}
```

https://colah.github.io/posts/2015-09-NN-Types-FP/

NEURO NETWORK IS FUNCTIONAL PROGRAMMING

Deep Learning Name	Functional Name
Learned Vector	Constant in metaprogramming
Encoding RNN	Fold
Generating RNN	Unfold
General RNN	Accumulating Map
Bidirectional RNN	Zipped Left/Right Accumulating Maps
Conv Layer	“Window Map”
TreeNet	Catamorphism

WHAT

BIOLOGICAL? YES

FEATURE ENGINEERING? YES

FUNCTIONAL PROGRAMMING? YES

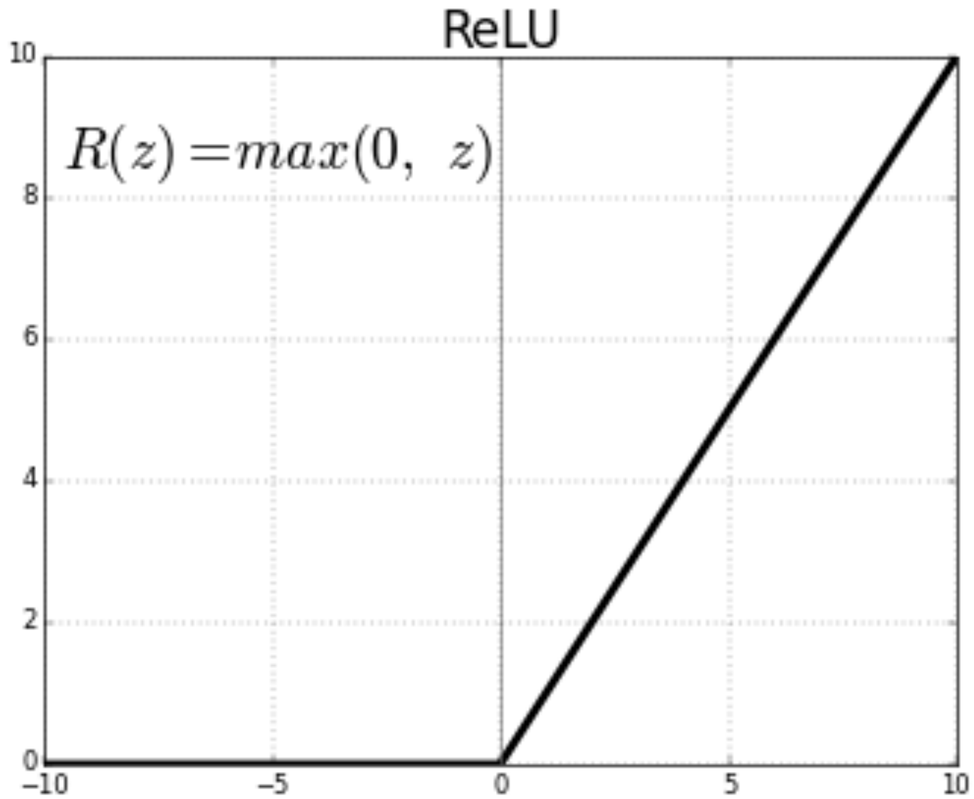
METAPROGRAMMING? YES

ThoughtWorks®

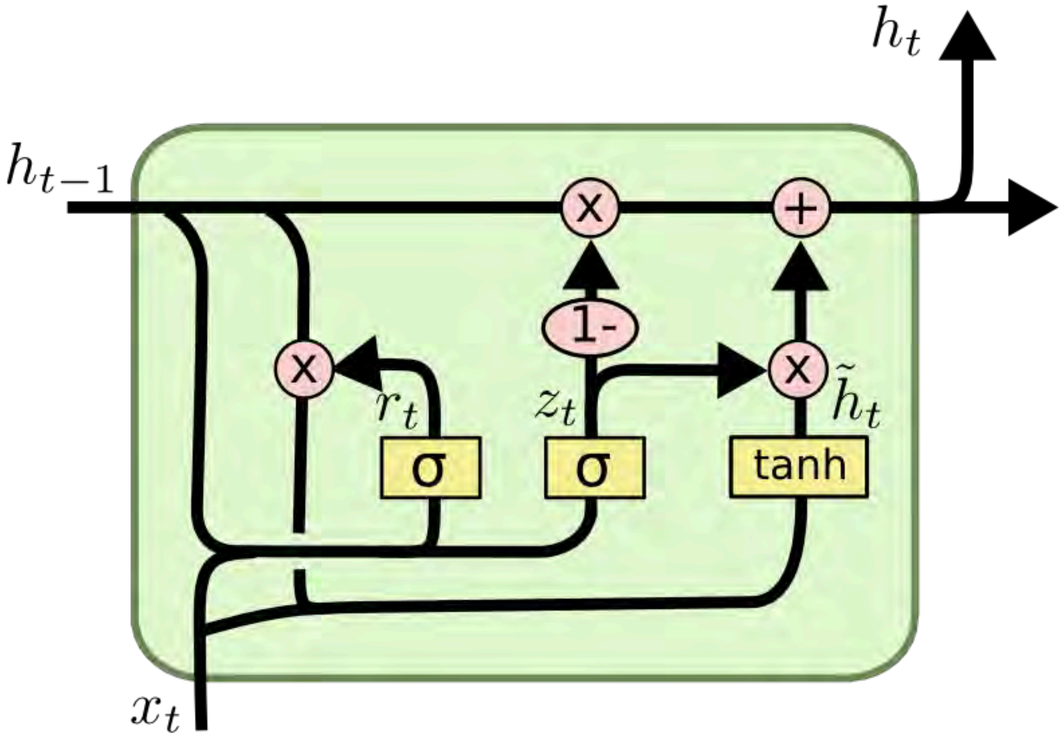
MONADIC DEEP LEARNING

we have done ...

IF ... ELSE ...



IF ... ELSE ...



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

http://colah.github.io/posts/2015-08-Understanding-LSTMs

MIXTURE OF EXPERTS

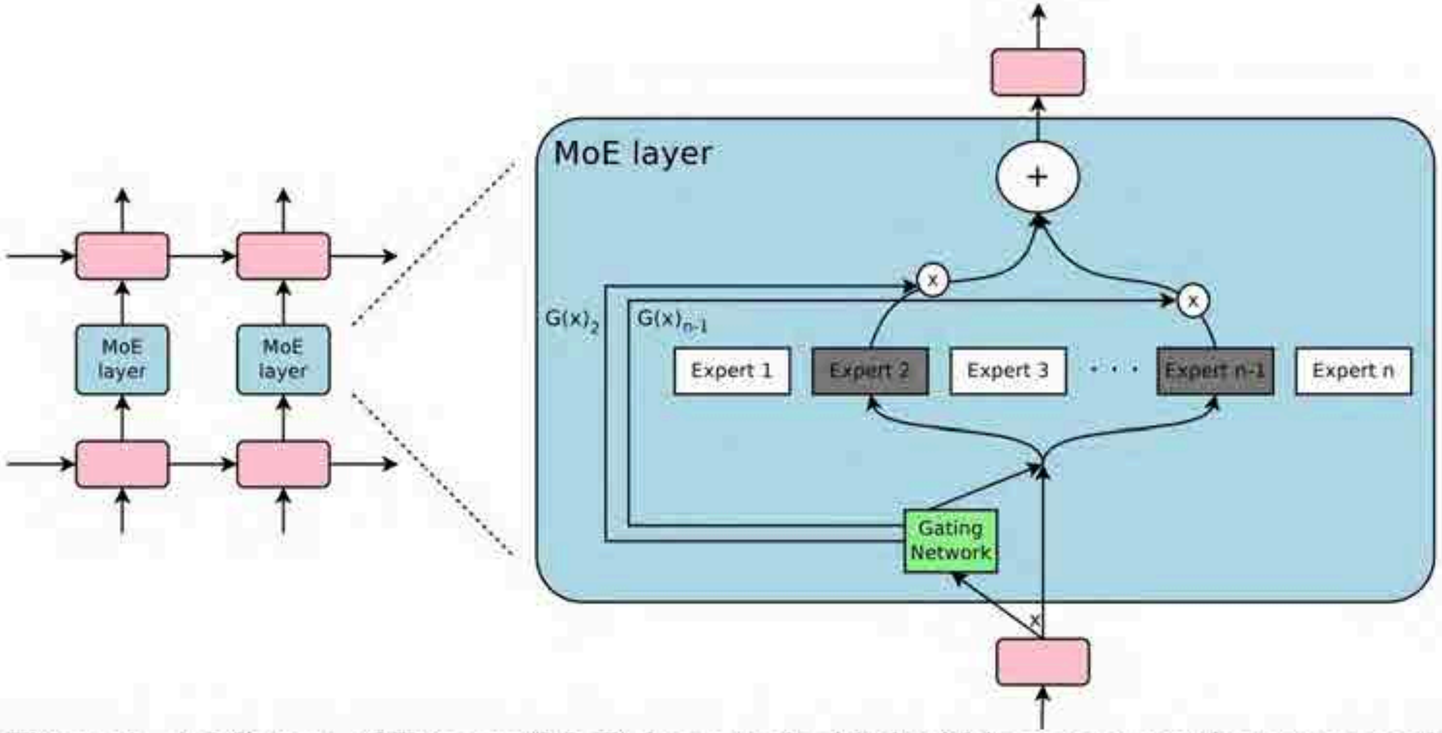


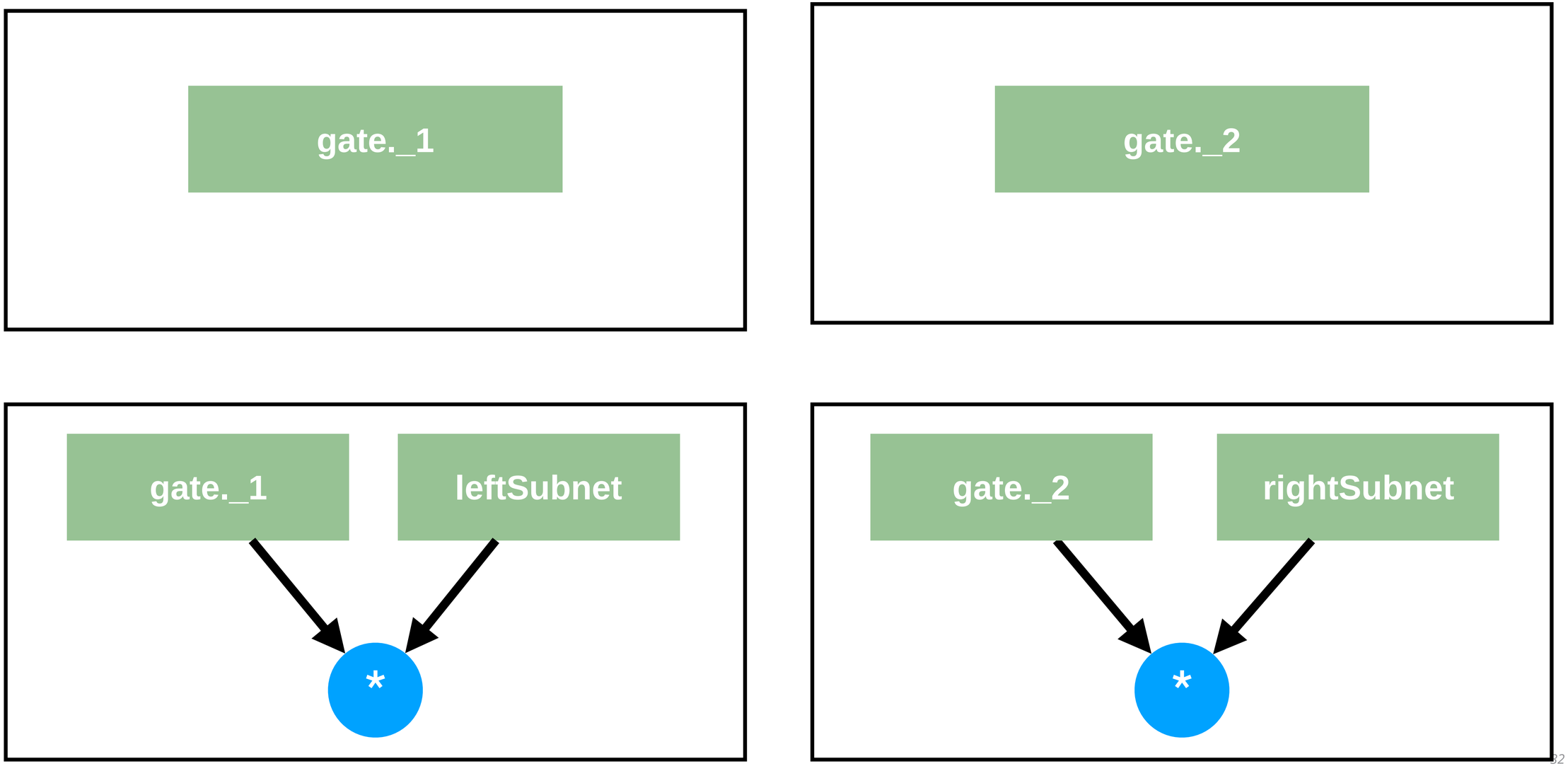
Figure 1: A Mixture of Experts (MoE) layer embedded within a recurrent language model. In this case, the sparse gating function selects two experts to perform computations. Their outputs are modulated by the outputs of the gating network.

https://arxiv.org/pdf/1701.06538.pdf

NAIVE GATED NET

```
type Input = INDArrayLayer
type Output = INDArrayLayer
type Condition = (DoubleLayer, DoubleLayer)

def leftSubnet(input: Input): Output = ???
def rightSubnet(input: Input): Output = ???
def gate(input: Input): Condition = ???

def naiveGatedNet(input: Input): Output = {
  val condition = gate(input)
  if (condition._1.predict.blockingAwait > condition._2.predict.blockingAwait) {
    conditionLeft * leftSubnet(input)
  } else {
    conditionRight * rightSubnet(input)
  }
}
```

NAIVE GATED NET



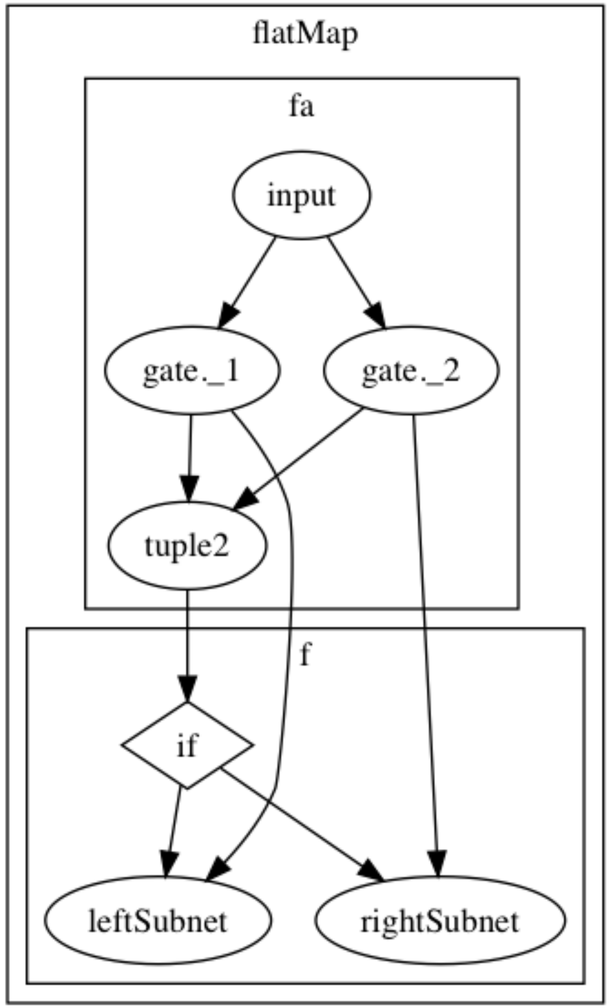
MONADIC GATED NET

```
def monadicGatedNet(input: Input): Output = {
  val condition = gate(input)
  val gatedForward =
(condition._1.forward.tuple2(condition._2.forward)).flatMap { pair =>
    if (pair._1.data > pair._2.data) {
      (condition._1 * leftSubnet(input)).forward
    } else {
      (condition._2 * rightSubnet(input)).forward
    }
  }
  INDArrayLayer(gatedForward)
}
```

MONADIC GATED NET

```
package scalaz
trait Monad[F[_]] extends Apply[F[_]] {
  def point(a: => A): F[A]
  def bind[A, B](fa: F[A])(f: A => F[B]): F[B]
}
```

MONADIC GATED NET




```
def monadicGatedNet(input: Input): Output = {
  val condition = gate(input)
  val gatedForward =
    (condition._1.forward.tuple2(condition._2.forward))
      .flatMap { pair =>
        if (pair._1.data > pair._2.data) {
          (condition._1 * leftSubnet(input)).forward
        } else {
          (condition._2 * rightSubnet(input)).forward
        }
      }
  INDArrayLayer(gatedForward)
}
```

PARALLEL GATED NET

```
def parallelGatedNet(input: Input): Output = {
  val condition: (DoubleLayer, DoubleLayer) = gate(input)
  val parallelCondition1Forward = Parallel(condition._1.forward)
  val parallelCondition2Forward = Parallel(condition._2.forward)
  val Parallel(tupled) = condition1ForwardParallel.tuple2(condition2ForwardParallel)
  val gatedForward = tupled.flatMap { pair =>
    if (pair._1.data > pair._2.data) {
      (condition._1 * leftSubnet(input)).forward
    } else {
      (condition._2 * rightSubnet(input)).forward
    }
  }
  INDArrayLayer(gatedForward)
}
```

THOUGHTWORKS EACH

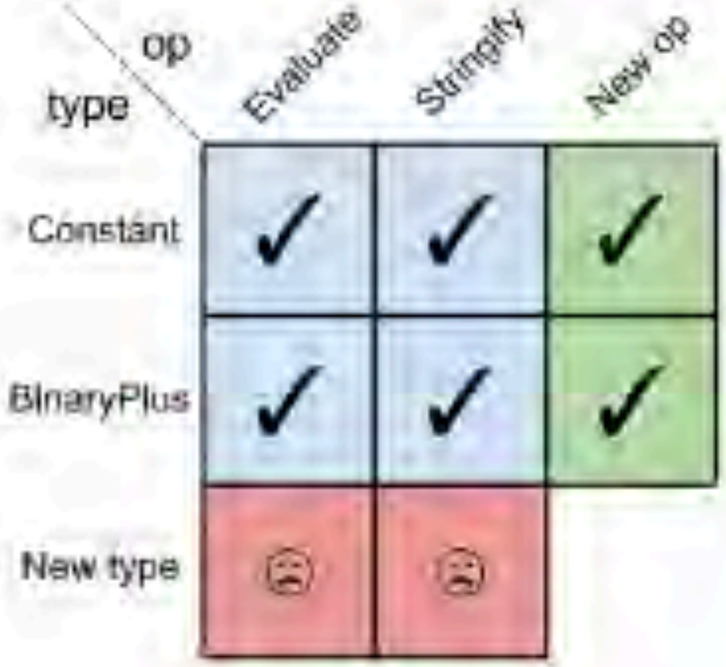
```
def eachGatedNet(input: Input): Output = INDArrayLayer(monadic[Do] {
  val condition: (DoubleLayer, DoubleLayer) = gate(input)
  val parallelCondition1Forward = Parallel(condition._1.forward)
  val parallelCondition2Forward = Parallel(condition._2.forward)
  val Parallel(tupled) = condition1ForwardParallel.tuple2(condition2ForwardParallel)
  val pair: (Tape[Double, Double], Tape[Double, Double]) = tupled.each
  if (pair._1.data > pair._2.data) {
    (condition._1 * leftSubnet(input)).forward.each
  } else {
    (condition._2 * rightSubnet(input)).forward.each
  }
})
```

ThoughtWorks®

EXPRESSION PROBLEM

we have done ...

THE EXPRESSION PROBLEM



Attila Magyar, 2016
Microsec ltd.

HYPERPARAMETER IN DEEPLEARNING.SCALA

```
val hyperparameters1 = Factory[Builtins with FixLearningRate].newInstance(
  learningRate = 0.001
)
val weight1 = hyperparameters1.INDArrayWeight(Nd4j.randn(10, 10))
val layer1[Input](input: Input): hyperparameters1.INDArrayLayer = {
  tanh(input dot weight1)
}

val hyperparameters2 = Factory[Builtins with FixLearningRate].newInstance(
  learningRate = 0.002
)
val weight2 = hyperparameters2.INDArrayWeight(Nd4j.randn(10, 10))
val layer2[Input](input: Input): hyperparameters2.INDArrayLayer = {
  tanh(input dot weight2)
}

val hyperparameters3 = Factory[Builtins].newInstance()
def twoLayerNeuralNetwork(input: INDArray): hyperparameters3.INDArrayLayer = {
  val layer1Output: hyperparameters1.INDArrayLayer = layer1(input)
  layer2(layer1Output)
}
```

PLUGIN

```
val hyperparameters = Factory[
  Plugin1 with Plugin2 with Plugin3
].newInstance(
  hyperparameterKey1 = hyperparameterValue1,
  hyperparameterKey2 = hyperparameterValue2,
  hyperparameterKey3 = hyperparameterValue3
)
```

CREATING PLUGIN

```
trait Softmax extends Builtins {
  def softmax[Scores: DeepLearning.Aux[?, INDArray, INDArray]]
(scores: Scores): INDArrayLayer = {
    val expScores = hyperparameters.exp(scores)
    expScores / expScores.sum(1)
  }
}
```

```
val hyperparameters = Factory[Softmax].newInstance()
val probabilities = hyperparameters.softmax(Nd4j.randn(10, 10))
```

CREATING PLUGIN

```
trait FixedLearningRate extends Builtins {
  def learningRate: Double
  trait INDArrayOptimizerApi extends super.INDArrayOptimizerApi {
    private lazy val delta0: INDArray = super.delta * learningRate
    override def delta: INDArray = delta0
  }
  override type INDArrayOptimizer <: Optimizer with INDArrayOptimizerApi
}
```

```
Factory[Builtins with FixLearningRate].newInstance(learningRate = 0.001)
```

AUTOMATIC HYPERPARAMETER TUNING

ThoughtWorks®

FUTURE WORK

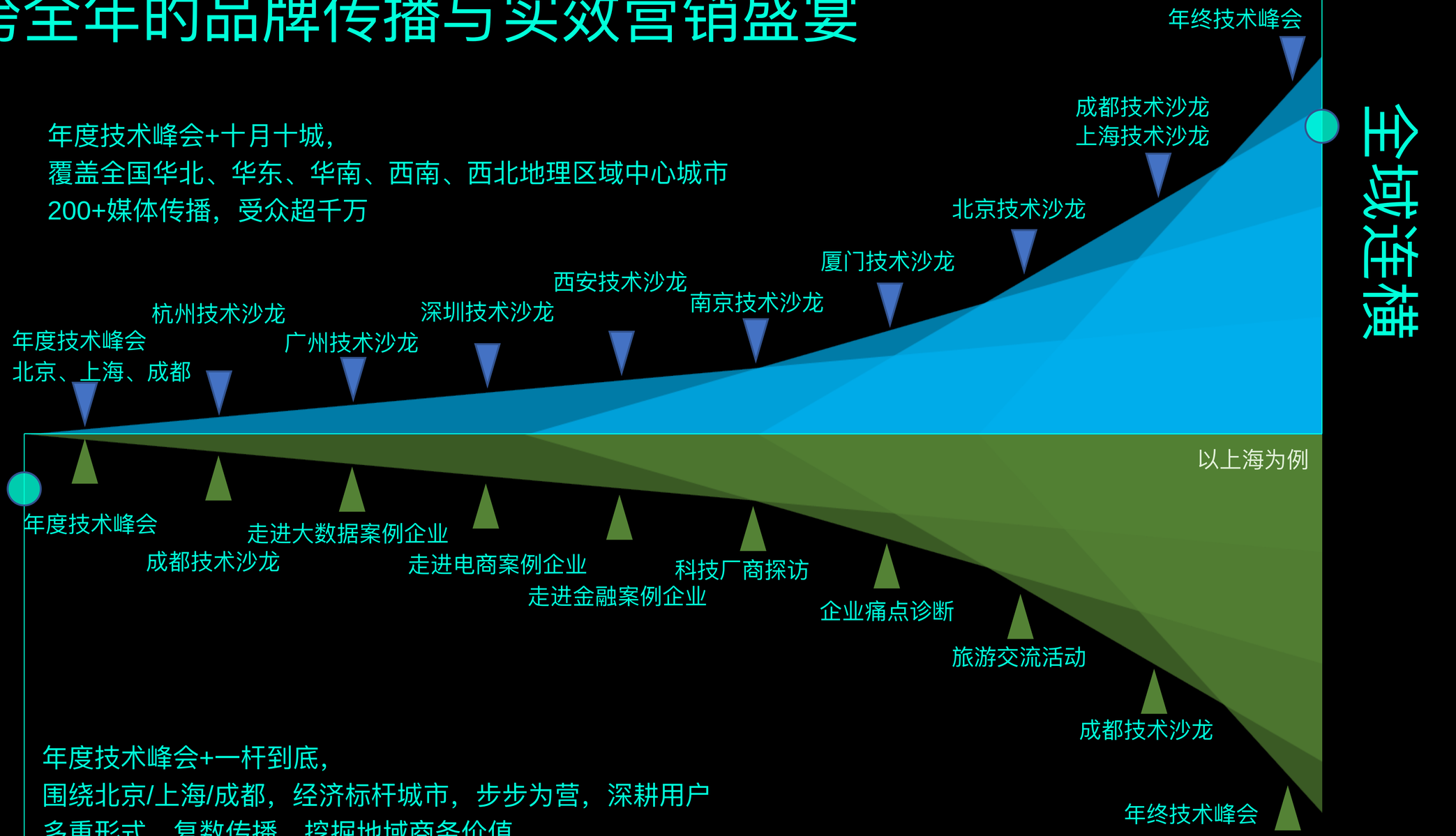
we have done ...

FUTURE WORK

1. NEW BACKEND

2. DISTRIBUTED MODEL

横跨全年的品牌传播与实效营销盛宴



年度技术峰会+十月十城，
覆盖全国华北、华东、华南、西南、西北地理区域中心城市
200+媒体传播，受众超千万

年度技术峰会+一杆到底，
围绕北京/上海/成都，经济标杆城市，步步为营，深耕用户
多重形式，复数传播，挖掘地域商务价值

合纵连横，在中国开发者群体中缔造品牌营销奇迹

中生代技术
FRESHMAN TECHNOLOGY

ArchData技术峰会全国巡回
上海9月，北京9月，成都10月，南京10月，
长沙11月，广州11月
中生代咨询内训
技术架构，研发管理，敏捷开发，大数据
微服务，AI，机器学习
中生代人才内推
对接研发主管，内推精准人才

ThoughtWorks®