Recurrent Neural Network with Caffe

Bonan CUAN

Recurrent Neural Network

- RNN: often handling sequential data

  to find patterns through time

- Weight sharing through time

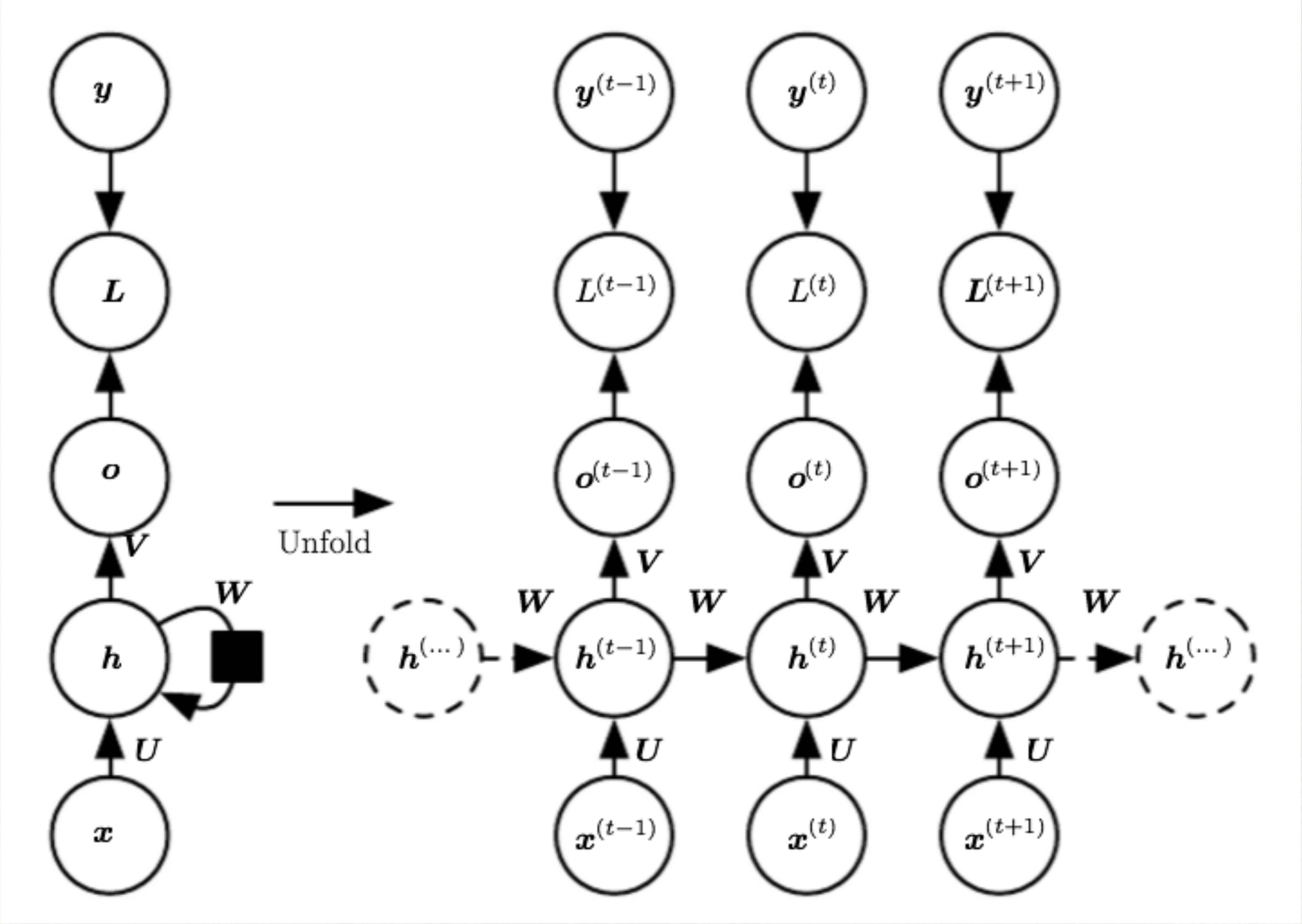
  same weight matrices, invariant to time shift

- How to design recurrence?

- Unfolding/unrolling of RNN

  unfolded RNN: sequence becomes serial states in graph

Unrolling & Weight Sharing



Caffe

- A widely-adopted deep learning framework

  Berkeley Vision & Learning Center (BVLC)

- Soft-coding with multiple interfaces

  command lines, Python, MATLAB

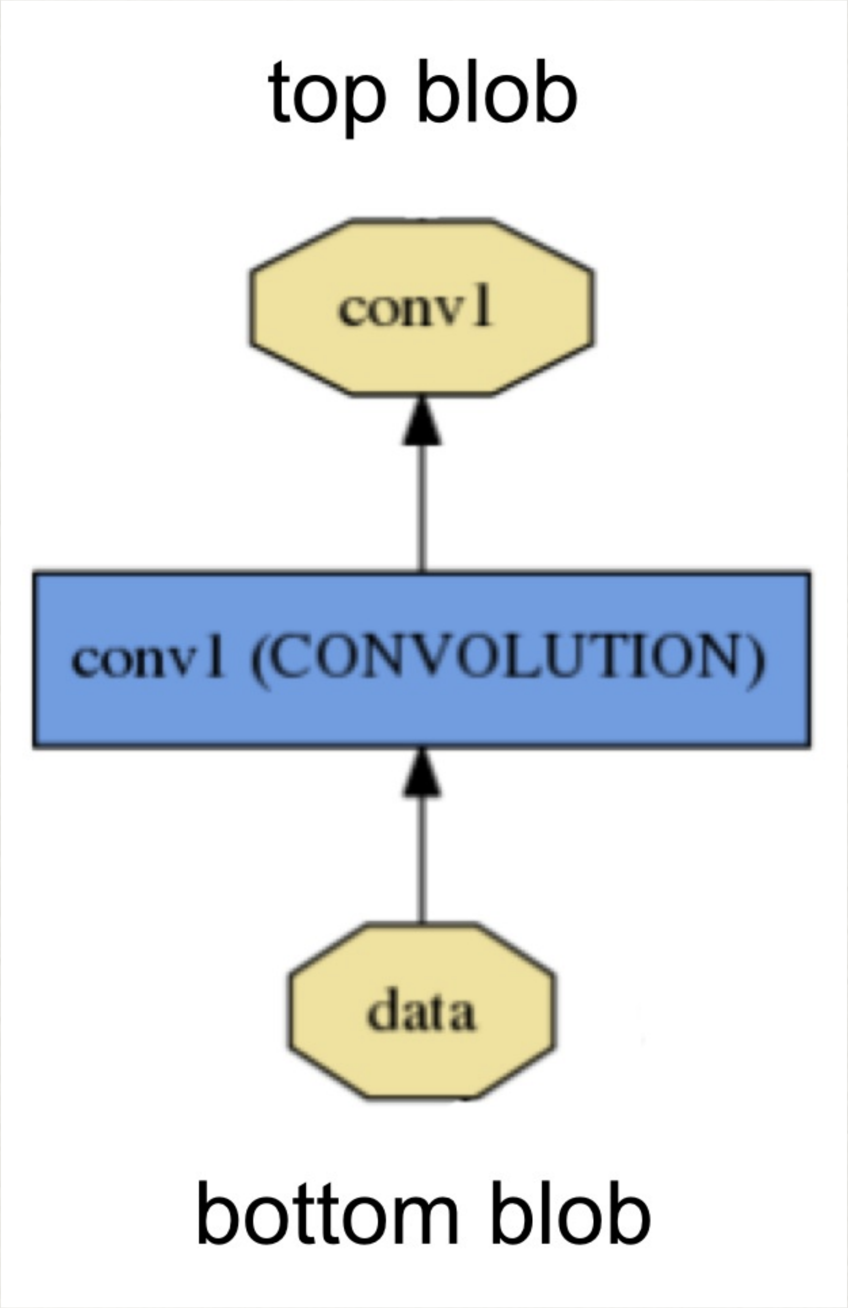
- Modularity and Extensibility

  easy to design customized nets

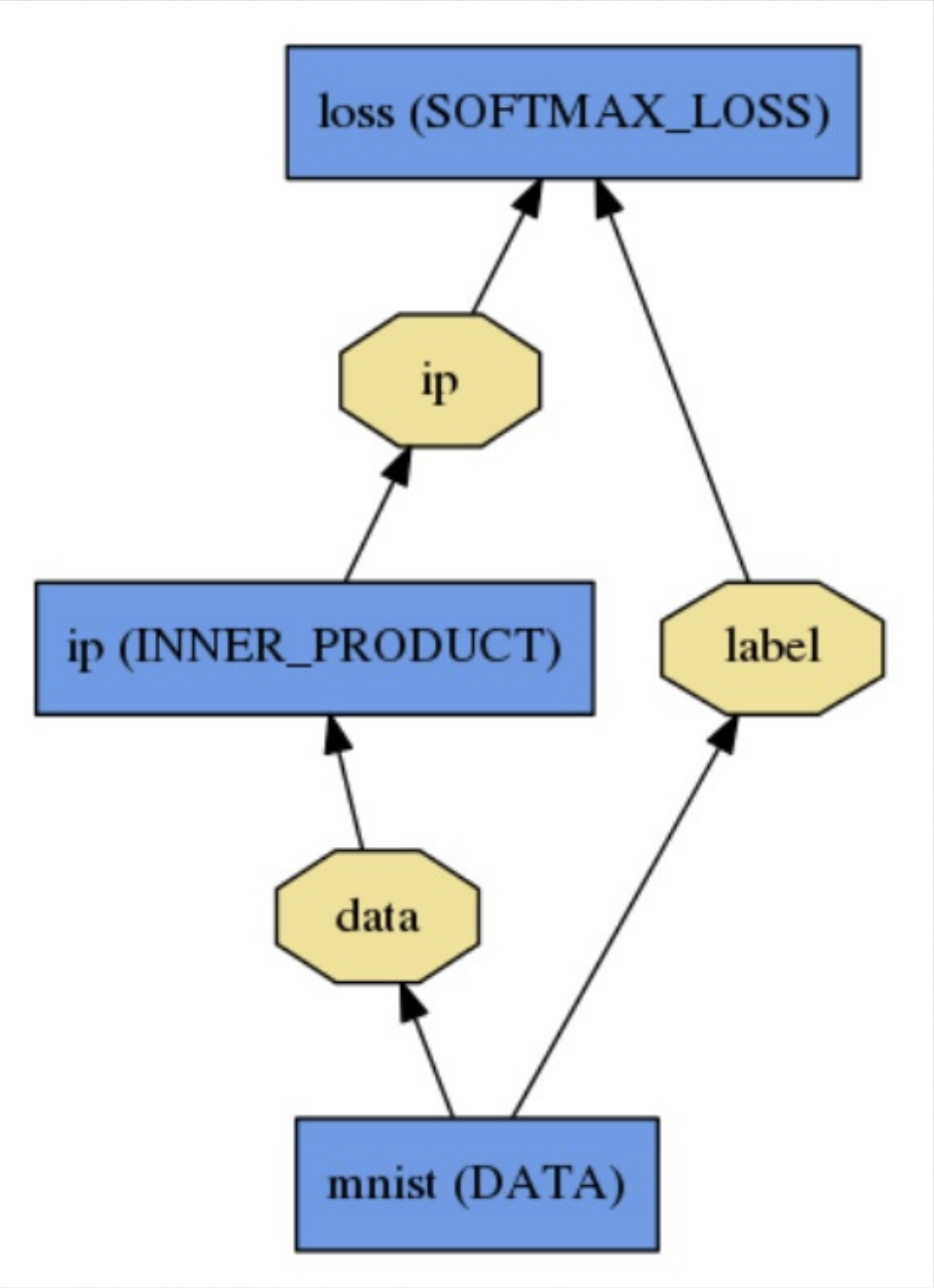
- Speed

  fastest open-source deep learning framework

Caffe Net Example



Blobs and Layer



Net Example

Caffe Net Example

```
layer {
  name: "data"
  type: "Data"
  top: "data"
  top: "label"
  transform_param {
    scale: 0.00392156862745
  }
  data_param {
    source: "examples/mnist/mnist_test_lmdb"
    batch_size: 100
    backend: LMDB
  }
}
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  convolution_param {
    num_output: 20
    kernel_size: 5
    weight_filler {
      type: "xavier"
    }
  }
}
layer {
  name: "pool1"
  type: "Pooling"
  bottom: "conv1"
  top: "pool1"
  pooling_param {
    pool: MAX
    kernel_size: 2
    stride: 2
  }
}
```

Blobs

Net

parameters
(weight or bias)

Layers

RNN with Caffe?

- No official version

  no built-in sequential parameter sharing mechanism

  possible with unrolling structure

- Manually shared parameters

  by using the same “name” attribute

  NOT convenient: too many “ctrl+c/v” when facing long sequences

- Third-party realization

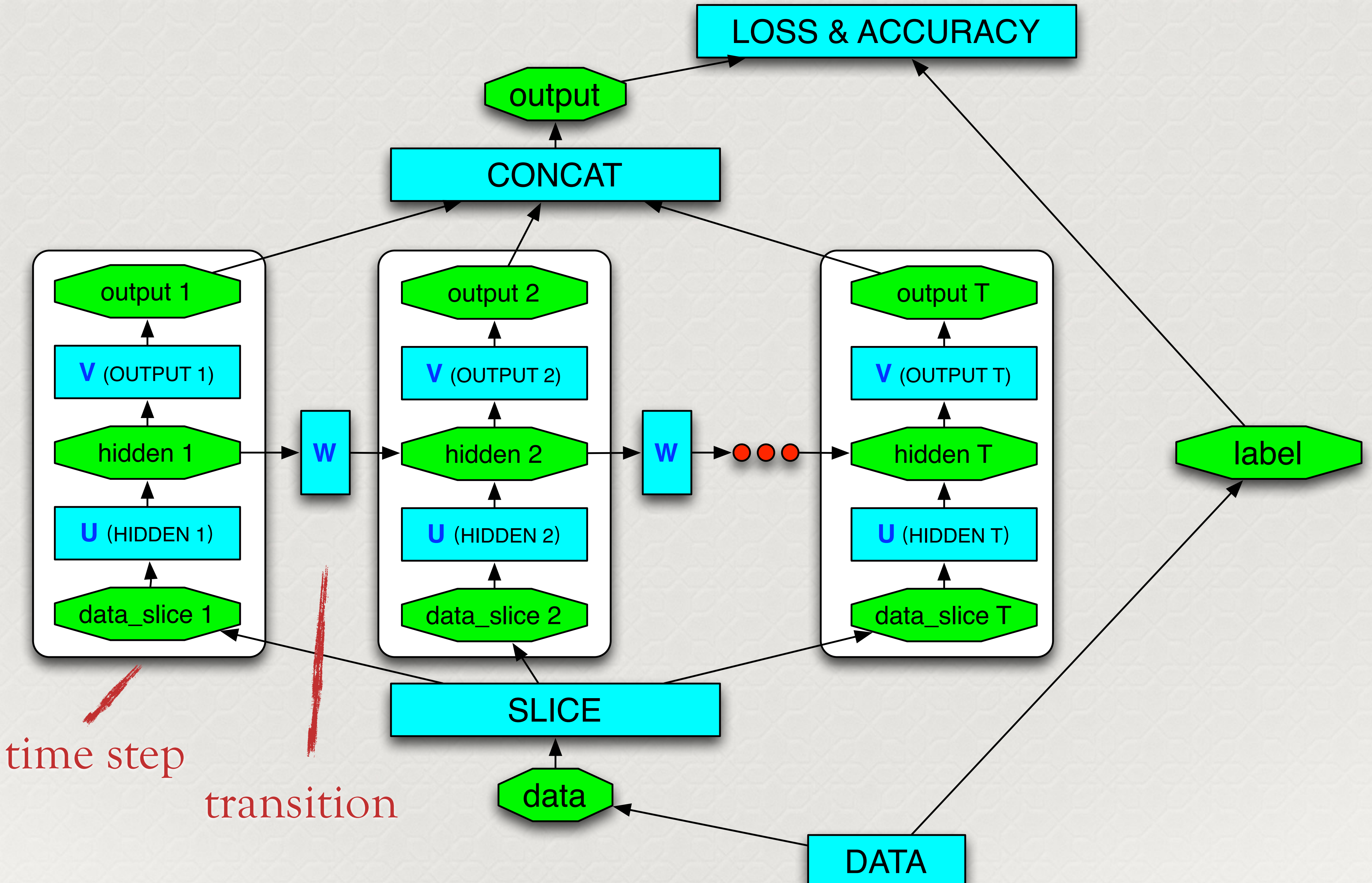
  e.g. Jeff Donahue’s Unrolled RNN, but out of date

  reinvent the wheel!

RNN with Caffe!

- **PyCaffe** or MatCaffe
  - rapid prototyping
  - easy access to nets, layers, blobs, parameters, etc.
- Automatic unrolling of RNN
  - by slicing the input to feed multiple time steps
  - by assigning different “name” attributes to the same layer in different time steps
  - by concatenating the outputs of all time steps
- Parameter sharing through time
  - by assigning the same “name” attribute to the parameters in different time steps

Unrolled RNN Structure



Binary Addition with RNN

```python
def binary_addition_train(num_steps, batch_size):
  # number of hidden units
  num_hu = 3

  # net spec
  ns = caffe.NetSpec()

  # inputs
  ns.data, ns.label = L.HDF5Data(name='HDF5', include=dict(phase=caffe.TRAIN), ntop=2,

  # slice
  X = L.Slice(ns.data, name='X', ntop=num_steps, slice_param=dict(axis=3, slice_point=

  #initial hidden units
  ns.H = L.DummyData(dummy_data_param=dict(shape=dict(dim=[batch_size, 1, num_hu]), da

  output = []
  for t in xrange(num_steps):
    step = str(t+1)

    zX = L.InnerProduct(X[t], param=[dict(name='WX',lr_mult=1), dict(name='bX',lr_mult
    ns.__setattr__('zX'+step, zX)

    zH = L.InnerProduct(ns.H, param=dict(name='WH',lr_mult=1), bias_term=False, num_ou
    ns.__setattr__('zH'+step, zH)

    z = L.Eltwise(zX, zH, eltwise_param=dict(operation=P.Eltwise.SUM))
    ns.__setattr__('z'+step, z)

    a = L.Sigmoid(z)
    ns.__setattr__('a'+step, a)

    out = L.InnerProduct(a, param=[dict(name='WO',lr_mult=1), dict(name='bO',lr_mult=1
    ns.__setattr__('out'+step, out)
    output.append(out)

    ns.__setattr__('H',a)

  ns.output = L.Concat(*output, concat_param=dict(axis=1))
  #ns.accuracy = L.Accuracy(ns.output, ns.label)
  ns.loss = L.EuclideanLoss(ns.output, ns.label)

  return ns.to_proto()
```

slice the input

assign different names to layers in different time steps

but parameter name remains the same

concatenate the output

Some Experiments

- GPU mode overwhelms CPU mode
- Difference of CPU changes little
- MKL is much faster than other BLASs, e.g. ATLAS & OpenBLAS