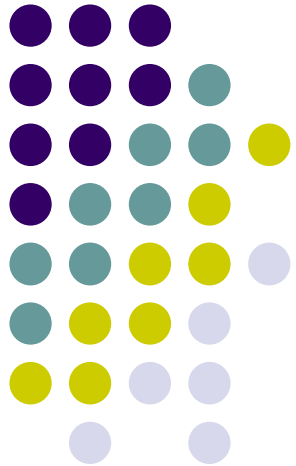
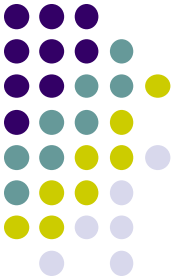
深度学习实现

张梅山

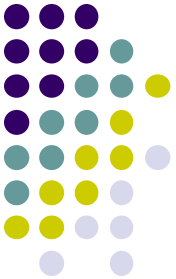


三个阶段



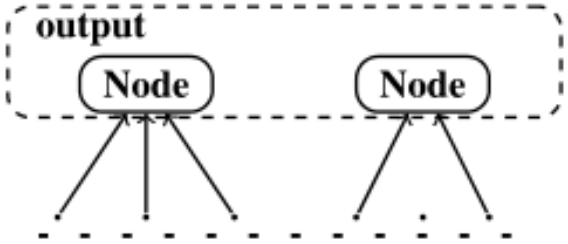
- **Layer-wise**
  - 不少个人代码
- **Operation-wise**
  - TensorFlow, Theano
- **Dynamic Graph**
  - Dynet, Pytorch, Tensorflow fold

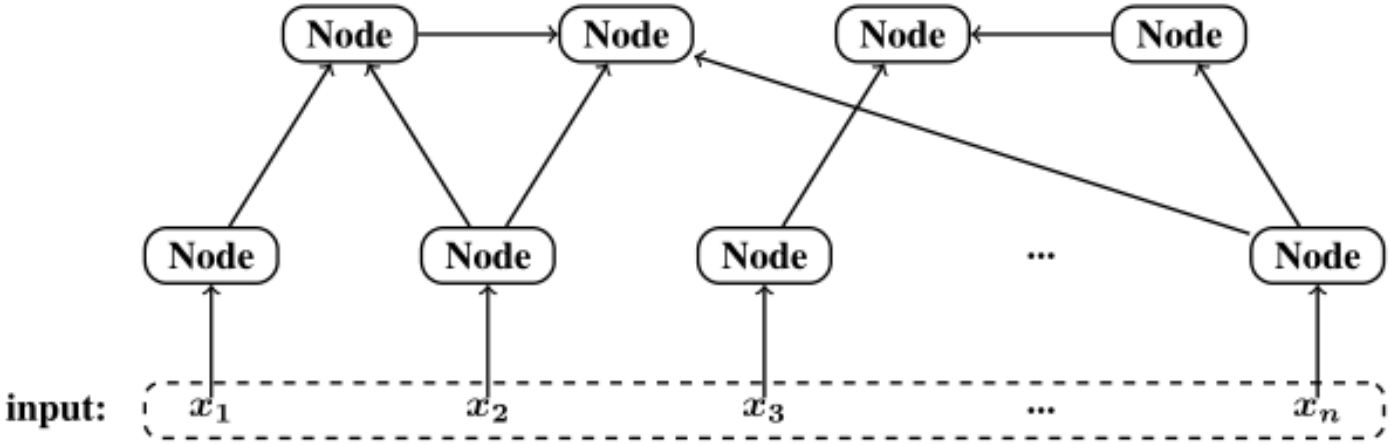
LibN3L 2.0简介



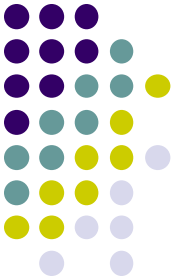
- *机器学习*
  - *目的：预测未知*
  - *如何预测：有指导的学习方法*
  - *如何指导：错误驱动(损失)*

LibN3L 2.0简介


output
Node
Node


Node
Node
Node
Node
Node
Node
Node
...
Node
input:
x_1
x_2
x_3
...
x_n

LibN3L 2.0简介



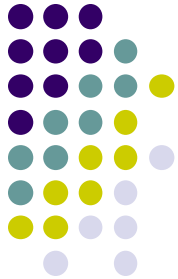
- ***Node***

$$y = f(W_1^p, x_1^q)$$

$$lW_i = ly \frac{\partial y}{\partial W_i}$$

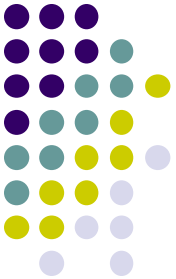
$$lx_i = ly \frac{\partial y}{\partial x_i}$$

LibN3L 2.0简介



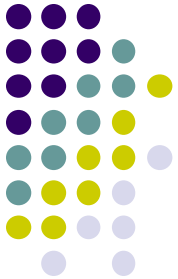
- ***Node***
  - *value　　前向传递*
  - *loss　　反向反馈*
  - *参数集合（若干Node共享）*
  - *如何计算?*
    - *forward（.....）*
    - *backward()*

LibN3L 2.0简介



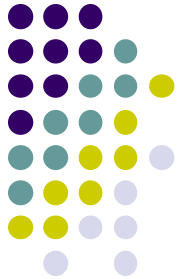
- ***Graph***
  - *一堆nodes*
  - *forward：定义图结构*
  - *backward: 自动*

LibN3L 2.0简介



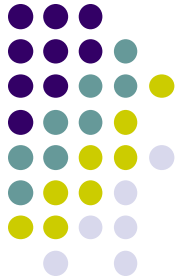
- ***Loss***
  - *output node*
  - *人工定义*

Node



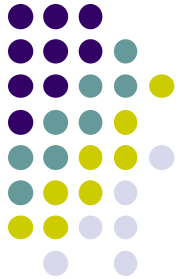
- *一个简单的Node*
  - $y = Wx+b$
  - *forward*　　*输入 x*
  - *backward*

Node



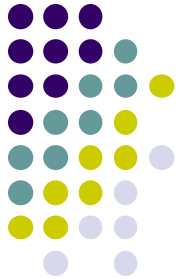
- *一个简单的Node*
  - $y = Wx+b$
  - *实现*

Node



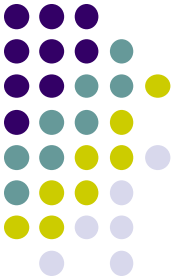
- ***Lookup Table***
  - *E*
  - *forward*
  - *backward*

Node



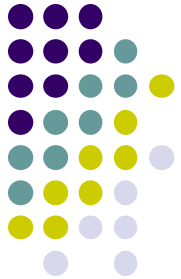
- *基本算子*
  - *Add*
  - *Elem-wise product*
  - *Matrix product*
  - *激活函数*

Node



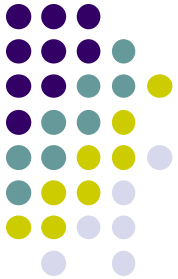
- ***Feed-forward***
  - $y = f(Wx + b)$
  - $y = f(W1x2 + W2x2 + b)$
  - $y = f(W1x2 + W2x2 + W3x3 + b)$
  - $y = f(W1x2 + W2x2 + W3x3 + W4x4 + b)$

Node



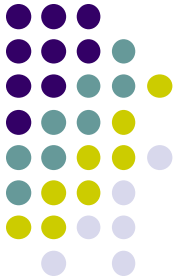
- *其它*
  - *pooling*
  - *concatenate*
  - *dropout*

Node



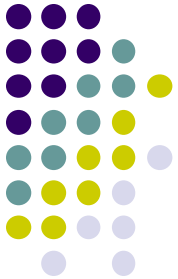
- *更复杂的*
  - *RNN*
  - *LSTM*

Node



- ***Sparse/Discrete***
  - *AP*
  - *普通discrete*

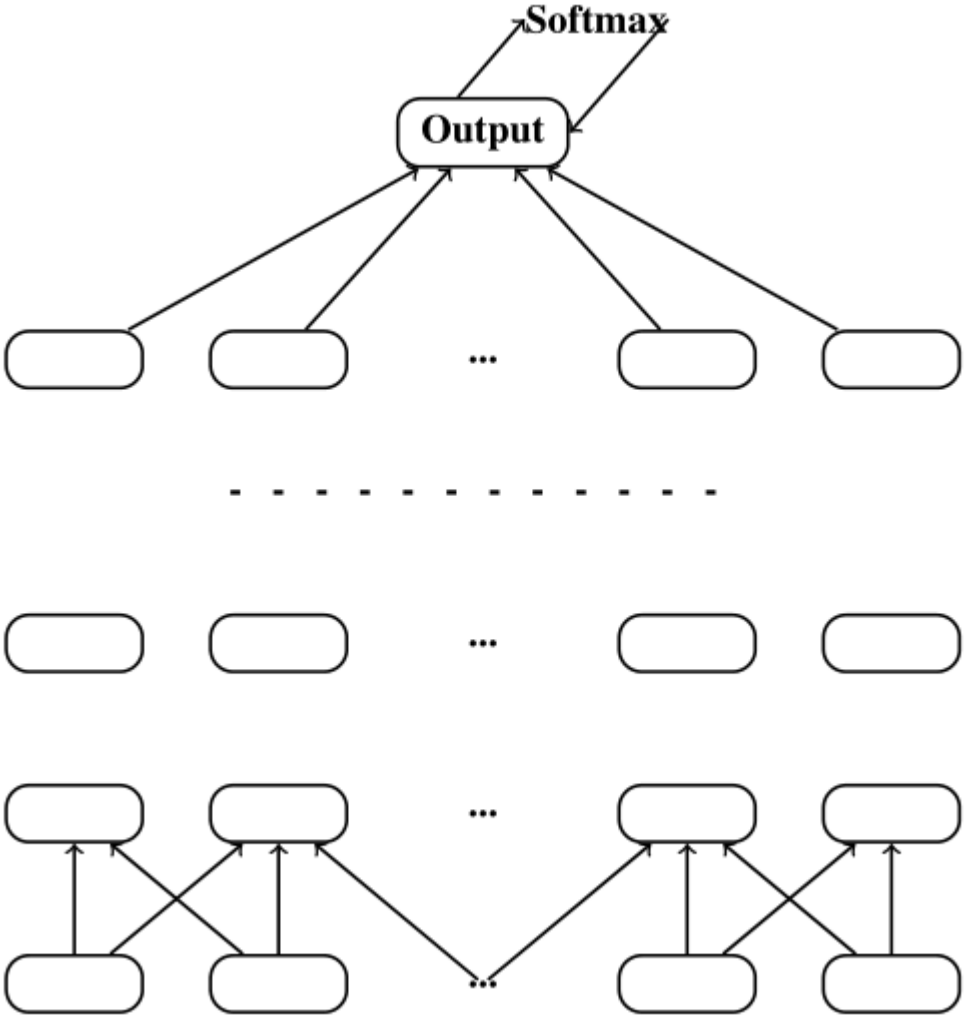
常用损失函数



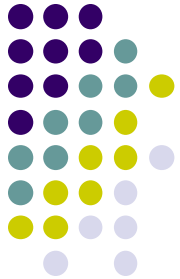
分类

- *softmax*
- *max-margin*

分类



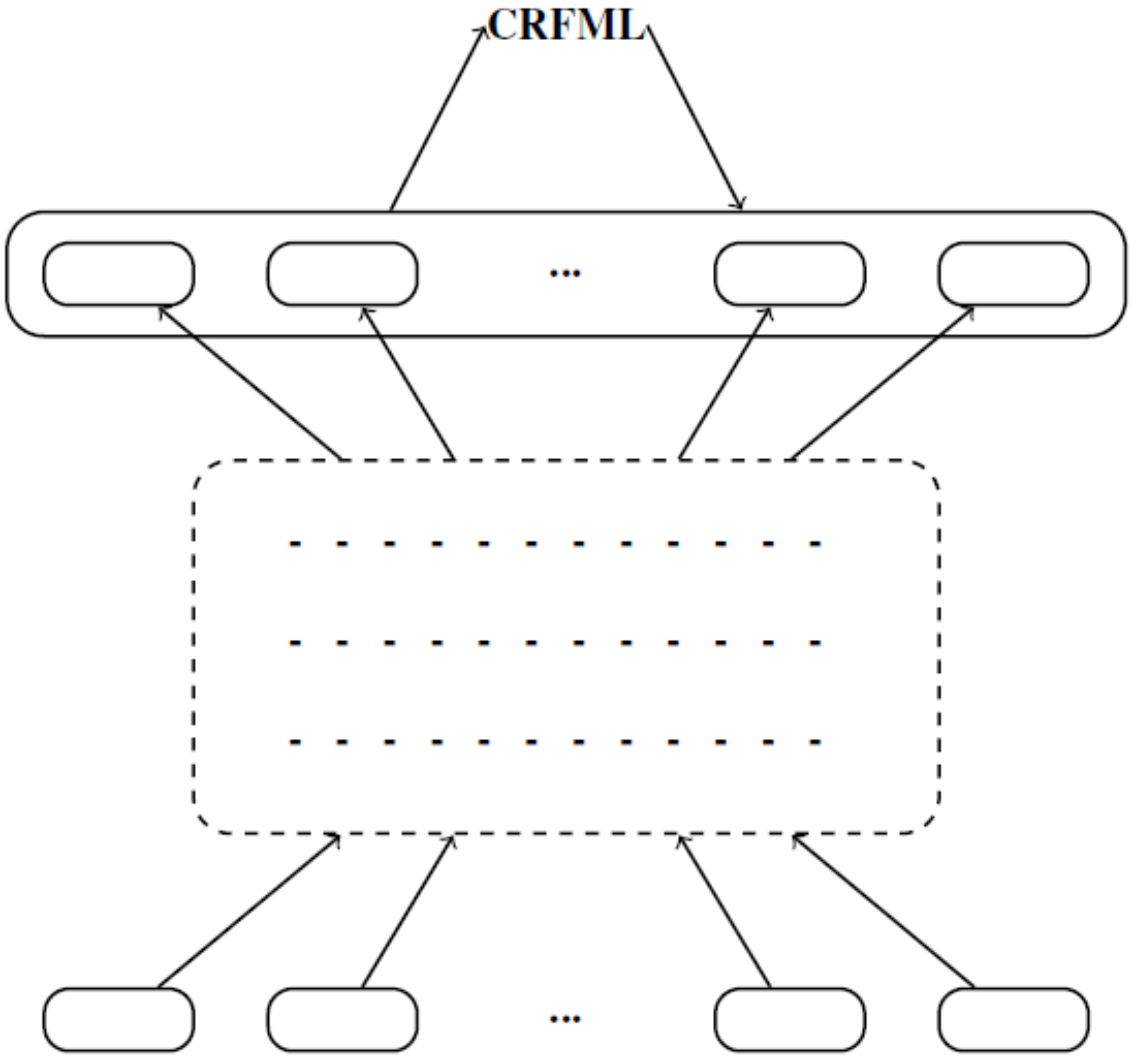
常用损失函数



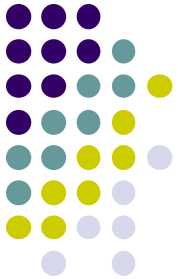
CRF

- *Max likelihood*
- *Max margin*

条件随机场



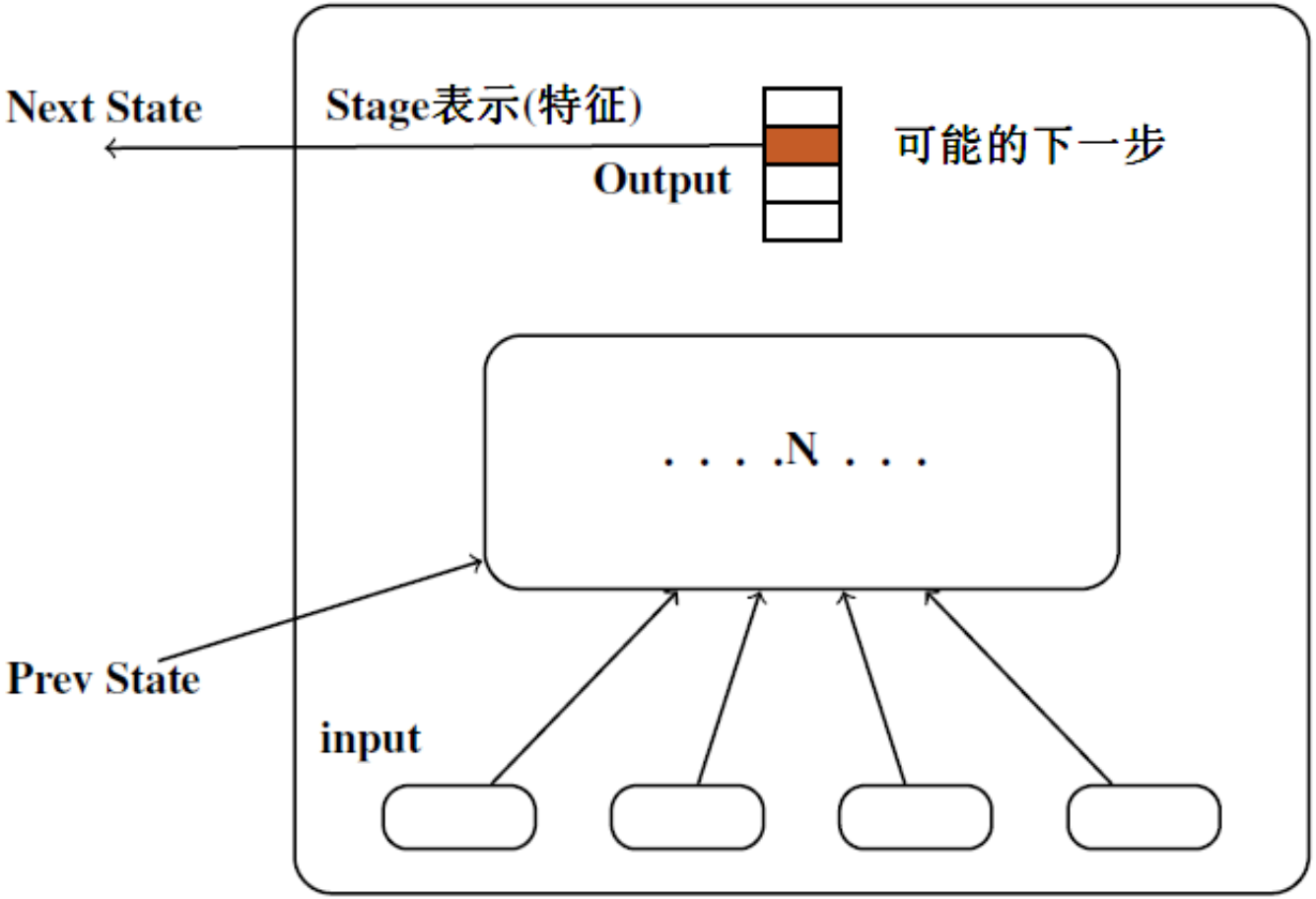
常用损失函数



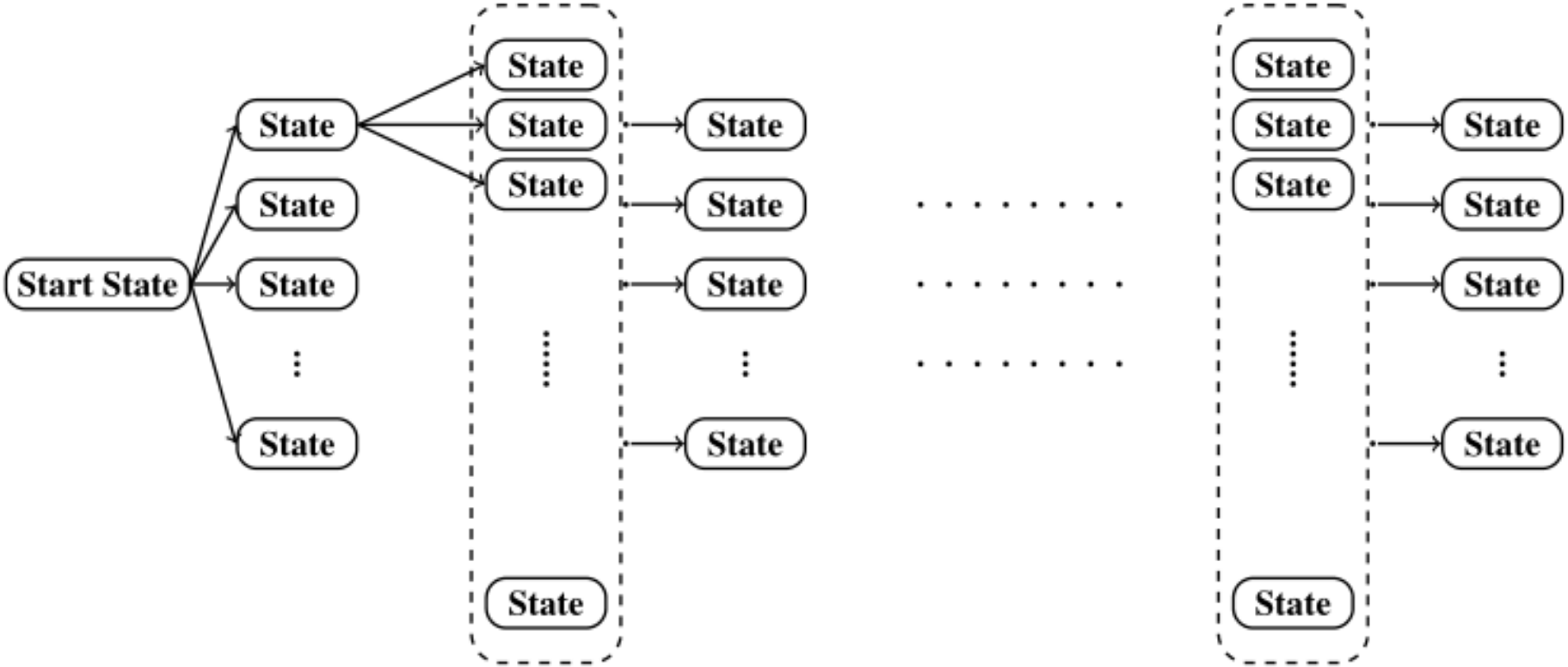
Transition-based or seq-seq

- *Normalized likelihood*
- *Max margin*

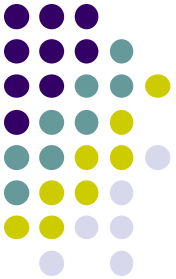
基于转移的算法



基于转移的算法


Start State
State
State
State
State
State
State
State
State
State
State
State
State
State
State
State
State
State
State
State
State
State
State

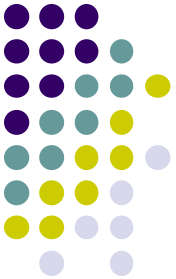
LibN3L 2.0缺点



速度慢！

- 矩阵乘法过多
- 小矩阵运算，很难被优化 (mkl, cuda)

解决方法

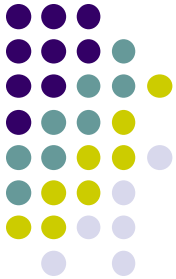


Node 在forward时不执行真正运算

Execute类，对node进行批量运算

```
struct Execute {
public:
  vector<PNode> batch;

public:
  virtual inline void forward() = 0;
  virtual inline void backward() = 0;
```

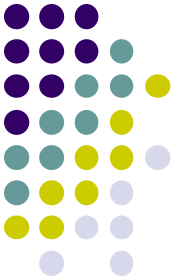
解决方法



如果必须要运算时，执行(Graph)：

```
while (free_nodes is not empty):
  execs = []
  for node in  free_nodes:
    bool find = false
    for exec in execs:
      if exec.canAdd(node):
        find = true
        exec.add(node)
        break

    if find is false:
      PExecute exec = node->generate()
      execs.push_back(exec)

  for exec in execs:
    exec->forward()
    addToGraph(exec)

  update(free_nodes)
```

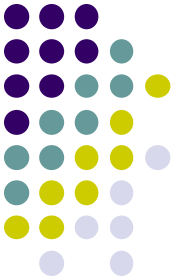
解决方法



typeEqual

```
inline bool typeEqual(PNode other) {
  bool result = Node::typeEqual(other);
  if (!result) return false;
  UniNode* conv_other = (UniNode*)other;
  if (param == conv_other->param
  && activate == conv_other->activate) {
    return true;
  }
}
```

解决方法



Execute generator

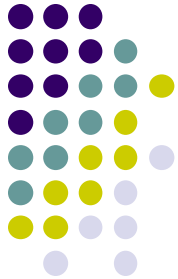
```
inline PExecute UniNode::generate() {
    UniExecute* exec = new UniExecute();
    exec->batch.push_back(this);
    exec->inDim = param->W.inDim();
    exec->outDim = param->W.outDim();
    exec->param = param;
    exec->activate = activate;
    exec->derivate = derivate;
    return exec;
}
```

解决方法

forward

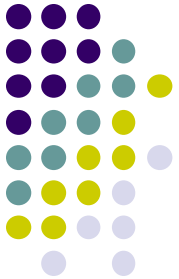
```
合并:
  x = [b[0].x, b[1].x, ...]

计算:
  ty.mat() = param->W.val.mat() * x.mat();

  if (param->bUseB) {
    ty.vec() = ty.vec() + b.vec();
  }

  y.vec() = ty.vec().unaryExpr(ptr_fun(activate));

分发:
  [b[0].y, b[1].y, ...] = y
```

解决方法



backward

```
合并:
  ly = [b[0].ly, b[1].ly, ...]

计算:
  lty.vec() = ly.vec() * ty.vec().binaryExpr(y.vec(), ptr_fun(derivate));
  param->W.grad.mat() += lty.mat() * x.mat().transpose();

  if (param->bUseB) {
    for (int idx = 0; idx < size; idx++)
      param->b.grad.vec += lty[idx];
  }

分发:
  [b[0].lx, b[1].lx, ...] = lx
```



谢谢

Q/A?