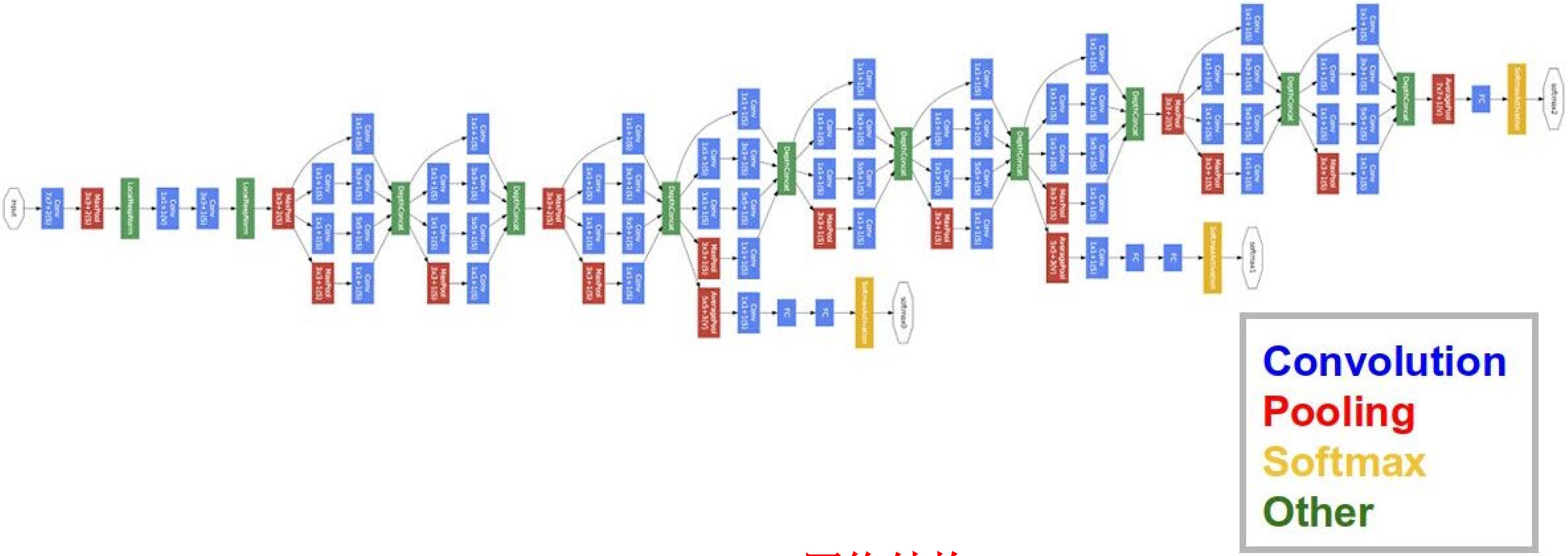
上节课回顾

- GoogLeNet (Inception module)
- ResNet (Residual network)
- 网络可视化
- Caffe软件中超参数文件solver.prototxt

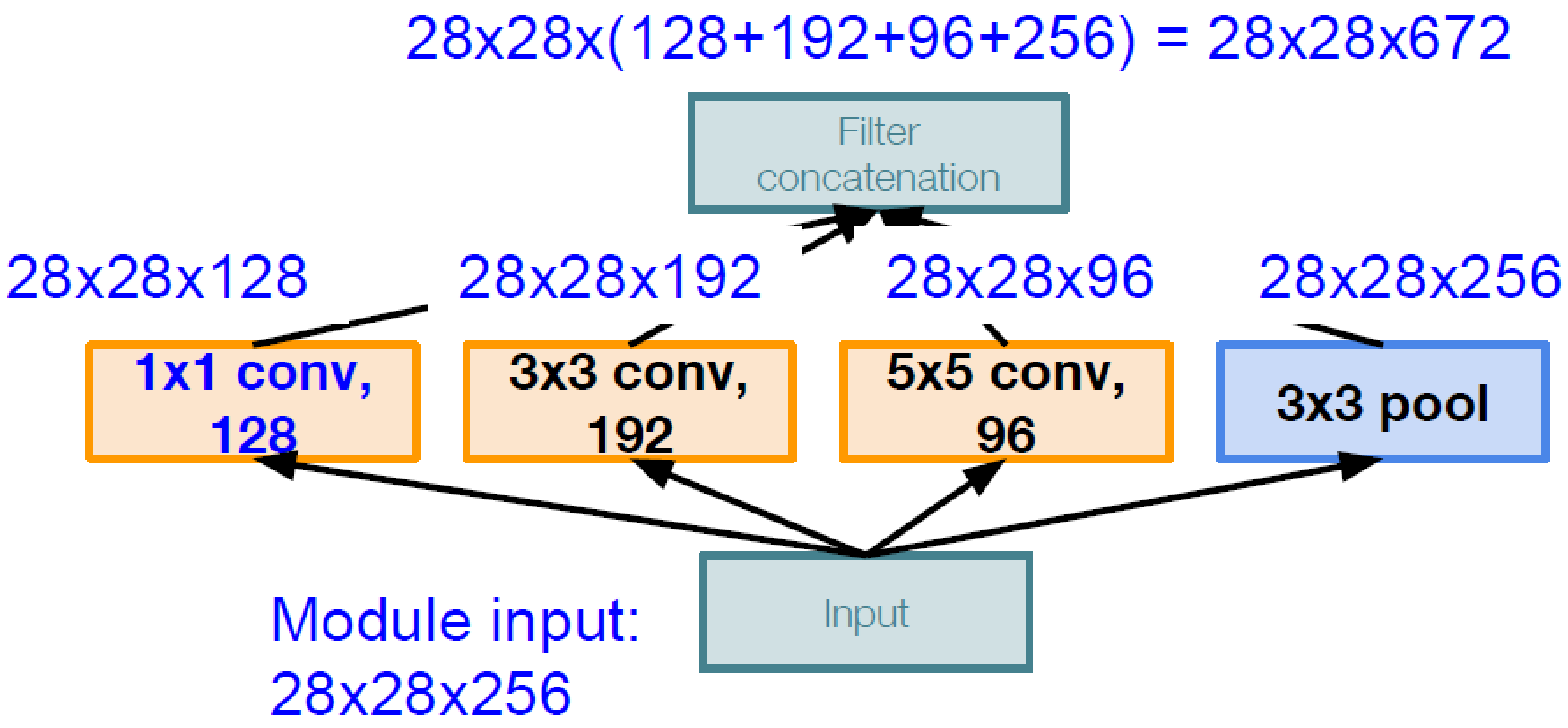
5. GoogLeNet



GoogLeNet网络结构

Szegedy C., LiuW., Jia Y., Sermanet P., Reed S., Anguelov D., Erhan D., Vanhoucke V., Rabinovich A. (2015) Going deeper with convolutions. In: CVPR.

5. GoogLeNet



卷积操作的计算量：

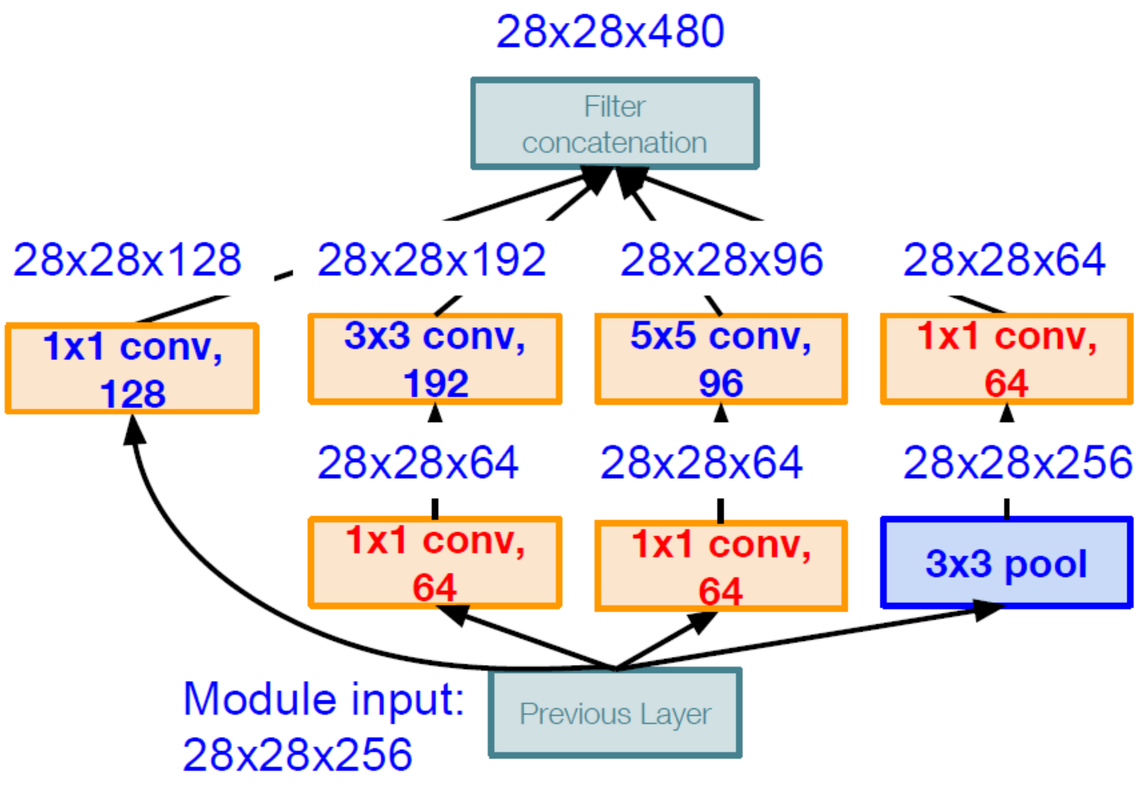
[1x1 conv, 128] 28x28x128x1x1x256
[3x3 conv, 192] 28x28x192x3x3x256
[5x5 conv, 96] 28x28x96x5x5x256

854M 次操作

Inception module 基础版

Szegedy C., LiuW., Jia Y., Sermanet P., Reed S., Anguelov D., Erhan D., Vanhoucke V., Rabinovich A. (2015) Going deeper with convolutions. In: CVPR.

5. GoogLeNet



Inception module 降维版

卷积操作的计算量：

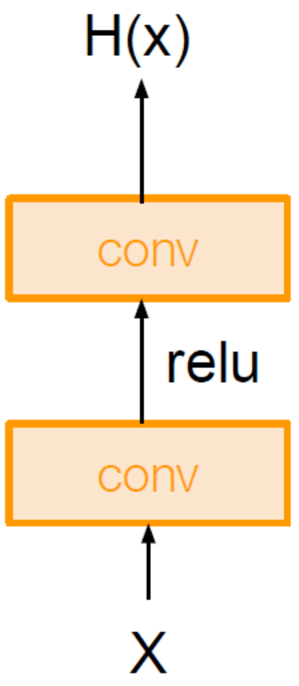
[1x1 conv, 64] 28x28x64x1x1x256
[1x1 conv, 64] 28x28x64x1x1x256
[1x1 conv, 128] 28x28x128x1x1x256
[3x3 conv, 192] 28x28x192x3x3x64
[5x5 conv, 96] 28x28x96x5x5x64
[1x1 conv, 64] 28x28x64x1x1x256

358M 次操作

Szegedy C., LiuW., Jia Y., Sermanet P., Reed S., Anguelov D., Erhan D., Vanhoucke V., Rabinovich A. (2015) Going deeper with convolutions. In: CVPR.

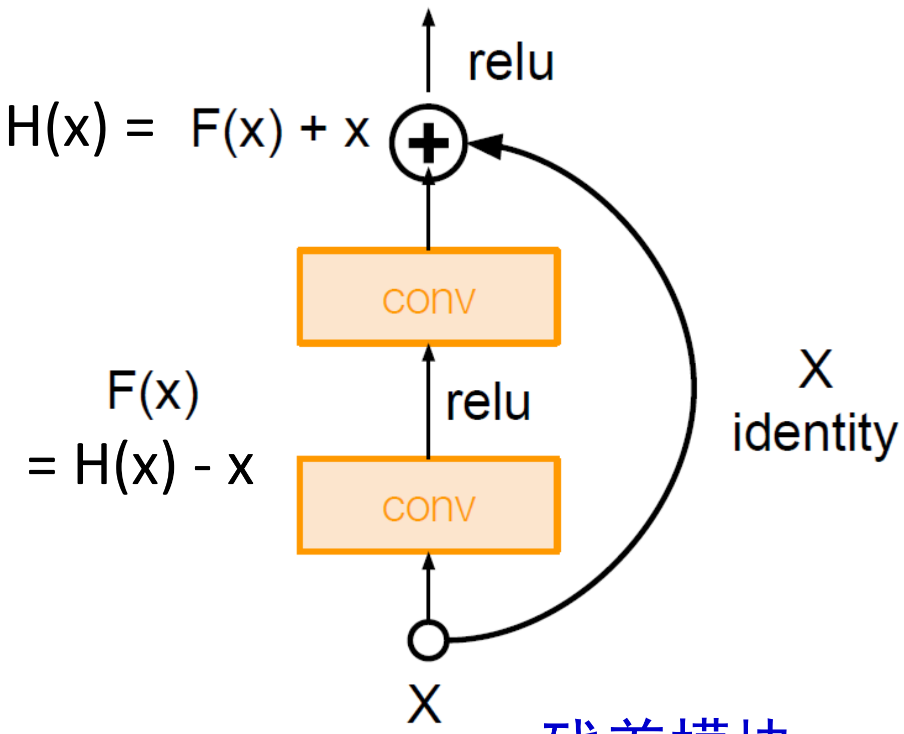
6. ResNet（Residual Network）

为了解决深度增加出现的梯度消失，提出残差网络**ResNet**



基础模块

使用两层卷积网络
来拟合残差H(x)-x,
用以代替直接拟合
H(x)的基础版



残差模块

He K., Zhang X., Ren S., Sun J. (2016) Deep residual learning for image recognition. In: CVPR.

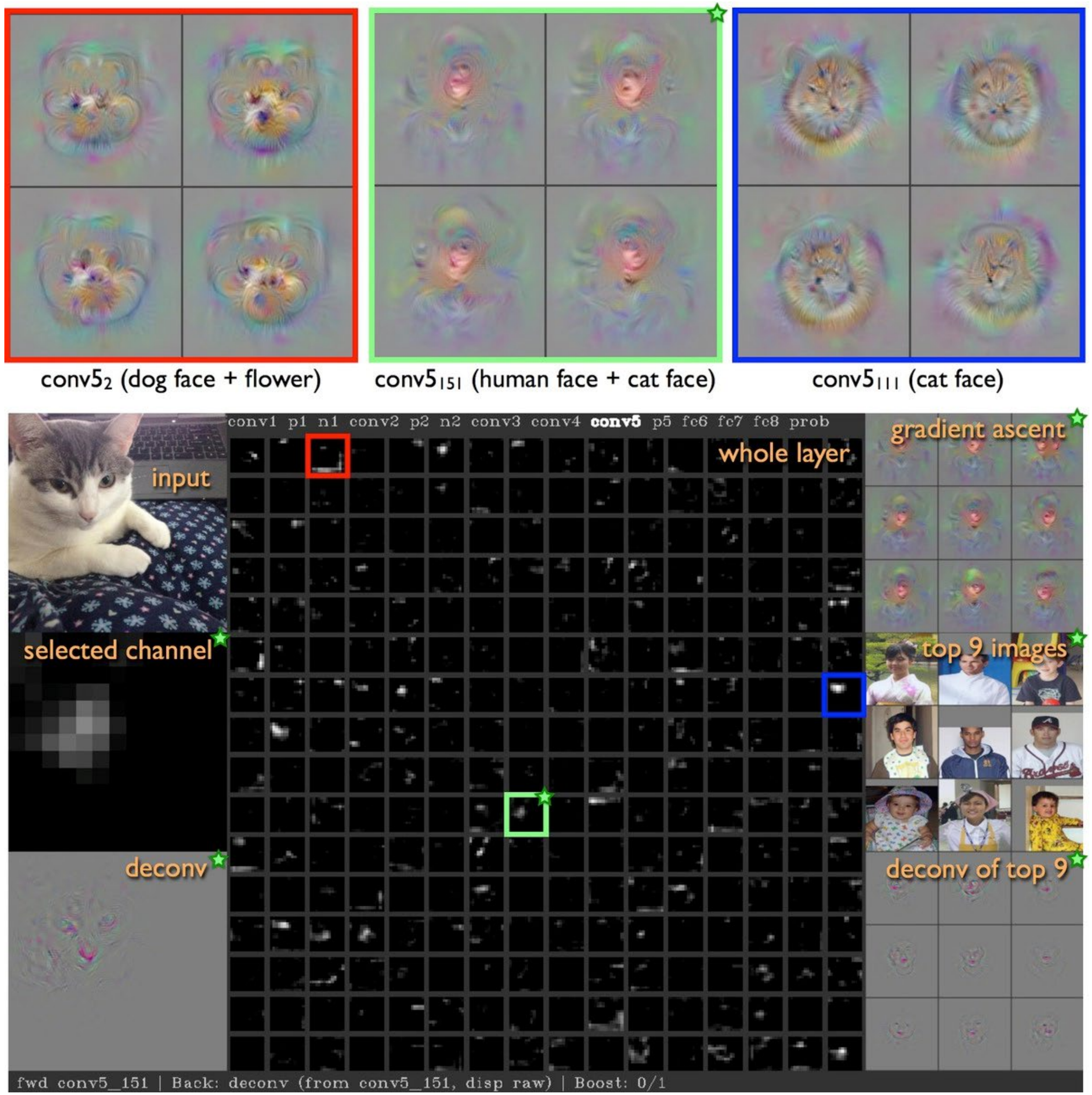
卷积网络可视化

基于Caffe实现可视化

http://yosinski.com/deepvis

40行Python代码实现可视化

https://towardsdatascience.com/how-to-visualize-convolutional-features-in-40-lines-of-code-70b7d87b0030



Caffe网络的超参数solver.prototxt

```
net: "examples/mnist/lenet_train_test.prototxt“
test_iter: 100
test_interval: 500
base_lr: 0.01
momentum: 0.9
weight_decay: 0.0005
lr_policy: "inv“
gamma: 0.0001
power: 0.75
display: 100
max_iter: 10000
snapshot: 5000
snapshot_prefix: "examples/mnist/lenet“
solver_mode: GPU
```

Caffe网络架构的定义train_val.prototxt

```
name: "AlexNet"
layer {
  name: "data"
  type: "Data"
  top: "data"
  top: "label"
  include {
    phase: TRAIN
  }
  transform_param {
    mirror: true
    crop_size: 227
    mean_file: "data/ilsvrc12/imagenet_mean.binaryproto"
  }
  data_param {
    source: "examples/imagenet/ilsvrc12_train_lmdb"
    batch_size: 256
    backend: LMDB
```

```
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 96
    kernel_size: 11
    stride: 4
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
```

https://ethereon.github.io/netscope/#/editor

https://github.com/BVLC/caffe/blob/master/models/bvlc_alexnet/train_val.prototxt

Caffe训练网络

Caffe支持三种接口：命令行，python，matlab

1. 命令行：> caffe train -solver lenet_solver.prototxt

2. Python接口：

```
import sys
sys.path.insert(0,'/path/to/caffe/python')
import caffe
caffe.set_device(1)
caffe.set_mode_gpu()
solver=caffe.SGDSolver('/path/to/solver.prototxt')
solver.net.copy_from('pretrained.caffemodel')
solver.solve()
```

solver=caffe.SGDSolver('/path/to/solver.prototxt')
solver.net.copy_from('pretrained.caffemodel')
solver.solve()

3. Matlab接口：

```
clear; clc; close all;
addpath('/home/cwang/caffe/matlab/');
opt = config();
caffe.reset_all();
caffe.set_mode_gpu();
caffe.set_device(0);
solver = caffe.Solver(opt.solver_prototxt);
train_dataset = load(opt.train_data_filename);
valid_dataset = load(opt.valid_data_filename);

do_train(train_dataset, valid_dataset, solver, opt);
```

solver = caffe.Solver(opt.solver_prototxt);

Caffe使用流程

1. 数据准备以及格式转换（现成脚本程序）

2. 定义网络文件（train_val.prototxt）

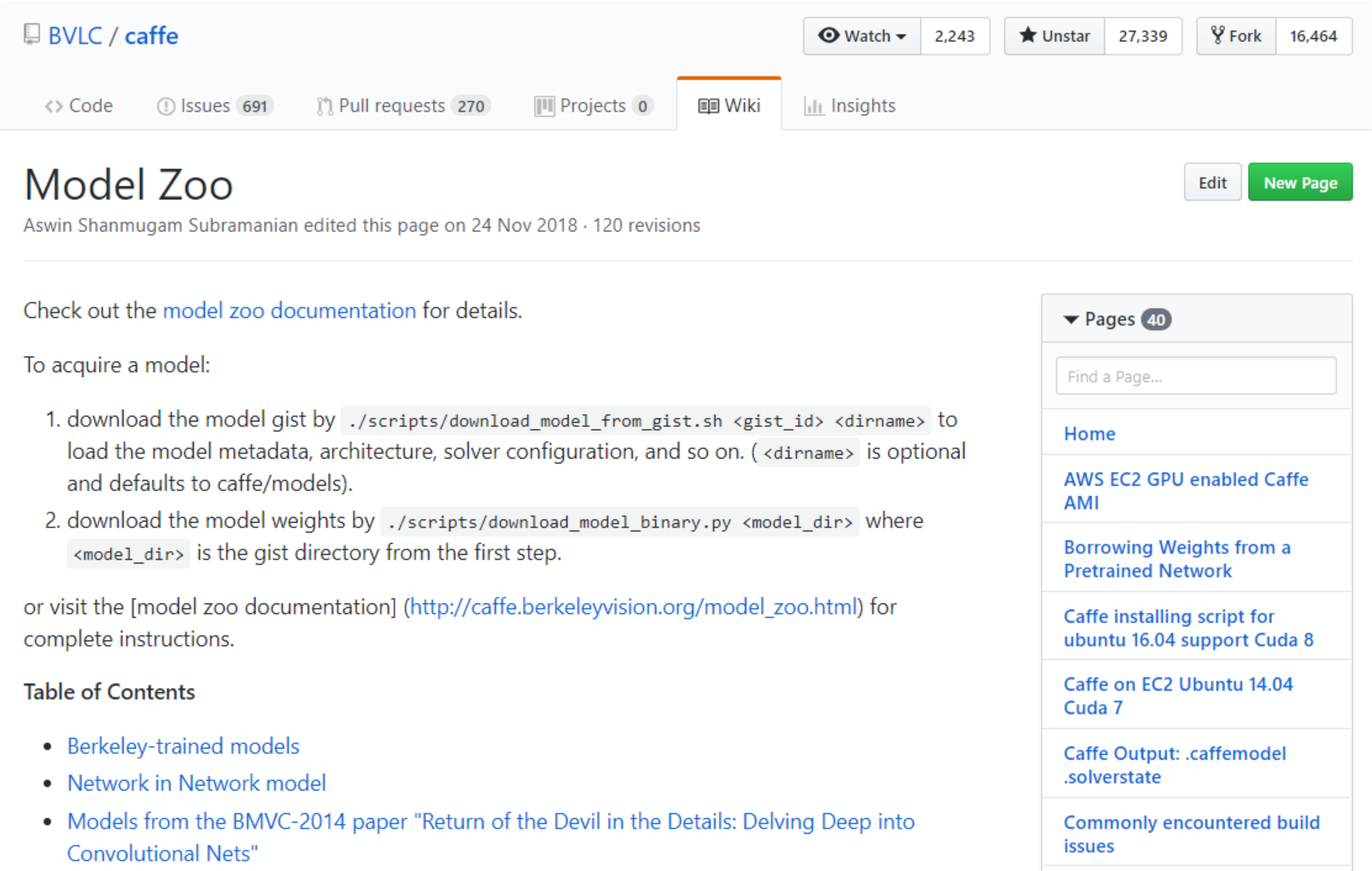
3. 定义训练中的超参数（solver.prototxt）

4. 训练模型（三种方式运行）

Caffe Model Zoo

现成已经学习好的模型参数，包括：
AlexNet，VGG，
GoogLeNet，ResNet

https://github.com/BVLC/caffe/wiki/Model-Zoo



BVLC / caffe

Watch 2,243 | Unstar 27,339 | Fork 16,464

Code | Issues 691 | Pull requests 270 | Projects 0 | Wiki | Insights

Model Zoo

Edit | New Page

Aswin Shanmugam Subramanian edited this page on 24 Nov 2018 · 120 revisions

Check out the model zoo documentation for details.

To acquire a model:

1. download the model gist by `./scripts/download_model_from_gist.sh <gist_id> <dirname>` to load the model metadata, architecture, solver configuration, and so on. (`<dirname>` is optional and defaults to caffe/models).
2. download the model weights by `./scripts/download_model_binary.py <model_dir>` where `<model_dir>` is the gist directory from the first step.

or visit the [model zoo documentation] (http://caffe.berkeleyvision.org/model_zoo.html) for complete instructions.

Table of Contents

- Berkeley-trained models
- Network in Network model
- Models from the BMVC-2014 paper "Return of the Devil in the Details: Delving Deep into Convolutional Nets"

Pages 40

Find a Page...

- Home
- AWS EC2 GPU enabled Caffe AMI
- Borrowing Weights from a Pretrained Network
- Caffe installing script for ubuntu 16.04 support Cuda 8
- Caffe on EC2 Ubuntu 14.04 Cuda 7
- Caffe Output: .caffemodel .solverstate
- Commonly encountered build issues

TensorFlow

开发者：Google Brain

初始版本：2015年11月9日

官网：https://tensorflow.google.cn/

Github：https://github.com/tensorflow/tensorflow

TensorFlow

一. 简介

二. Tensor

三. 图和会话

四. 实例

一、简介

- TensorFlow是一个基于数据流编程（dataflow programming）的符号数学系统，被广泛应用的机器学习领域，支持GPU和TPU加速。

- 截至版本1.12.0，绑定完成并支持版本兼容运行的语言为C和Python，其它（试验性）绑定完成的语言为JavaScript、C++、Java、Go和Swift，依然处于开发阶段的包括C#、Haskell、Julia、Ruby、Rust和Scala。

- 支持平台，Linux、MacOS、Windows、Raspbian

二、Tensor：张量

- 正如名称所示，TensorFlow 这一框架定义和运行涉及张量的计算。张量是对矢量和矩阵向潜在的更高维度的泛化。TensorFlow 在内部将张量表示为基本数据类型的 n 维数组。

- 在使用TensorFlow编程时，操作和传递的主要对象是 tf.Tensor。

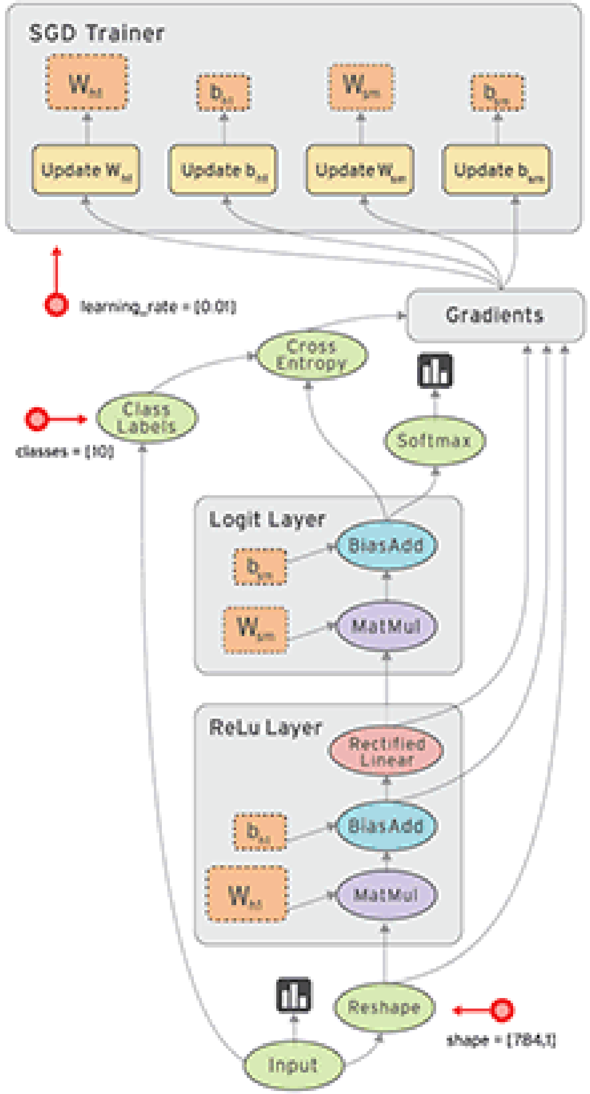
- 某些类型的张量是特殊的。主要的特殊张量有：tf.Variable，tf.constant，tf.placeholder，tf.SparseTensor，除tf.Variable外，其他张量的值均为不可变的。

```
>>> import tensorflow as tf
>>> mystr = tf.Variable(["Hello"], tf.string)
>>> cool_numbers  = tf.Variable([3.14159, 2.71828], tf.float32)
>>> first_primes = tf.Variable([2, 3, 5, 7, 11], tf.int32)
>>> its_very_complicated = tf.Variable([12.3 - 4.85j, 7.5 - 6.23j], tf.complex64)
>>>
```

三、图和会话

- TensorFlow 程序可看成两个互相独立的部分组成
  1. 构建计算图（tf.Graph）
  2. 运行计算图（tf.Session）

- 计算图由两种类型的对象组成
  1. 操作(简称‘op’)：图的节点。描述了对张量的计算。
  2. 张量：图的边。它们代表将流经图的值。大多数 TensorFlow 函数会返回 tf.Tensors。

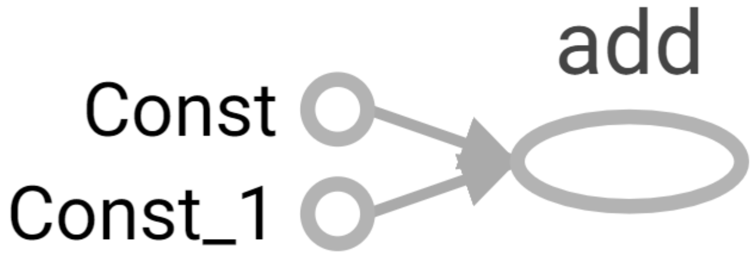


图实例与会话

- 使用tensorflow定义了一个非常简单的计算图，构建两个常量，并将它们相加。

```
>>> import tensorflow as tf
>>> a = tf.constant(3.0, dtype=tf.float32)
>>> b = tf.constant(4.0, dtype=tf.float32)
>>> total=a+b
```

- 可以使用TensorBoard进行可视化。



- 实例化一个tf.Session来运行计算图。

- 构建一个tf.Session会话，并调用run方法来计算total张量。

```
>>> sess = tf.Session()
>>> print(sess.run(total))
7.0
```

构建一个tf.Session对象

四、实例

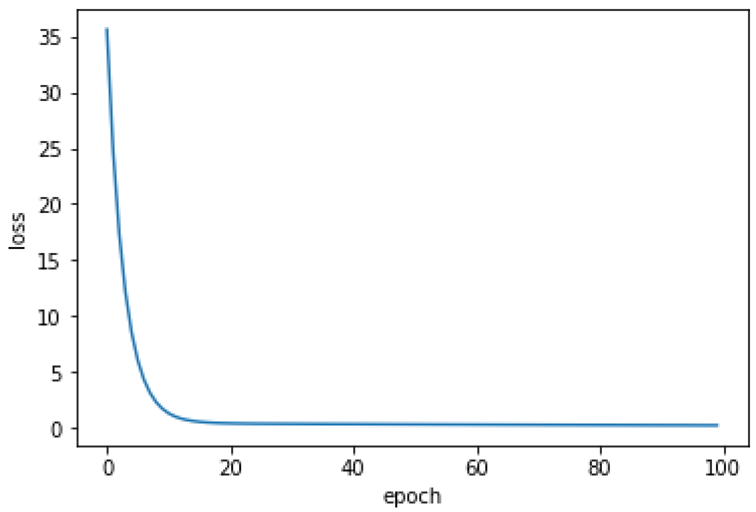
- 使用TensorFlow实现一个简单的回归模型
- 假如有一批输入值1，2，3，4；它们对应的输出值是0，-1，-2，-3。
- 构建一个简单的线性模型，该模型对应不同的输入值，给出相应输出的预测值。

代码分析

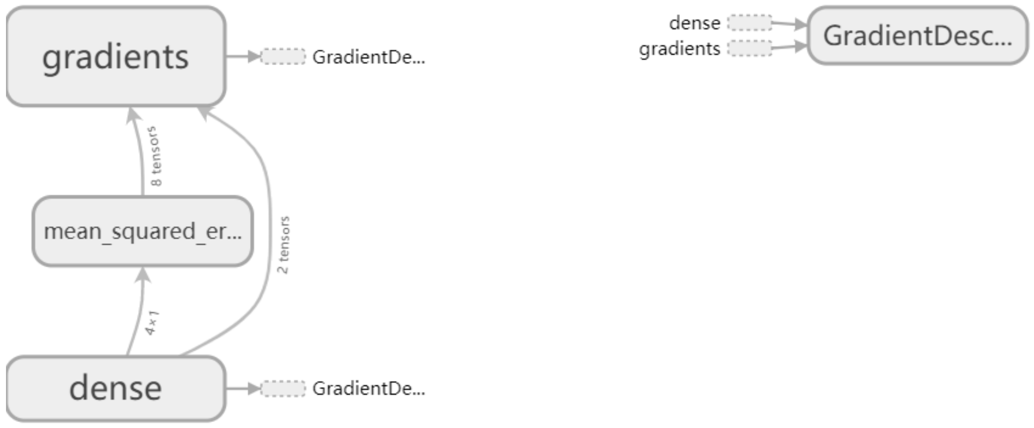
构建线性模型 $y = Wx$+b。神经元结点数为1，无激活函数。

```python
import tensorflow as tf

#定义数据
x = tf.constant([[1], [2], [3], [4]], dtype=tf.float32)
y_true = tf.constant([[0], [-1], [-2], [-3]], dtype=tf.float32)

#定义模型
linear_model = tf.layers.Dense(units=1)
y_pred = linear_model(x)

#定义损失函数，这里使用均方误差
loss = tf.losses.mean_squared_error(labels=y_true, predictions=y_pred)

#构建优化器。这里使用随机梯度下降.
optimizer = tf.train.GradientDescentOptimizer(0.01)
train = optimizer.minimize(loss)

init = tf.global_variables_initializer()

#训练。优化器来执行标准的优化算法，梯度下降
sess = tf.Session()
sess.run(init)
for i in range(100):
  _, loss_value = sess.run((train, loss))
  print(loss_value)

#输出预测值
print(sess.run(y_pred))
```

- 训练曲线，可以看出训练过程中的loss值是趋于收敛的。



- 使用TensorBoard可视化的模型图，可以看出这个回归模型的计算图结构。



PYTORCH

开发者：Facebook Artificial Intelligence Research Group （FAIR）

网站： https://pytorch.org/

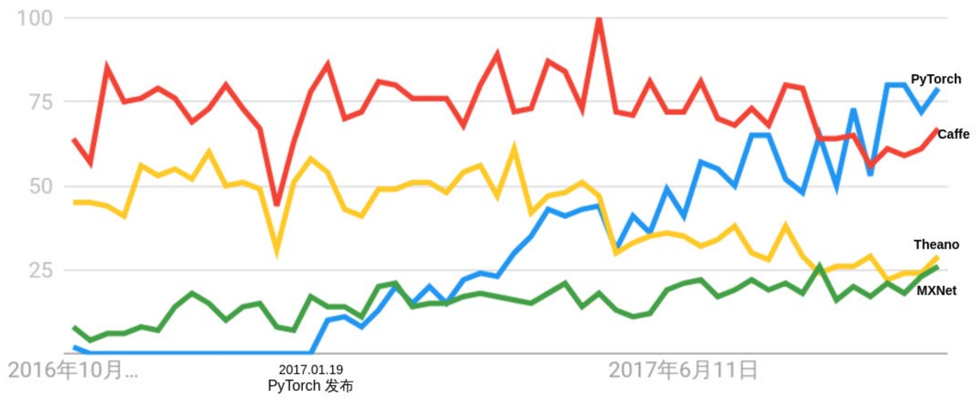
GitHub：https://github.com/pytorch/pytorch

目录

一、简介

- PyTorch的历史可追溯到2002年就诞生于纽约大学的Torch。Torch使用了一种不是很大众的语言Lua作为接口。

- 2017年1月，Facebook人工智能研究院（FAIR）团队在GitHub上开源了PyTorch，并迅速占领GitHub热度榜榜首。



为什么选择PyTorch？

- 简洁：PyTorch的设计遵循tensor→variable(autograd)→nn.Module 三个由低到高的抽象层次，分别代表高维数组（张量）、自动求导（变量）和神经网络（层/模块），而且这三个抽象之间联系紧密，可以同时进行修改和操作。

- 速度：框架的运行速度和程序员的编码水平有极大关系，但同样的算法，使用PyTorch实现的那个更有可能快过用其他框架实现的。

- 易用：PyTorch的设计最符合人们的思维，它让用户尽可能地专注于实现自己的想法，即所思即所得，不需要考虑太多关于框架本身的束缚。

- 活跃的社区：PyTorch提供了完整的文档，循序渐进的指南，作者亲自维护的论坛供用户交流和求教问题。

二、PyTorch Tensor

- Tensor是一种包含单一数据类型元素的多维矩阵。类似于numpy中的ndarrays，但是Tensor可以使用GPU进行加速。

- Pytorch定义了七种CPU Tensor类型和八种GPU Tensor类型。

- 一般情况下，使用torch.Tensor，是默认的Tensor类型torch.FloatTensor

三、自动求导机制

- PyTorch中每个变量都有两个标志：requires_grad和volatile。
- 当设置requires_grad=True时，表示该变量需要梯度。当输入中的某个变量需要梯度时，其输出也需要梯度；只有当输入的所有变量都不需要梯度时，输出才不需要梯度，这个时候子图中反向传播不会执行。
- reguires_grad标志非常有用，当我们只需要使用预训练好的模型进行微调时，想要冻结模型中的某部分参数，只需要将这些参数reguires_grad=False就行了。
- 有时不需要反向传播，比如对模型进行测试的时候，这时只需要将volatile设置为True就可以了。它将使用绝对最小的内存来评估模型，此时require_grad =False。

四、常用的PyTorch包

Package	作用
torch.Tensor	定义张量
torch.nn	其中包括许多类，torch.nn.Parameter被用于模块参数，torch.nn.Module是所有网络模型的基类
torch.nn.functional	函数库，包含卷积函数、池化函数等诸多函数
torch.autograd	提供了类和函数用来对任意标量函数进行求导
torch.optim	实现了各种优化算法的库。大部分常用的方法得到支持，并且接口具备足够的通用性，使得未来能够集成更加复杂的方法。
torch.utils.data	包含诸多处理数据集的库

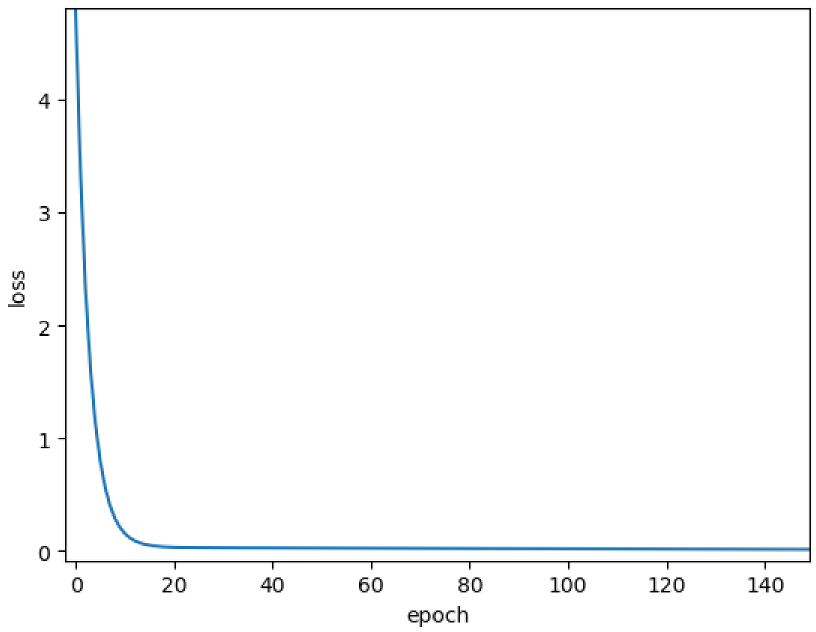
五、实例

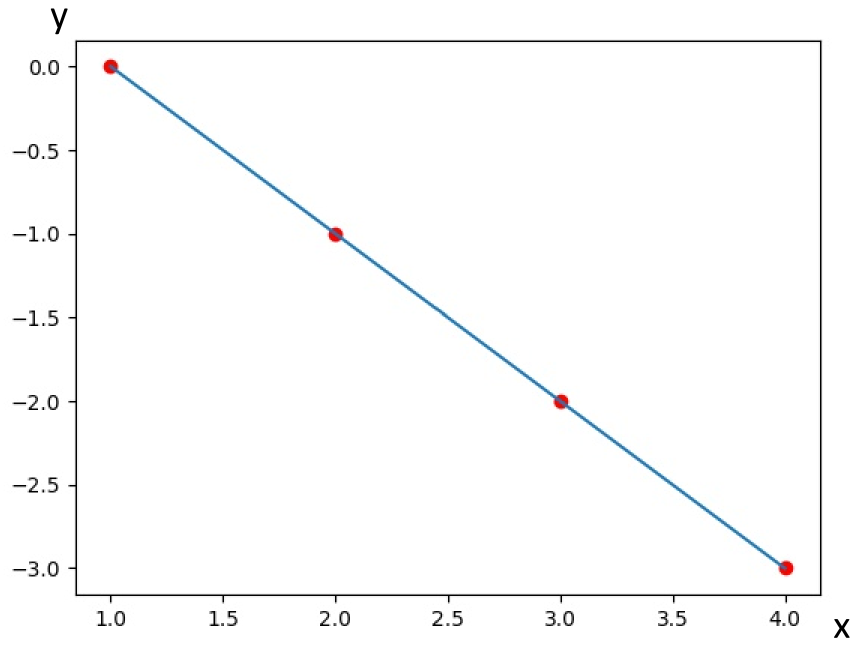
- 实现一个简单的基于PyTorch的线性回归
- 假如有一批输入值1，2，3，4；它们对应的输出值是0，-1，-2，-3。
- 构建一个简单的线性模型，该模型对应不同的输入值，给出相应输出的预测值。

代码分析

扩展nn

```python
import torch
import torch.optim as optim
import matplotlib.pyplot as plt

#定义数据，tf.Tensor
x=torch.Tensor([[1], [2], [3], [4]])
y=torch.Tensor([[0], [-1], [-2], [-3]])

#定义模型，扩展nn，线性回归
class LinerRegress(torch.nn.Module):
    def __init__(self):
        super(LinerRegress, self).__init__()
        self.fc1 = torch.nn.Linear(1, 1)

    def forward(self, x):
        return self.fc1(x)


net = LinerRegress()

#定义损失函数，使用均方误差
loss_func = torch.nn.MSELoss()

#使用随机梯度下降优化器
optimzer = optim.SGD(net.parameters(),lr=0.01)

#训练
for i in range(40000):
    optimzer.zero_grad()

    out = net(x)
    loss = loss_func(out, y)
    loss.backward()

    optimzer.step()
```





图中展示的是前140个epoch的训练曲线，可以看出loss是趋于收敛的

图中红色点为真实值，蓝色的线为拟合的线

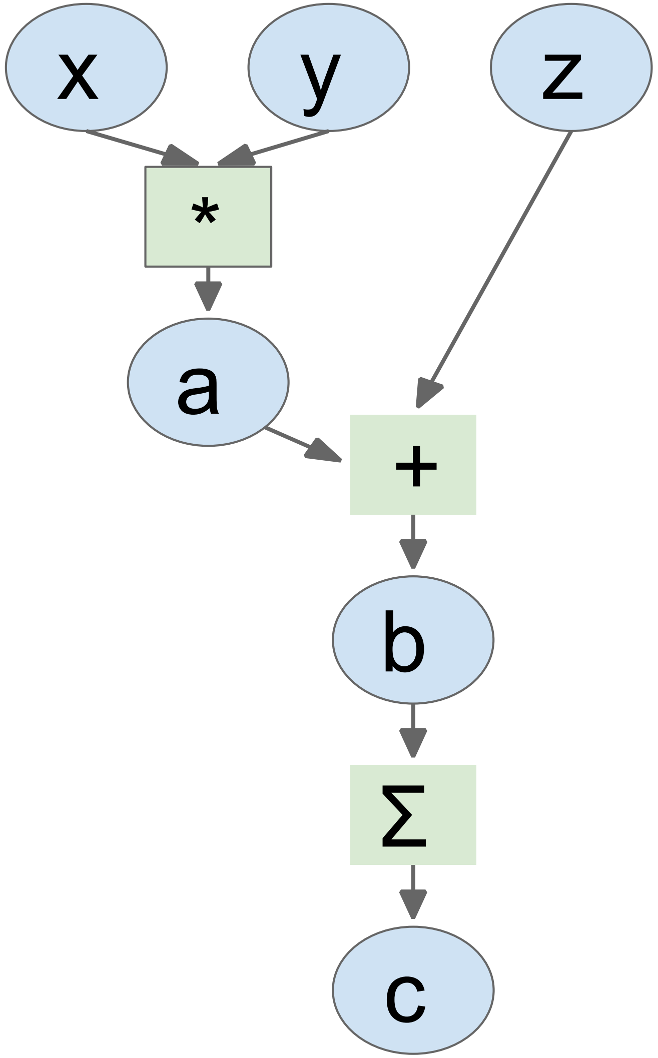
Numpy, TensorFlow, Pytorch

Numpy

计算图

```
import numpy as np
np.random.seed(0)

N, D = 3, 4

x = np.random.randn(N,D)
y = np.random.randn(N,D)
z = np.random.randn(N,D)

a = x * y
b = a + z
c = np.sum(b)
```

```
grad_c = 1.0
grad_b = grad_c * np.ones((N, D))
grad_a = grad_b.copy()
grad_z = grad_b.copy()
grad_x = grad_a * y
grad_y = grad_a * x
```



问题：

- **不能在GPU上运行**
- **必须自己计算梯度**

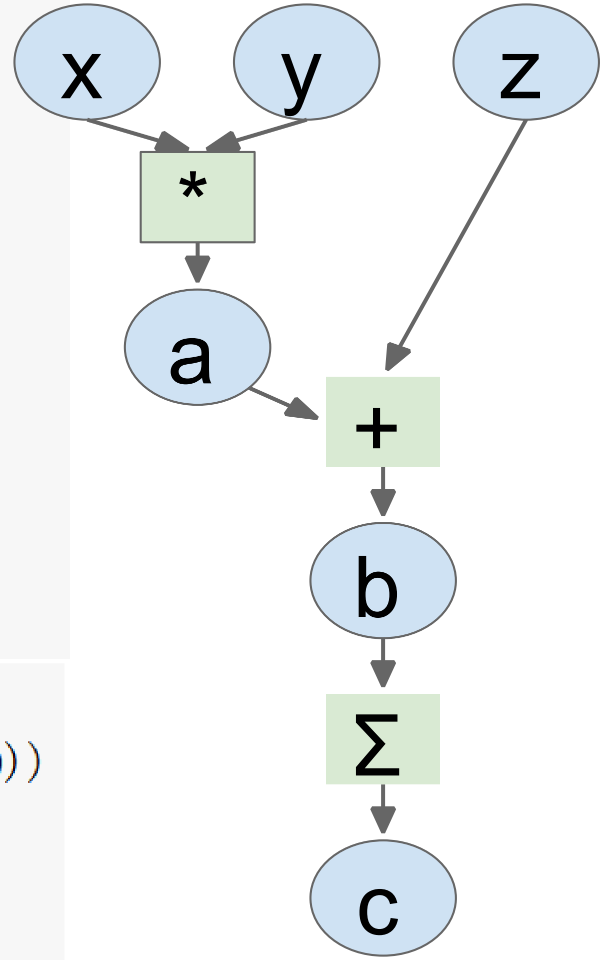
Numpy

计算图

TensorFlow

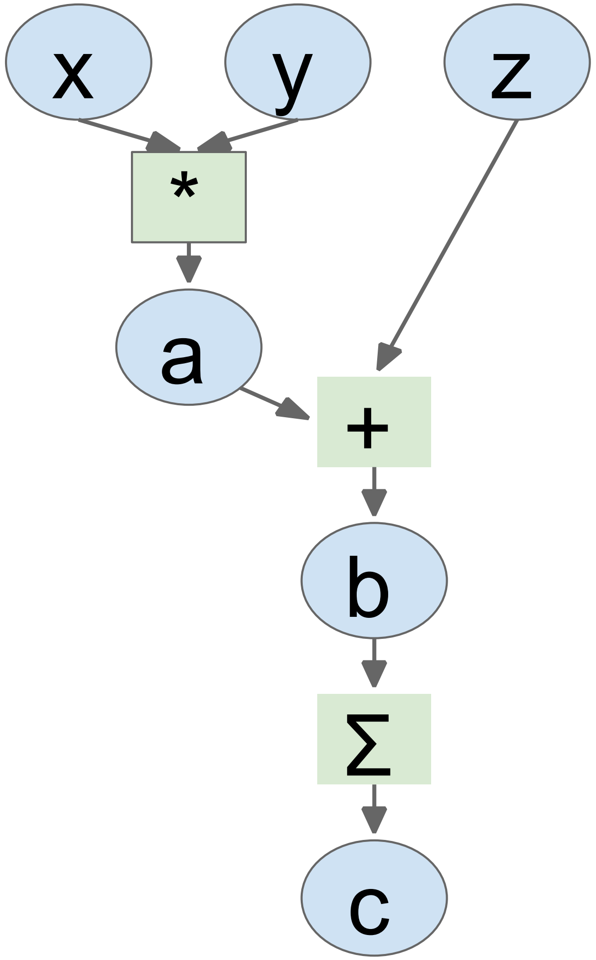
```python
import numpy as np
np.random.seed(0)

N, D = 3, 4

x = np.random.randn(N,D)
y = np.random.randn(N,D)
z = np.random.randn(N,D)

a = x * y
b = a + z
c = np.sum(b)
```

```python
grad_c = 1.0
grad_b = grad_c * np.ones((N, D))
grad_a = grad_b.copy()
grad_z = grad_b.copy()
grad_x = grad_a * y
grad_y = grad_a * x
```




```python
# Basic computational graph
import numpy as np
np.random.seed(0)
import tensorflow as tf

N, D = 3, 4

x = tf.placeholder(tf.float32)
y = tf.placeholder(tf.float32)
z = tf.placeholder(tf.float32)

a = x * y
b = a + z
c = tf.reduce_sum(b)

grad_x, grad_y, grad_z = tf.gradients(c, [x, y, z])

with tf.Session() as sess:
    values = {
        x: np.random.randn(N, D),
        y: np.random.randn(N, D),
        z: np.random.randn(N, D),
    }
    out = sess.run([c, grad_x, grad_y,grad_z],
                   feed_dict=values)
    c_val, grad_x_val, grad_y_val, grad_z_val = out
```

计算图



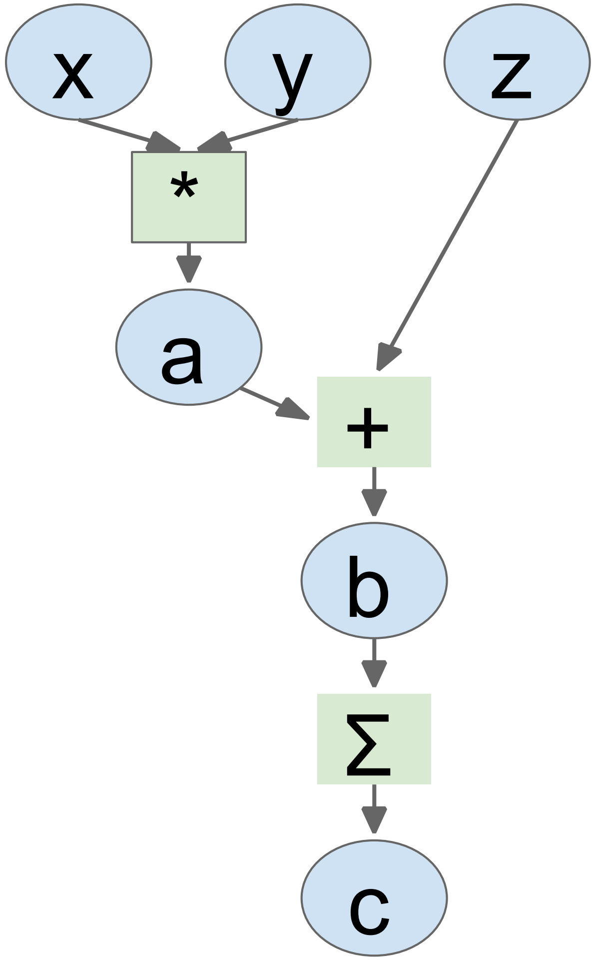
TensorFlow

创建前向计算图

利用TensorFlow计算梯度

```python
# Basic computational graph
import numpy as np
np.random.seed(0)
import tensorflow as tf

N, D = 3, 4

x = tf.placeholder(tf.float32)
y = tf.placeholder(tf.float32)
z = tf.placeholder(tf.float32)

a = x * y
b = a + z
c = tf.reduce_sum(b)

grad_x, grad_y, grad_z = tf.gradients(c, [x, y, z])

with tf.Session() as sess:
    values = {
        x: np.random.randn(N, D),
        y: np.random.randn(N, D),
        z: np.random.randn(N, D),
    }
    out = sess.run([c, grad_x, grad_y,grad_z],
                   feed_dict=values)
    c_val, grad_x_val, grad_y_val, grad_z_val = out
```

计算图

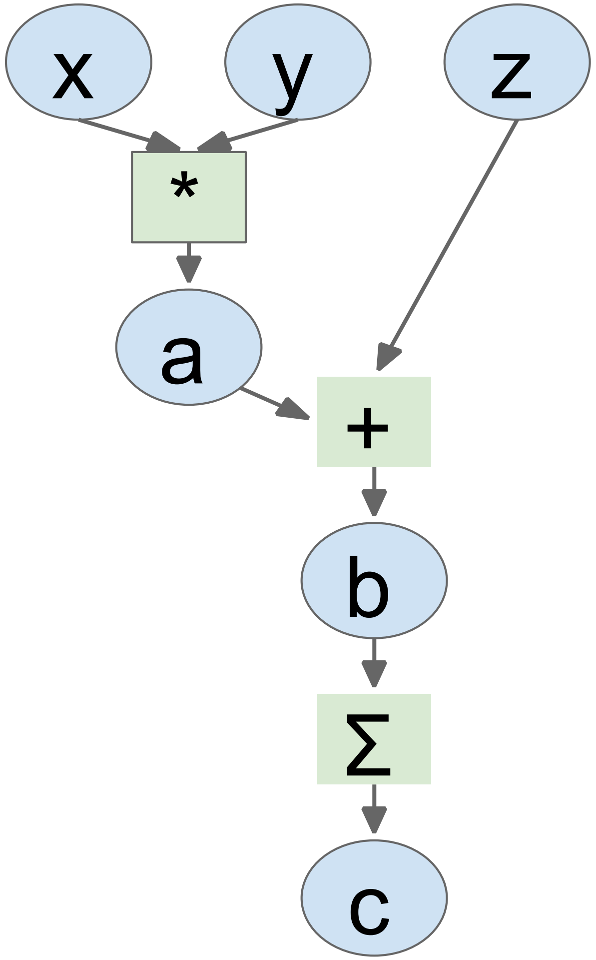


使用CPU运行

TensorFlow

```
import numpy as np
np.random.seed(0)
import tensorflow as tf

N, D = 3000, 4000

with tf.device('/cpu:0')
    x = tf.placeholder(tf.float32)
    y = tf.placeholder(tf.float32)
    z = tf.placeholder(tf.float32)

    a = x * y
    b = a + z
    c = tf.reduce_sum(b)

grad_x, grad_y, grad_z = tf.gradients(c, [x, y, z])

with tf.Session() as sess:
    values = {
        x: np.random.randn(N, D),
        y: np.random.randn(N, D),
        z: np.random.randn(N, D),
    }
    out = sess.run([c, grad_x, grad_y,grad_z],
                   feed_dict=values)
    c_val, grad_x_val, grad_y_val, grad_z_val = out
```

计算图

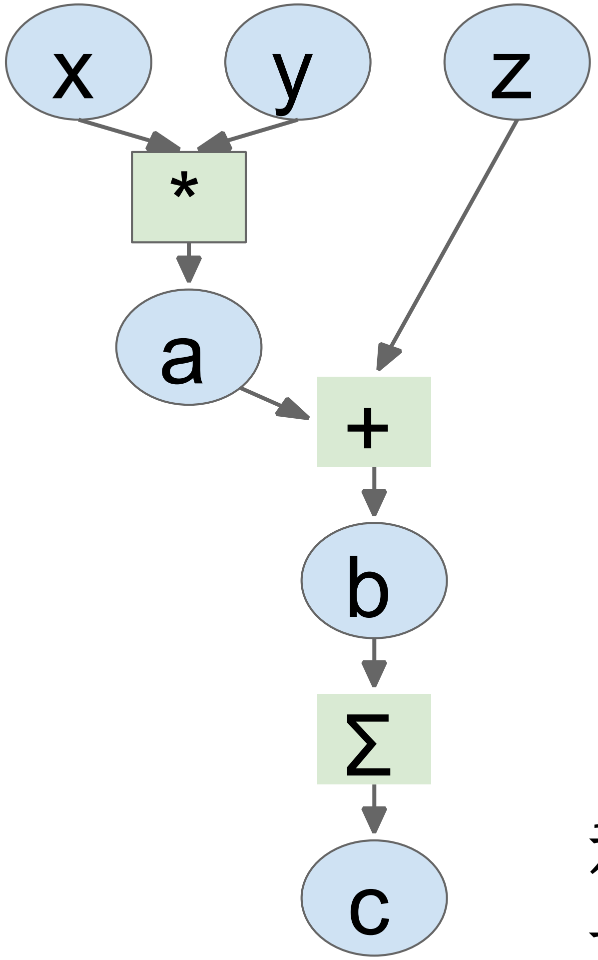


TensorFlow

使用GPU运行

```
import numpy as np
np.random.seed(0)
import tensorflow as tf

N, D = 3000, 4000

with tf.device('/gpu:0')
    x = tf.placeholder(tf.float32)
    y = tf.placeholder(tf.float32)
    z = tf.placeholder(tf.float32)

    a = x * y
    b = a + z
    c = tf.reduce_sum(b)

grad_x, grad_y, grad_z = tf.gradients(c, [x, y, z])

with tf.Session() as sess:
    values = {
        x: np.random.randn(N, D),
        y: np.random.randn(N, D),
        z: np.random.randn(N, D),
    }
    out = sess.run([c, grad_x, grad_y,grad_z],
                   feed_dict=values)
    c_val, grad_x_val, grad_y_val, grad_z_val = out
```

计算图



PyTorch

```
import torch
from torch.autograd import Variable

N, D = 3, 4
```

定义变量
开始构建计算图

```
x = Variable(torch.randn(N, D),
             requires_grad=True)
y = Variable(torch.randn(N, D),
             requires_grad=True)
z = Variable(torch.randn(N, D),
             requires_grad=True)
```

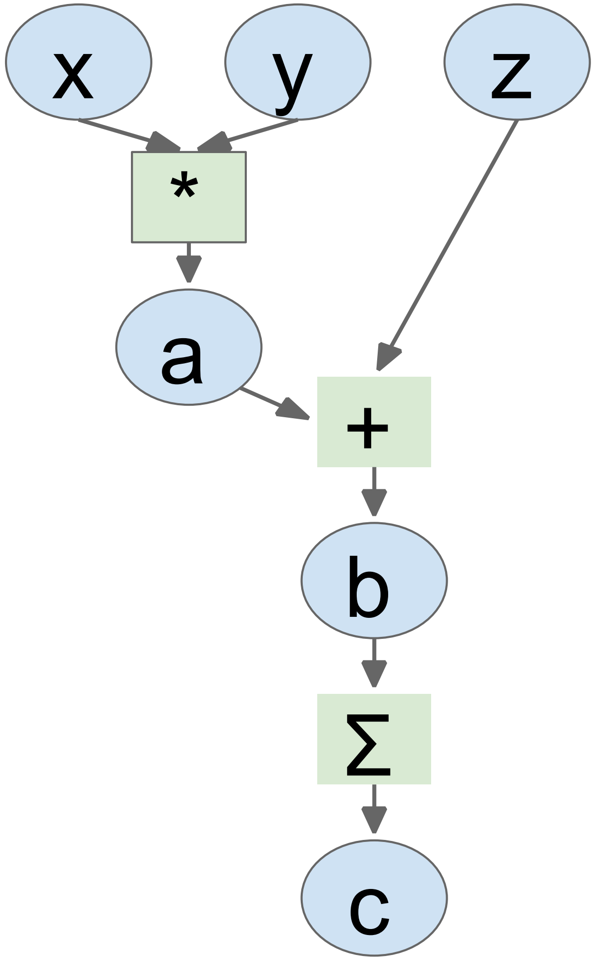
类似numpy的
前向计算

```
a = x * y
b = a + z
c = torch.sum(b)
```

利用c.backward()
计算所有梯度

```
c.backward()

print(x.grad.data)
print(y.grad.data)
print(z.grad.data)
```

计算图



通过.cuda()
在GPU上运行

PyTorch

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

N, D = 3, 4

x = Variable(torch.randn(N, D).cuda(),
             requires_grad=True)
y = Variable(torch.randn(N, D).cuda(),
             requires_grad=True)
z = Variable(torch.randn(N, D).cuda(),
             requires_grad=True)

a = x * y
b = a + z
c = torch.sum(b)

c.backward()

print(x.grad.data)
print(y.grad.data)
print(z.grad.data)
```

Numpy

```python
import numpy as np
np.random.seed(0)

N, D = 3, 4

x = np.random.randn(N,D)
y = np.random.randn(N,D)
z = np.random.randn(N,D)

a = x * y
b = a + z
c = np.sum(b)

grad_c = 1.0
grad_b = grad_c * np.ones((N, D))
grad_a = grad_b.copy()
grad_z = grad_b.copy()
grad_x = grad_a * y
grad_y = grad_a * x
```

TensorFlow

```python
import numpy as np
np.random.seed(0)
import tensorflow as tf

N, D = 3000, 4000

with tf.device('/gpu:0'):
    x = tf.placeholder(tf.float32)
    y = tf.placeholder(tf.float32)
    z = tf.placeholder(tf.float32)

    a = x * y
    b = a + z
    c = tf.reduce_sum(b)

grad_x, grad_y, grad_z = tf.gradients(c, [x, y, z])

with tf.Session() as sess:
    values = {
        x: np.random.randn(N, D),
        y: np.random.randn(N, D),
        z: np.random.randn(N, D),
    }
    out = sess.run([c, grad_x, grad_y,grad_z],
                   feed_dict=values)
    c_val, grad_x_val, grad_y_val, grad_z_val = out
```

PyTorch

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

N, D = 3, 4

x = Variable(torch.randn(N, D).cuda(),
             requires_grad=True)
y = Variable(torch.randn(N, D).cuda(),
             requires_grad=True)
z = Variable(torch.randn(N, D).cuda(),
             requires_grad=True)

a = x * y
b = a + z
c = torch.sum(b)

c.backward()

print(x.grad.data)
print(y.grad.data)
print(z.grad.data)
```

TensorFlow：神经网络

Running example：

采用L2 loss，在随机数据上训练一个两层的ReLU网络

```
N, D, H = 64, 1000, 100
x = tf.placeholder(tf.float32, shape=(N, D))
y = tf.placeholder(tf.float32, shape=(N, D))
w1 = tf.placeholder(tf.float32, shape=(D, H))
w2 = tf.placeholder(tf.float32, shape=(H, D))

h = tf.maximum(tf.matmul(x, w1), 0)
y_pred = tf.matmul(h, w2)
diff = y_pred - y
loss = tf.reduce_mean(tf.reduce_sum(diff ** 2, axis=1))

grad_w1, grad_w2 = tf.gradients(loss, [w1, w2])

with tf.Session() as sess:
    values = {x: np.random.randn(N, D),
              w1: np.random.randn(D, H),
              w2: np.random.randn(H, D),
              y: np.random.randn(N, D),}
    out = sess.run([loss, grad_w1, grad_w2],
                   feed_dict=values)
    loss_val, grad_w1_val, grad_w2_val = out
```

TensorFlow: 神经网络

```
import numpy as np
import tensorflow as tf
```

假设每个代码段顶部都有 imports

```
N, D, H = 64, 1000, 100
x = tf.placeholder(tf.float32, shape=(N, D))
y = tf.placeholder(tf.float32, shape=(N, D))
w1 = tf.placeholder(tf.float32, shape=(D, H))
w2 = tf.placeholder(tf.float32, shape=(H, D))

h = tf.maximum(tf.matmul(x, w1), 0)
y_pred = tf.matmul(h, w2)
diff = y_pred - y
loss = tf.reduce_mean(tf.reduce_sum(diff ** 2, axis=1))

grad_w1, grad_w2 = tf.gradients(loss, [w1, w2])

with tf.Session() as sess:
    values = {x: np.random.randn(N, D),
              w1: np.random.randn(D, H),
              w2: np.random.randn(H, D),
              y: np.random.randn(N, D),}
    out = sess.run([loss, grad_w1, grad_w2],
                   feed_dict=values)
    loss_val, grad_w1_val, grad_w2_val = out
```

TensorFlow: 神经网络

首先定义计算图

```
N, D, H = 64, 1000, 100
x = tf.placeholder(tf.float32, shape=(N, D))
y = tf.placeholder(tf.float32, shape=(N, D))
w1 = tf.placeholder(tf.float32, shape=(D, H))
w2 = tf.placeholder(tf.float32, shape=(H, D))

h = tf.maximum(tf.matmul(x, w1), 0)
y_pred = tf.matmul(h, w2)
diff = y_pred - y
loss = tf.reduce_mean(tf.reduce_sum(diff ** 2, axis=1))

grad_w1, grad_w2 = tf.gradients(loss, [w1, w2])
```

然后多次执行计算图

```
with tf.Session() as sess:
    values = {x: np.random.randn(N, D),
              w1: np.random.randn(D, H),
              w2: np.random.randn(H, D),
              y: np.random.randn(N, D),}
    out = sess.run([loss, grad_w1, grad_w2],
                   feed_dict=values)
    loss_val, grad_w1_val, grad_w2_val = out
```

TensorFlow: 神经网络

为输入x，权重w1,w2以及目标y创建占位符

```
N, D, H = 64, 1000, 100
x = tf.placeholder(tf.float32, shape=(N, D))
y = tf.placeholder(tf.float32, shape=(N, D))
w1 = tf.placeholder(tf.float32, shape=(D, H))
w2 = tf.placeholder(tf.float32, shape=(H, D))

h = tf.maximum(tf.matmul(x, w1), 0)
y_pred = tf.matmul(h, w2)
diff = y_pred - y
loss = tf.reduce_mean(tf.reduce_sum(diff ** 2, axis=1))

grad_w1, grad_w2 = tf.gradients(loss, [w1, w2])

with tf.Session() as sess:
    values = {x: np.random.randn(N, D),
              w1: np.random.randn(D, H),
              w2: np.random.randn(H, D),
              y: np.random.randn(N, D),}
    out = sess.run([loss, grad_w1, grad_w2],
                   feed_dict=values)
    loss_val, grad_w1_val, grad_w2_val = out
```

TensorFlow: 神经网络

前向传播:
计算y的预测值
以及loss（y与
y_pred的L2距离）

此处没有计算,
只是构建计算图

```python
N, D, H = 64, 1000, 100
x = tf.placeholder(tf.float32, shape=(N, D))
y = tf.placeholder(tf.float32, shape=(N, D))
w1 = tf.placeholder(tf.float32, shape=(D, H))
w2 = tf.placeholder(tf.float32, shape=(H, D))

h = tf.maximum(tf.matmul(x, w1), 0)
y_pred = tf.matmul(h, w2)
diff = y_pred - y
loss = tf.reduce_mean(tf.reduce_sum(diff ** 2, axis=1))

grad_w1, grad_w2 = tf.gradients(loss, [w1, w2])

with tf.Session() as sess:
    values = {x: np.random.randn(N, D),
              w1: np.random.randn(D, H),
              w2: np.random.randn(H, D),
              y: np.random.randn(N, D),}
    out = sess.run([loss, grad_w1, grad_w2],
                   feed_dict=values)
    loss_val, grad_w1_val, grad_w2_val = out
```

TensorFlow: 神经网络

告知TensorFlow计算w1, w2的梯度损失

此处仍然没有计算，只是构建计算图

```
N, D, H = 64, 1000, 100
x = tf.placeholder(tf.float32, shape=(N, D))
y = tf.placeholder(tf.float32, shape=(N, D))
w1 = tf.placeholder(tf.float32, shape=(D, H))
w2 = tf.placeholder(tf.float32, shape=(H, D))

h = tf.maximum(tf.matmul(x, w1), 0)
y_pred = tf.matmul(h, w2)
diff = y_pred - y
loss = tf.reduce_mean(tf.reduce_sum(diff ** 2, axis=1))

grad_w1, grad_w2 = tf.gradients(loss, [w1, w2])

with tf.Session() as sess:
    values = {x: np.random.randn(N, D),
              w1: np.random.randn(D, H),
              w2: np.random.randn(H, D),
              y: np.random.randn(N, D),}
    out = sess.run([loss, grad_w1, grad_w2],
                   feed_dict=values)
    loss_val, grad_w1_val, grad_w2_val = out
```

TensorFlow：神经网络

此时计算图构建完毕，可以进入会话执行计算图

```
N, D, H = 64, 1000, 100
x = tf.placeholder(tf.float32, shape=(N, D))
y = tf.placeholder(tf.float32, shape=(N, D))
w1 = tf.placeholder(tf.float32, shape=(D, H))
w2 = tf.placeholder(tf.float32, shape=(H, D))

h = tf.maximum(tf.matmul(x, w1), 0)
y_pred = tf.matmul(h, w2)
diff = y_pred - y
loss = tf.reduce_mean(tf.reduce_sum(diff ** 2, axis=1))

grad_w1, grad_w2 = tf.gradients(loss, [w1, w2])

with tf.Session() as sess:
    values = {x: np.random.randn(N, D),
              w1: np.random.randn(D, H),
              w2: np.random.randn(H, D),
              y: np.random.randn(N, D),}
    out = sess.run([loss, grad_w1, grad_w2],
                   feed_dict=values)
    loss_val, grad_w1_val, grad_w2_val = out
```

TensorFlow: 神经网络

创建numpy数组来填充上面的占位符

```
N, D, H = 64, 1000, 100
x = tf.placeholder(tf.float32, shape=(N, D))
y = tf.placeholder(tf.float32, shape=(N, D))
w1 = tf.placeholder(tf.float32, shape=(D, H))
w2 = tf.placeholder(tf.float32, shape=(H, D))

h = tf.maximum(tf.matmul(x, w1), 0)
y_pred = tf.matmul(h, w2)
diff = y_pred - y
loss = tf.reduce_mean(tf.reduce_sum(diff ** 2, axis=1))

grad_w1, grad_w2 = tf.gradients(loss, [w1, w2])

with tf.Session() as sess:
    values = {x: np.random.randn(N, D),
              w1: np.random.randn(D, H),
              w2: np.random.randn(H, D),
              y: np.random.randn(N, D),}
    out = sess.run([loss, grad_w1, grad_w2],
                   feed_dict=values)
    loss_val, grad_w1_val, grad_w2_val = out
```

TensorFlow: 神经网络

```
N, D, H = 64, 1000, 100
x = tf.placeholder(tf.float32, shape=(N, D))
y = tf.placeholder(tf.float32, shape=(N, D))
w1 = tf.placeholder(tf.float32, shape=(D, H))
w2 = tf.placeholder(tf.float32, shape=(H, D))

h = tf.maximum(tf.matmul(x, w1), 0)
y_pred = tf.matmul(h, w2)
diff = y_pred - y
loss = tf.reduce_mean(tf.reduce_sum(diff ** 2, axis=1))

grad_w1, grad_w2 = tf.gradients(loss, [w1, w2])

with tf.Session() as sess:
    values = {x: np.random.randn(N, D),
              w1: np.random.randn(D, H),
              w2: np.random.randn(H, D),
              y: np.random.randn(N, D),}
    out = sess.run([loss, grad_w1, grad_w2],
                   feed_dict=values)
    loss_val, grad_w1_val, grad_w2_val = out
```

执行计算图：将numpy数组喂给x, y, w1, w2，获得loss以及w1, w2的梯度

深度学习软件框架建议

- **TensorFlow** 是当前最流行的，适用大部分工作
- **PyTorch** 适合学术研究
- **Caffe**, **TensorFlow** 适合工程项目
- **TensorFlow** or **Caffe2** 适合移动端

参考资料：

- TensorFlow官网：https://tensorflow.google.cn/
- TensorFlow中文社区：http://www.tensorfly.cn/
- Stanford CS231n