



A framework for fast, dynamic deep learning and scientific computing

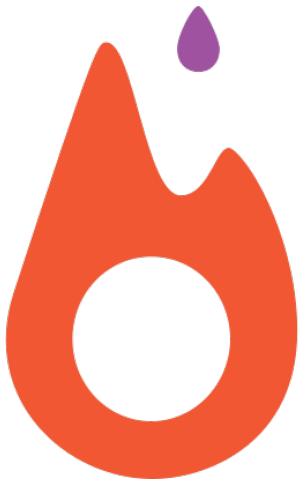
Adam Paszke, Sam Gross, **Soumith Chintala**, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Alban Desmaison, Andreas Kopf, Edward Yang, Zach Devito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy & Team



What is PyTorch?



What is PyTorch?



automatic differentiation
engine

**Ndarray library
with GPU support**

gradient based
optimization package

Utilities
(data loading, etc.)

Deep Learning

Numpy-alternative

Reinforcement Learning



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.

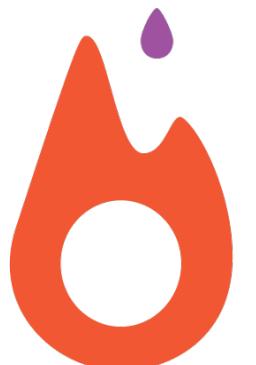
```
from __future__ import print_function  
import torch
```

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:

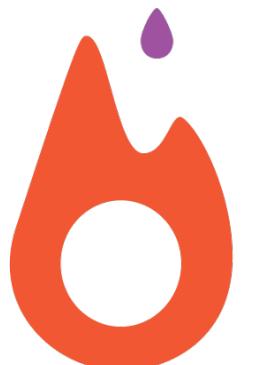
```
0.2598  0.7231  0.8534
0.3928  0.1244  0.5110
0.5476  0.2700  0.5856
0.7288  0.9455  0.8749
0.6663  0.8230  0.2713
[torch.FloatTensor of size 5x3]
```

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)
```

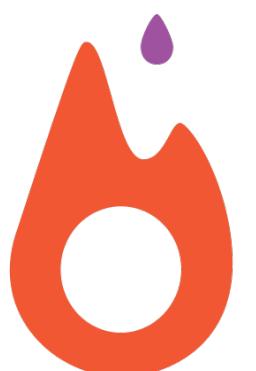
Out:

```
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:

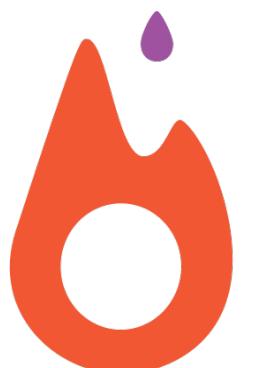
```
1  
1  
1  
1  
1  
[torch.FloatTensor of size 5]
```

**Zero memory-copy
very efficient**

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

Out:

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



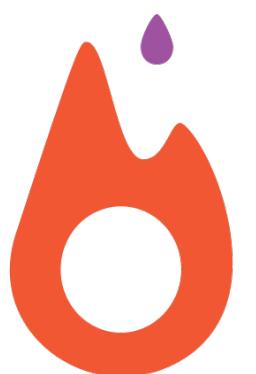
NumPy bridge

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
```



Deep Learning 101

a quick introduction



Types of typical operators

Convolution

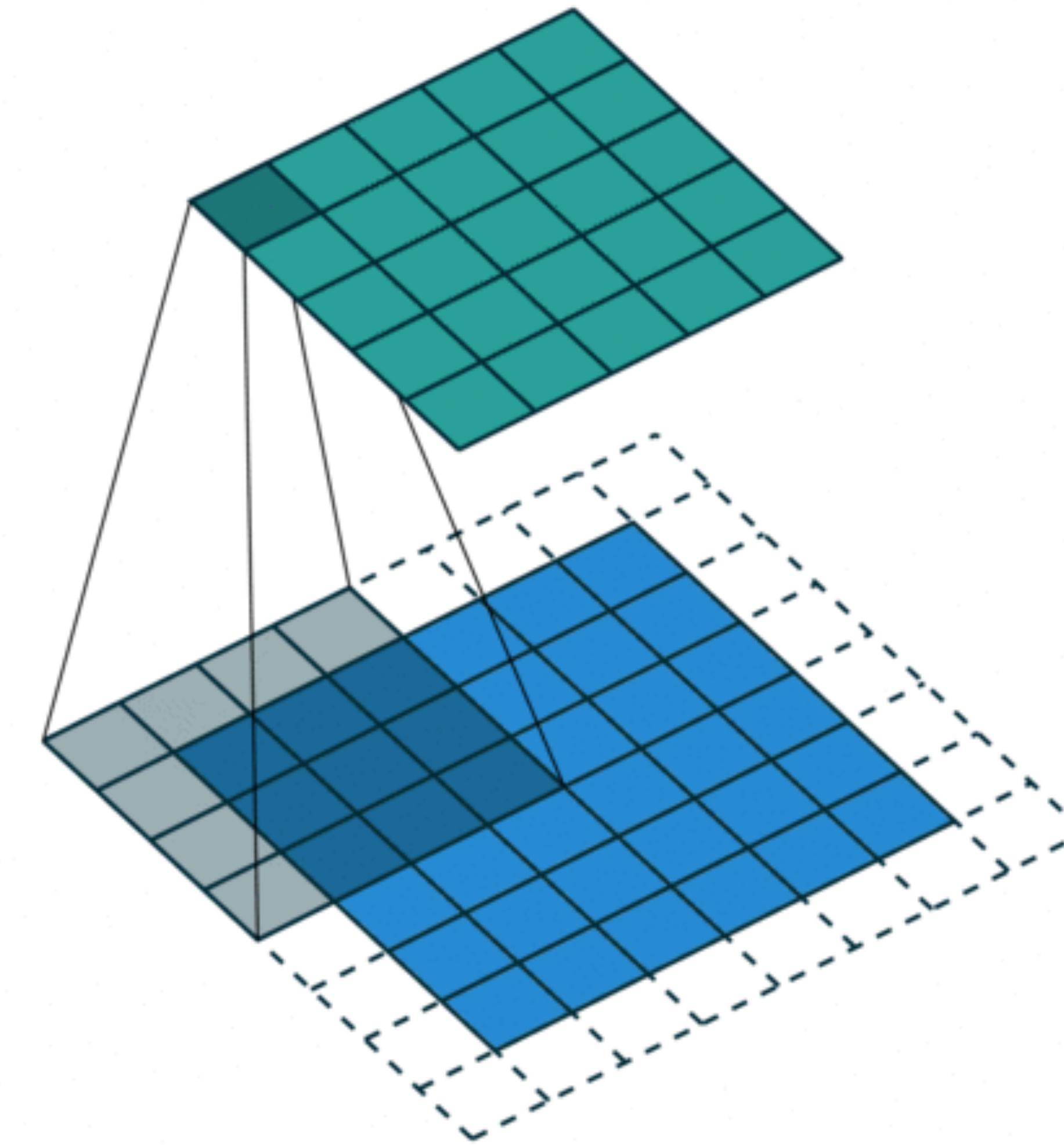
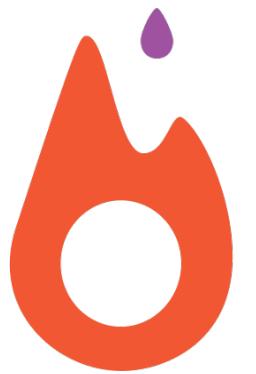


Figure by Vincent Dumolin: https://github.com/vdumoulin/conv_arithmetic



Types of typical operators

Convolution

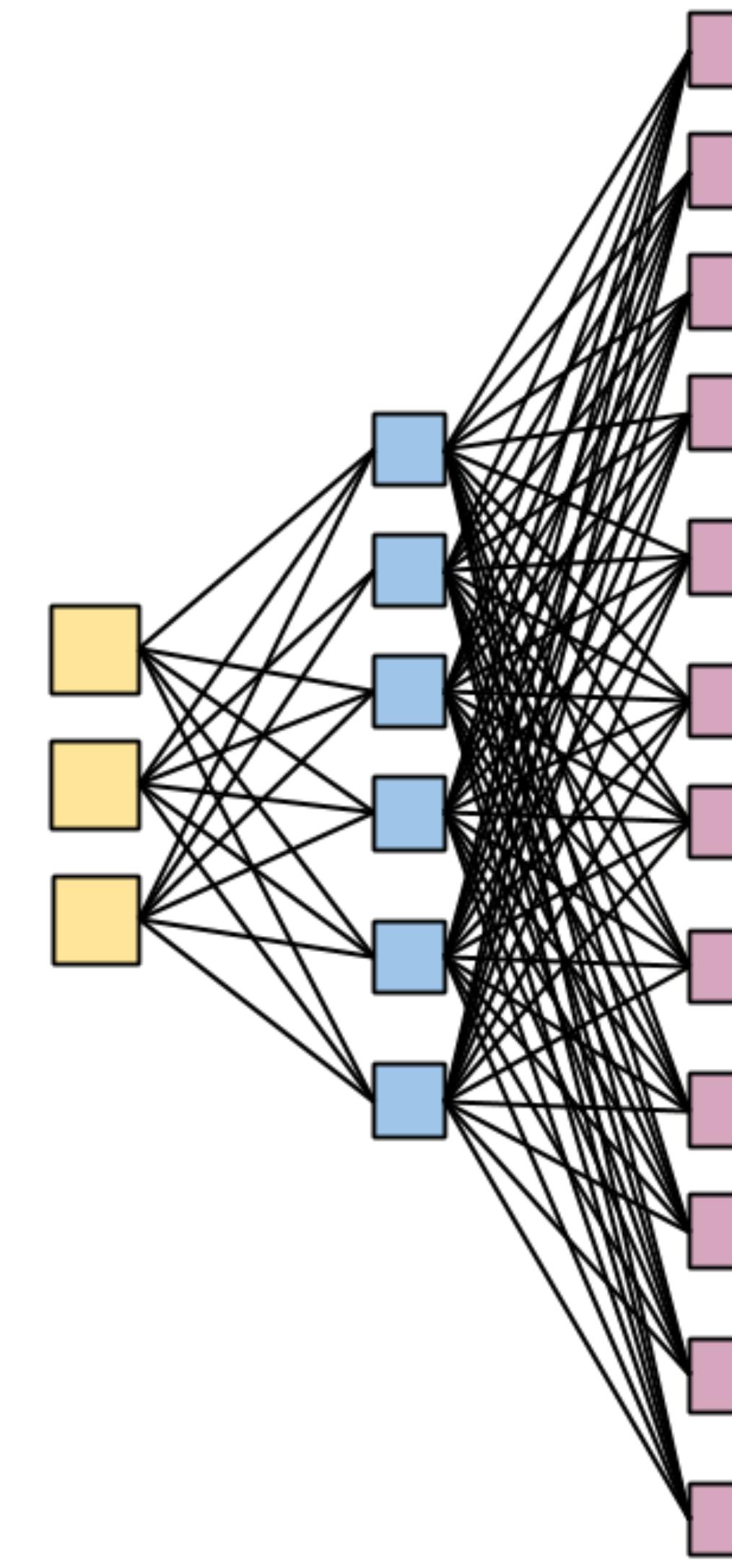
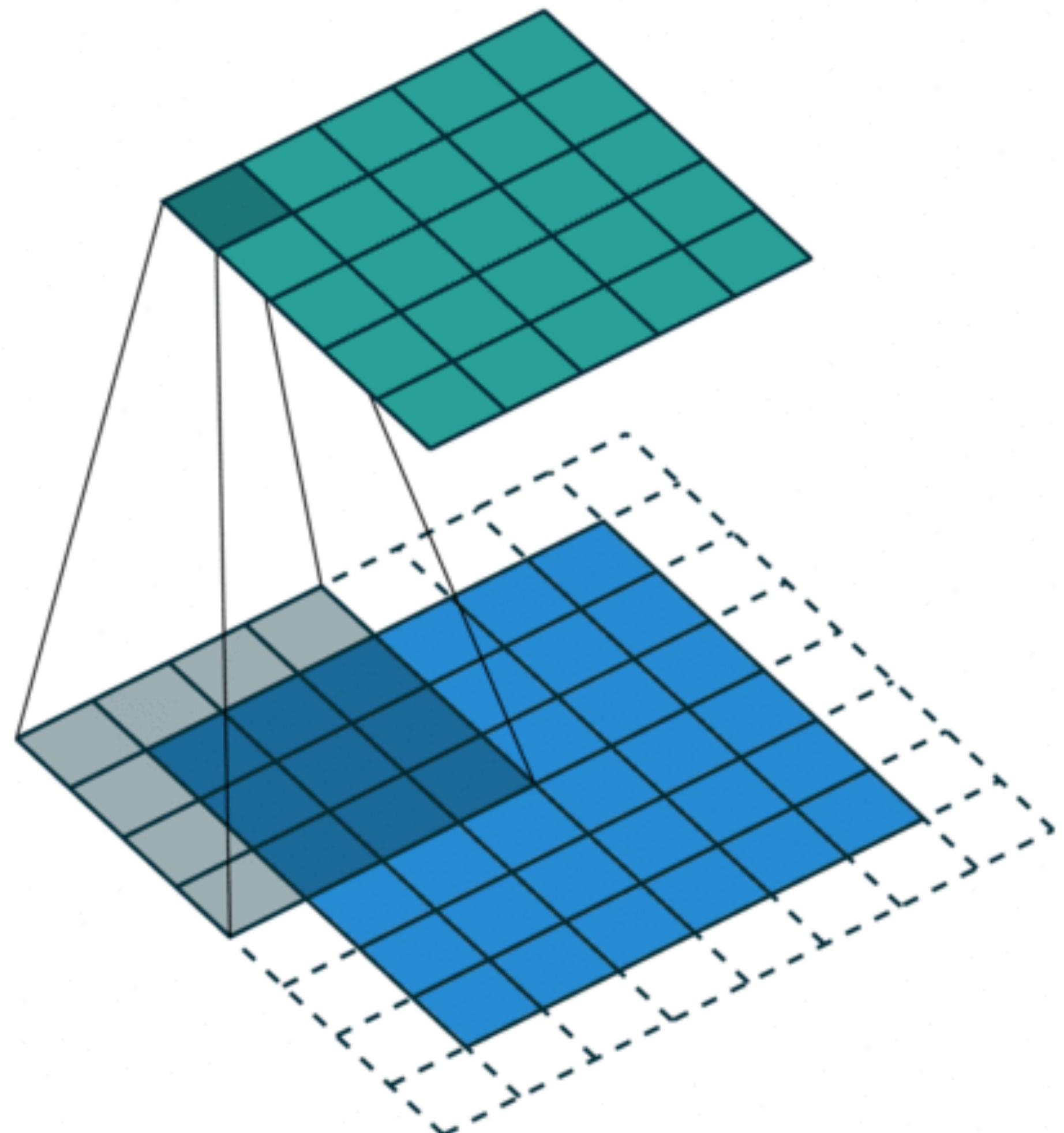
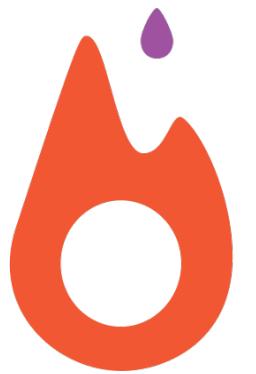


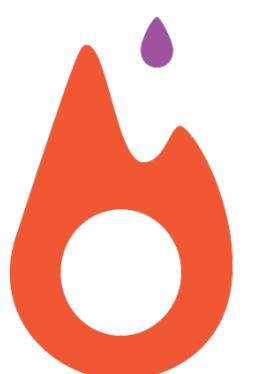
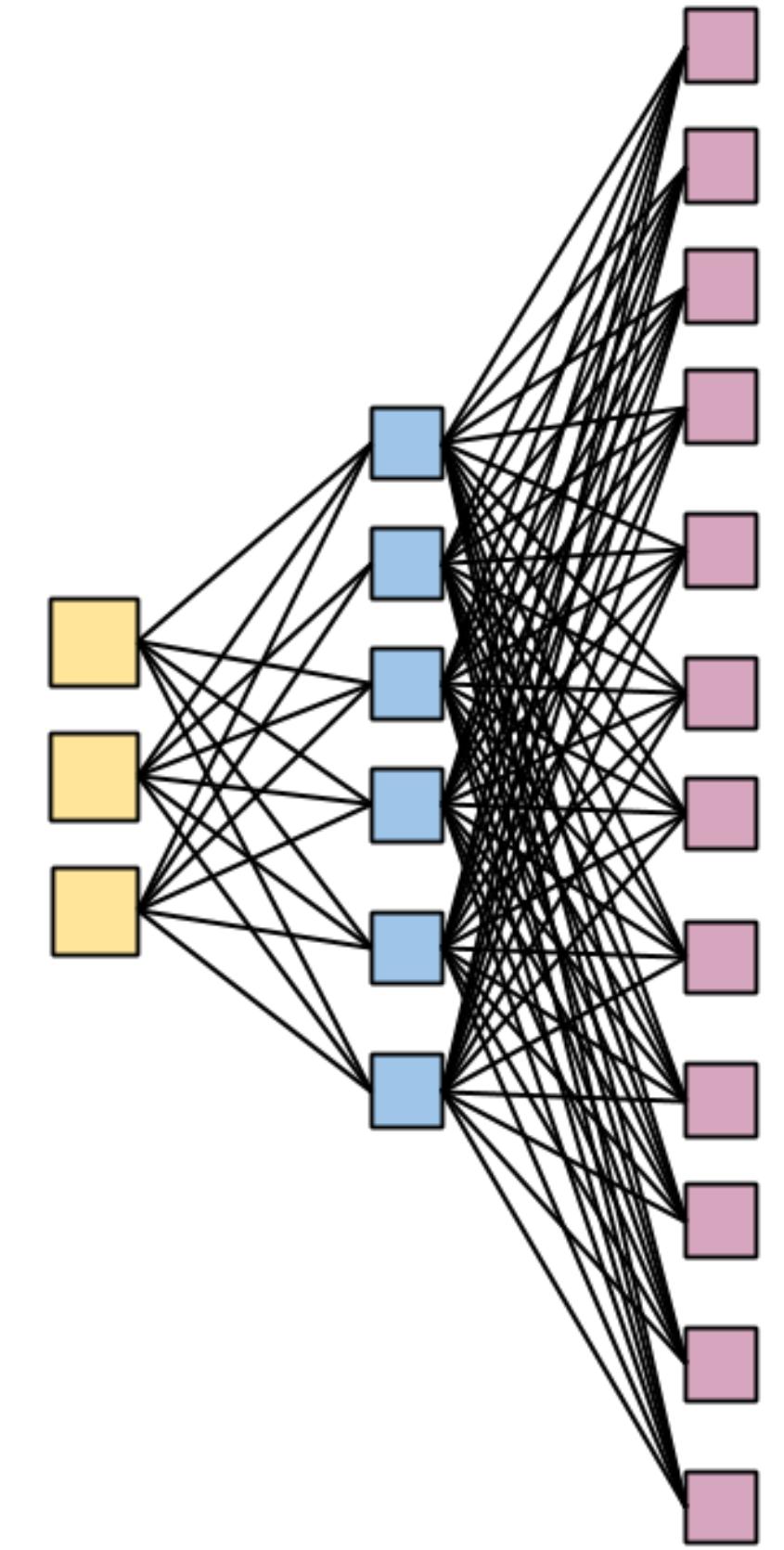
Figure by Vincent Dumolin: https://github.com/vdumoulin/conv_arithmetic



Types of typical operators

Convolution

```
for oc in output_channel:  
    for ic in input_channel:  
        for h in output_height:  
            for w in output_width:  
                for kh in kernel_height:  
                    for kw in kernel_width:  
                        output_pixel += input_pixel * kernel_value
```



Types of typical operators

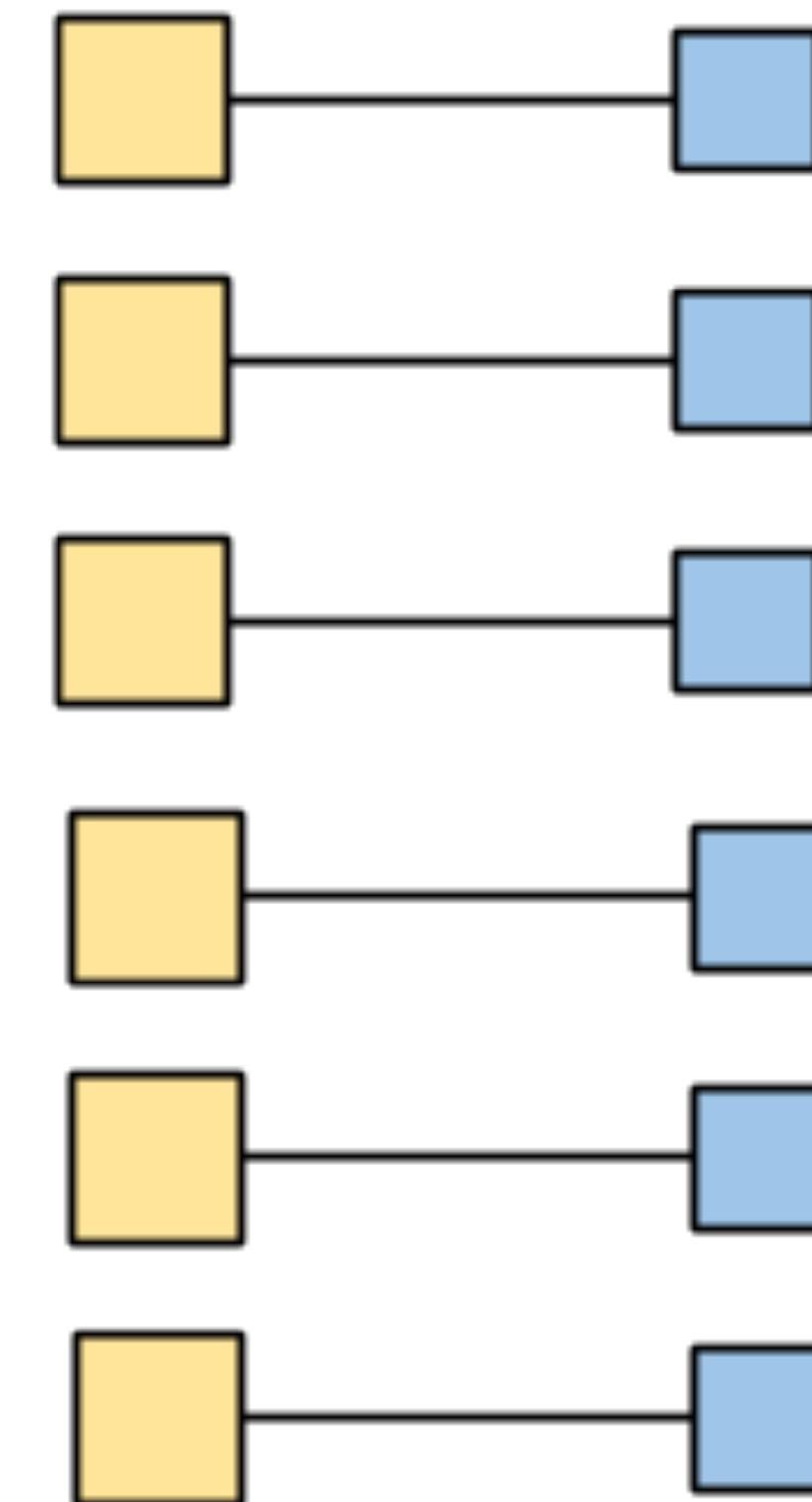
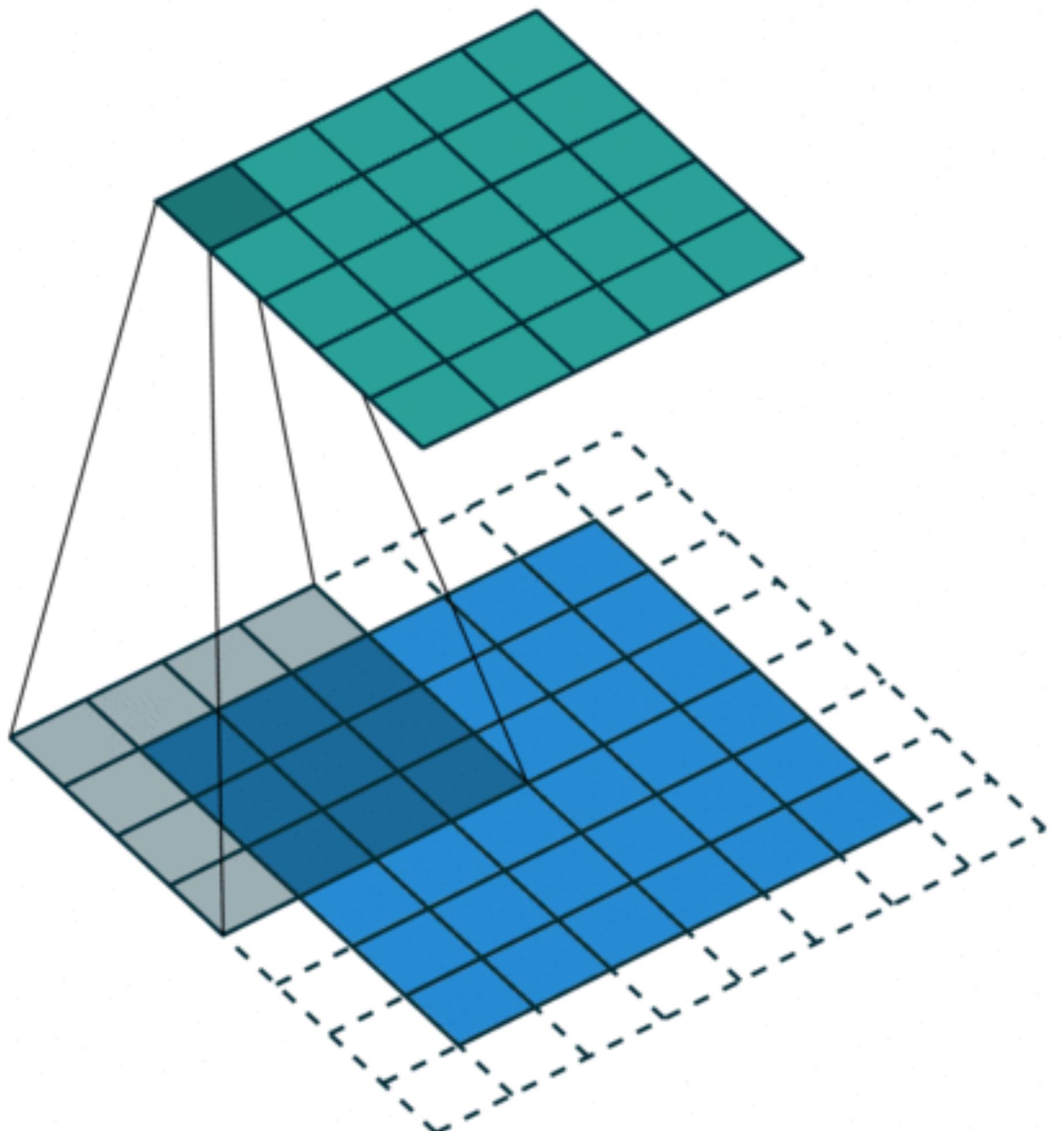


Figure by Vincent Dumolin: https://github.com/vdumoulin/conv_arithmetic



Types of typical operators

Matrix Multiply

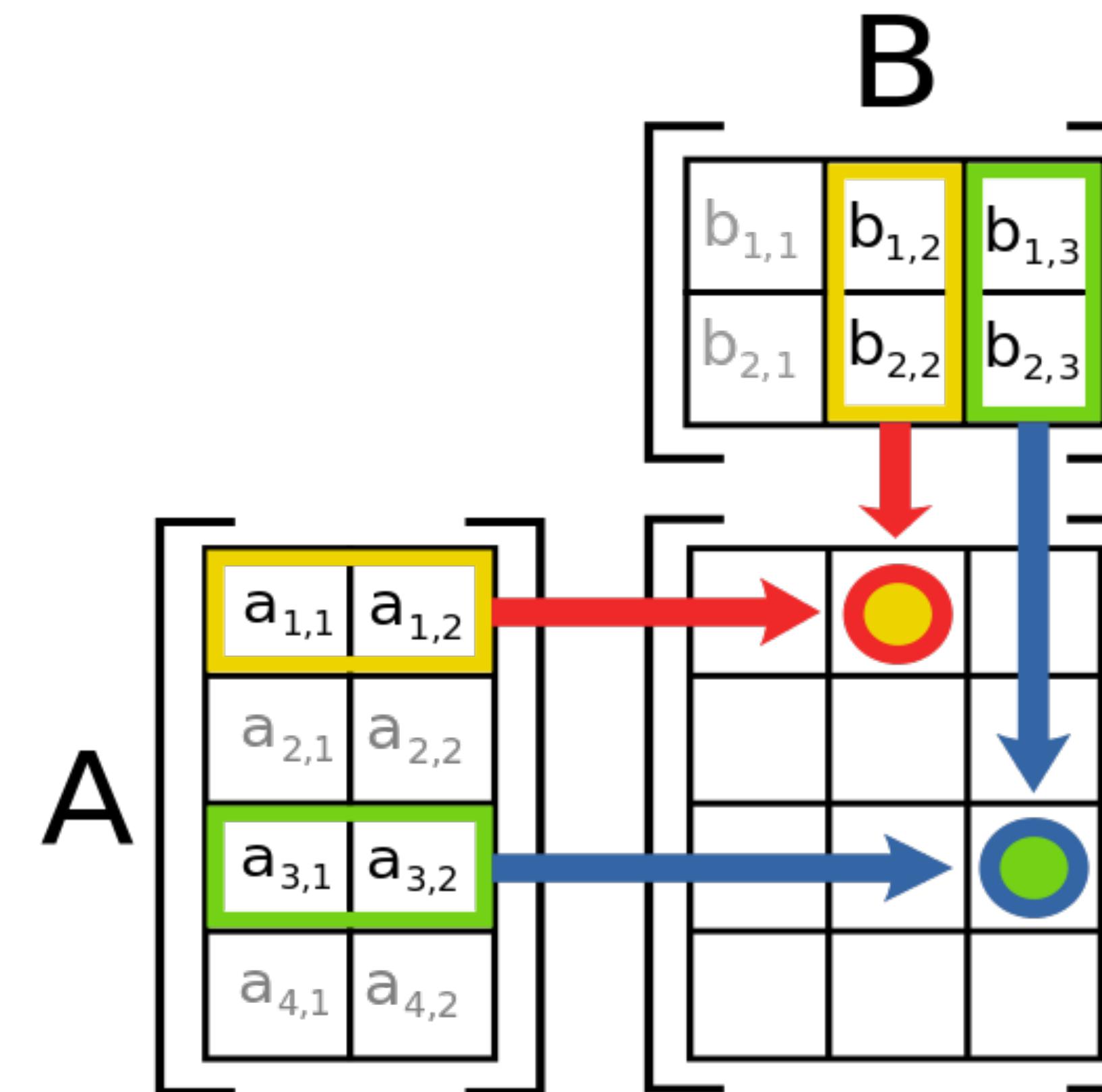
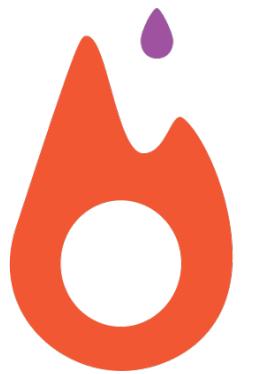


Figure by Wikipedia: https://en.wikipedia.org/wiki/Matrix_multiplication



Types of typical operators

Pointwise operations

```
for (i=0; i < data_length; i++) {  
    output[i] = input1[i] + input2[i]  
}
```



Types of typical operators

Reduction operations

```
double sum = 0.0;  
for (i=0; i < data_length; i++) {  
    sum += input[i];  
}
```



Chained Together

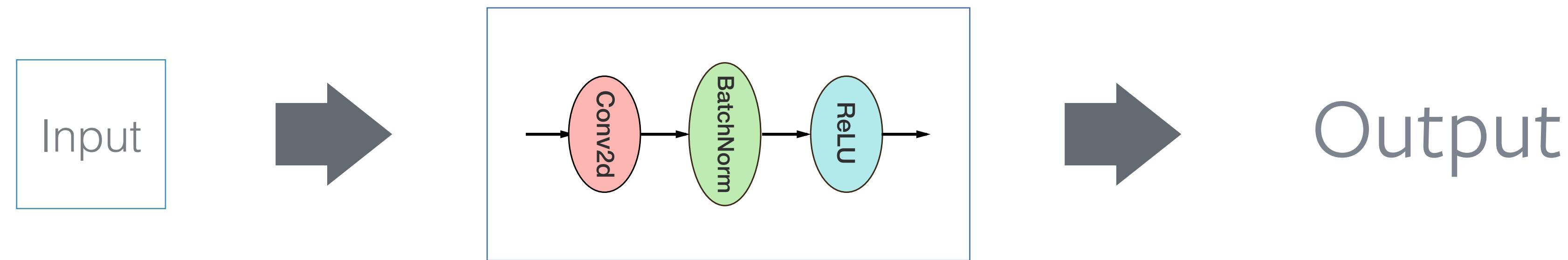
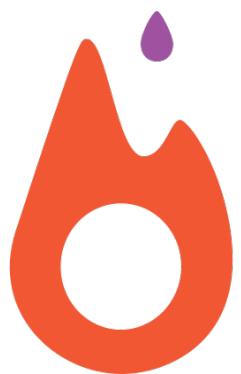


Figure by Wikipedia: https://en.wikipedia.org/wiki/Matrix_multiplication



Chained Together

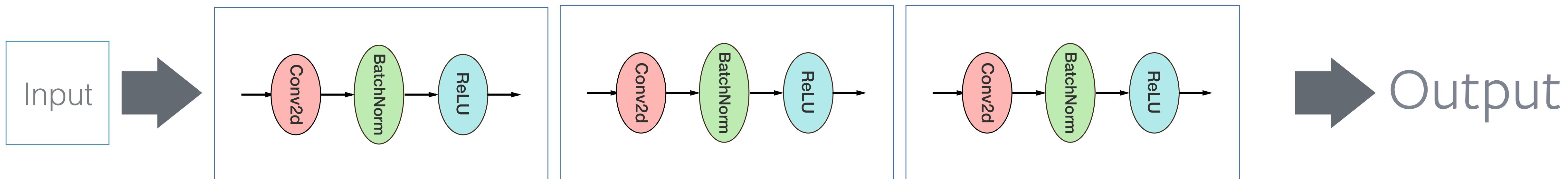


Figure by Wikipedia: https://en.wikipedia.org/wiki/Matrix_multiplication



Chained Together

"deep"



Figure by Wikipedia: https://en.wikipedia.org/wiki/Matrix_multiplication

Chained Together

"deep"

recurrent

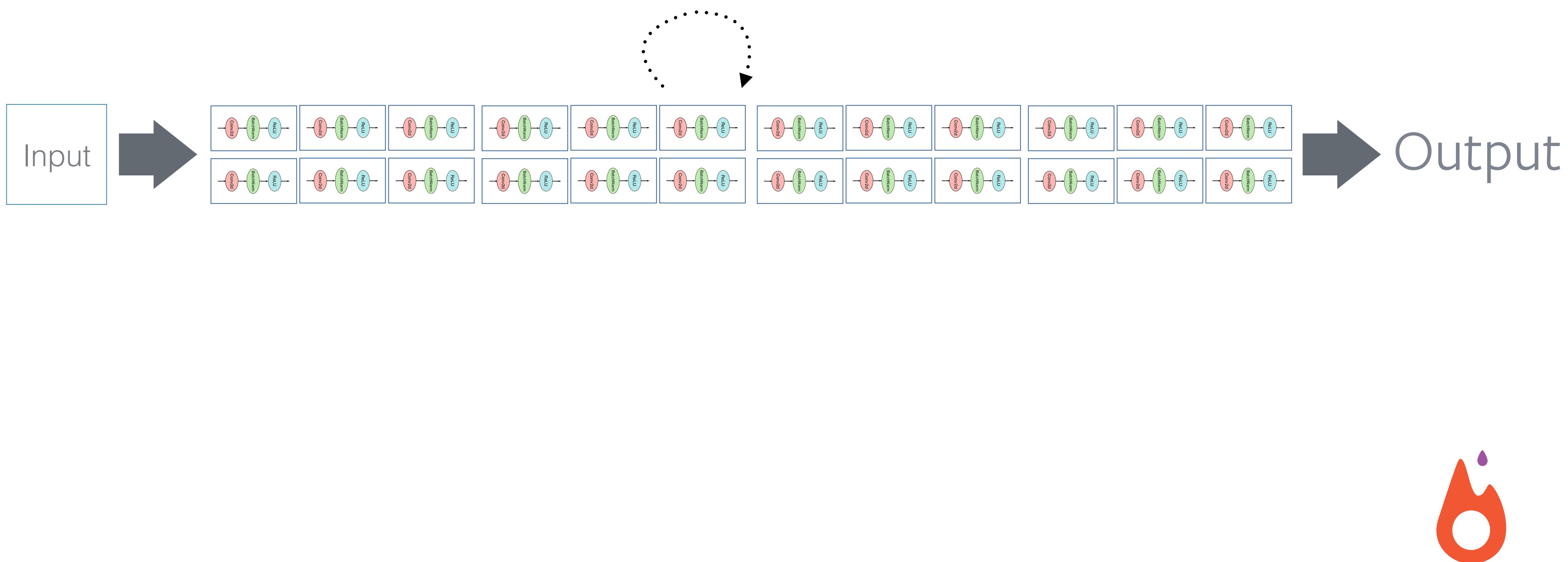


Figure by Wikipedia: https://en.wikipedia.org/wiki/Matrix_multiplication

Trained with Gradient Descent

"deep"

recurrent

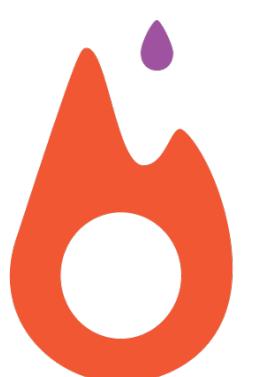
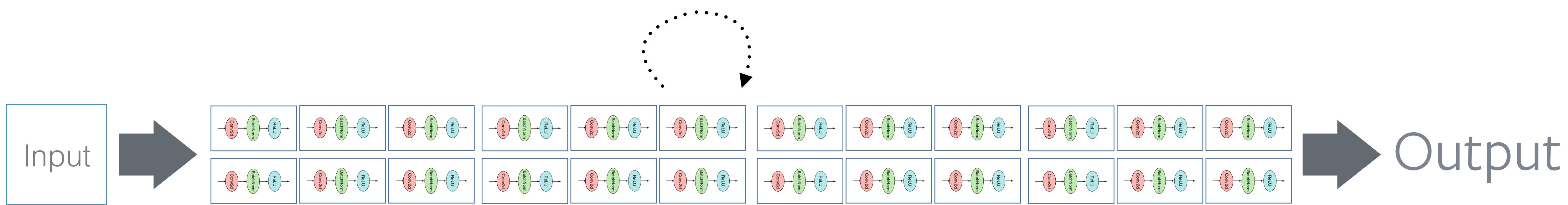
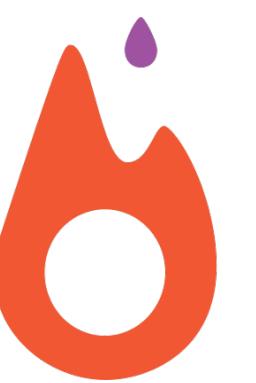
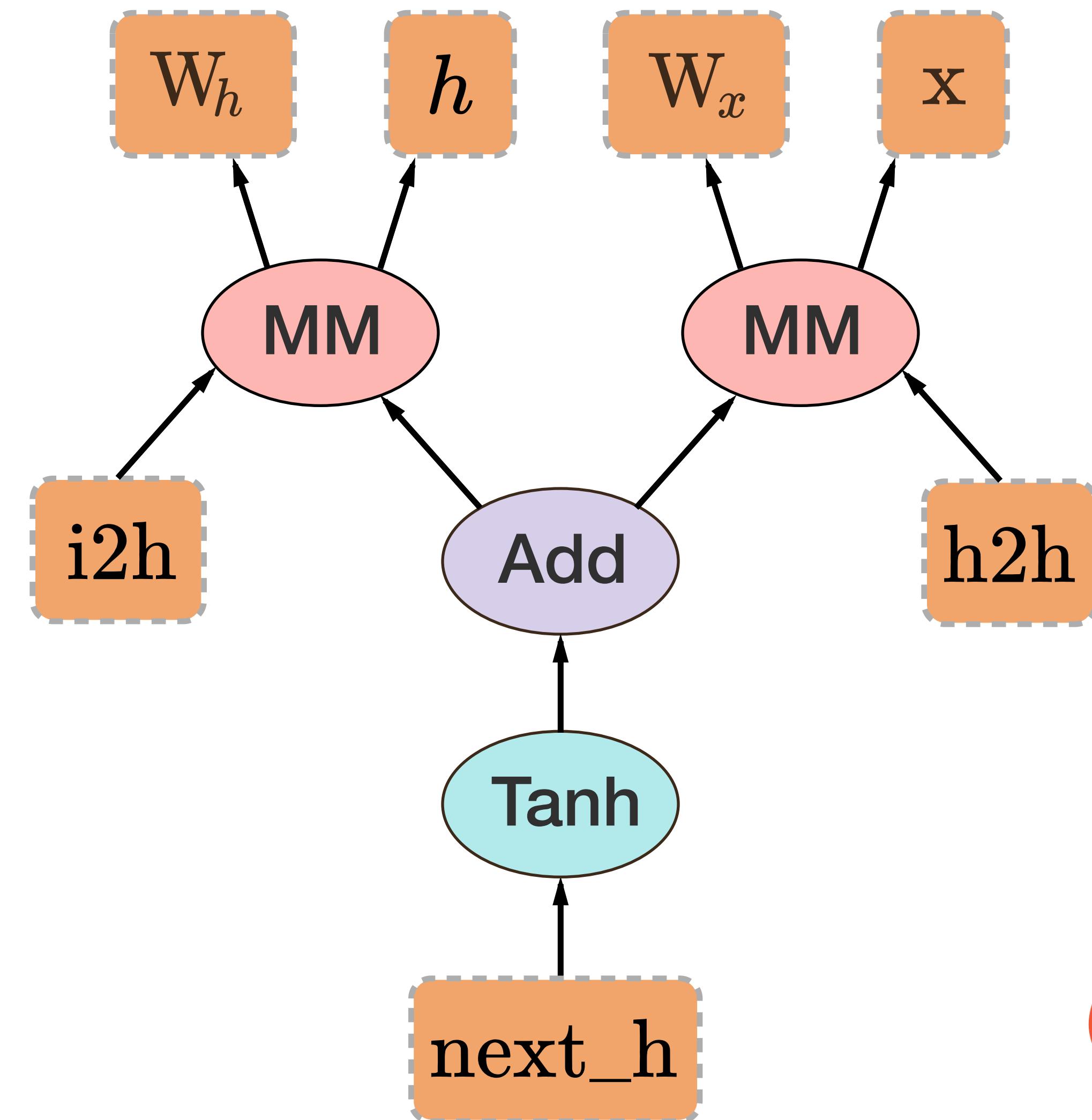


Figure by Wikipedia: https://en.wikipedia.org/wiki/Matrix_multiplication

Trained with Gradient Descent

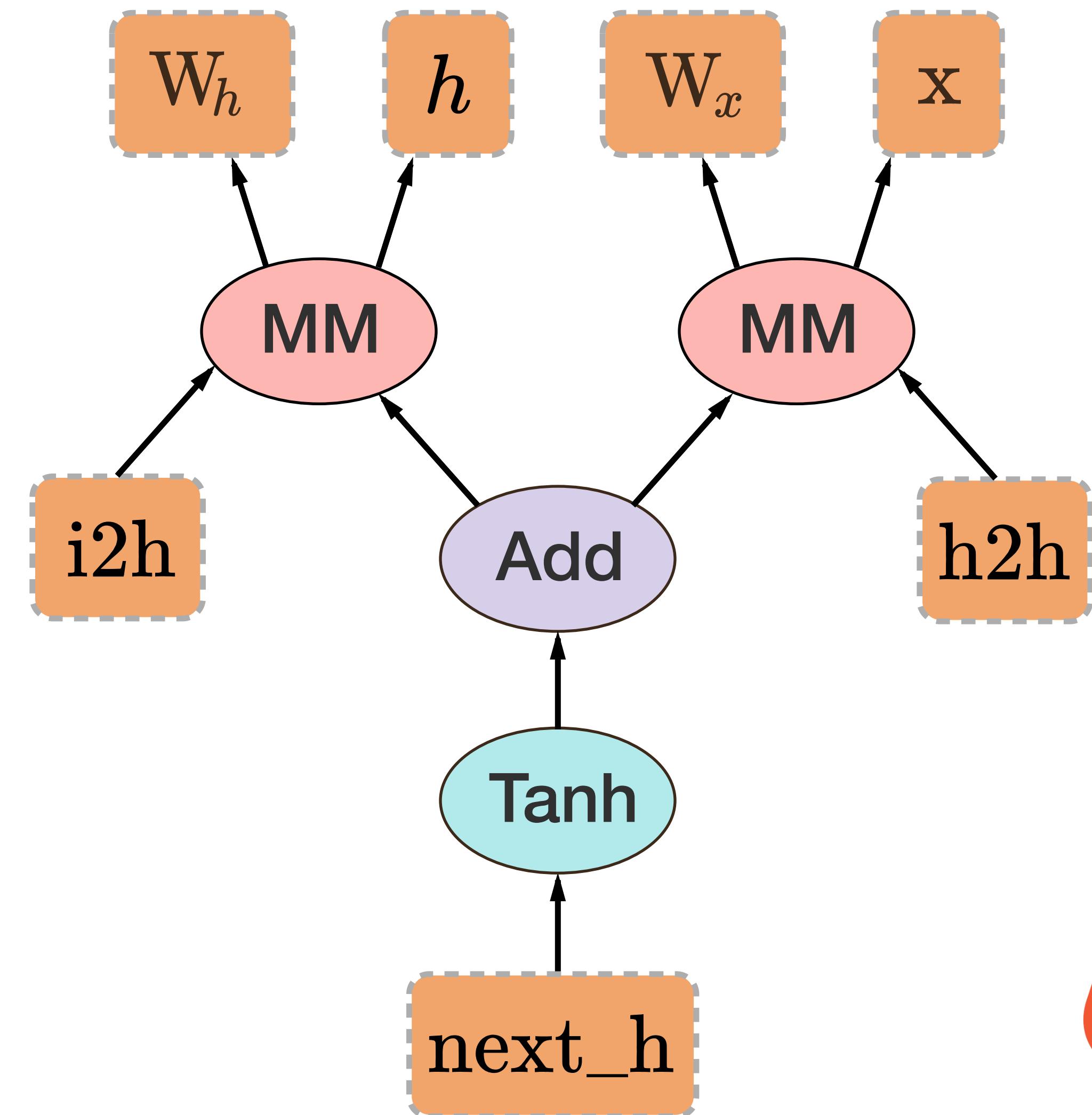
- Provide gradient computation
 - Gradient of one variable w.r.t. any variable in graph



Trained with Gradient Descent

- Provide gradient computation
 - Gradient of one variable w.r.t. any variable in graph
- For example, can compute:

$$\frac{d\text{next}_h}{dW_h}$$



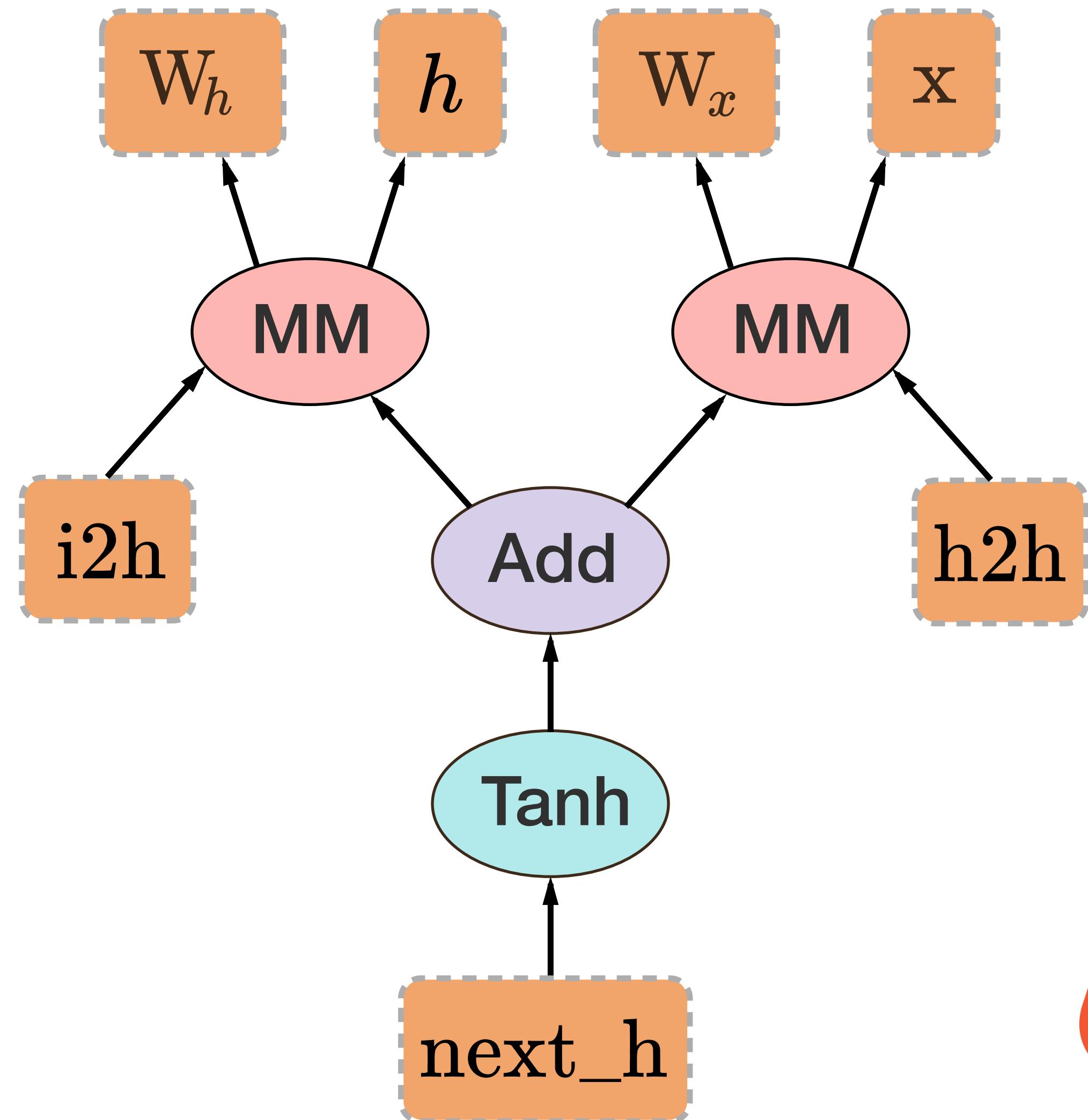
Trained with Gradient Descent

- Provide gradient computation
 - Gradient of one variable w.r.t. any variable in graph
- For example, can compute:

$$\frac{d\text{next}_h}{dW_h}$$

- And then you usually do SGD step:

$$W_h = W_h - \alpha * \frac{d\text{next}_h}{dW_h}$$



Computation Graph Toolkits



Computation Graph Toolkits

Declarative Toolkits



Caffe



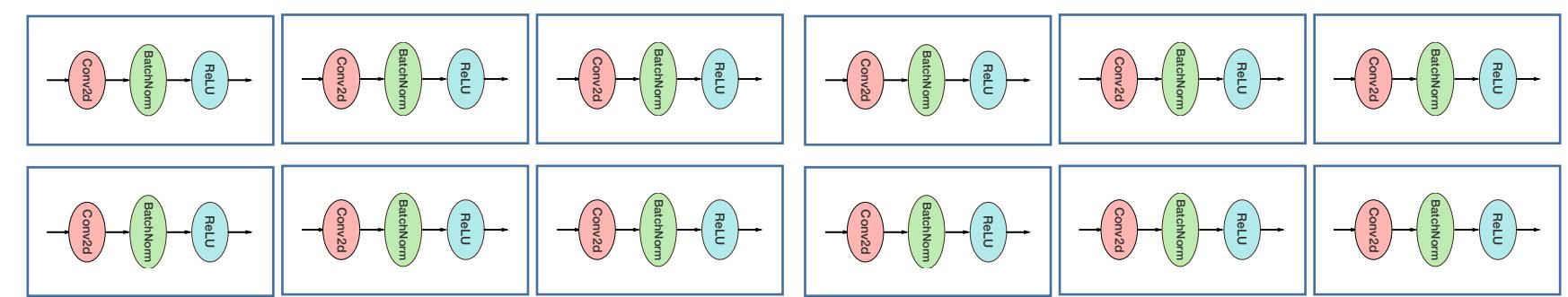
Declarative Toolkits

1. Declare and compile a model

typically in a high-level language like Python

2. Repeatedly execute the model in a VM

Python script



Toolkit
VM



Computation Graph

Declarative Toolkits

- Declare a computation
 - with placeholder variables
- Compile it
- Run it in a Session

```
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))
```

Input / output placeholders

Computation Graph

Declarative Toolkits

- Declare a computation
 - with placeholder variables
- Compile it
- Run it in a Session

```
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")                                Model definition
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))
```

Computation Graph

Declarative Toolkits

- Declare a computation
 - with placeholder variables
- Compile it
- Run it in a Session

A separate turing-complete
Virtual Machine

```
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))
```

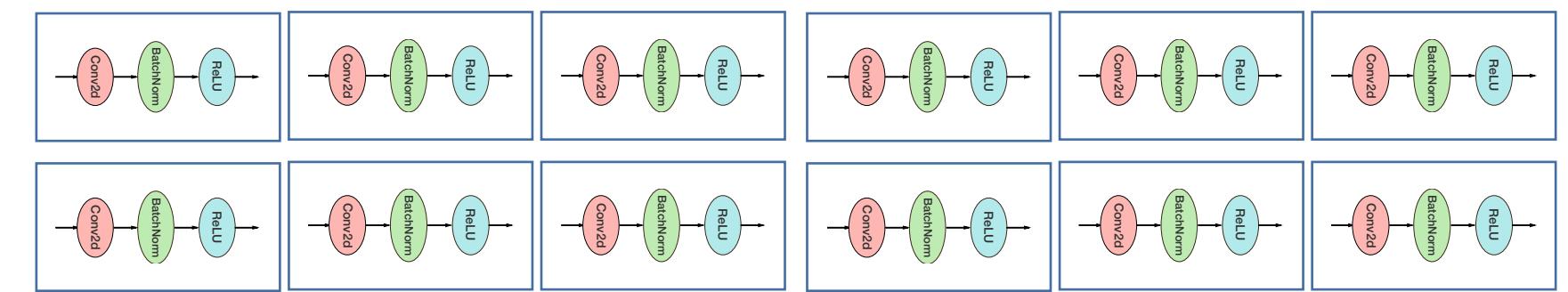
Declarative Toolkits

1. Declare and compile a model

typically in a high-level language like Python

2. Repeatedly execute the model in a VM

Python script



Toolkit
VM



Imperative Toolkits



Imperative Toolkits

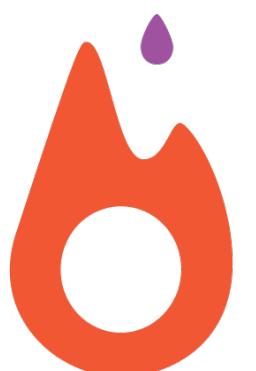
Run a computation

- computation is run



HIPS Autograd

Dynet



Imperative Toolkits

- Run a computation
- computation is run!

```
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)
```

Imperative Toolkits

- Run a computation
- computation is run!
- no separate execution engine

```
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)
```

Imperative Toolkits

- Run a computation
- computation is run!
- no separate execution engine
- great for Dynamic Graphs!

```
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)
```

Imperative Toolkits

- Run a computation
- computation is run!
- no separate execution engine
- great for Dynamic Graphs!

About Dynamic Graphs.....

```
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)
```

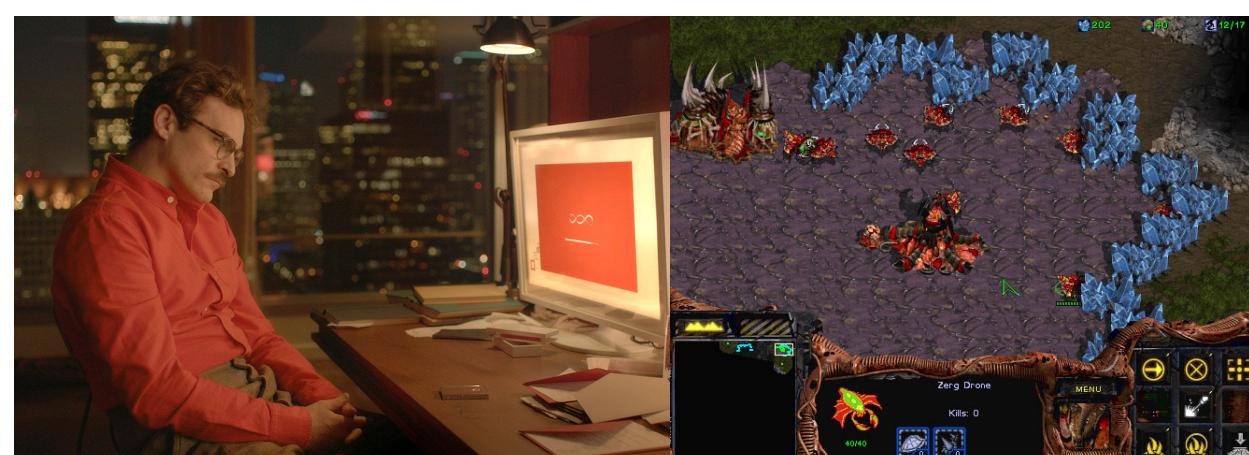
Dynamic Graphs: a paradigm shift in AI Research



Today's AI



Active Research &
Future AI



Tools for AI
keeping up with change

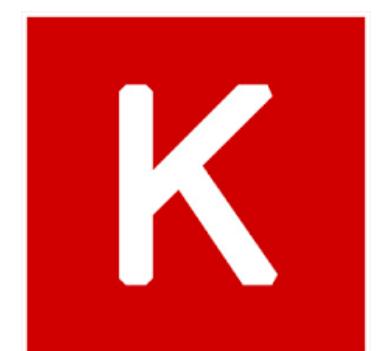
PYTORCH



theano



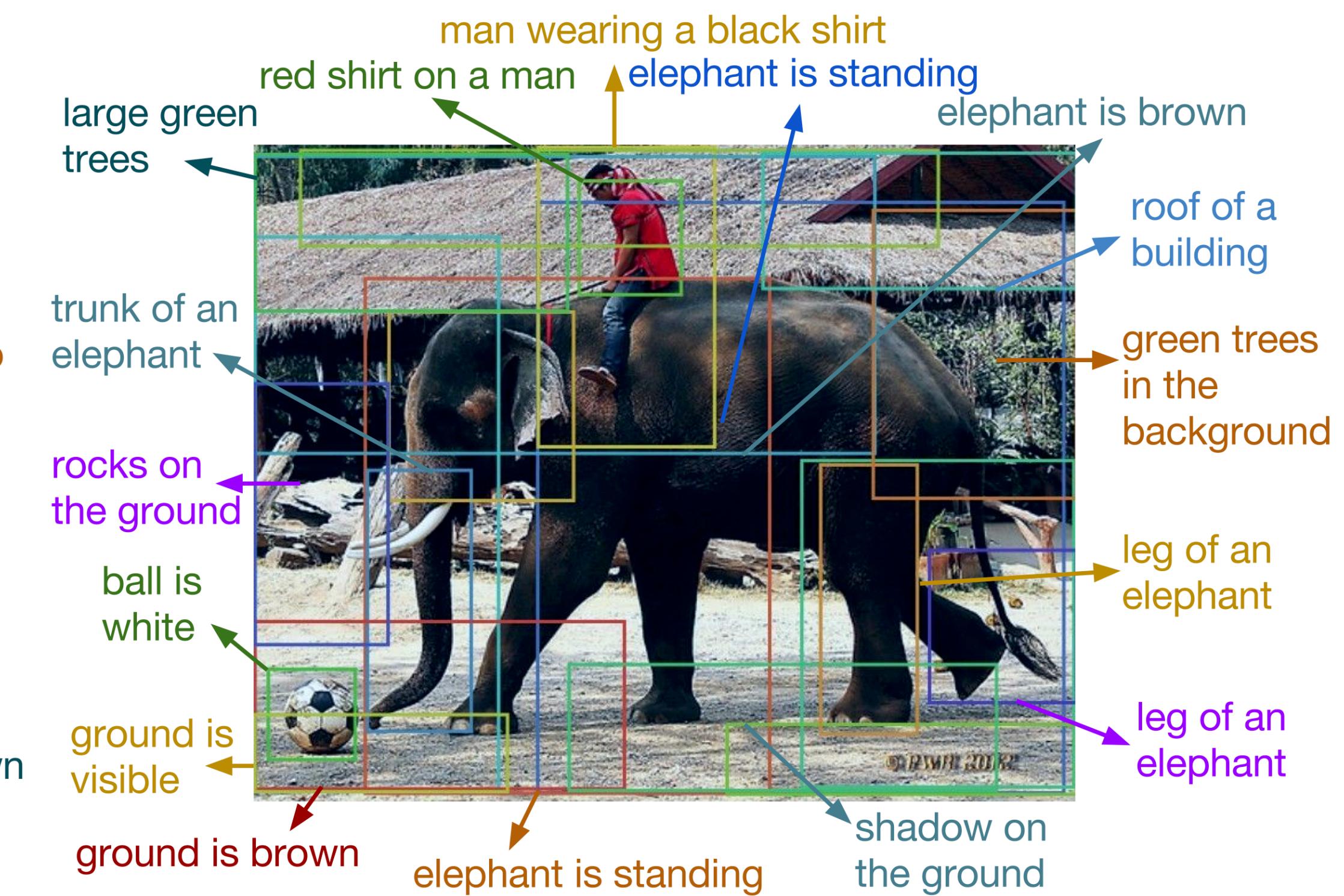
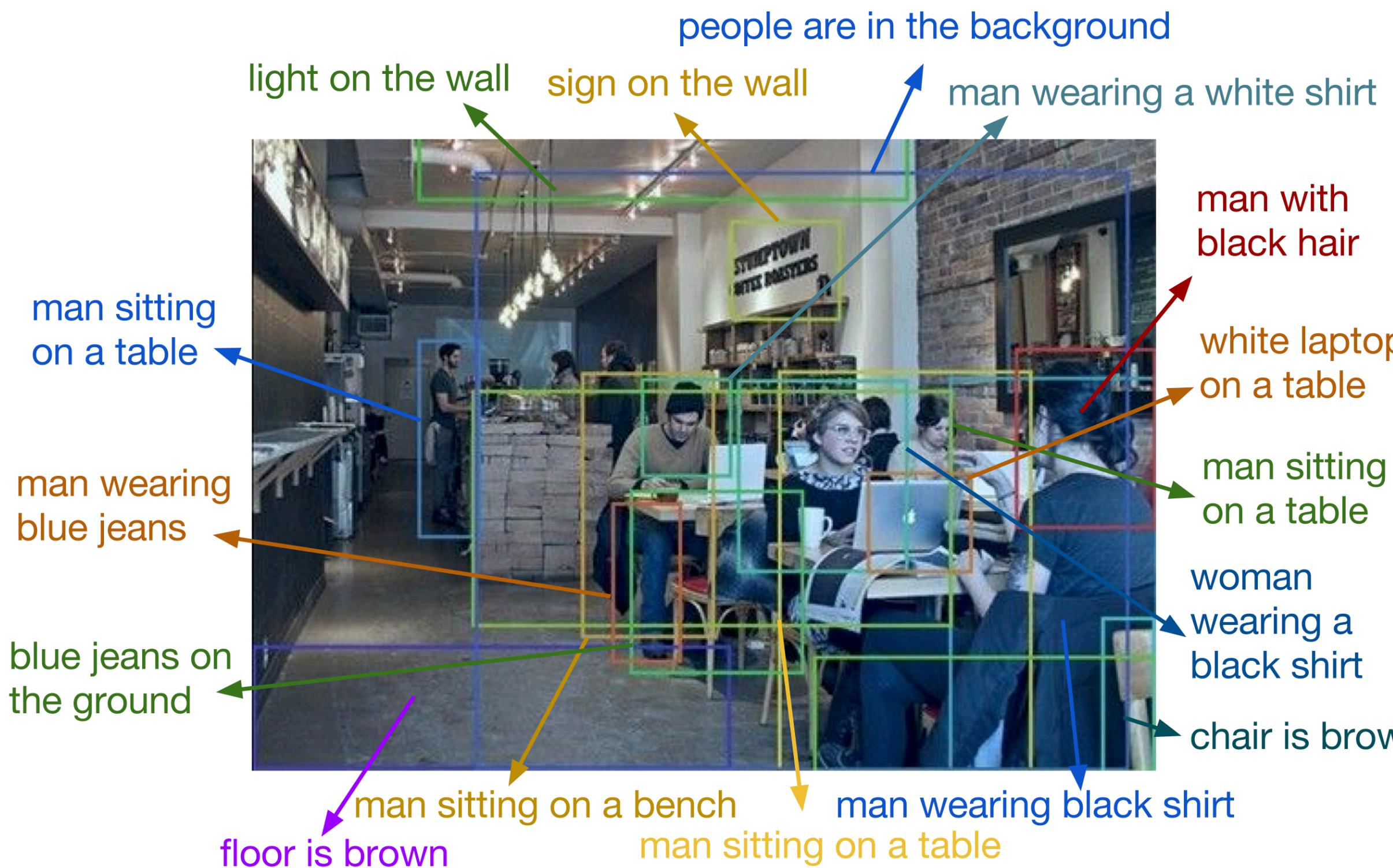
Caffe



Today's AI

DenseCap by Justin Johnson & group

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



Today's AI

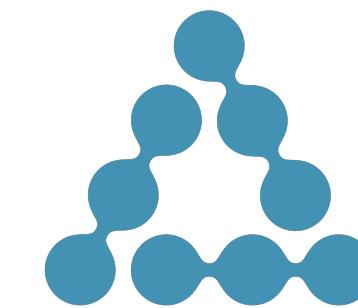
Future AI

Tools for AI

Today's AI



DeepMask by Pedro Pinhero & group



Today's AI



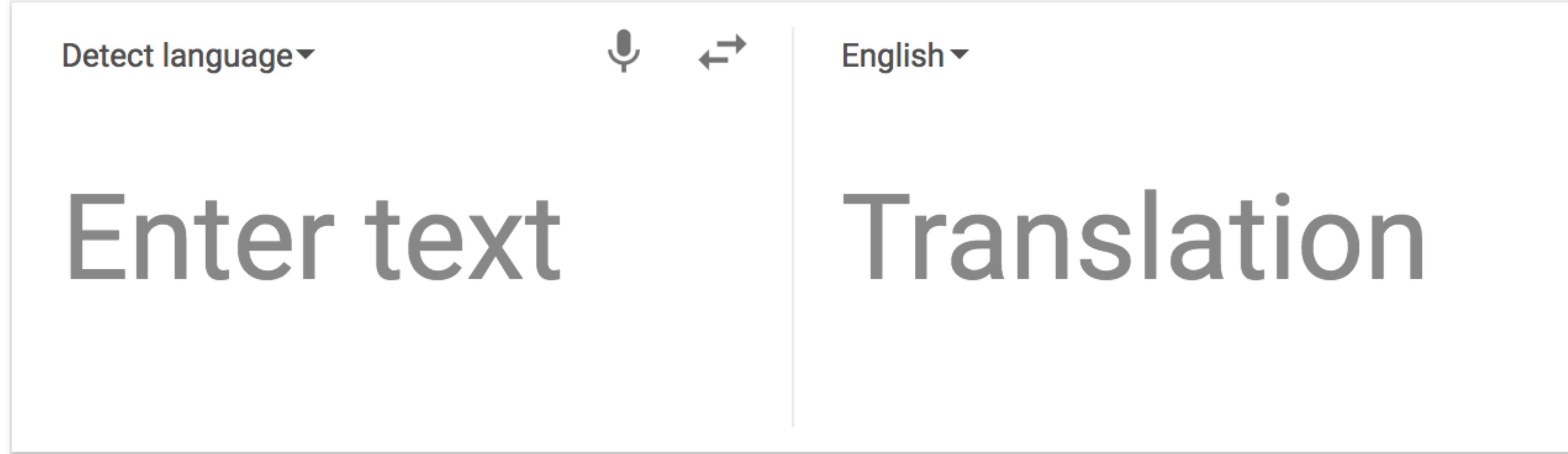
Future AI



Tools for AI

Today's AI

Machine Translation



The screenshot shows the Google Translate interface. At the top left is a "Detect language" dropdown with a downward arrow. Next to it are icons for microphone and a double-headed arrow. On the right is a "English" dropdown with a downward arrow. Below these are two large text fields: "Enter text" on the left and "Translation" on the right. The "Enter text" field is currently empty.



Google
Translate



BING TRANSLATOR

Today's AI



Future AI



Tools for AI

Today's AI

Text Classification (sentiment analysis etc.)

Text Embeddings

Graph embeddings

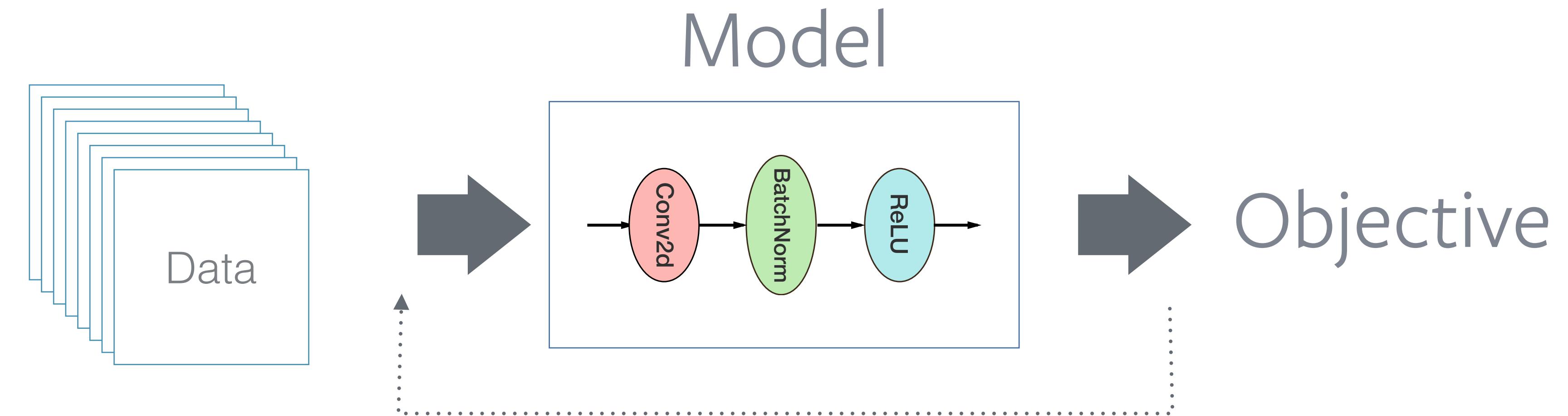
Machine Translation

Ads ranking



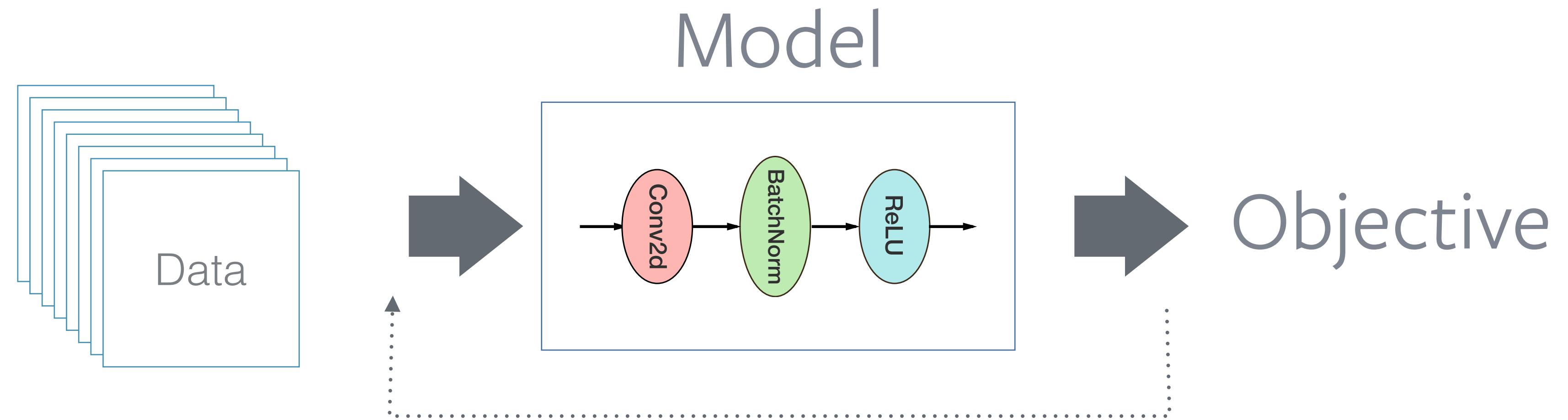
Today's AI

Train Model

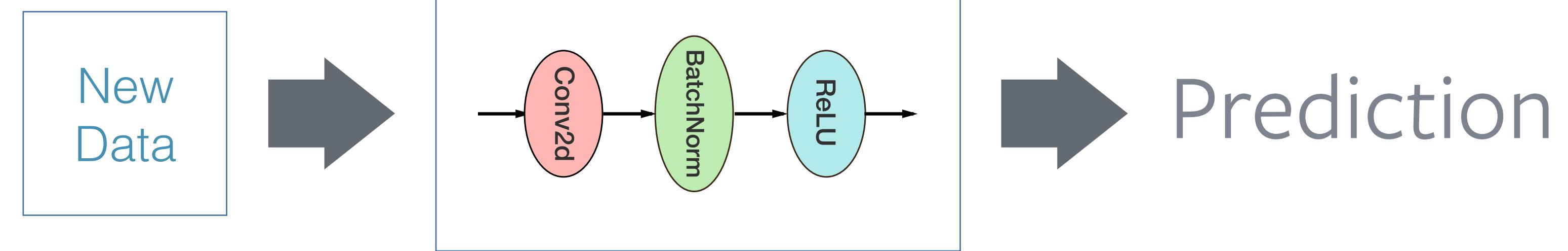


Today's AI

Train Model



Deploy & Use



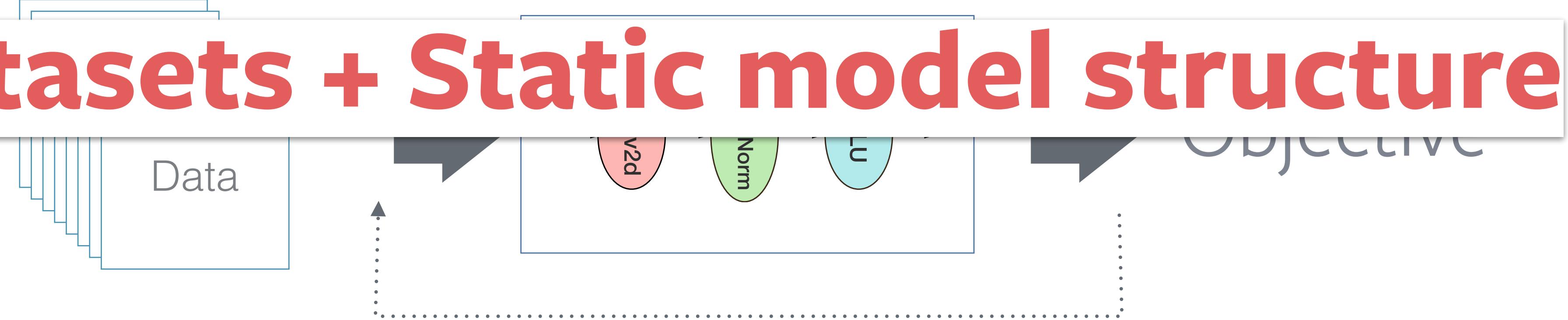
Today's AI

Future AI

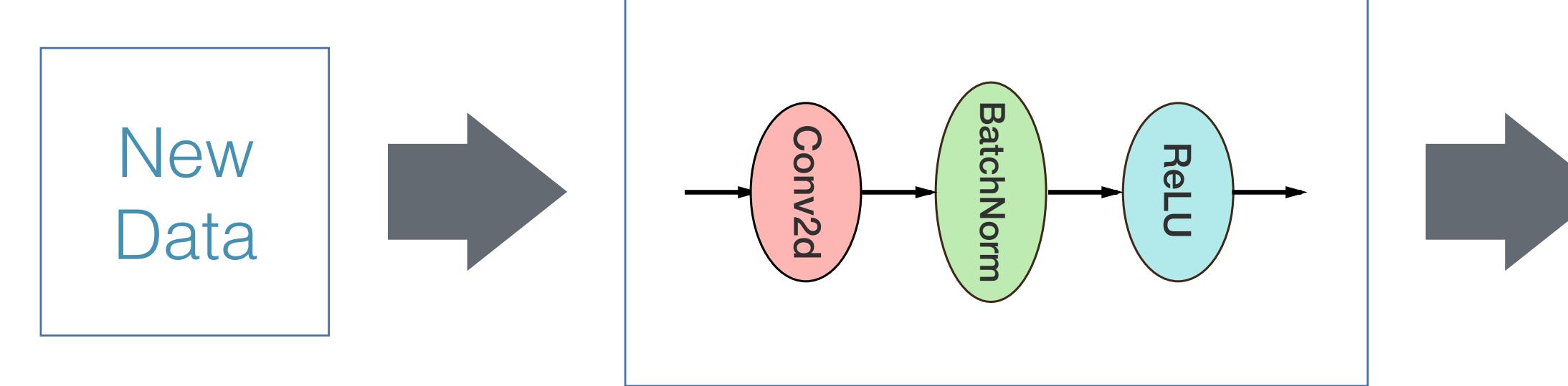
Tools for AI

Today's AI

- Static datasets + Static model structure

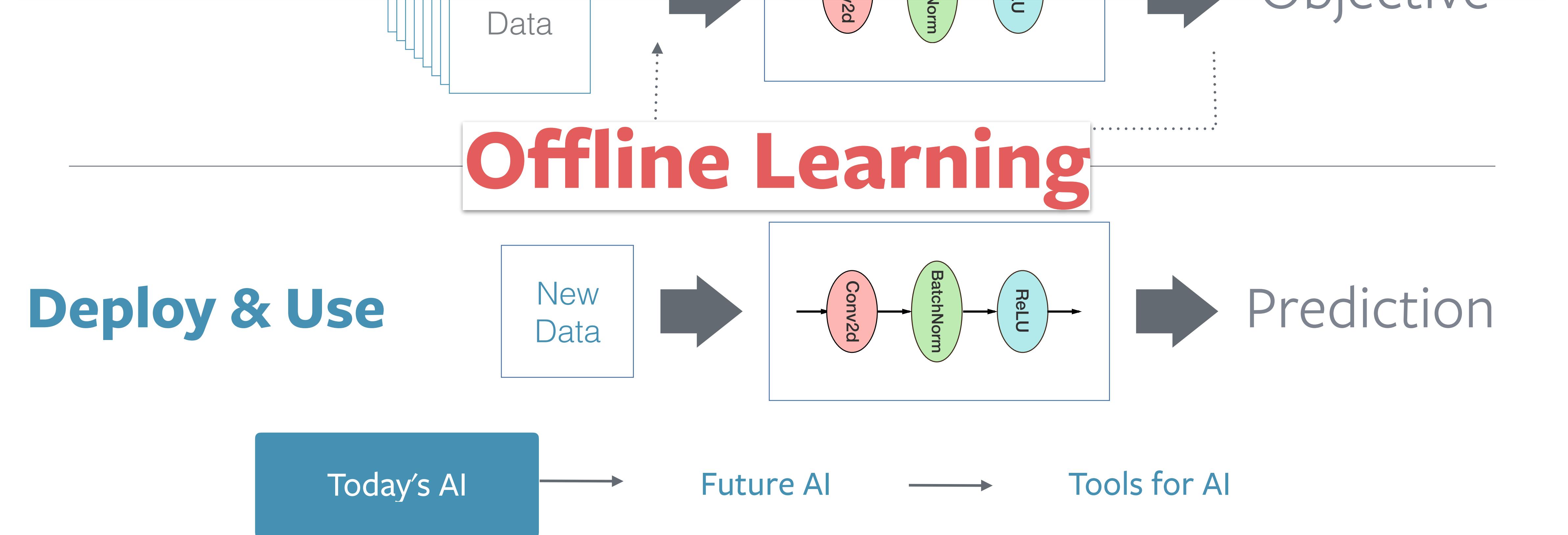


Deploy & Use



Today's AI

- Static datasets + Static model structure



Current AI Research / Future AI

Self-driving Cars



Today's AI



Future AI



Tools for AI

Current AI Research / Future AI

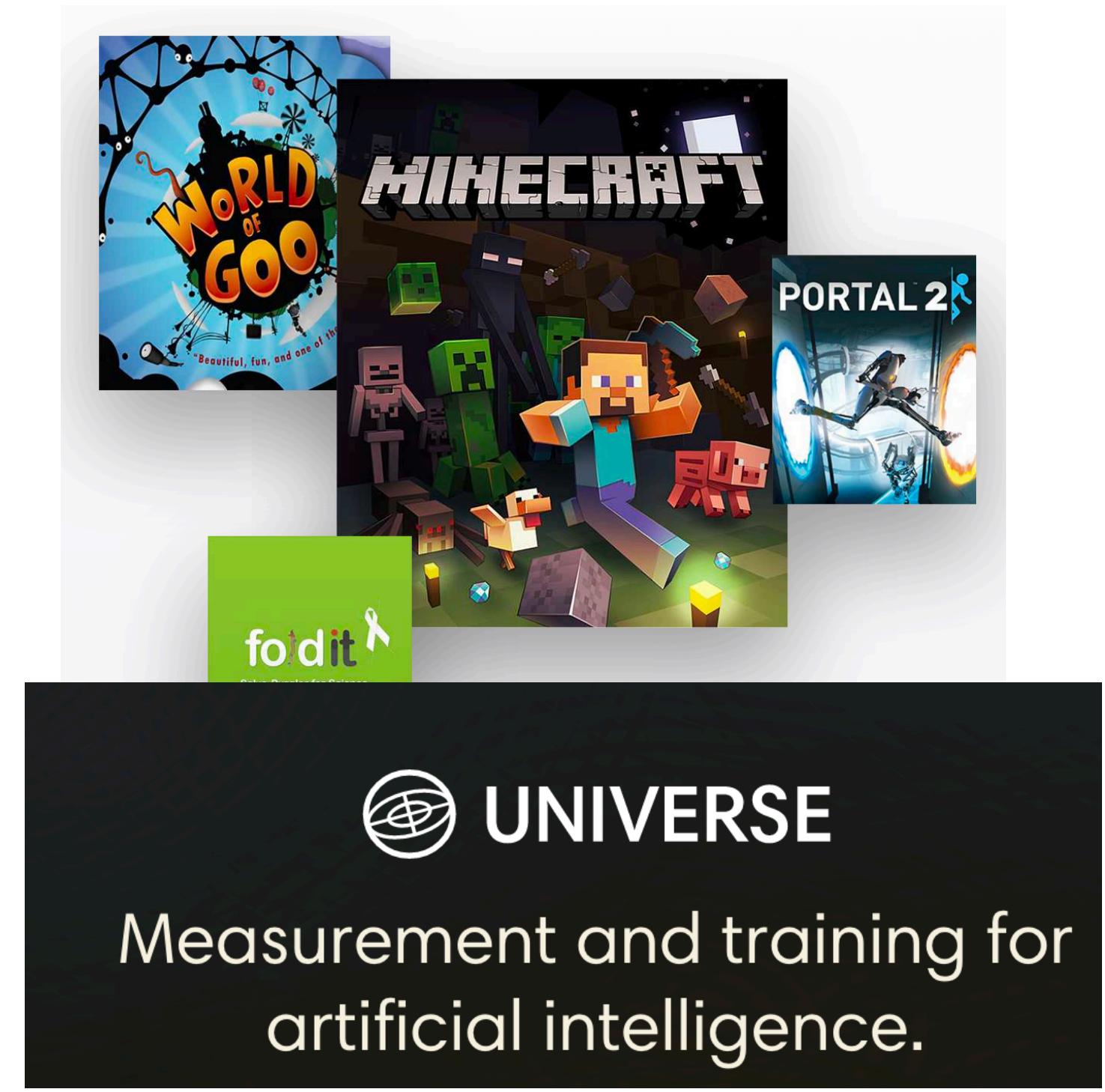
Agents trained in many environments



Cars



Video games



Internet

Today's AI



Future AI

Tools for AI

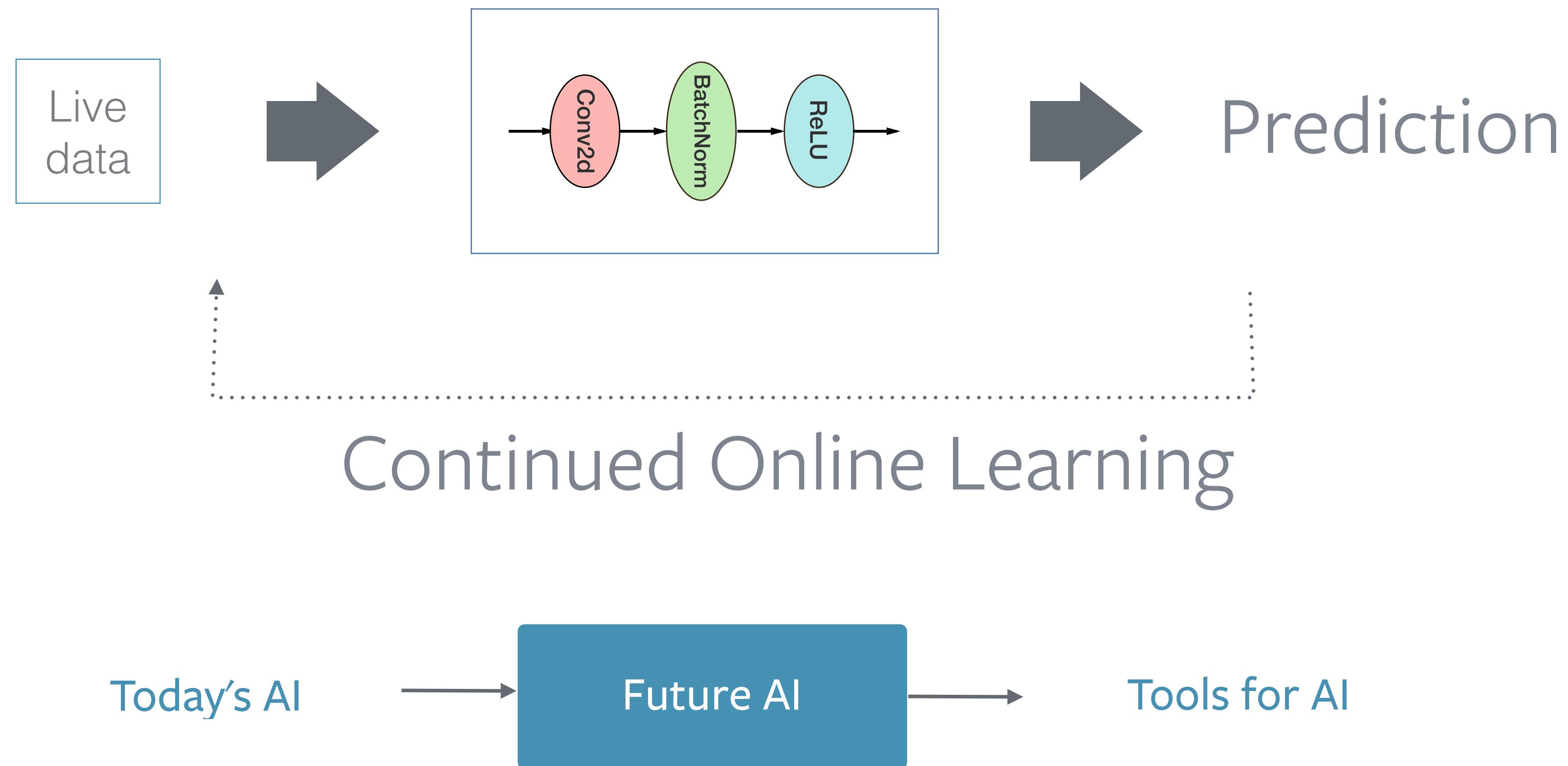
Current AI Research / Future AI

Dynamic Neural Networks

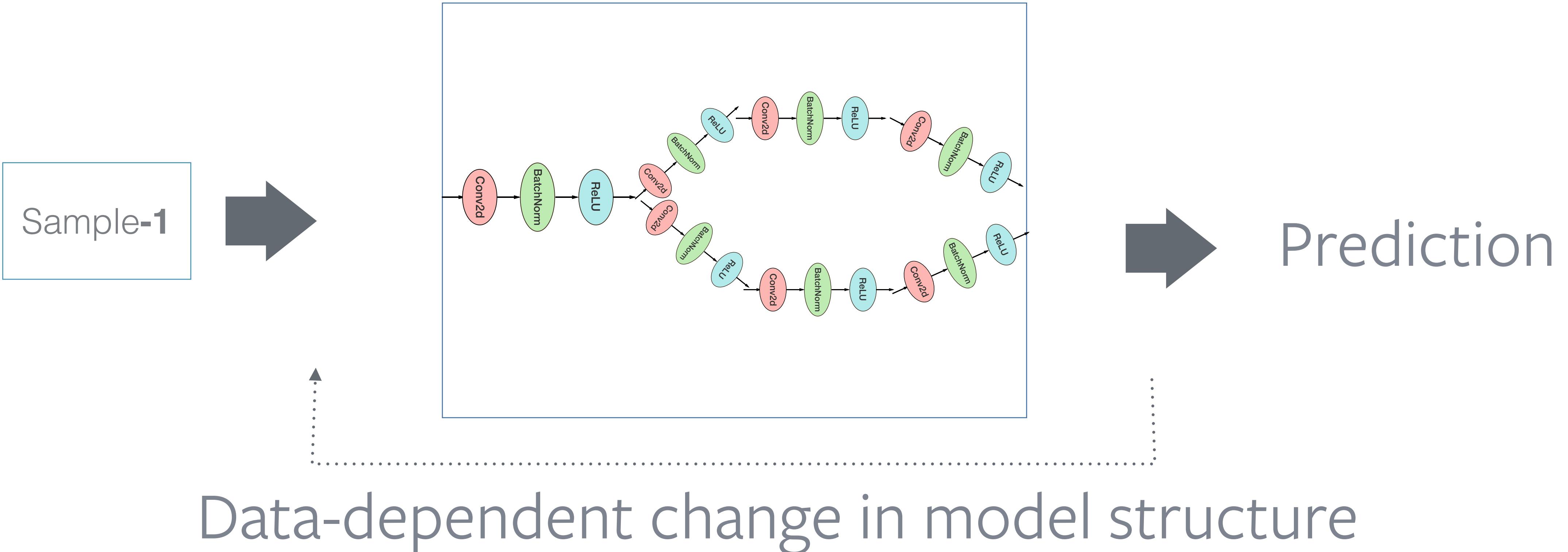
self-adding new memory or layers
changing evaluation path based on inputs



Current AI Research / Future AI



Current AI Research / Future AI

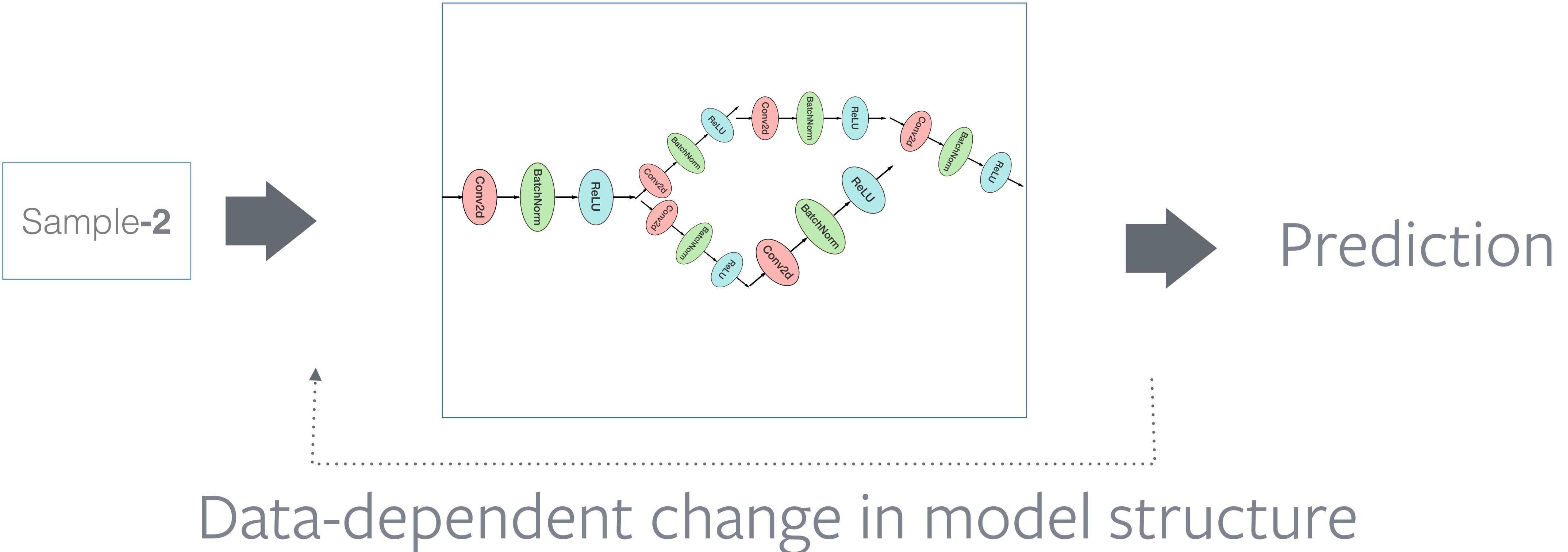


Today's AI

Future AI

Tools for AI

Current AI Research / Future AI

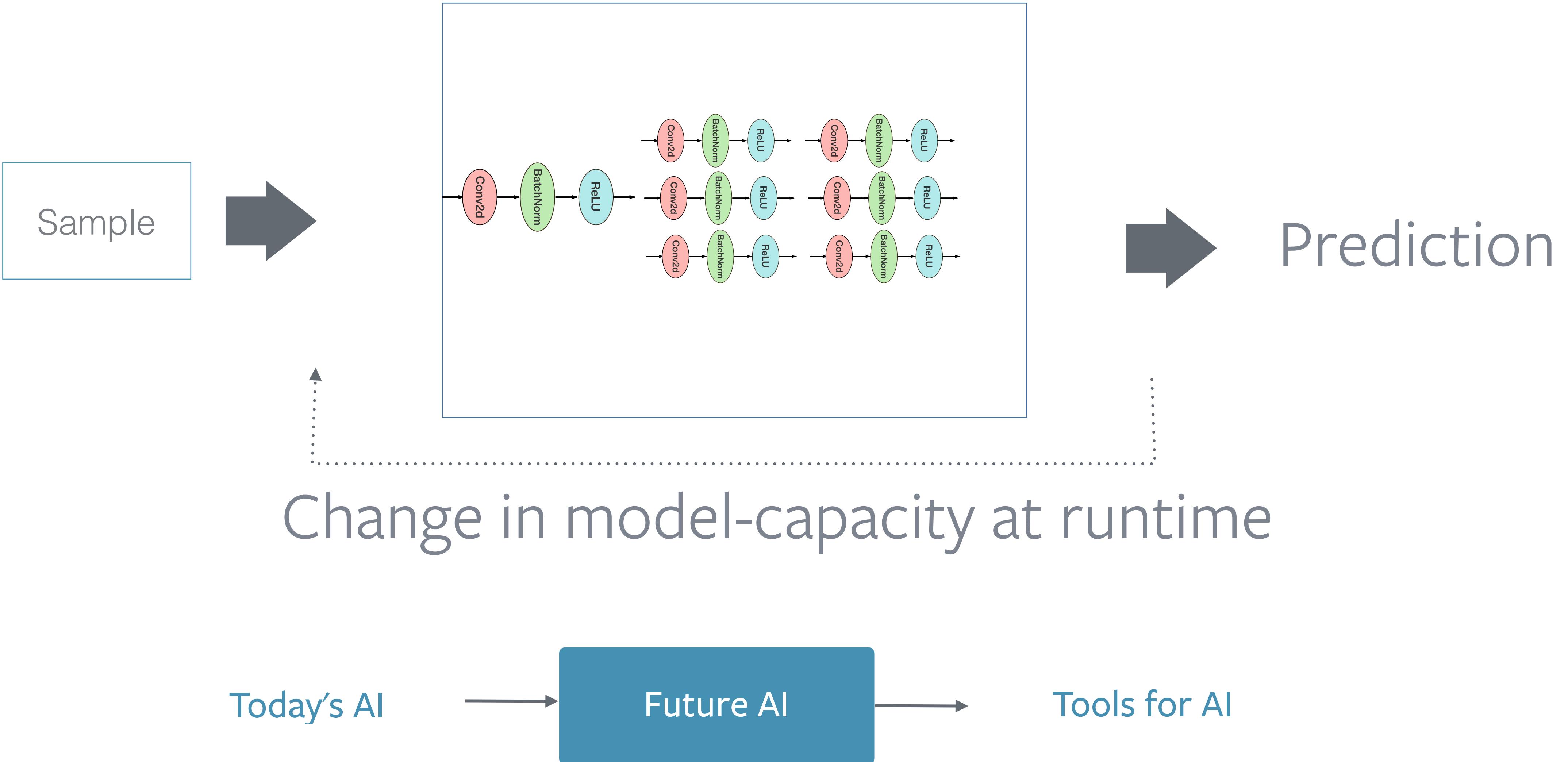


Today's AI

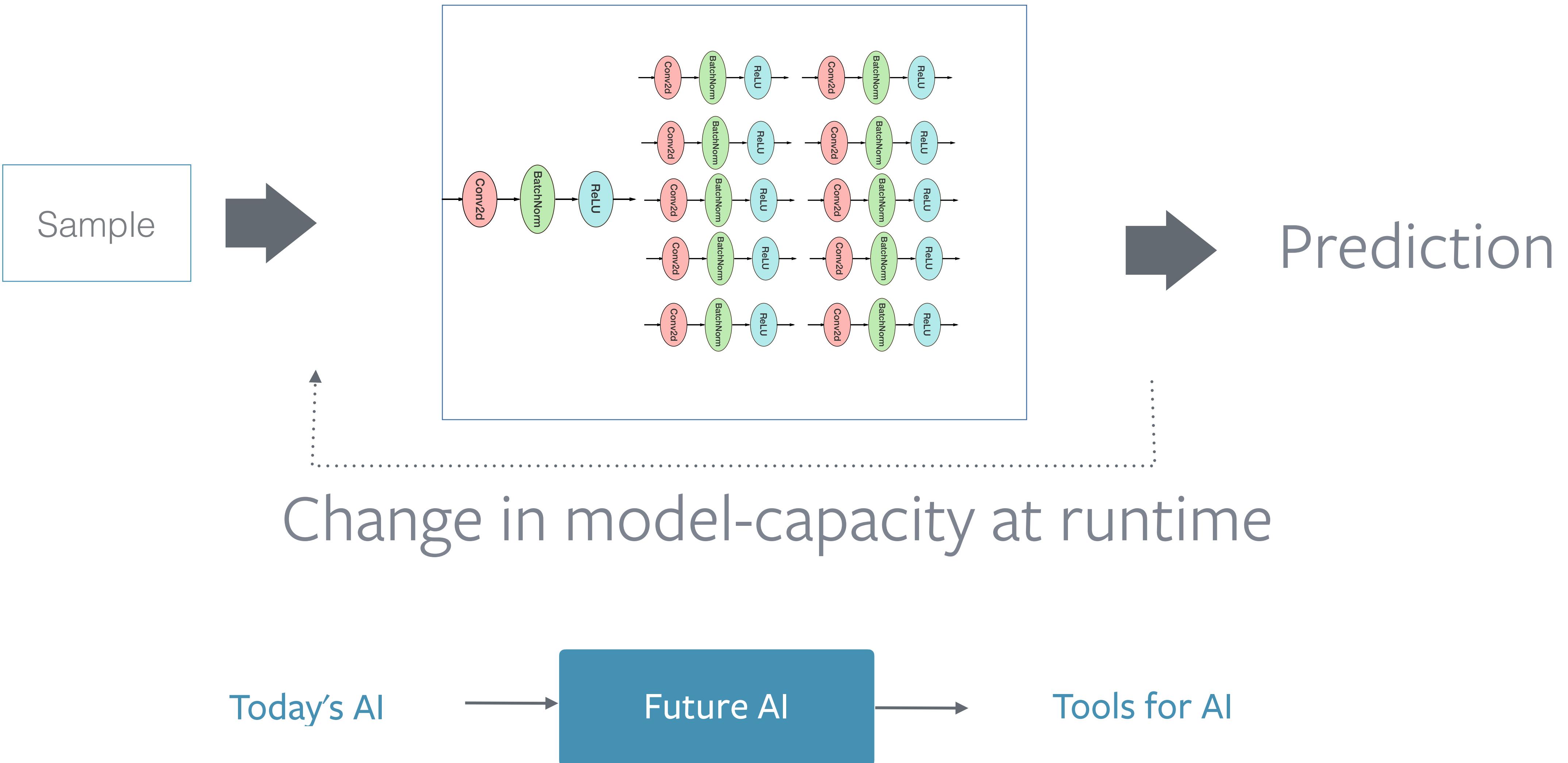
Future AI

Tools for AI

Current AI Research / Future AI



Current AI Research / Future AI



The need for a dynamic framework

- 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



Imperative Toolkits

- Run a computation
- computation is run!
- no separate execution engine
- great for Dynamic Graphs!

About Dynamic Graphs.....

```
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)
```

automatic differentiation engine

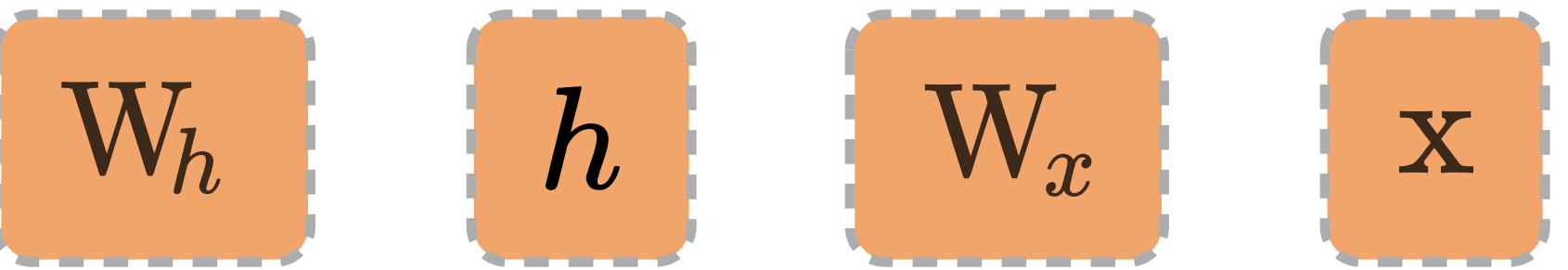
for deep learning and reinforcement learning



PyTorch Autograd

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

PyTorch Autograd

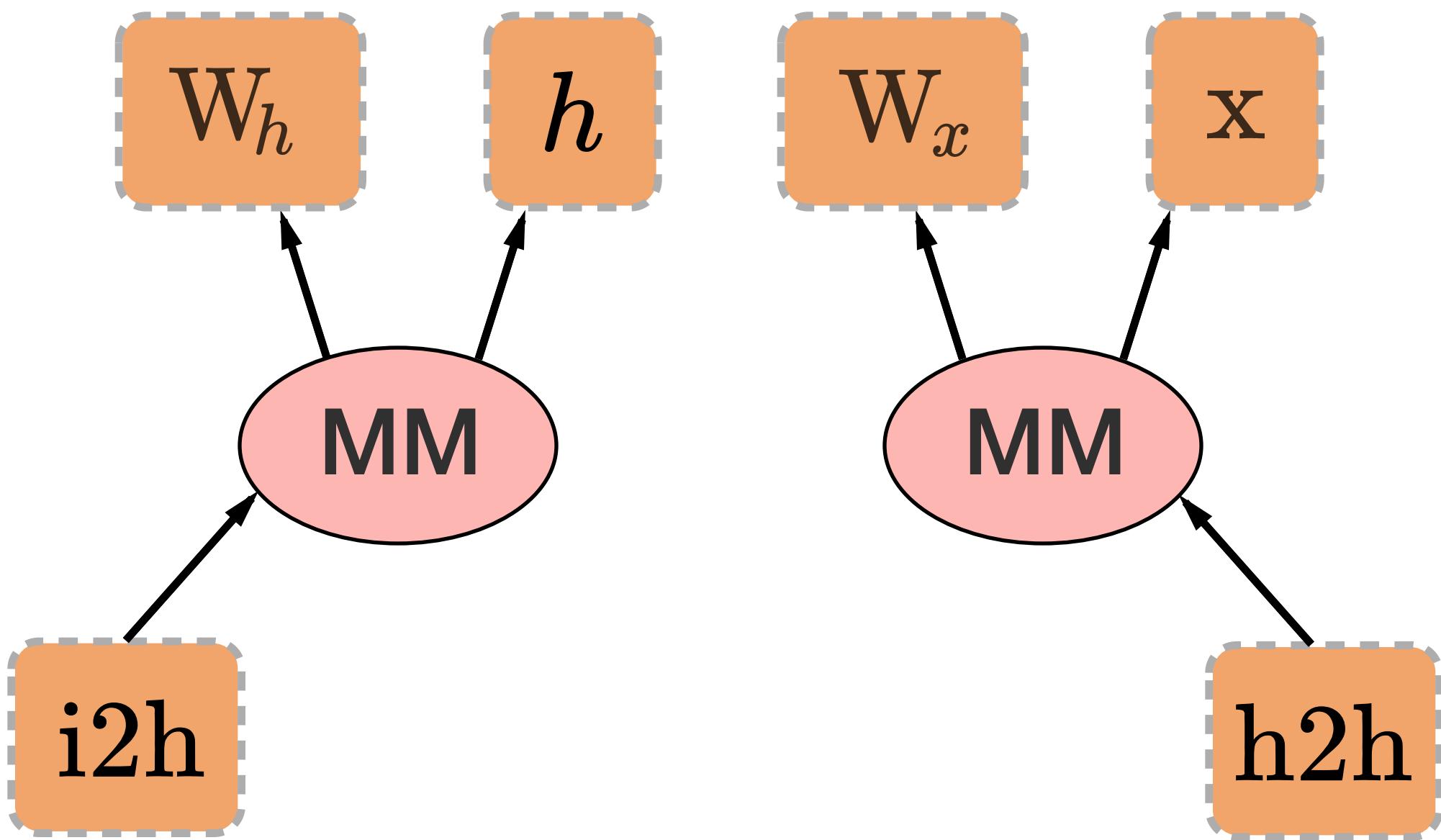


```
from torch.autograd import Variable
```

```
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))
```

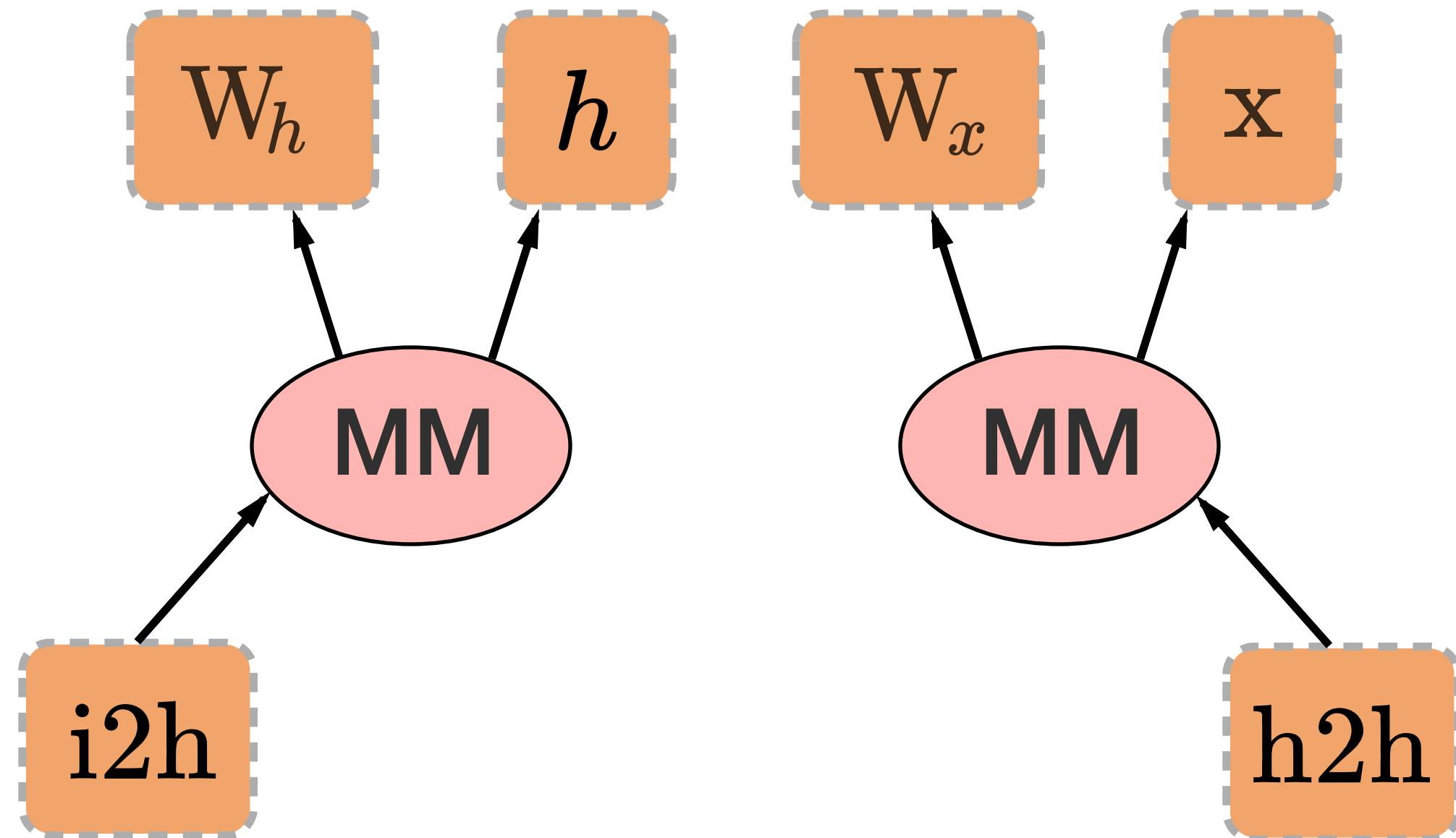
PyTorch Autograd

```
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())
```



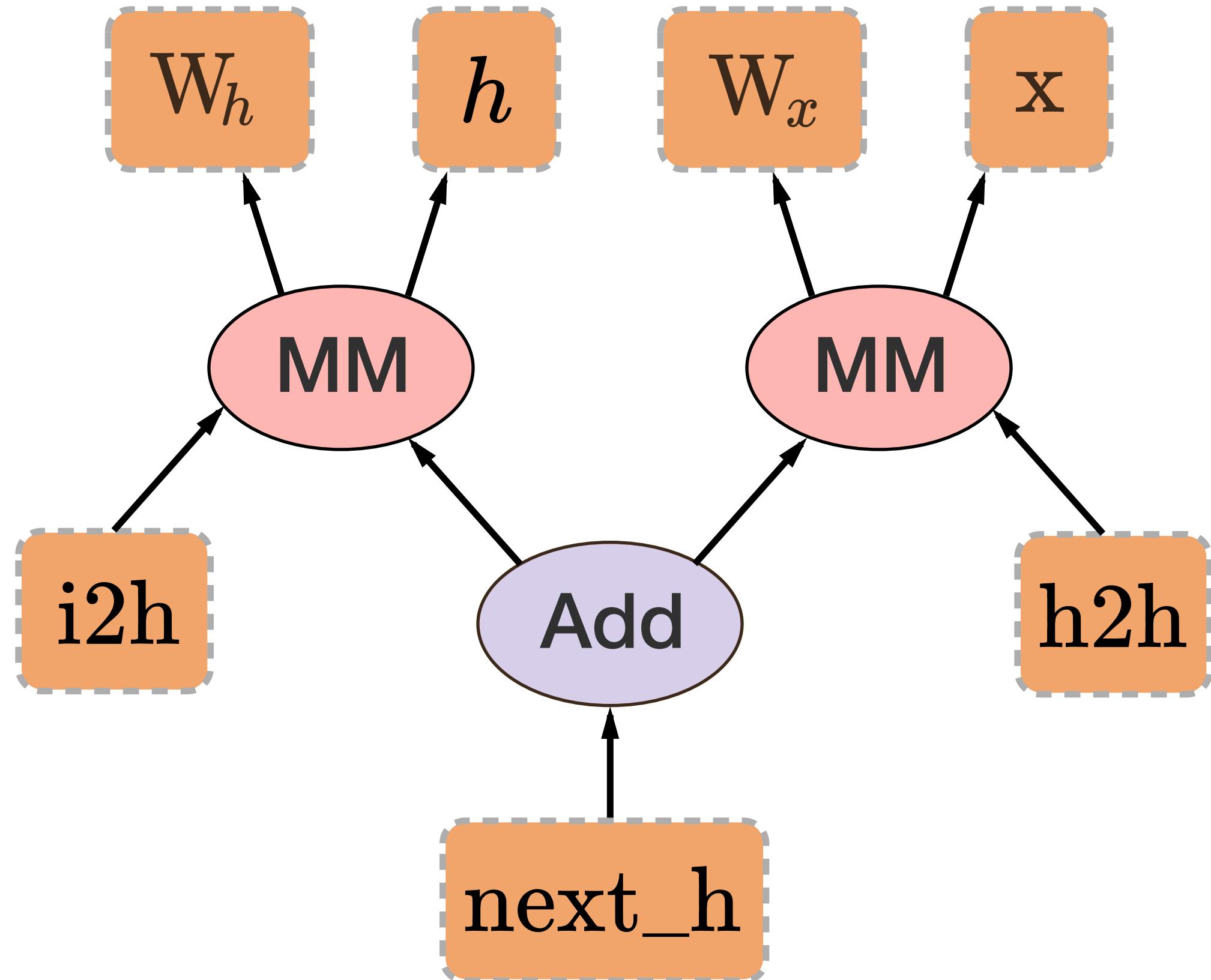
PyTorch Autograd

```
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
```



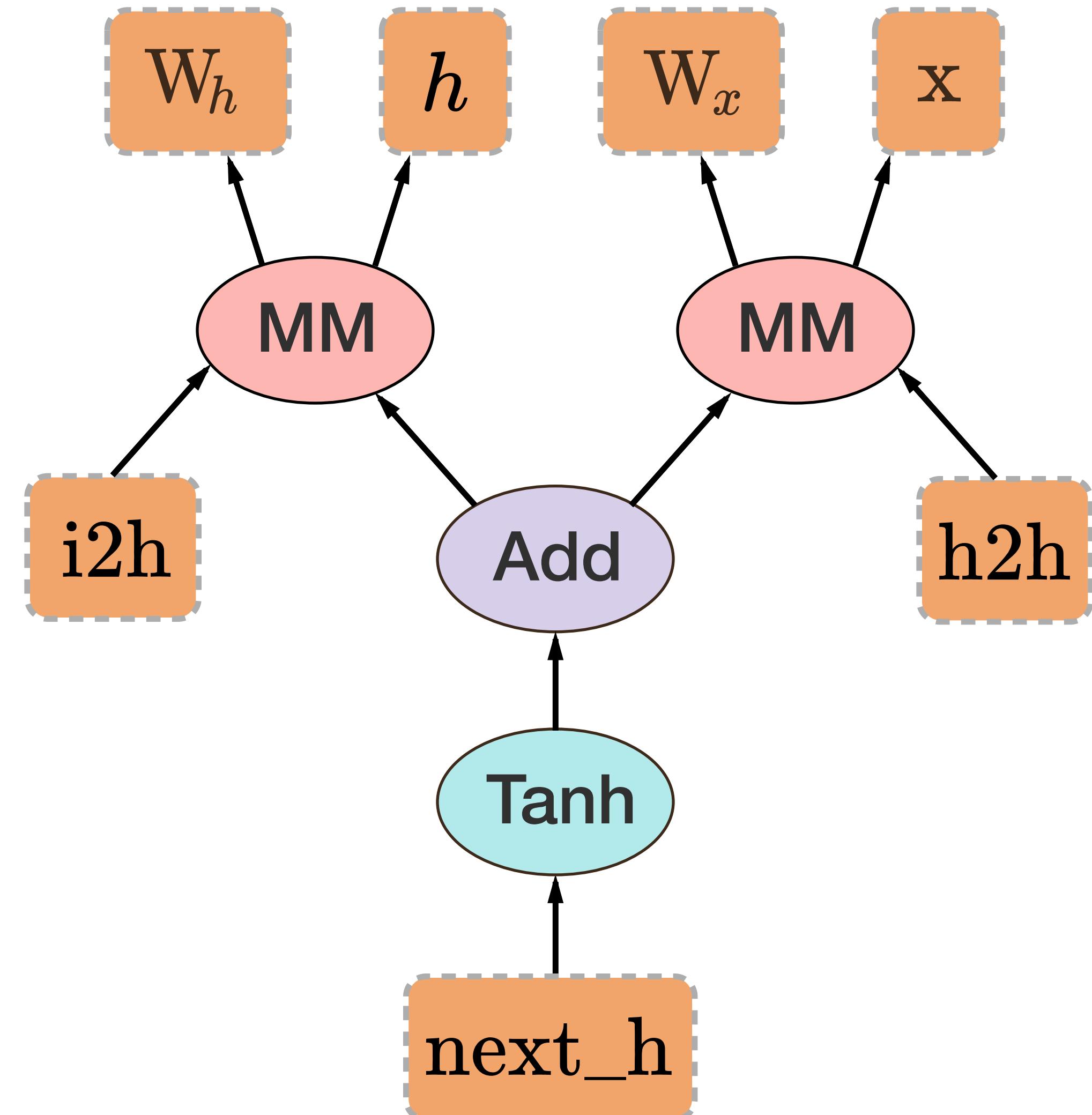
PyTorch Autograd

```
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
```



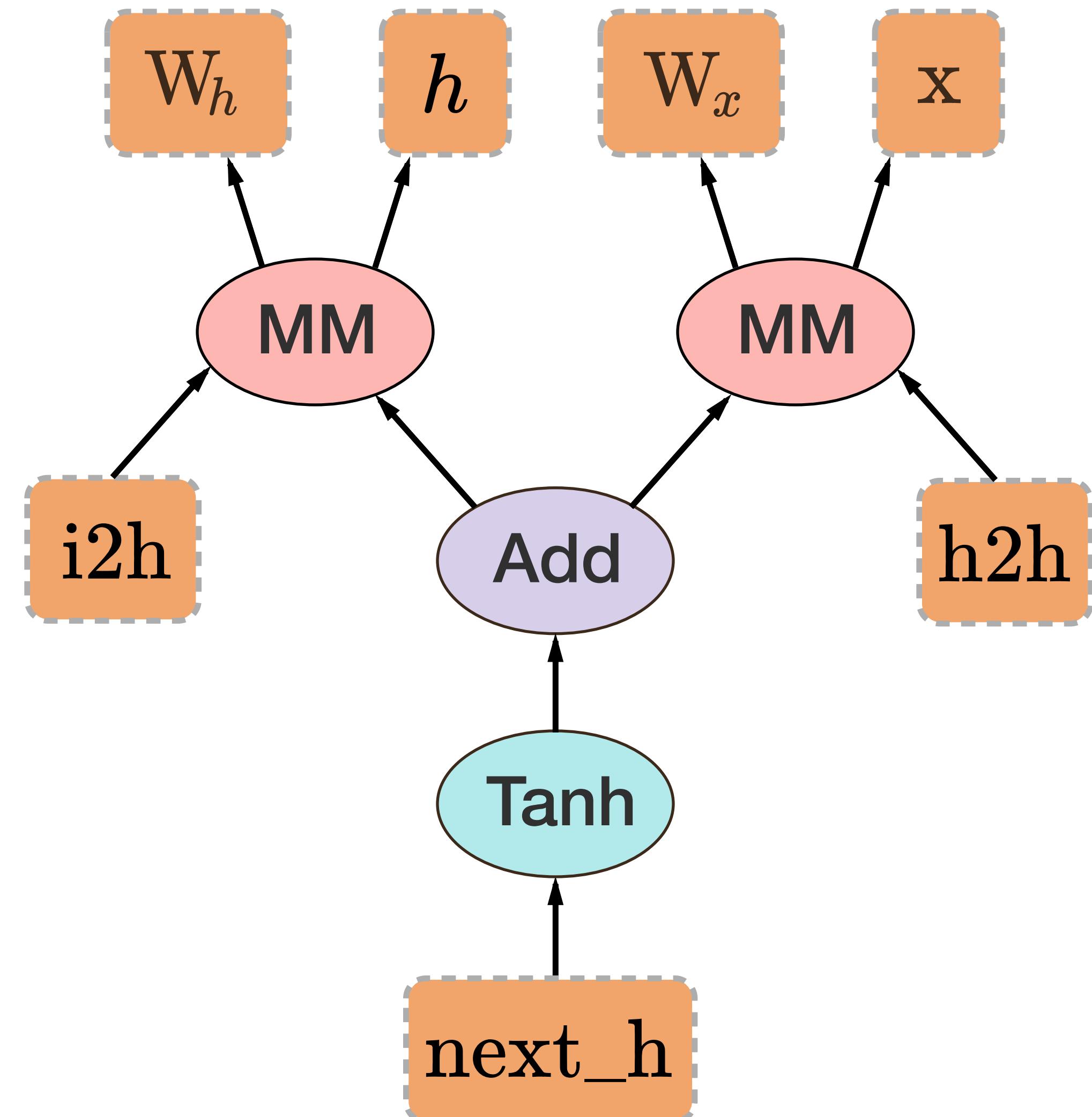
PyTorch Autograd

```
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))
```



Neural Networks

```
1  class Net(nn.Module):
2      def __init__(self):
3          super(Net, self).__init__()
4          self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
5          self.conv2 = nn.Conv2d(10, 20, kernel_size=5)
6          self.conv2_drop = nn.Dropout2d()
7          self.fc1 = nn.Linear(320, 50)
8          self.fc2 = nn.Linear(50, 10)
9
10     def forward(self, x):
11         x = F.relu(F.max_pool2d(self.conv1(x), 2))
12         x = F.relu(F.max_pool2d(self.conv2_drop(self.conv2(x)), 2))
13         x = x.view(-1, 320)
14         x = F.relu(self.fc1(x))
15         x = F.dropout(x, training=self.training)
16         x = self.fc2(x)
17         return F.log_softmax(x)
18
19 model = Net()
20 input = Variable(torch.randn(10, 20))
21 output = model(input)
```

Neural Networks

```
1  class Net(nn.Module):
2      def __init__(self):
3          super(Net, self).__init__()
4          self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
5          self.conv2 = nn.Conv2d(10, 20, kernel_size=5)
6          self.conv2_drop = nn.Dropout2d()
7          self.fc1 = nn.Linear(320, 50)
8          self.fc2 = nn.Linear(50, 10)
9
10     def forward(self, x):
11         x = F.relu(F.max_pool2d(self.conv1(x), 2))
12         x = F.relu(F.max_pool2d(self.conv2_drop(self.conv2(x)), 2))
13         x = x.view(-1, 320)
14         x = F.relu(self.fc1(x))
15         x = F.dropout(x, training=self.training)
16         x = self.fc2(x)
17         return F.log_softmax(x)
18
19 model = Net()
20 input = Variable(torch.randn(10, 20))
21 output = model(input)
```

Neural Networks

```
1  class Net(nn.Module):
2      def __init__(self):
3          super(Net, self).__init__()
4          self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
5          self.conv2 = nn.Conv2d(10, 20, kernel_size=5)
6          self.conv2_drop = nn.Dropout2d()
7          self.fc1 = nn.Linear(320, 50)
8          self.fc2 = nn.Linear(50, 10)
9
10     def forward(self, x):
11         x = F.relu(F.max_pool2d(self.conv1(x), 2))
12         x = F.relu(F.max_pool2d(self.conv2_drop(self.conv2(x)), 2))
13         x = x.view(-1, 320)
14         x = F.relu(self.fc1(x))
15         x = F.dropout(x, training=self.training)
16         x = self.fc2(x)
17         return F.log_softmax(x)
18
19 model = Net()
20 input = Variable(torch.randn(10, 20))
21 output = model(input)
```

Optimization package

SGD, Adagrad, RMSProp, LBFGS, etc.

```
1 net = Net()
2 optimizer = torch.optim.SGD(net.parameters(), lr=0.01, momentum=0.9)
3
4 for input, target in dataset:
5     optimizer.zero_grad()
6     output = model(input)
7     loss = F.cross_entropy(output, target)
8     loss.backward()
9     optimizer.step()
```

Work items in practice

Writing
Dataset loaders

Building models

Implementing
Training loop

Checkpointing
models

Interfacing with
environments

Building optimizers

Dealing with
GPUs

Building
Baselines



Work items in practice

Writing
Dataset loaders

Building models

Implementing
Training loop

Checkpointing
models

Python + PyTorch - an environment to do all of this

Interfacing with
environments

Building optimizers

Dealing with
GPUs

Building
Baselines



Writing Data Loaders

- every dataset is slightly differently formatted



Writing Data Loaders

- every dataset is slightly differently formatted
- have to be preprocessed and normalized differently



Writing Data Loaders

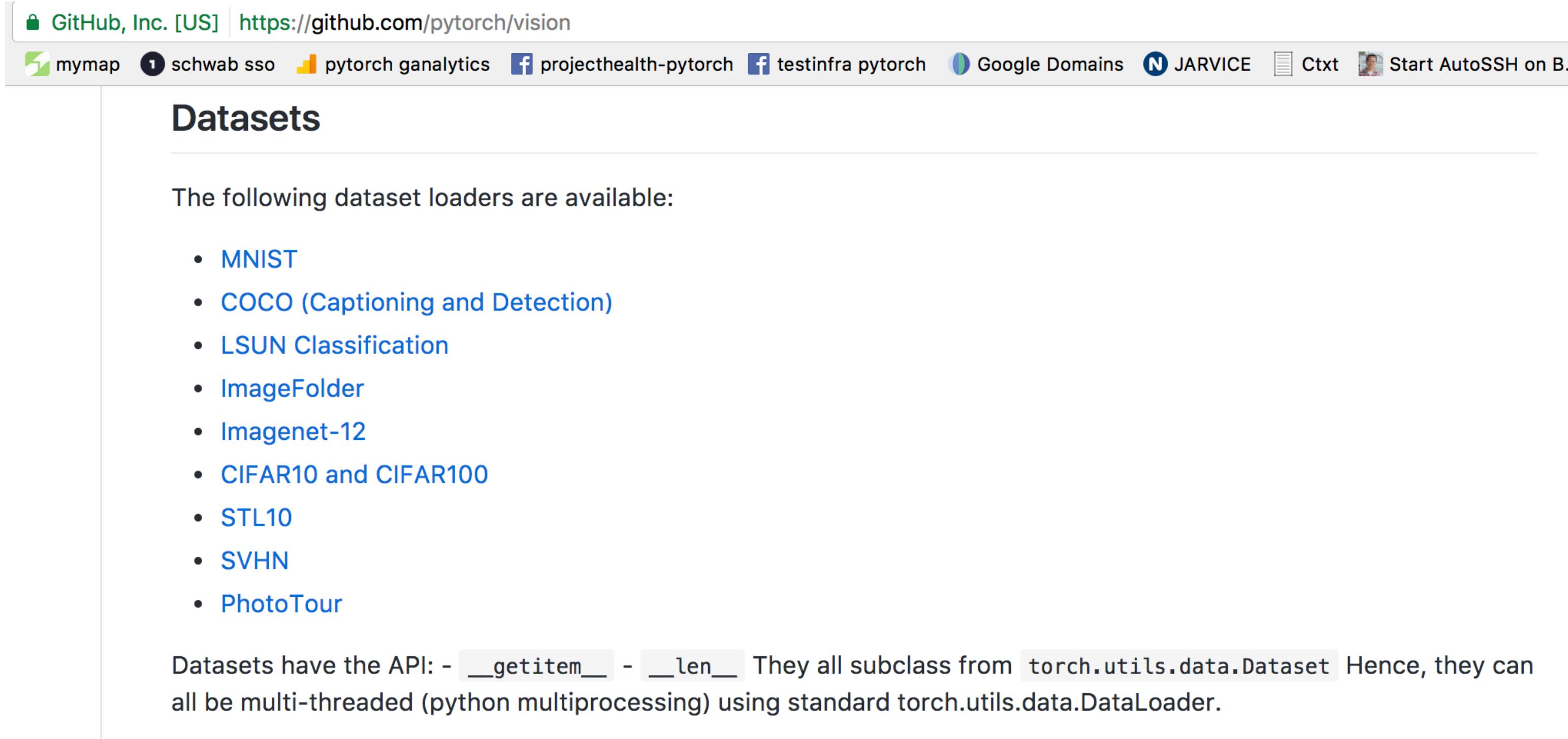
- every dataset is slightly differently formatted
- have to be preprocessed and normalized differently
- need a multithreaded Data loader to feed GPUs fast enough



Writing Data Loaders

PyTorch solution:

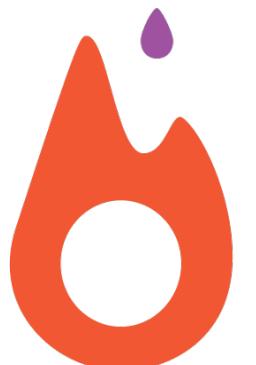
- share data loaders across the community!



The screenshot shows a GitHub page for the PyTorch Vision repository. The title is "Datasets". Below it, a list of available dataset loaders is provided:

- MNIST
- COCO (Captioning and Detection)
- LSUN Classification
- ImageFolder
- Imagenet-12
- CIFAR10 and CIFAR100
- STL10
- SVHN
- PhotoTour

At the bottom, a note states: "Datasets have the API: - `__getitem__` - `__len__`. They all subclass from `torch.utils.data.Dataset`. Hence, they can all be multi-threaded (python multiprocessing) using standard `torch.utils.data.DataLoader`".



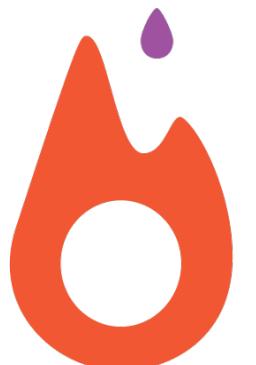
Writing Data Loaders

PyTorch solution:

- share data loaders across the community!

The screenshot shows the GitHub repository page for `pytorch/text`. At the top, it displays the repository name, URL (<https://github.com/pytorch/text>), and a lock icon indicating it's private. Below the header, there are several social sharing and monitoring links: mymap (with a location pin icon), schwab sso (with a user icon), pytorch ganalytics (with a bar chart icon), projecthealth-pytorch (with a Facebook icon), testinfra pytorch (with a Facebook icon), and Google Dom (with a globe icon). The main content area starts with the heading "This repository consists of:" followed by a bulleted list:

- `torchtext.data` : Generic data loaders, abstractions, and iterators for text
- `torchtext.datasets` : Pre-built loaders for common NLP datasets
- (maybe) `torchtext.models` : Model definitions and pre-trained models for p
situation is not the same as vision, where people can download a pretrained
make it useful for other tasks -- it might make more sense to leave NLP m



Writing Data Loaders

PyTorch solution:

- use regular Python to write Datasets:
leverage existing Python code



Writing Data Loaders

PyTorch solution:

- use regular Python to write Datasets:
leverage existing Python code

Example: ParlAI



ParlAI

ParlAI (pronounced "par-lay") is a framework for dialog AI research, implemented in Python.

Its goal is to provide researchers:

- a unified framework for training and testing dialog models
- multi-task training over many datasets at once
- seamless integration of [Amazon Mechanical Turk](#) for data collection and human evaluation

Over 20 tasks are supported in the first release, including popular datasets such as [SQuAD](#), [bAbI tasks](#), [MCTest](#), [WikiQA](#), [WebQuestions](#), [SimpleQuestions](#), [WikiMovies](#), [QACNN & QADailyMail](#), [CBT](#), [BookTest](#), [bAbI Dialog tasks](#), [Ubuntu Dialog](#), [OpenSubtitles](#), [Cornell Movie](#) and [VQA-COCO2014](#).



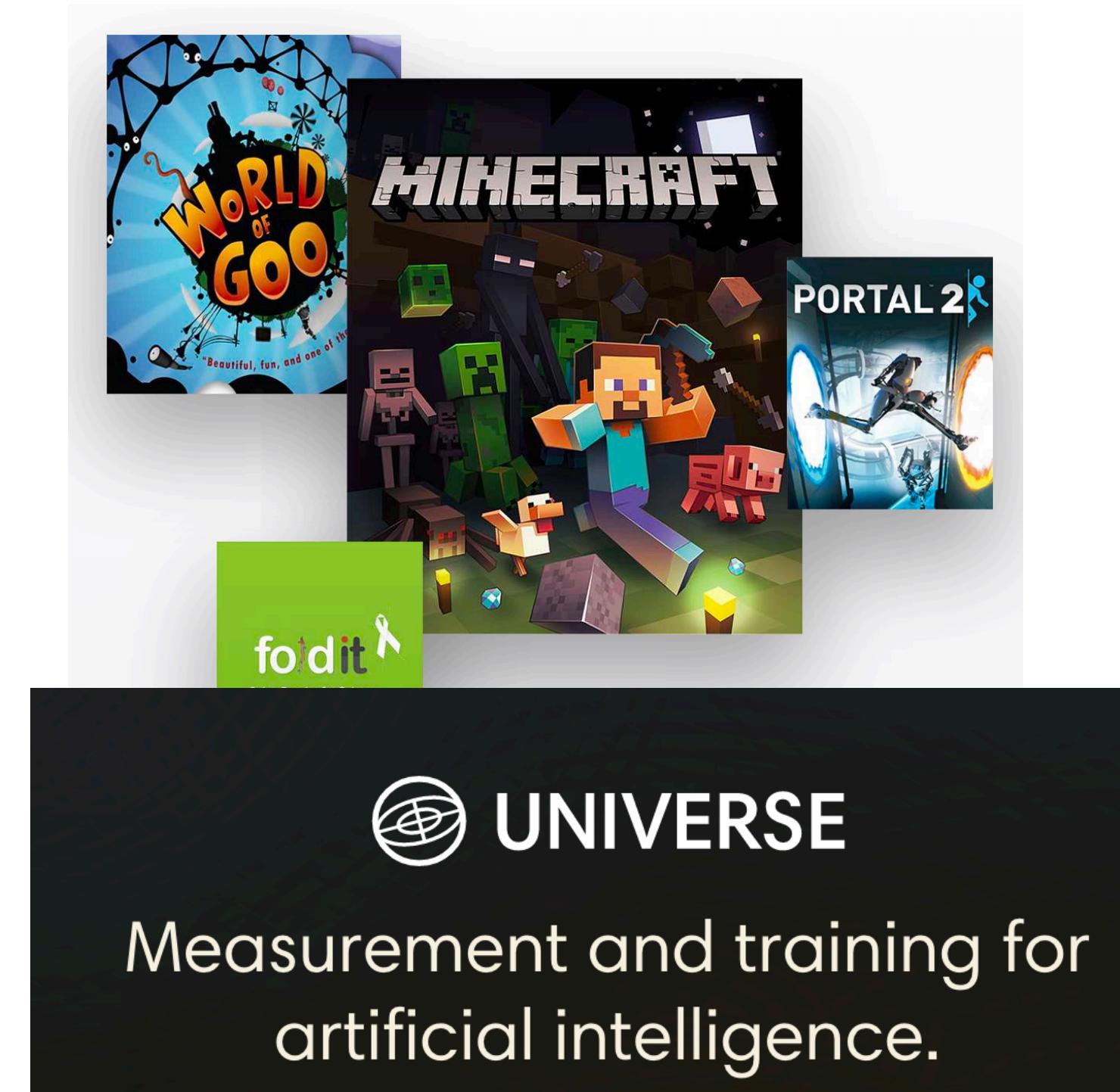
Interfacing with environments



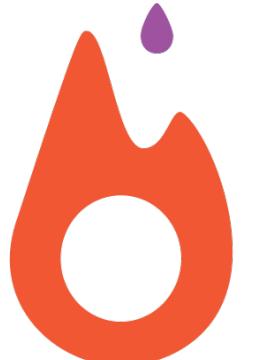
Cars



Video games



Pretty much every environment provides a Python API



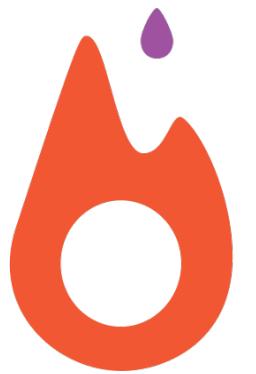
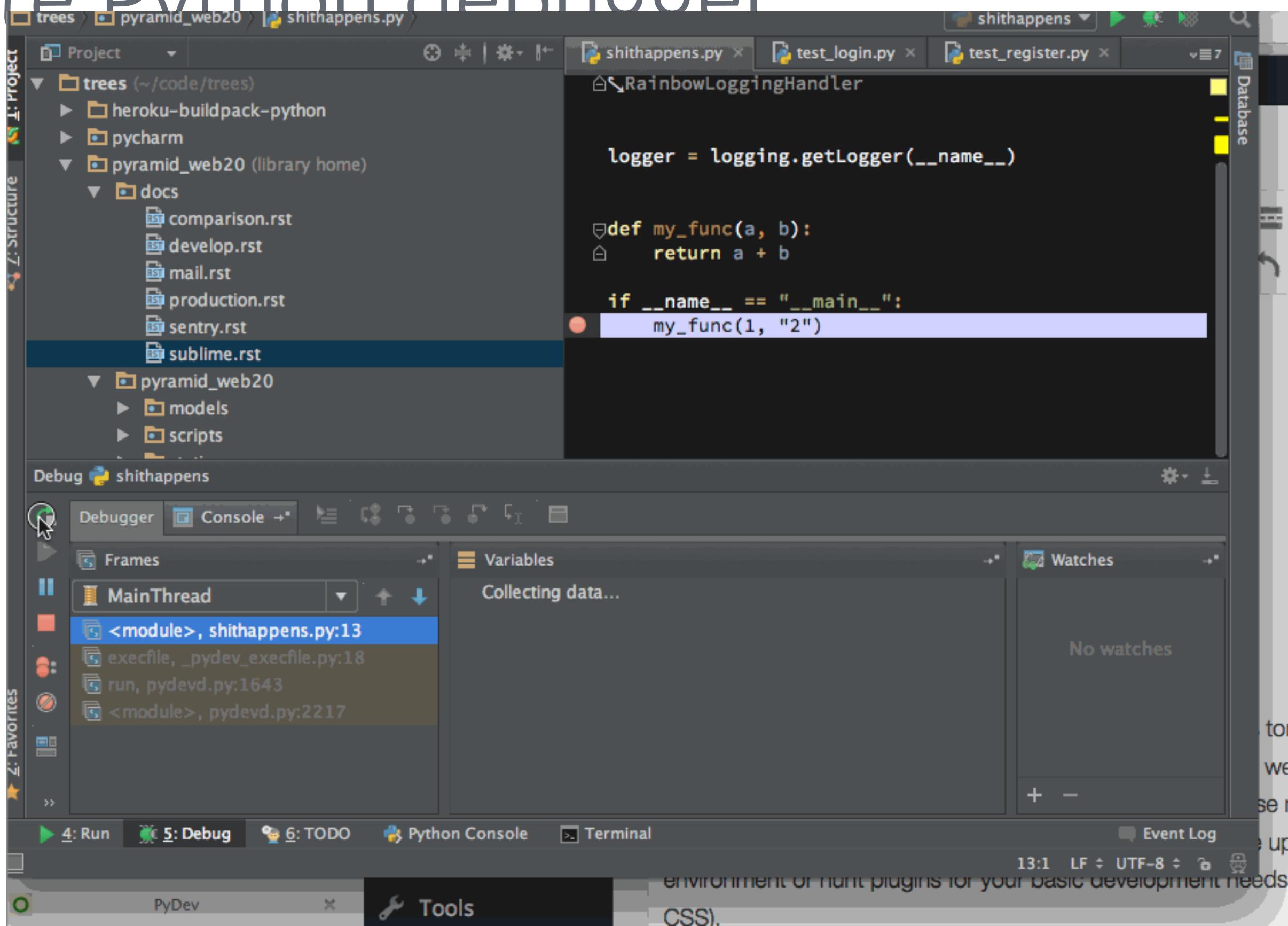
Debugging

- PyTorch is a Python extension
- Use your favorite Python debugger



Debugging

- PyTorch is a Python extension
- Use your favorite Python debugger



Ecosystem

- Use the entire Python ecosystem at your will
- Including SciPy, Scikit-Learn, etc.



Ecosystem

- Use the entire ecosystem
- Including SciPy



Brandon Amos @brandondamos · Jan 18

Why PyTorch's layer creation is powerful: Here's my layer that solves an optimization problem with a primal-dual interior point method.

```
def forward_single(input_i, Q, G, A, b, h, U_Q, U_S, R):
    nineq, nz = G.size()
    neq = A.size(0) if A.ndimension() > 0 else 0

    d = torch.ones(nineq).type_as(Q)
    nb = -b if b is not None else None
    factor_kkt(U_S, R, d, neq)
    x, s, z, y = solve_kkt(U_Q, d, G, A, U_S, input_i, torch.zeros(nineq).type_as(Q),
                           -h, nb, neq, nz)

    if torch.min(s) < 0:
        s -= torch.min(s) - 1
    if torch.min(z) < 0:
        z -= torch.min(z) - 1

    for i in range(20):
        rx = (torch.mv(A.t(),y) if neq > 0 else 0.) + \
             torch.mv(G.t(), z) + torch.mv(Q,x) + input_i
        rs = z
        ry = torch.mv(G,x) + s - h
        rz = torch.mv(A,x) - b if neq > 0 else torch.Tensor([0.])
        mu = torch.dot(s,z)/nineq
        pri_resid = torch.norm(ry) + torch.norm(rz)
        dual_resid = torch.norm(rx)
        d = z/s

        factor_kkt(U_S, R, d, neq)
        dx_aff, ds_aff, dz_aff, dy_aff = solve_kkt(U_Q, d, G, A, U_S,
                                                     rx, rs, ry, rz, neq, nz)

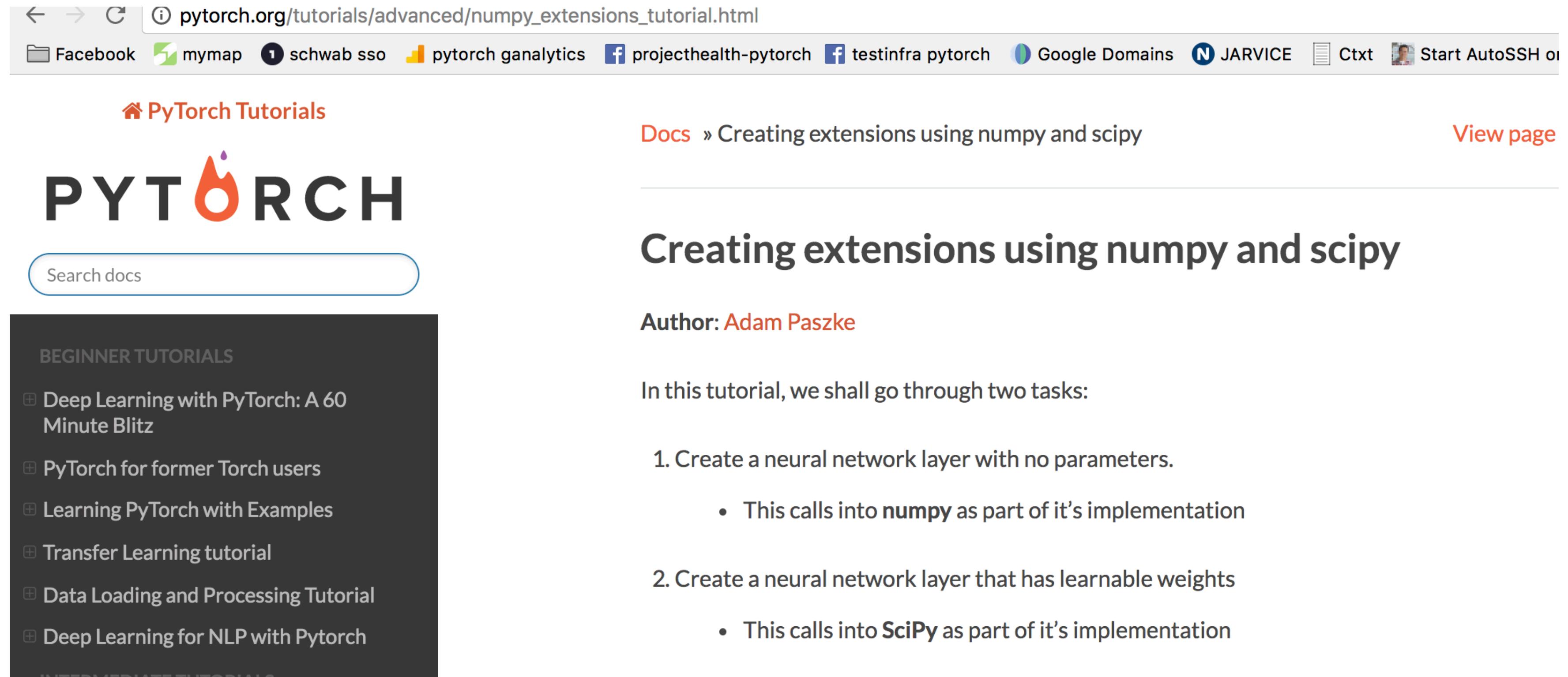
        # compute centering directions
        alpha = min(min(get_step(z,dz_aff), get_step(s, ds_aff)), 1.0)
        sig = (torch.dot(s + alpha*ds_aff, z + alpha*dz_aff)/(torch.dot(s,z)))**3
        dx_cor, ds_cor, dz_cor, dy_cor = solve_kkt(
            U_Q, d, G, A, U_S, torch.zeros(nz).type_as(Q),
            (-mu*sig*torch.ones(nineq).type_as(Q) + ds_aff*dz_aff)/s,
            torch.zeros(nineq).type_as(Q), torch.zeros(neq).type_as(Q), neq, nz)

        dx = dx_aff + dx_cor
        ds = ds_aff + ds_cor
        dz = dz_aff + dz_cor
        dy = dy_aff + dy_cor if neq > 0 else None
```



Ecosystem

- Use the entire Python ecosystem at your will
- Including SciPy, Scikit-Learn, etc.



The screenshot shows a web browser displaying the PyTorch documentation. The URL in the address bar is pytorch.org/tutorials/advanced/numpy_extensions_tutorial.html. The page title is "Creating extensions using numpy and scipy". The author is listed as "Author: Adam Paszke". The main content describes two tasks: creating a neural network layer with no parameters (calling into numpy) and creating one with learnable weights (calling into SciPy). The left sidebar contains "BEGINNER TUTORIALS" with links to "Deep Learning with PyTorch: A 60 Minute Blitz", "PyTorch for former Torch users", "Learning PyTorch with Examples", "Transfer Learning tutorial", "Data Loading and Processing Tutorial", and "Deep Learning for NLP with Pytorch".

← → ⌂ ⓘ pytorch.org/tutorials/advanced/numpy_extensions_tutorial.html

Facebook mymap schwab sso pytorch ganalytics projecthealth-pytorch testinfra pytorch Google Domains JARVICE Ctxt Start AutoSSH or

PyTorch Tutorials Docs » Creating extensions using numpy and scipy View page

PYTORCH

Search docs

BEGINNER TUTORIALS

- ⊕ Deep Learning with PyTorch: A 60 Minute Blitz
- ⊕ PyTorch for former Torch users
- ⊕ Learning PyTorch with Examples
- ⊕ Transfer Learning tutorial
- ⊕ Data Loading and Processing Tutorial
- ⊕ Deep Learning for NLP with Pytorch

INTERMEDIATE TUTORIALS

Creating extensions using numpy and scipy

Author: Adam Paszke

In this tutorial, we shall go through two tasks:

1. Create a neural network layer with no parameters.
 - This calls into **numpy** as part of it's implementation
2. Create a neural network layer that has learnable weights
 - This calls into **SciPy** as part of it's implementation



Ecosystem

- A shared model-zoo:

We provide pre-trained models for the ResNet variants and AlexNet, using the PyTorch `torch.utils.model_zoo`. These can be constructed by passing `pretrained=True`:

```
import torchvision.models as models  
resnet18 = models.resnet18(pretrained=True)  
alexnet = models.alexnet(pretrained=True)
```



Recent and Upcoming features



Distributed PyTorch

- MPI style distributed communication
- Broadcast Tensors to other nodes
- Reduce Tensors among nodes
 - for example: sum gradients among all nodes



Higher order derivatives

- `grad(grad(grad(grad(grad(torch.norm(x)))))))`



Higher order derivatives

- `grad(grad(grad(grad(grad(torch.norm(x)))))))`
- Useful to implement crazy ideas



Broadcasting and Advanced Indexing

- Numpy-style broadcasting
- Numpy-style indexing that covers advanced cases



JIT Compilation (upcoming)

- A full tracing JIT used to cache and compile subgraphs



A compiler for PyTorch



Add -> Mul

A simple use-case

```
d = (a + b) * c # , a, b, c, d are Tensors
```

```
for (i=0; i < data_length; i++) {  
    out1[i] = a[i] + b[i];  
}
```

```
for (i=0; i < data_length; i++) {  
    d[i] = out1[i] * c[i];  
}
```



Add -> Mul Fusion

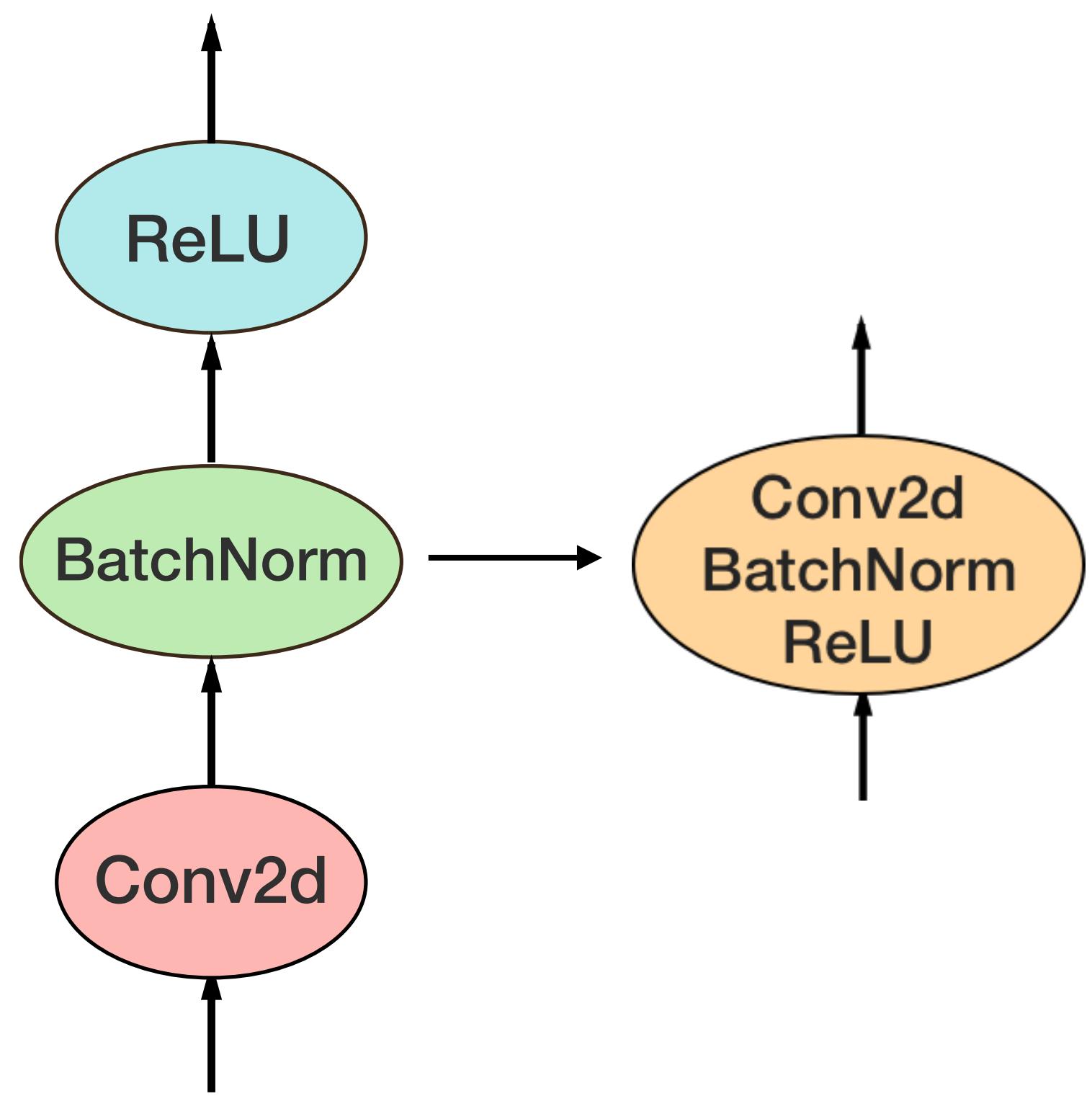
`d = (a + b) * c # , a, b, c, d are Tensors`

```
for (i=0; i < data_length; i++) {          for (i=0; i < data_length; i++) {  
    out1[i] = a[i] + b[i];                  out1[i] = a[i] + b[i];  
}  
d[i] = out1[i] * c[i];                      d[i] = out1[i] * c[i];  
}
```

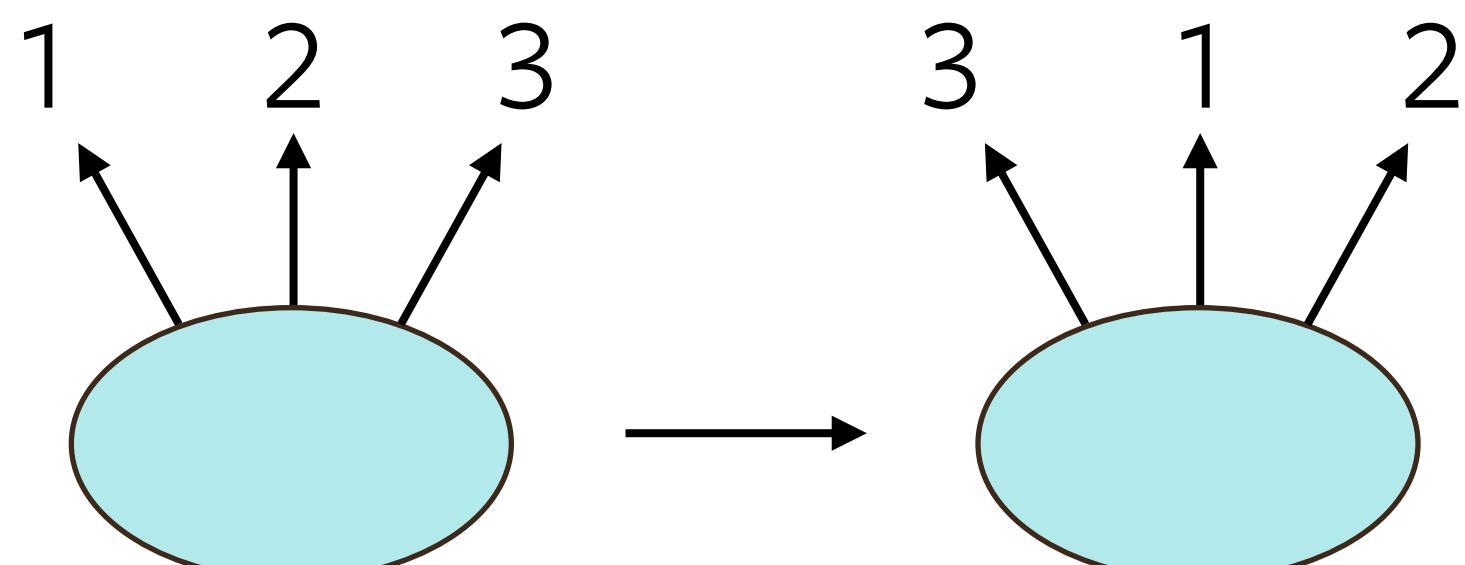


Compilation benefits

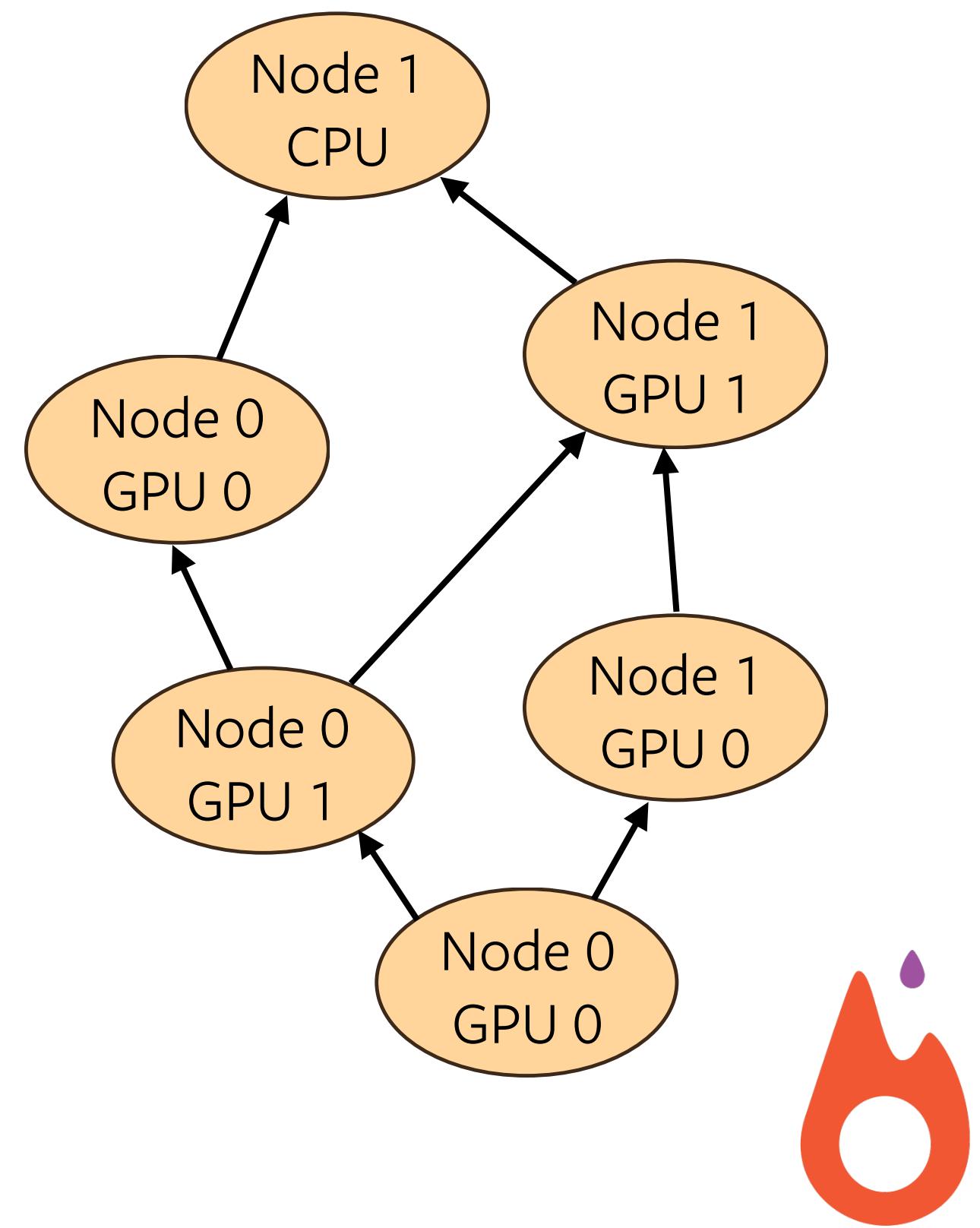
Kernel fusion



Out-of-order
execution



Automatic
work placement



Tracing JIT

```
def foo(x):  
    y = x + 1  
    z = y * 2  
    return z  
  
x = torch.Tensor([0, 1, 2, 3])  
  
y = foo(x) # 2, 4, 6, 8
```



Tracing JIT

```
@torch.jit.trace
def foo(x):
    y = x + 1
    z = y * 2
    return z

x = torch.Tensor([0, 1, 2, 3])
```

```
y1 = foo(x)
y2 = foo(x)
```



Tracing JIT

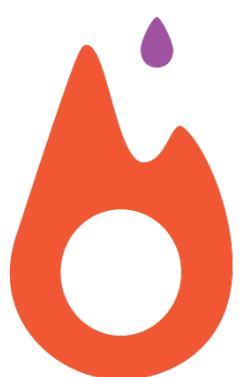
```
@torch.jit.trace
def foo(x):
    y = x + 1
    z = y * 2
    return z

x = torch.Tensor([0, 1, 2, 3])

y1 = foo(x)
y2 = foo(x)
```

Trace

Add(x, 1) -> t1



Tracing JIT

```
@torch.jit.trace  
def foo(x):  
    y = x + 1  
    z = y * 2  
    return z
```

```
x = torch.Tensor([0, 1, 2, 3])
```

```
y1 = foo(x)  
y2 = foo(x)
```

Trace

Add(x, 1) -> t1

Mul(t1, 2) -> retval



Tracing JIT

```
@torch.jit.trace
def foo(x):
    y = x + 1
    z = y * 2
    return z
```

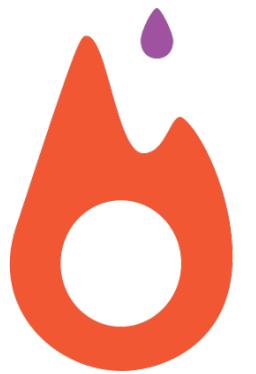
```
x = torch.Tensor([0, 1, 2, 3])
```

```
y1 = foo(x)
y2 = foo(x)
```

Trace

Add(x, 1) -> t1

Mul(t1, 2) -> retval



Tracing JIT

```
@torch.jit.trace
def foo(x):
    y = x + 1
    z = y * 2
    return z
```

```
x = torch.Tensor([0, 1, 2, 3])
```

```
y1 = foo(x)
y2 = foo(x)
```

Trace

```
Add(x, 1) -> t1
```

```
Mul(t1, 2) -> retval
```



Tracing JIT

```
@torch.jit.trace
def foo(x):
    y = x + 1
    z = y * 2
    return z
```

```
x = torch.Tensor([0, 1, 2, 3])
```

```
y1 = foo(x)
y2 = foo(x)
```

Trace

Add(x, 1) -> t1

Mul(t1, 2) -> retval



Tracing JIT

more details

- An IR with Tensors as first class types



Tracing JIT

more details

- An IR with Tensors as first class types
- optimized for reusable and cached traces
 - (unlike javascript tracers)



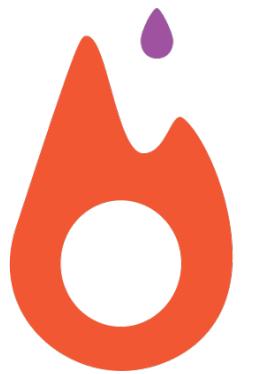
Tracing JIT

Gotcha: cannot handle
control-flow yet

```
@torch.jit.trace
def foo(x):
    sum = x.sum().data[0]
    if sum < 5:
        return x + 1
    else:
        return x + 2

x = torch.Tensor([-1, 0, 1, 2])
x2 = torch.Tensor([1, 2, 3, 4])

y = foo(x) # 0, 1, 2, 3
y2 = foo(x2) # 2, 3, 4, 5
```





<http://pytorch.org>

Released Jan 2017

275,000+ downloads

1900+ community repos

13,500+ user posts

314 contributors

facebook



NVIDIA

salesforce

ParisTech
INSTITUT DES SCIENCES ET TECHNOLOGIE
PARIS INSTITUTE OF TECHNOLOGY

Carnegie
Mellon
University

UNIVERSITE
PIERRE & MARIE CURIE
LA SCIENCE A PARIS

Digital
Reasoning

Stanford
University

UNIVERSITY OF
OXFORD

NYU

Inria

ENS
ÉCOLE NORMALE
SUPÉRIEURE

EPFL
ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Berkeley
UNIVERSITY OF CALIFORNIA

With ❤ from