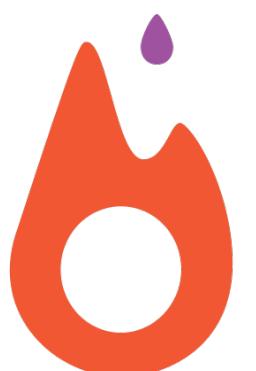




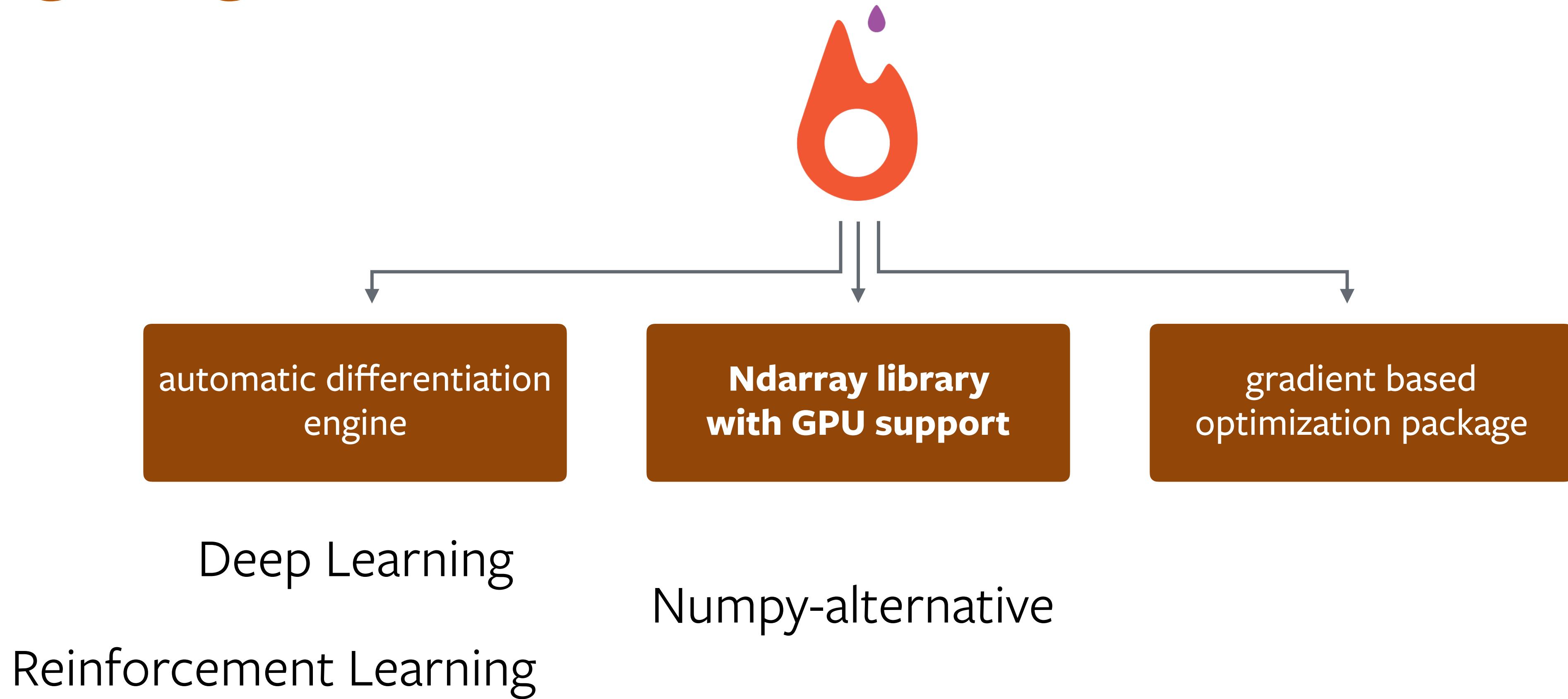
**an ndarray library with GPU support
a deep learning framework**

Soumith Chintala, Adam Paszke, Sam Gross

Facebook AI Research



Overview



ndarray library

with GPU support



ndarray library

- np.ndarray <-> torch.Tensor
- 200+ operations, similar to numpy
- very fast acceleration on NVIDIA GPUs



```

# -*- coding: utf-8 -*-
import numpy as np

# N is batch size; D_in is input dimension;
# H is hidden dimension; D_out is output dimension.
N, D_in, H, D_out = 64, 1000, 100, 10

# Create random input and output data
x = np.random.randn(N, D_in)
y = np.random.randn(N, D_out)

# Randomly initialize weights
w1 = np.random.randn(D_in, H)
w2 = np.random.randn(H, D_out)

learning_rate = 1e-6
for t in range(500):
    # Forward pass: compute predicted y
    h = x.dot(w1)
    h_relu = np.maximum(h, 0)
    y_pred = h_relu.dot(w2)

    # Compute and print loss
    loss = np.square(y_pred - y).sum()
    print(t, loss)

    # Backprop to compute gradients of w1 and w2 with respect to loss
    grad_y_pred = 2.0 * (y_pred - y)
    grad_w2 = h_relu.T.dot(grad_y_pred)
    grad_h_relu = grad_y_pred.dot(w2.T)
    grad_h = grad_h_relu.copy()
    grad_h[h < 0] = 0
    grad_w1 = x.T.dot(grad_h)

    # Update weights
    w1 -= learning_rate * grad_w1
    w2 -= learning_rate * grad_w2

```

Numpy

```

import torch
dtype = torch.FloatTensor
# dtype = torch.cuda.FloatTensor # Uncomment this to run on GPU

# N is batch size; D_in is input dimension;
# H is hidden dimension; D_out is output dimension.
N, D_in, H, D_out = 64, 1000, 100, 10

# Create random input and output data
x = torch.randn(N, D_in).type(dtype)
y = torch.randn(N, D_out).type(dtype)

# Randomly initialize weights
w1 = torch.randn(D_in, H).type(dtype)
w2 = torch.randn(H, D_out).type(dtype)

learning_rate = 1e-6
for t in range(500):
    # Forward pass: compute predicted y
    h = x.mm(w1)
    h_relu = h.clamp(min=0)
    y_pred = h_relu.mm(w2)

    # Compute and print loss
    loss = (y_pred - y).pow(2).sum()
    print(t, loss)

    # Backprop to compute gradients of w1 and w2 with respect to loss
    grad_y_pred = 2.0 * (y_pred - y)
    grad_w2 = h_relu.t().mm(grad_y_pred)
    grad_h_relu = grad_y_pred.mm(w2.t())
    grad_h = grad_h_relu.clone()
    grad_h[h < 0] = 0
    grad_w1 = x.t().mm(grad_h)

    # Update weights using gradient descent
    w1 -= learning_rate * grad_w1
    w2 -= learning_rate * grad_w2

```

PyTorch

ndarray / Tensor library

Tensors are similar to numpy's ndarrays, with the addition being that Tensors can also be used on a GPU to accelerate computing.

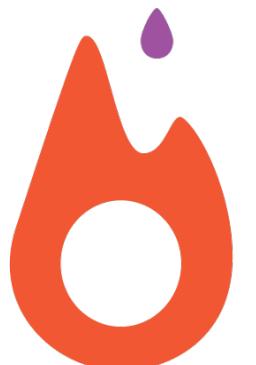
```
from __future__ import print_function  
import torch
```

Construct a 5x3 matrix, uninitialized:

```
x = torch.Tensor(5, 3)  
print(x)
```

Out:

```
1.00000e-25 *  
 0.4136  0.0000  0.0000  
 0.0000  1.6519  0.0000  
 1.6518  0.0000  1.6519  
 0.0000  1.6518  0.0000  
 1.6520  0.0000  1.6519  
[torch.FloatTensor of size 5x3]
```



ndarray / Tensor library

Construct a randomly initialized matrix

```
x = torch.rand(5, 3)
print(x)
```

Out:

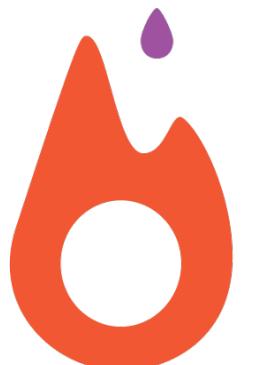
```
0.2598  0.7231  0.8534
0.3928  0.1244  0.5110
0.5476  0.2700  0.5856
0.7288  0.9455  0.8749
0.6663  0.8230  0.2713
[torch.FloatTensor of size 5x3]
```

Get its size

```
print(x.size())
```

Out:

```
torch.Size([5, 3])
```



ndarray / Tensor library

You can use standard numpy-like indexing with all bells and whistles!

```
print(x[:, 1])
```

Out:

```
0.7231
0.1244
0.2700
0.9455
0.8230
[torch.FloatTensor of size 5]
```



ndarray / Tensor library

```
y = torch.rand(5, 3)  
print(x + y)
```

Out:

```
0.7931  1.1872  1.6143  
1.1946  0.4669  0.9639  
0.7576  0.8136  1.1897  
0.7431  1.8579  1.3400  
0.8188  1.1041  0.8914  
[torch.FloatTensor of size 5x3]
```



NumPy bridge

Converting torch Tensor to numpy Array

```
a = torch.ones(5)  
print(a)
```

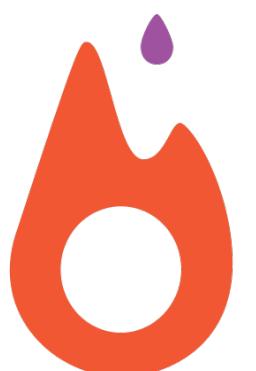
Out:

```
1  
1  
1  
1  
1  
[torch.FloatTensor of size 5]
```

```
b = a.numpy()  
print(b)
```

Out:

```
[ 1.  1.  1.  1.  1.]
```



NumPy bridge

Converting torch Tensor to numpy Array

```
a = torch.ones(5)  
print(a)
```

Out:

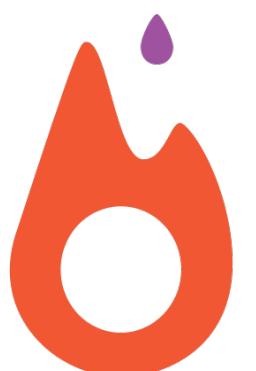
```
1  
1  
1  
1  
1  
[torch.FloatTensor of size 5]
```

**Zero memory-copy
very efficient**

```
b = a.numpy()  
print(b)
```

Out:

```
[ 1.  1.  1.  1.  1.]
```



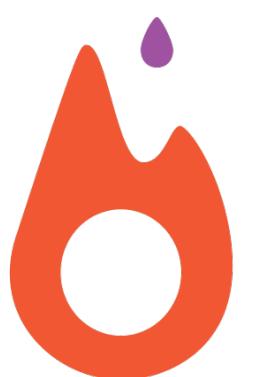
NumPy bridge

See how the numpy array changed in value.

```
a.add_(1)  
print(a)  
print(b)
```

Out:

```
2  
2  
2  
2  
2  
[torch.FloatTensor of size 5]  
[ 2.  2.  2.  2.  2.]
```



NumPy bridge

Converting numpy Array to torch Tensor

See how changing the np array changed the torch Tensor automatically

```
import numpy as np
a = np.ones(5)
b = torch.from_numpy(a)
np.add(a, 1, out=a)
print(a)
print(b)
```

Out:

```
[ 2.  2.  2.  2.  2.]
2
2
2
2
2
[torch.DoubleTensor of size 5]
```

All the Tensors on the CPU except a CharTensor support converting to NumPy and back.



Seamless GPU Tensors

CUDA Tensors

Tensors can be moved onto GPU using the `.cuda` function.

```
# let us run this cell only if CUDA is available
if torch.cuda.is_available():
    x = x.cuda()
    y = y.cuda()
    x + y
```



GPUs are fast

- Buy \$700 NVIDIA 1080
- 100x faster matrix multiply
- 10x faster operations in general on matrices



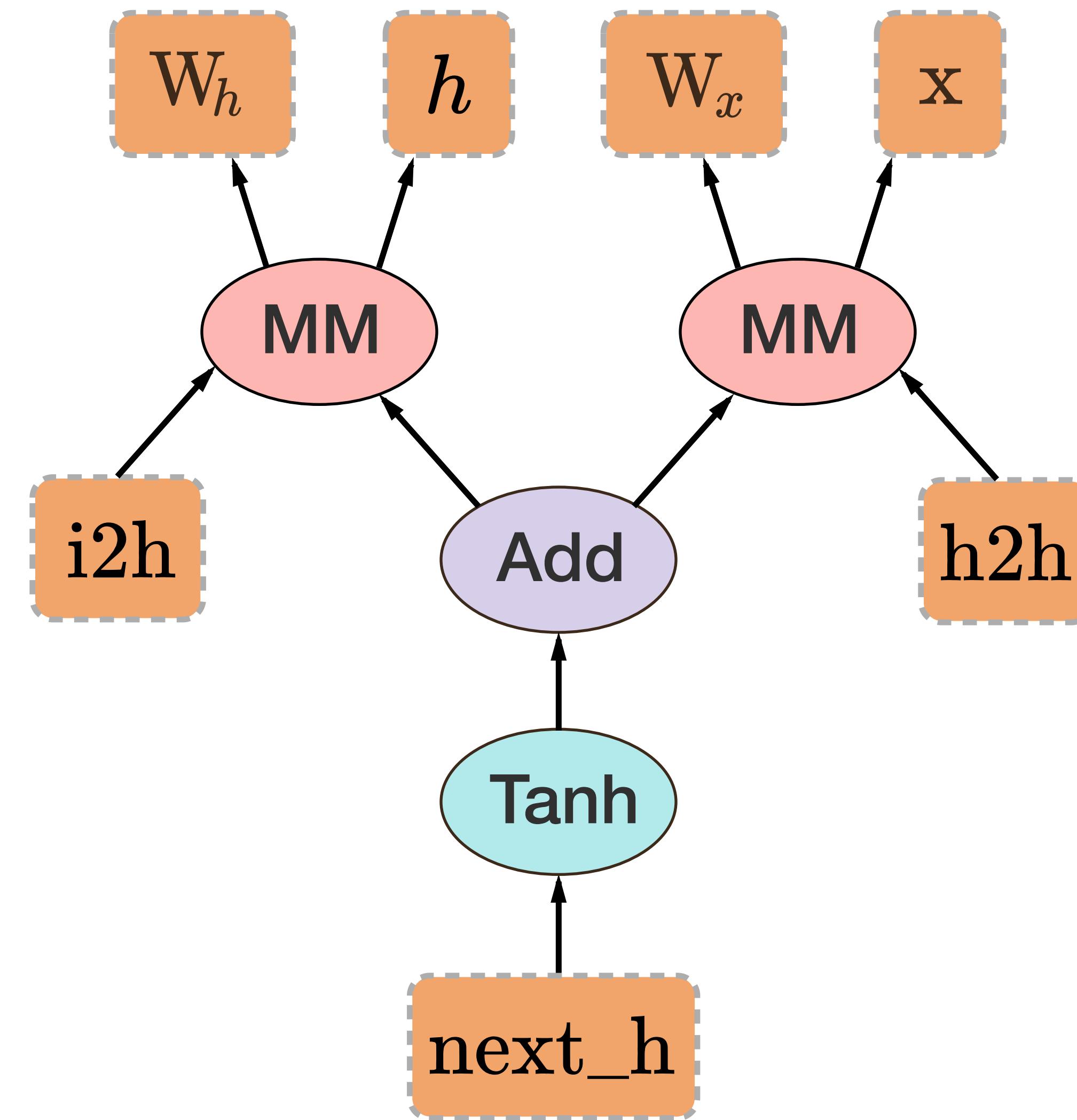
automatic differentiation engine

for deep learning and reinforcement learning



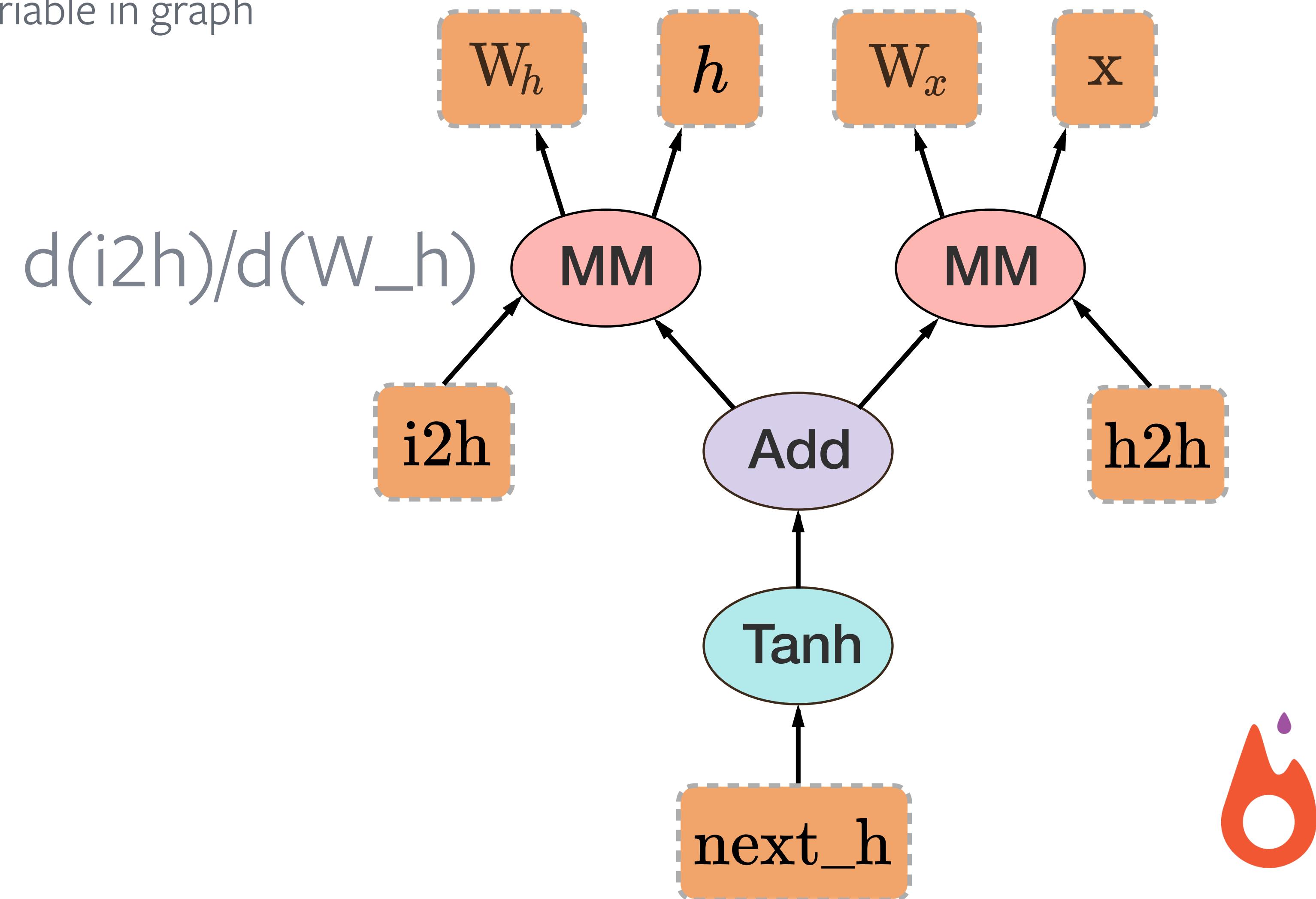
Deep Learning Frameworks

- Provide gradient computation
- Gradient of one variable w.r.t. any variable in graph



Deep Learning Frameworks

- Provide gradient computation
 - Gradient of one variable w.r.t. any variable in graph

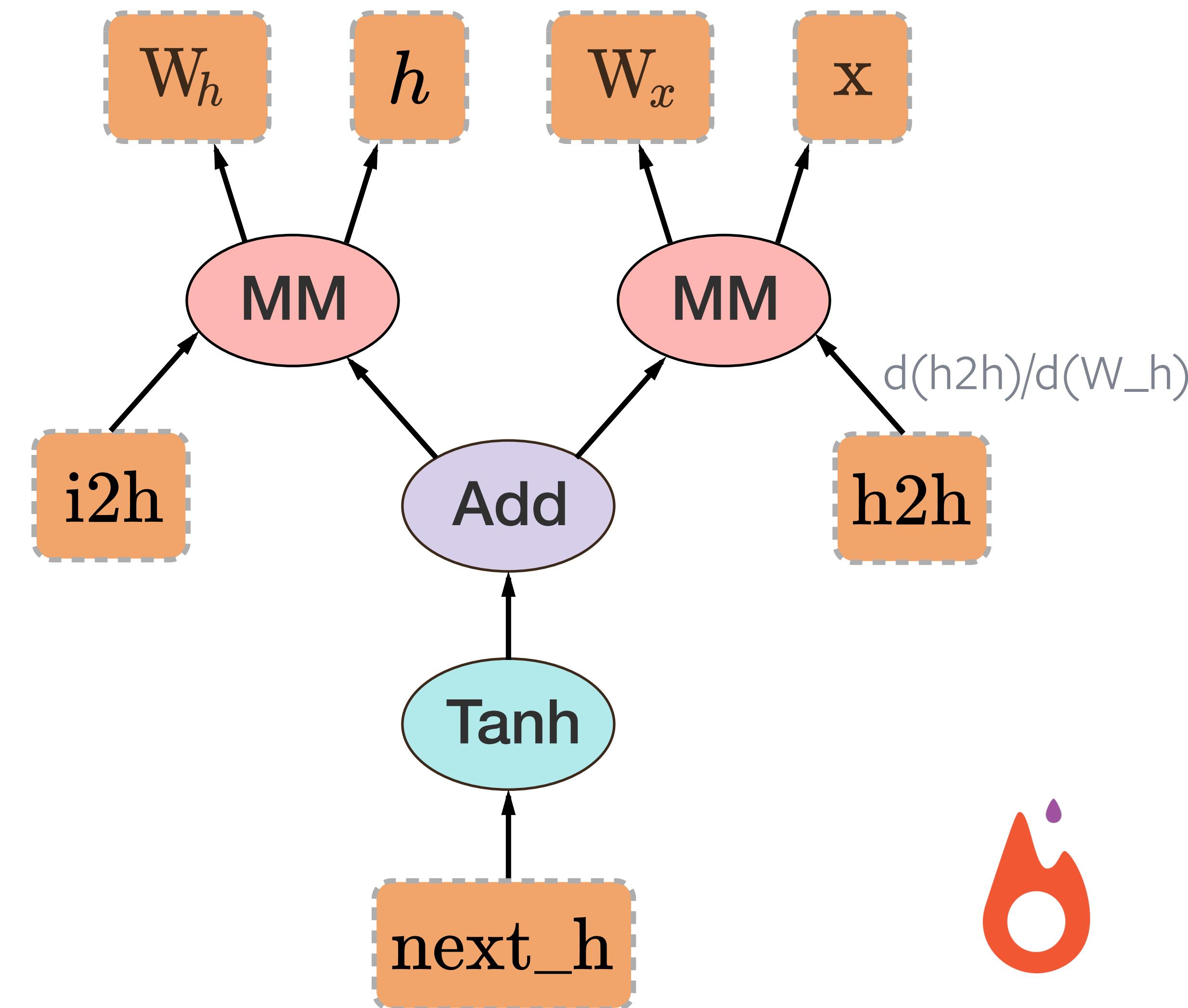


Deep Learning Frameworks

In addition to Computation

Graph Toolkits

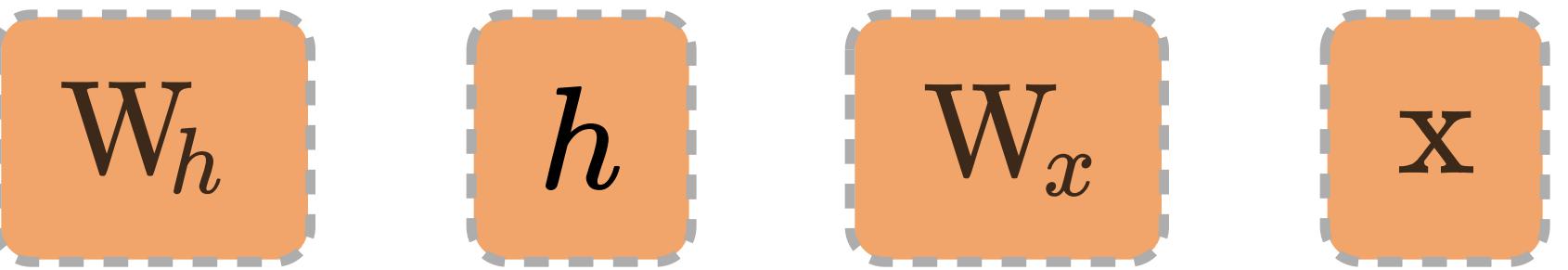
- Provide gradient computation
 - Gradient of one variable w.r.t. any variable in graph
- Provide integration with high performance DL libraries like CuDNN



PyTorch Autograd

```
from torch.autograd import Variable
```

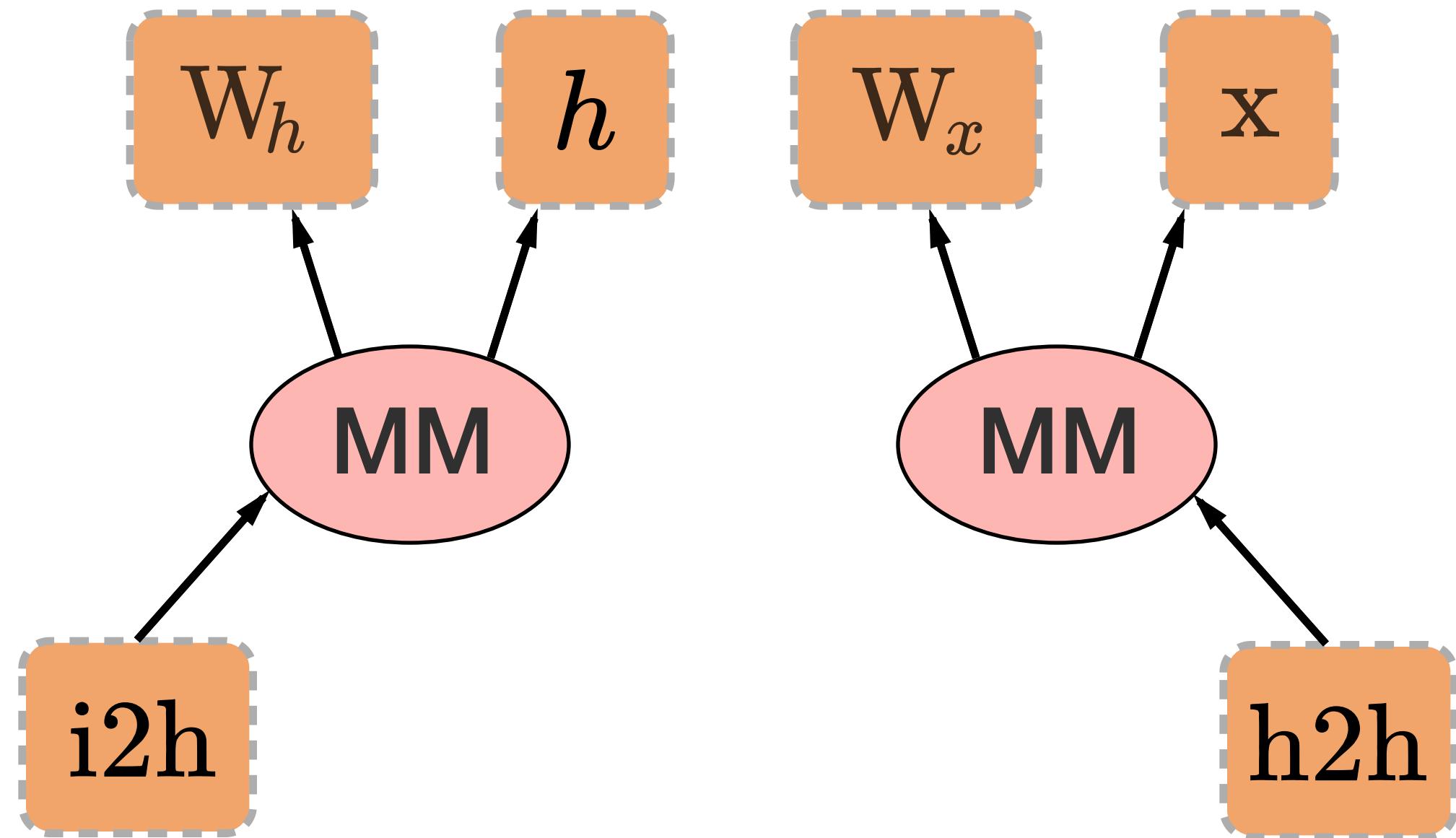
PyTorch Autograd



```
from torch.autograd import Variable  
  
x = Variable(torch.randn(1, 10))  
prev_h = Variable(torch.randn(1, 20))  
W_h = Variable(torch.randn(20, 20))  
W_x = Variable(torch.randn(20, 10))
```

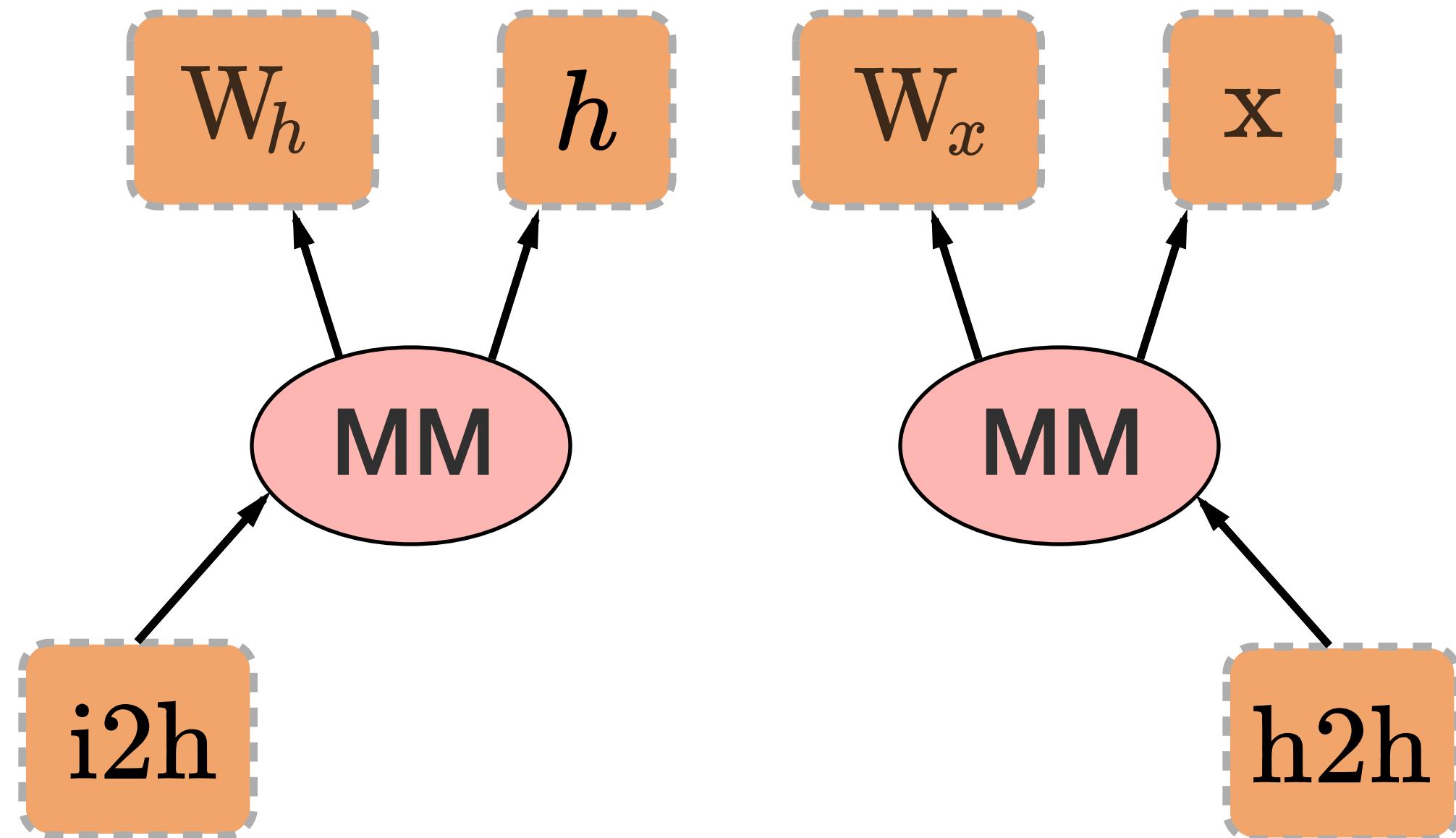
PyTorch Autograd

```
from torch.autograd import Variable  
  
x = Variable(torch.randn(1, 10))  
prev_h = Variable(torch.randn(1, 20))  
W_h = Variable(torch.randn(20, 20))  
W_x = Variable(torch.randn(20, 10))  
  
i2h = torch.mm(W_x, x.t())  
h2h = torch.mm(W_h, prev_h.t())
```



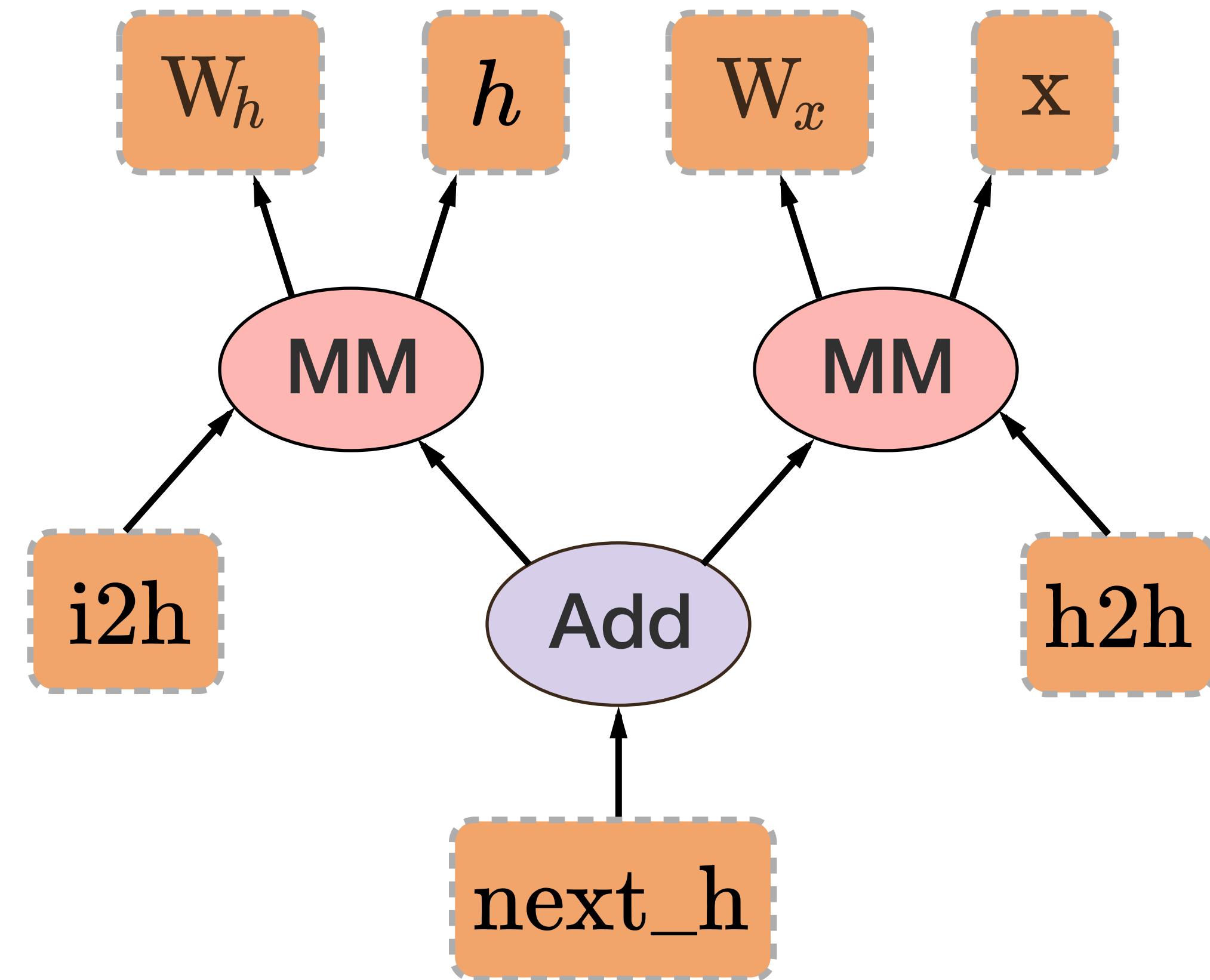
PyTorch Autograd

```
from torch.autograd import Variable  
  
x = Variable(torch.randn(1, 10))  
prev_h = Variable(torch.randn(1, 20))  
W_h = Variable(torch.randn(20, 20))  
W_x = Variable(torch.randn(20, 10))  
  
i2h = torch.mm(W_x, x.t())  
h2h = torch.mm(W_h, prev_h.t())  
next_h = i2h + h2h
```



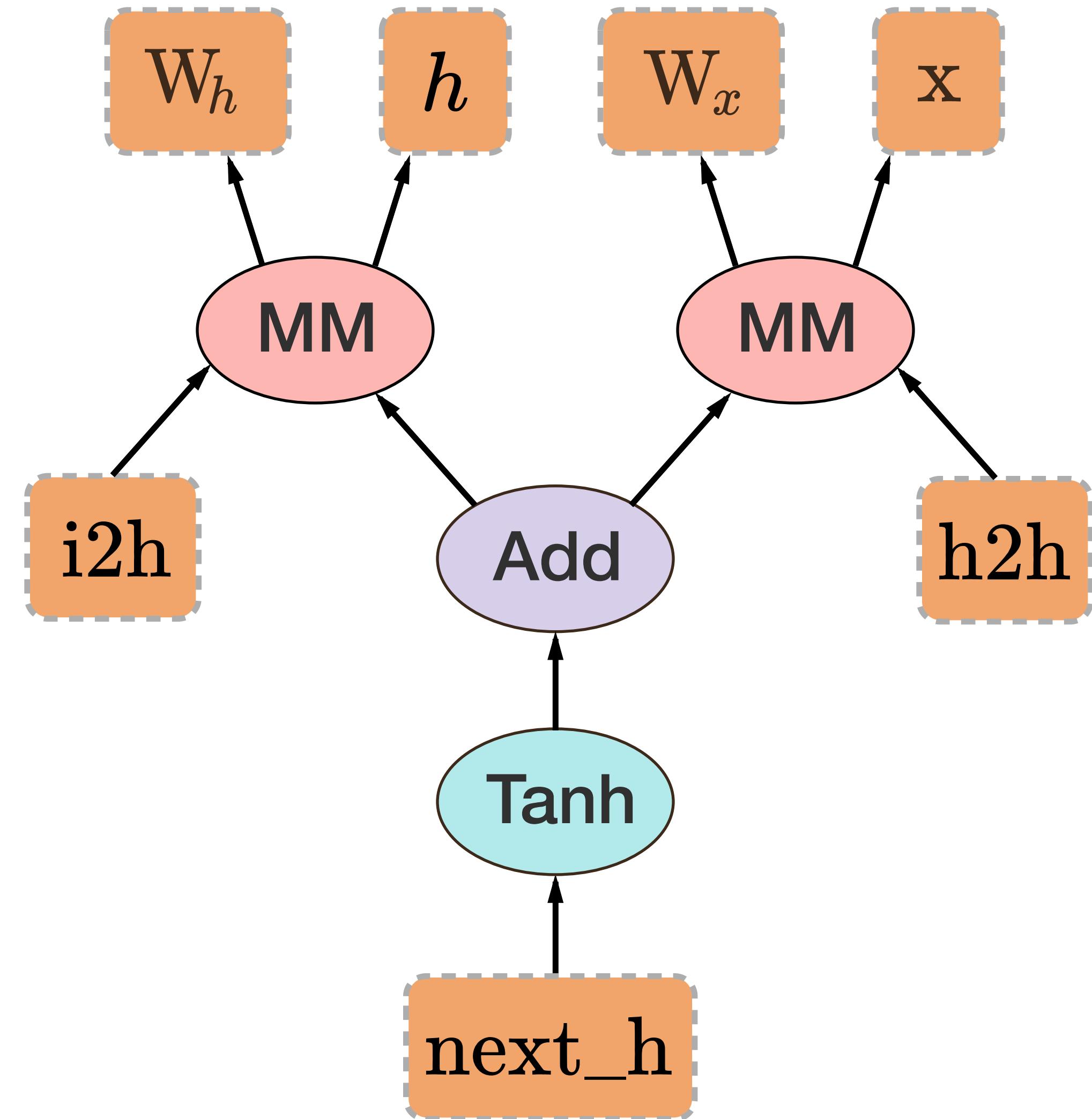
PyTorch Autograd

```
from torch.autograd import Variable  
  
x = Variable(torch.randn(1, 10))  
prev_h = Variable(torch.randn(1, 20))  
W_h = Variable(torch.randn(20, 20))  
W_x = Variable(torch.randn(20, 10))  
  
i2h = torch.mm(W_x, x.t())  
h2h = torch.mm(W_h, prev_h.t())  
next_h = i2h + h2h
```



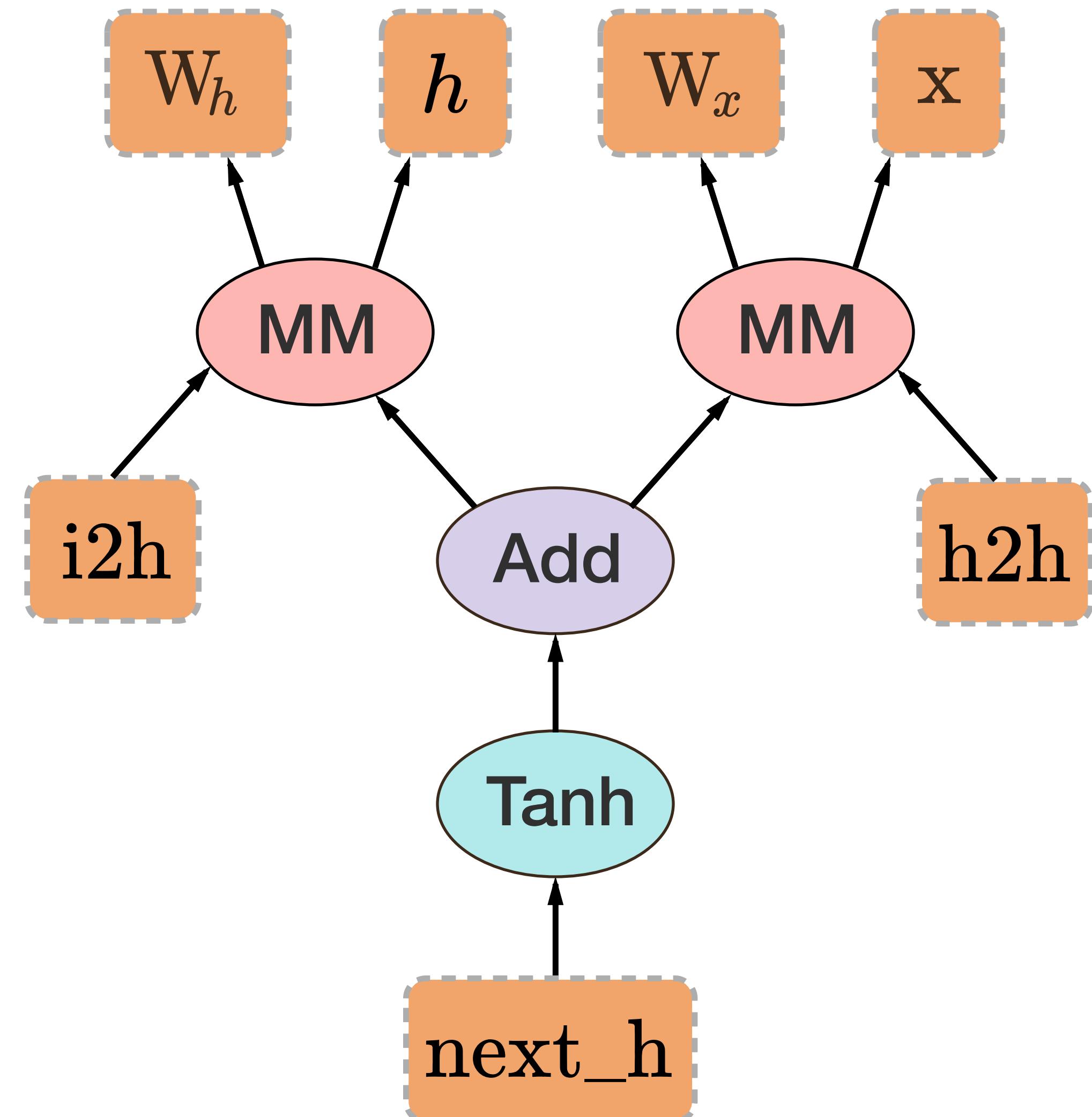
PyTorch Autograd

```
from torch.autograd import Variable  
  
x = Variable(torch.randn(1, 10))  
prev_h = Variable(torch.randn(1, 20))  
W_h = Variable(torch.randn(20, 20))  
W_x = Variable(torch.randn(20, 10))  
  
i2h = torch.mm(W_x, x.t())  
h2h = torch.mm(W_h, prev_h.t())  
next_h = i2h + h2h  
next_h = next_h.tanh()
```



PyTorch Autograd

```
from torch.autograd import Variable  
  
x = Variable(torch.randn(1, 10))  
prev_h = Variable(torch.randn(1, 20))  
W_h = Variable(torch.randn(20, 20))  
W_x = Variable(torch.randn(20, 10))  
  
i2h = torch.mm(W_x, x.t())  
h2h = torch.mm(W_h, prev_h.t())  
next_h = i2h + h2h  
next_h = next_h.tanh()  
  
next_h.backward(torch.ones(1, 20))
```



side by side: TensorFlow and PyTorch

```
1 import tensorflow as tf
2 import numpy as np
3
4 trX = np.linspace(-1, 1, 101)
5 trY = 2 * trX + np.random.randn(*trX.shape) * 0.33
6
7 X = tf.placeholder("float")
8 Y = tf.placeholder("float")
9
10 def model(X, w):
11     return tf.multiply(X, w)
12
13 w = tf.Variable(0.0, name="weights")
14 y_model = model(X, w)
15
16 cost = tf.square(Y - y_model)
17
18 train_op = tf.train.GradientDescentOptimizer(0.01).minimize(cost)
19
20 with tf.Session() as sess:
21     tf.global_variables_initializer().run()
22
23     for i in range(100):
24         for (x, y) in zip(trX, trY):
25             sess.run(train_op, feed_dict={X: x, Y: y})
26
27 print(sess.run(w))
```

```
1 import torch
2 from torch.autograd import Variable
3
4 trX = torch.linspace(-1, 1, 101)
5 trY = 2 * trX + torch.randn(*trX.size()) * 0.33
6
7 w = Variable(trX.new([0.0]), requires_grad=True)
8
9 for i in range(100):
10     for (x, y) in zip(trX, trY):
11         X = Variable(x)
12         Y = Variable(y)
13
14         y_model = X * w.expand_as(X)
15         cost = (Y - Y_model) ** 2
16         cost.backward(torch.ones(*cost.size()))
17
18         w.data = w.data + 0.01 * w.grad.data
19
20 print(w)
```

motivation behind design of torch.autograd

why is it more convenient?



Today's AI



Active Research &
Future AI



Tools for AI
keeping up with change

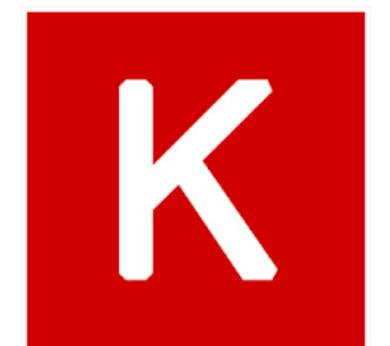
PYTORCH



theano



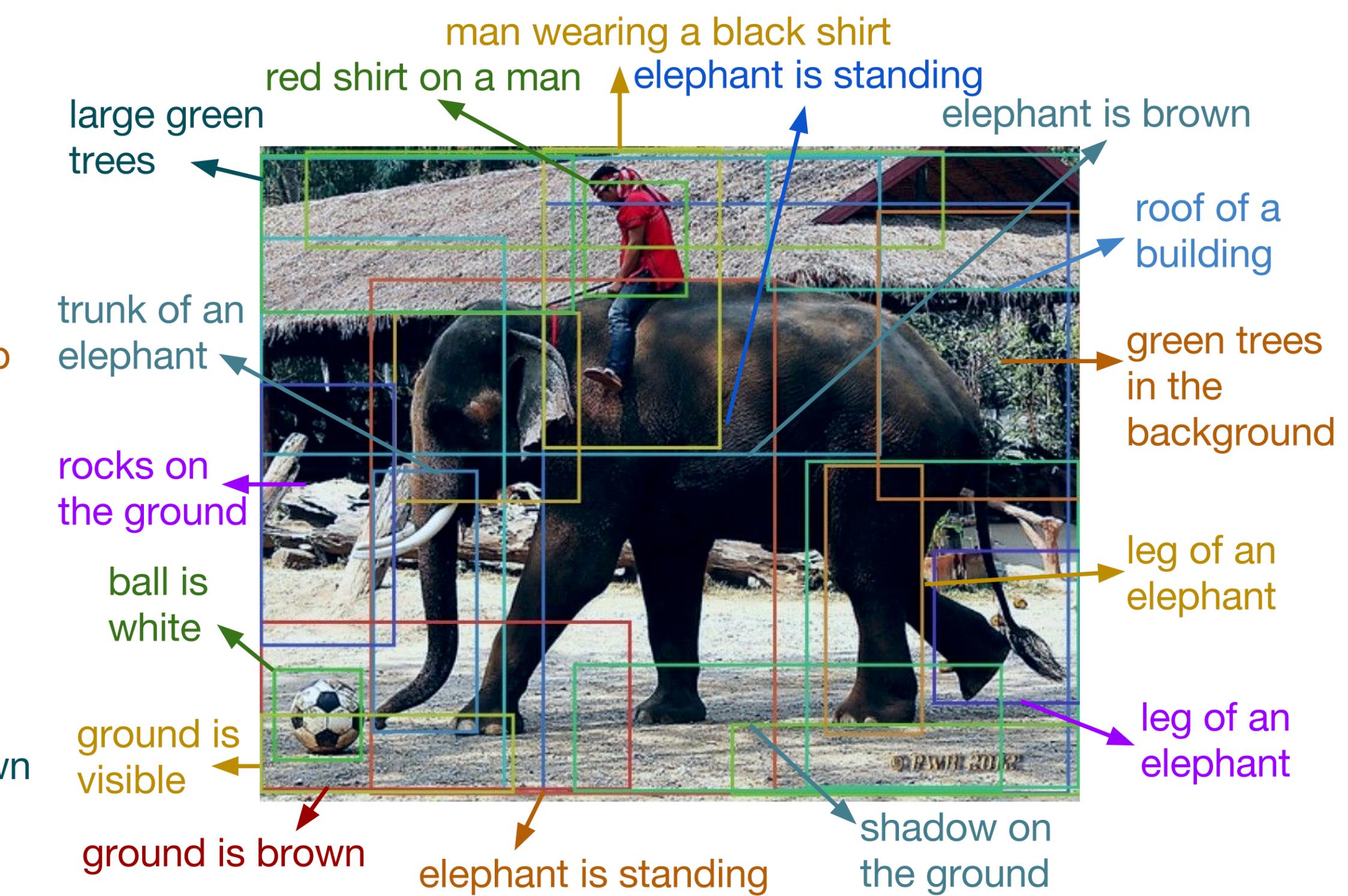
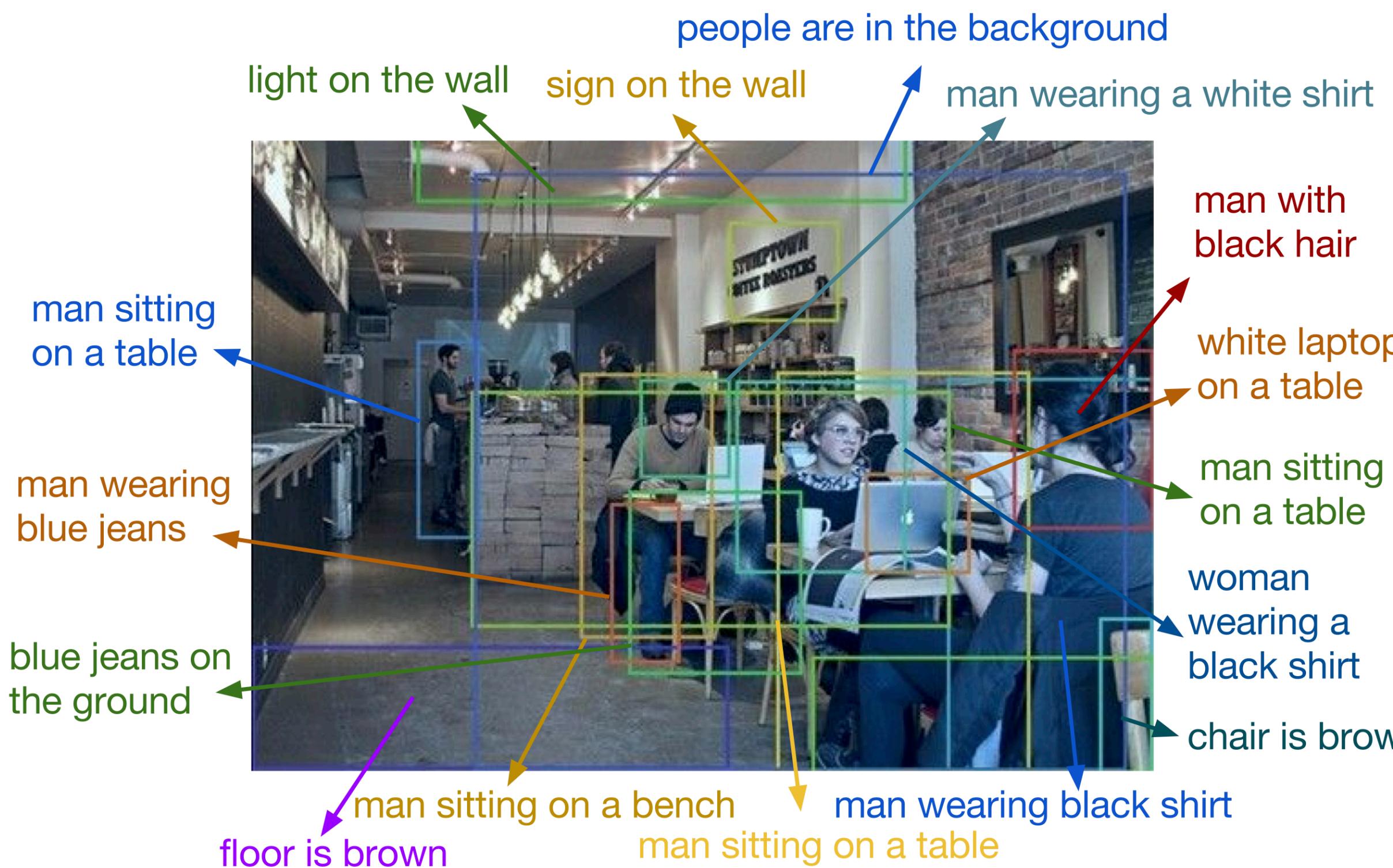
Caffe



Today's AI

DenseCap by Justin Johnson & group

<https://github.com/jcjohnson/densecap>



Today's AI

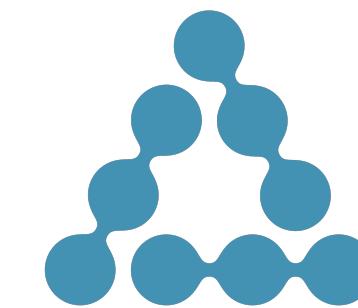
Future AI

Tools for AI

Today's AI



DeepMask by Pedro Pinhero & group



Today's AI



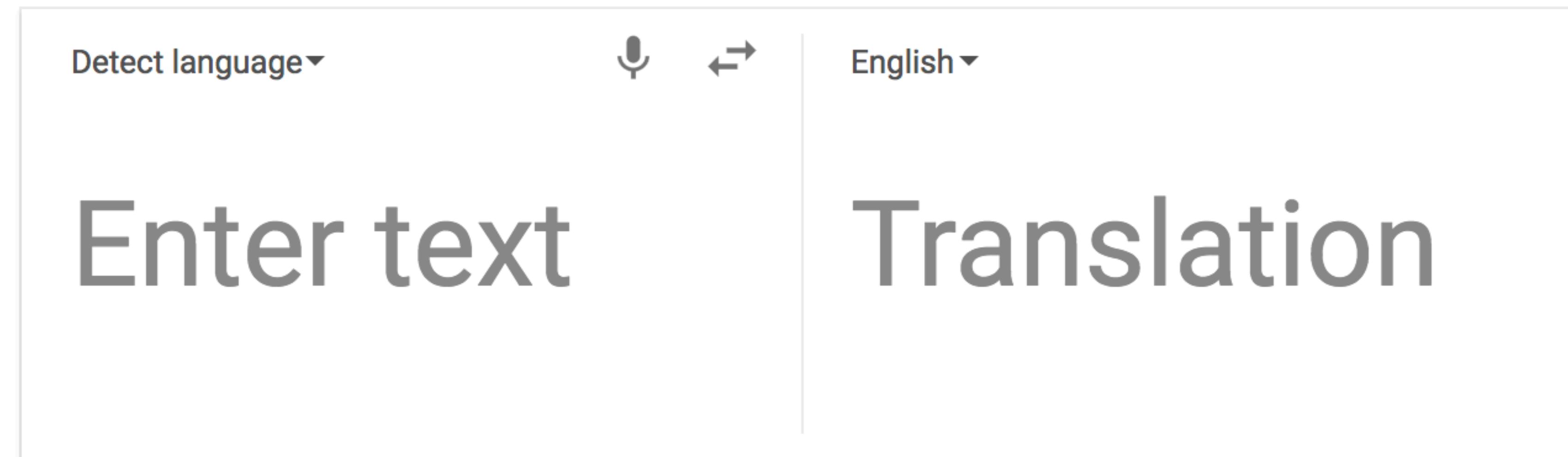
Future AI



Tools for AI

Today's AI

Machine Translation



Google
Translate



BING TRANSLATOR

Today's AI



Future AI



Tools for AI

Today's AI

Text Classification (sentiment analysis etc.)

Text Embeddings

Graph embeddings

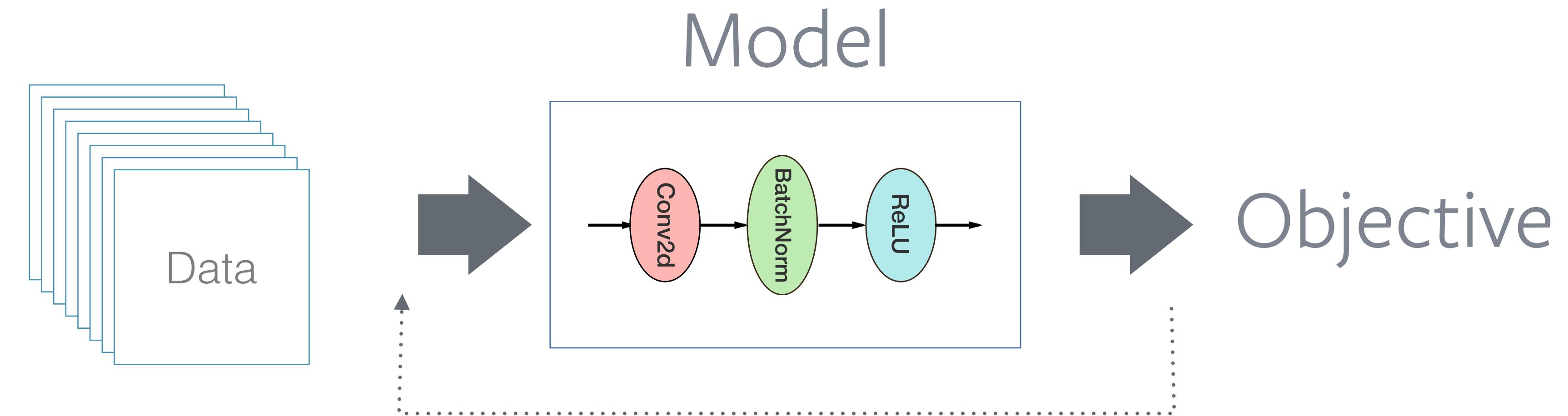
Machine Translation

Ads ranking



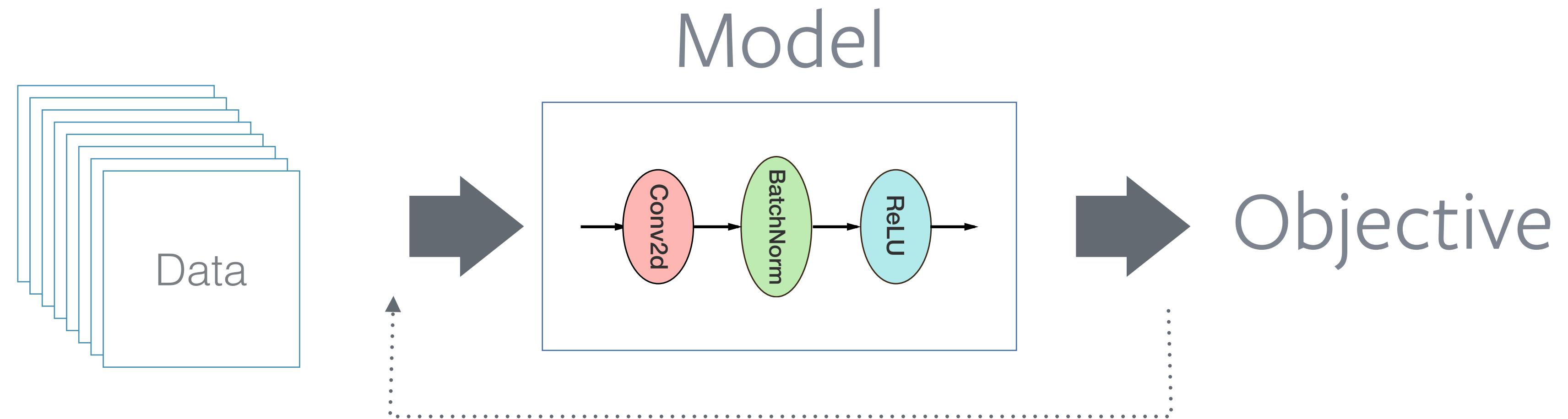
Today's AI

Train Model

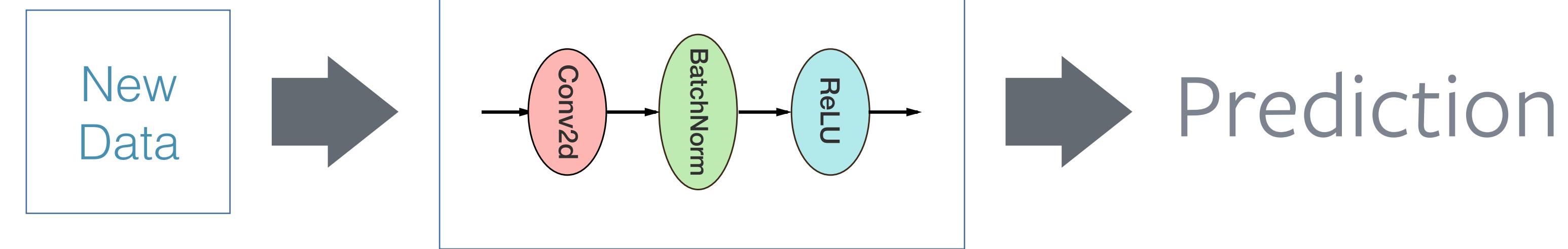


Today's AI

Train Model



Deploy & Use



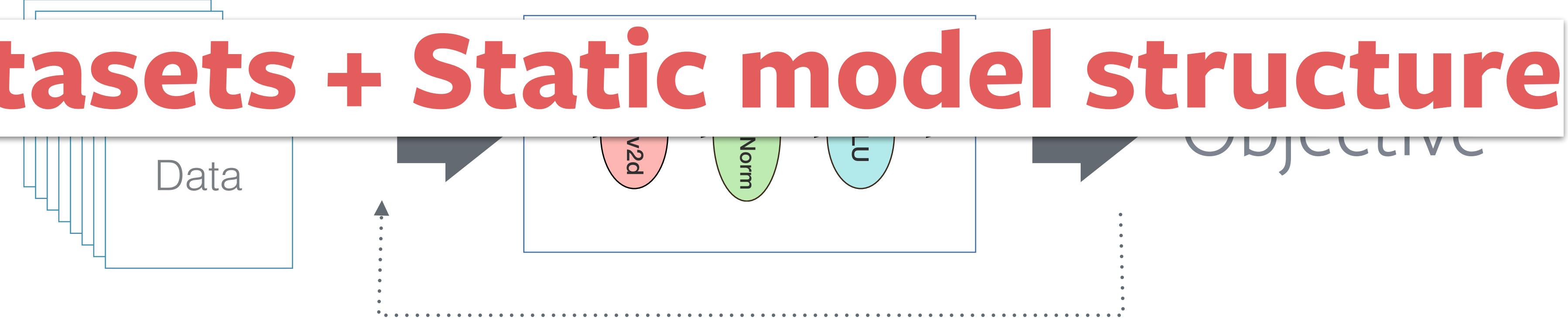
Today's AI

Future AI

Tools for AI

Today's AI

- Static datasets + Static model structure



Deploy & Use

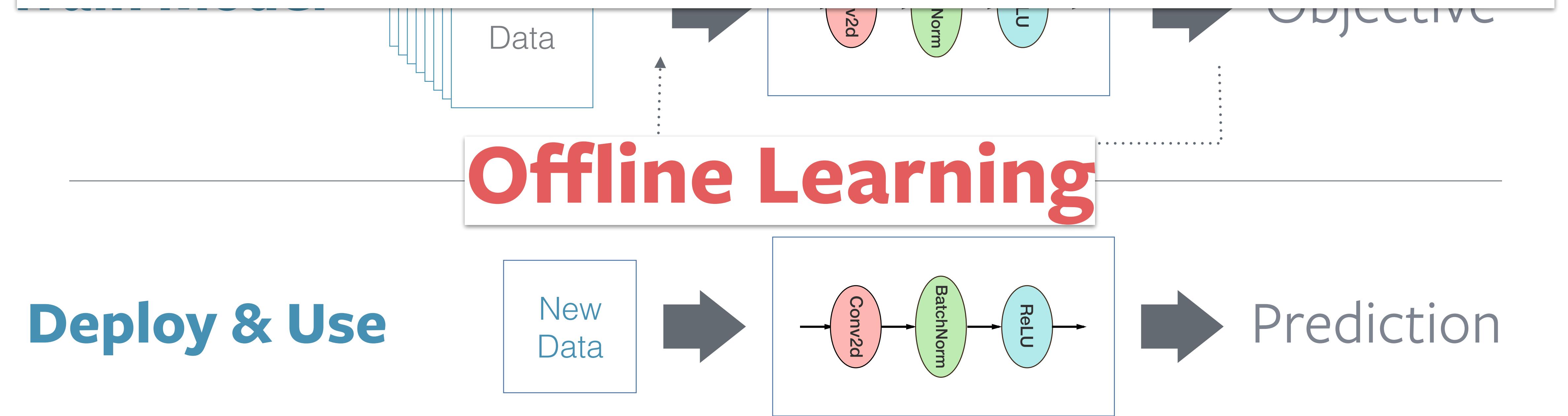


Prediction



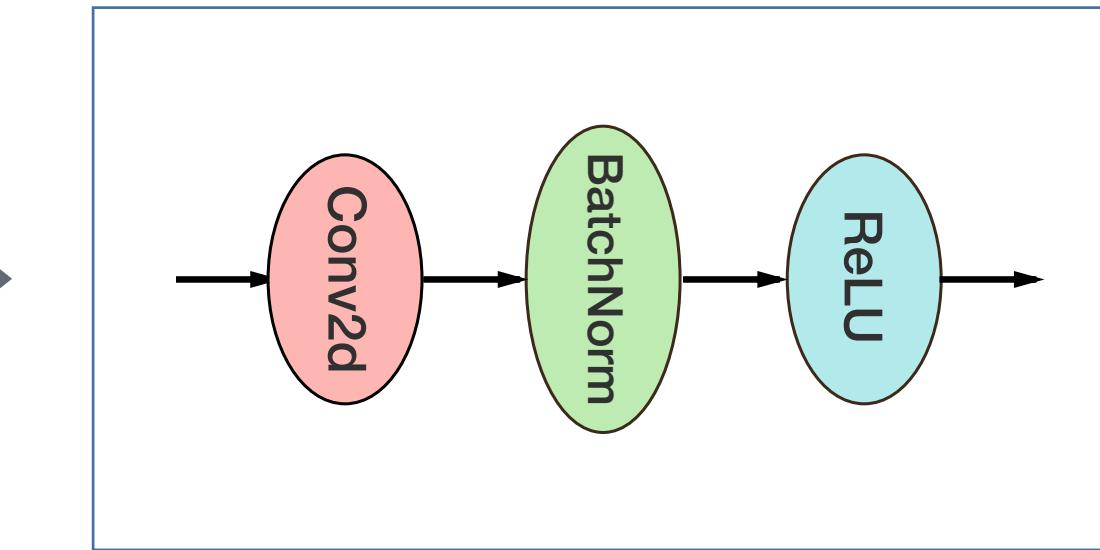
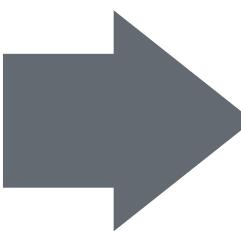
Today's AI

- Static datasets + Static model structure



Deploy & Use

New
Data



Prediction

Today's AI



Future AI



Tools for AI

Current AI Research / Future AI

Self-driving Cars



Today's AI



Future AI



Tools for AI

Current AI Research / Future AI

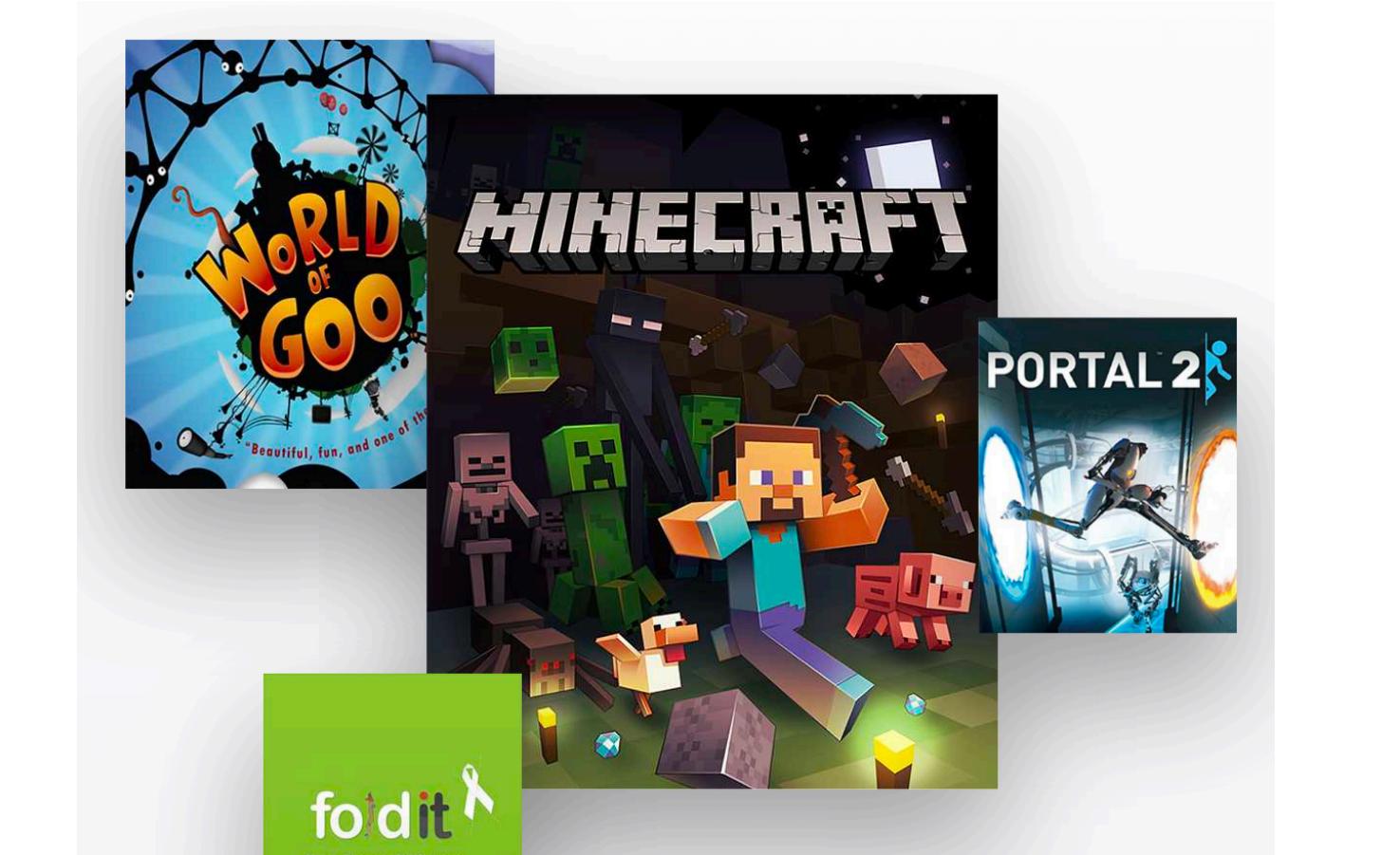
Agents trained in many environments



Cars



Video games



UNIVERSE

Measurement and training for
artificial intelligence.

Internet

Today's AI



Tools for AI

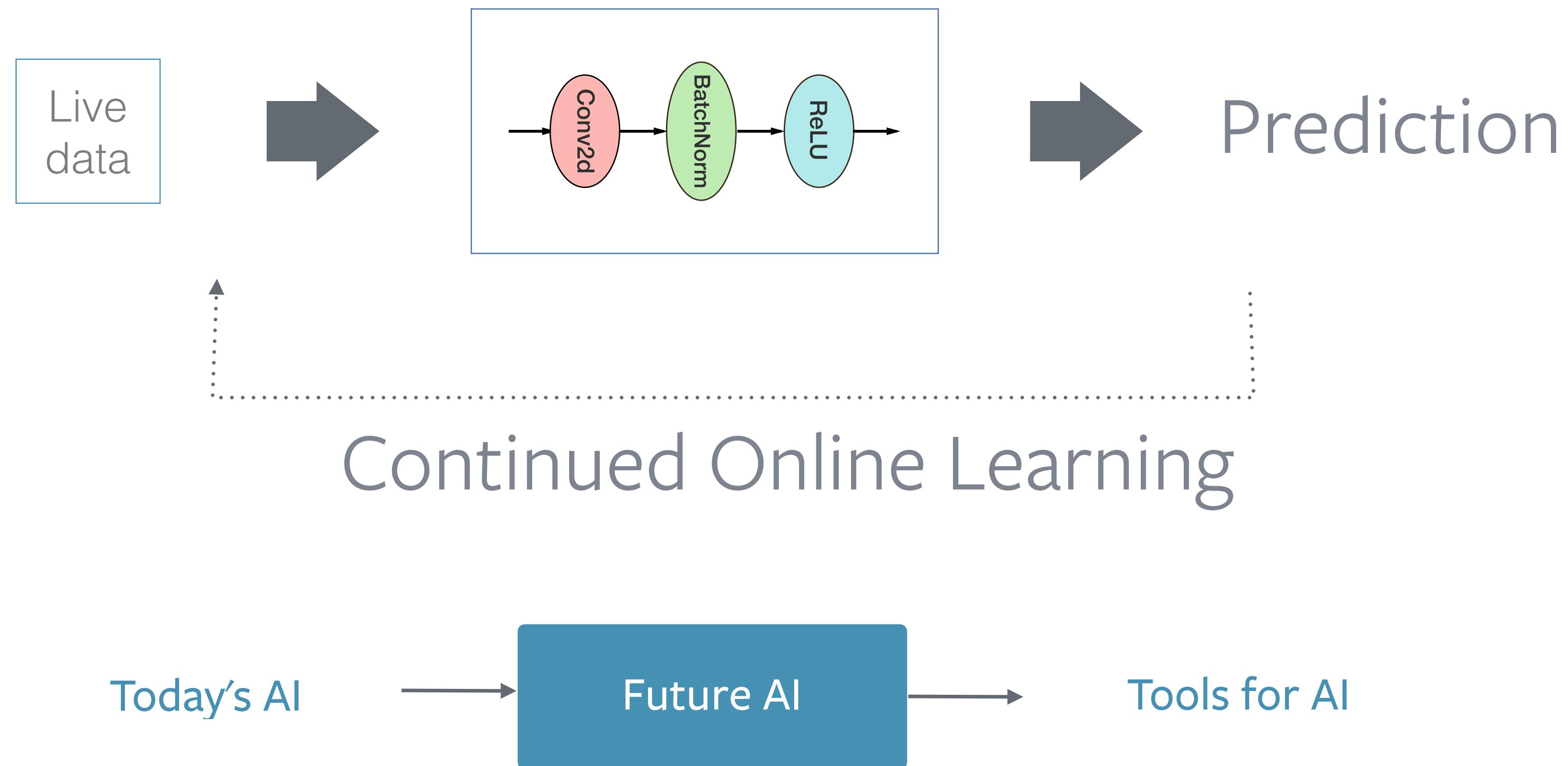
Current AI Research / Future AI

Dynamic Neural Networks

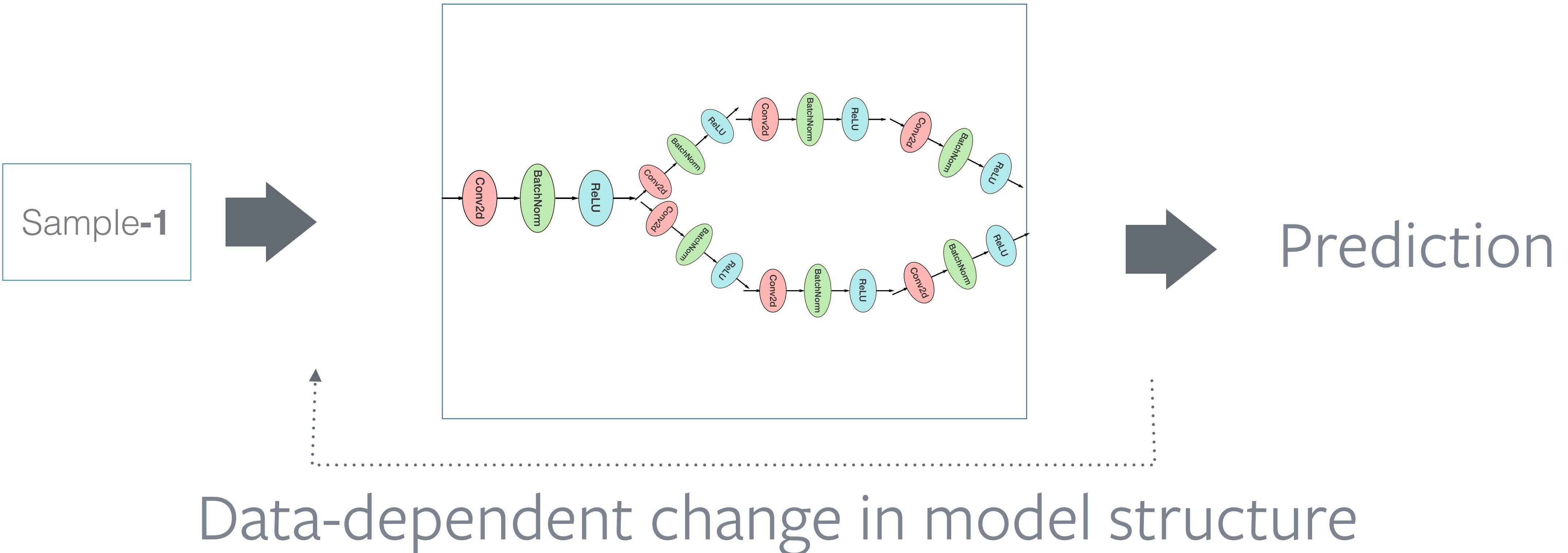
self-adding new memory or layers
changing evaluation path based on inputs



Current AI Research / Future AI



Current AI Research / Future AI

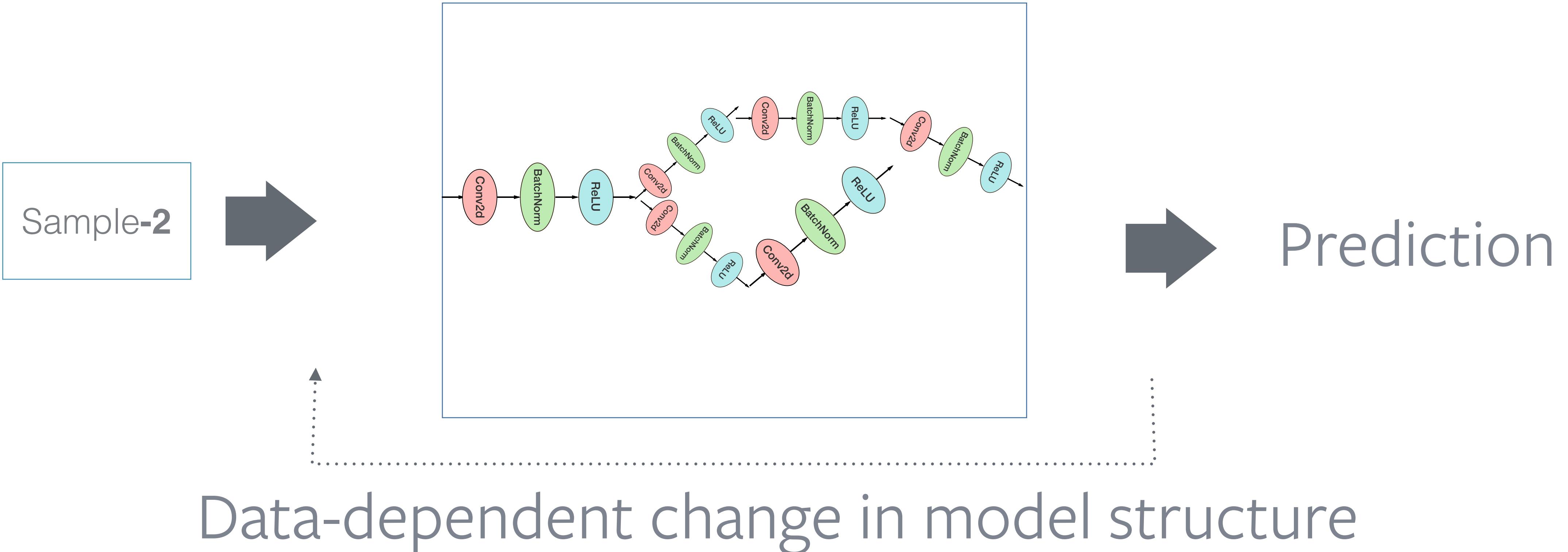


Today's AI

Future AI

Tools for AI

Current AI Research / Future AI

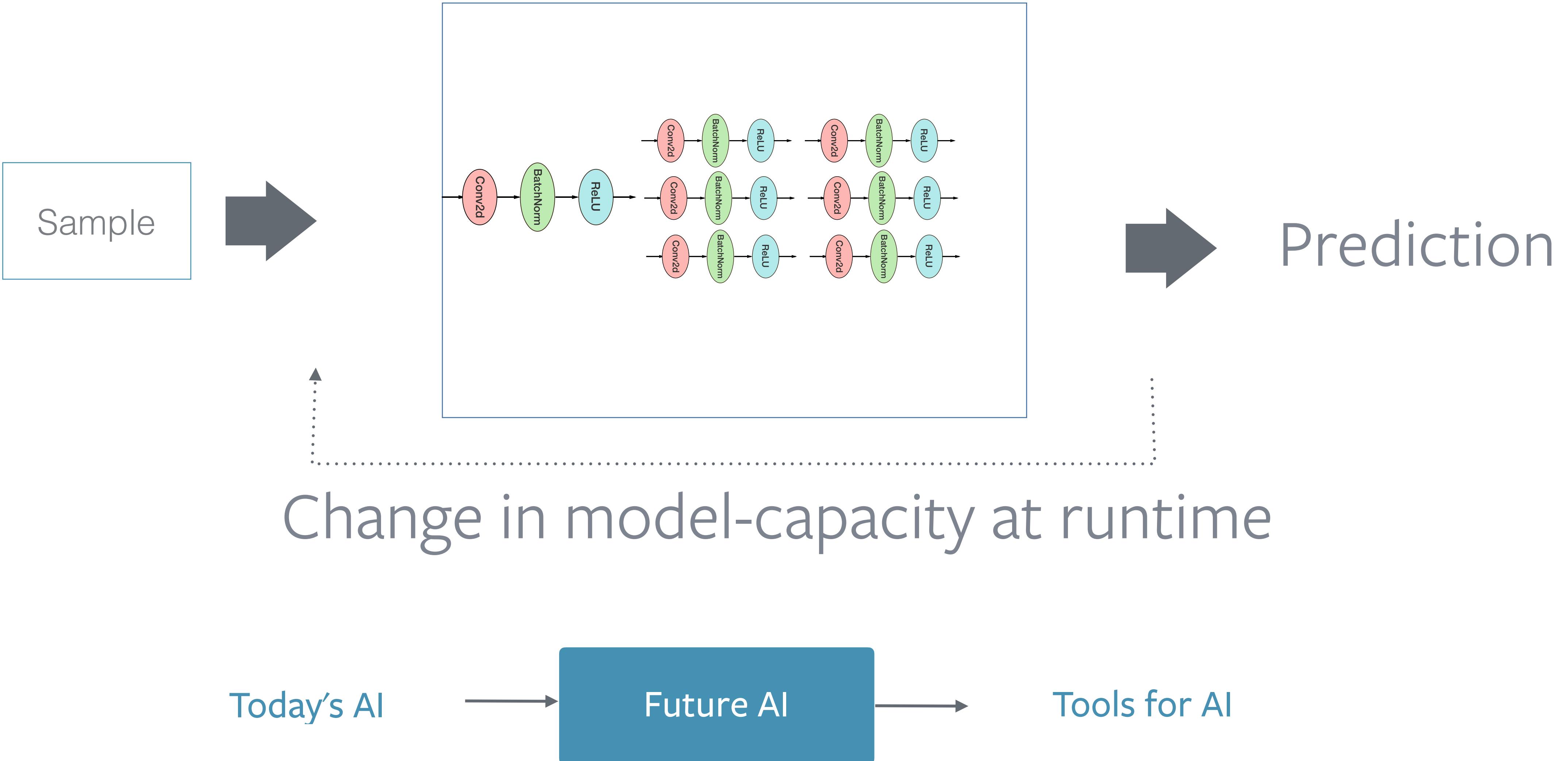


Today's AI

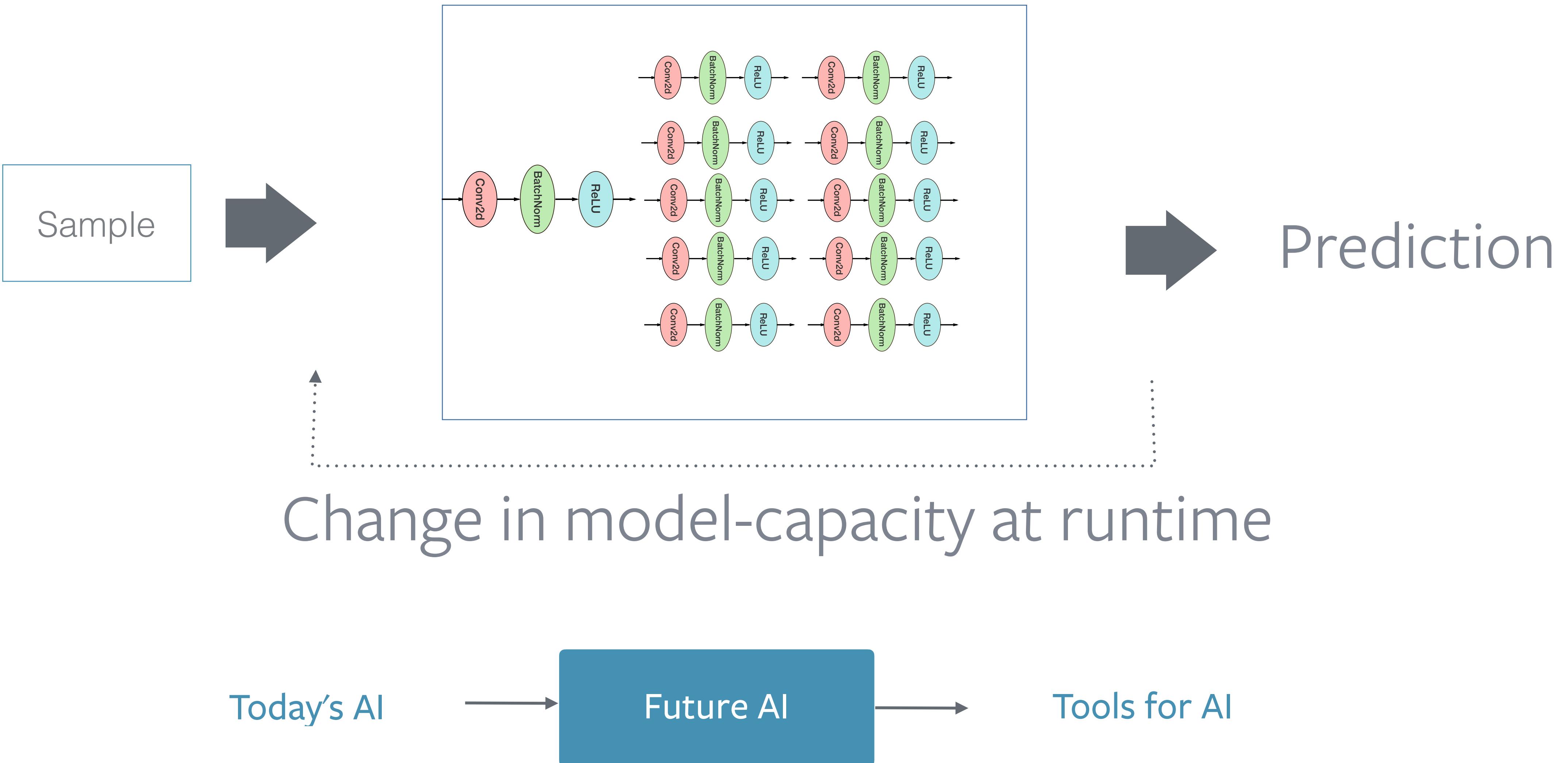
Future AI

Tools for AI

Current AI Research / Future AI



Current AI Research / Future AI



The need for a dynamic auto-diff

- Interop with many dynamic environments
 - Connecting to car sensors should be as easy as training on a dataset
 - Connect to environments such as OpenAI Universe
- Dynamic Neural Networks
 - Change behavior and structure of neural network at runtime
- Minimal Abstractions
 - more complex AI systems means harder to debug without a simple API



Tools for AI research and deployment

Many machine learning tools and deep learning frameworks



theano

Caffe



Today's AI



Future AI



Tools for AI

Tools for AI research and deployment

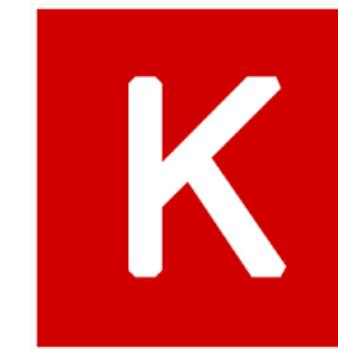
Static graph frameworks



TensorFlow



theano



Caffe



Today's AI



Future AI



Tools for AI

Dynamic graph frameworks



Dynamic graph Frameworks

- Model is constructed on the fly at runtime
- Change behavior, structure of model
- Imperative style of programming





- Fast ndarray library with GPU support
- Build the latest neural networks and do gradient based learning using the autograd and neural network package
- large community of people, many companies using and contributing



<http://pytorch.org>

Released Jan 18th

50,000+ downloads

250+ community repos

5400+ user posts

485k+ forum views

facebook



ParisTech
INSTITUT DES SCIENCES ET TECHNOLOGIE
PARIS INSTITUTE OF TECHNOLOGY

Carnegie
Mellon
University

NVIDIA.

salesforce

Digital
Reasoning

Stanford
University

UNIVERSITY OF
OXFORD

NYU

1794
ENS
ÉCOLE NORMALE
SUPÉRIEURE

You?

