Neural Network Abstractions

Programming Abstractions

一个框架所提供的编程接口定义了实现、扩展、执行模型计算的最常用的方法

Forward and Backward Layer Interface

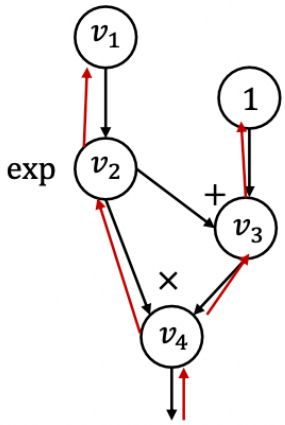
Example framework: Caffe 1.0

特点：图中的每个节点称为 Layer. 在 `Layer` 类中定义了 `forward` 和 `backward` 两种方法

forward pass 和 backward pass 在同一个计算图上进行 (in-place)

可以执行 forward / backward 传播操作

缺点：将梯度计算与模块组成结合，耦合度高



Computational Graph and Declarative Programming

Example framework: Tensorflow 1.0

特点：先声明计算图，再提供输入并执行计算

```
import tensorflow as tf

v1 = tf.Variable()
v2 = tf.exp(v1)
v3 = v2 + 1
v4 = v2 * v3

sess = tf.Session()
value4 = sess.run(v4, feed_dict={v1: numpy.array([1]})
```

可以根据需要的数据，只执行计算图的局部，跳过不必要的部分

优点：

- 提前给出计算图（静态计算图），优化的可能性更大
- 可放在远程服务器上计算，计算与开发分离
- 很多推理模型的选择，因为内存复用和优化的空间大

缺点：

- 交互式 (print法) debug 不方便
- 静态计算图灵活性差
- 必须使用描述计算图的语言，无法内嵌 Python 原生的分支语句（例如，想在 Tensorflow 1.0 中引入 if 语句必须加入新的节点 `tf.if`

Imperative Automatic Differentiation

Example framework: PyTorch (needle)

特点：

- define by run: 在构建计算图的同时进行计算
- 允许加入控制流（动态计算图）

优点：

- 更友好，允许混用 Python (host language)
- 可以直接用 print 大法 debug（交互式）
- 灵活性强，可以在运行时动态决定生成神经网络的深度（在 NLP 中有用）
- 更高层的抽象，将梯度计算和模块组成分离，变成对 tensor 的计算图抽象和自微分处理两部分

缺点：

- 针对计算图的优化空间小

灵活性与优化能力的 tradeoff

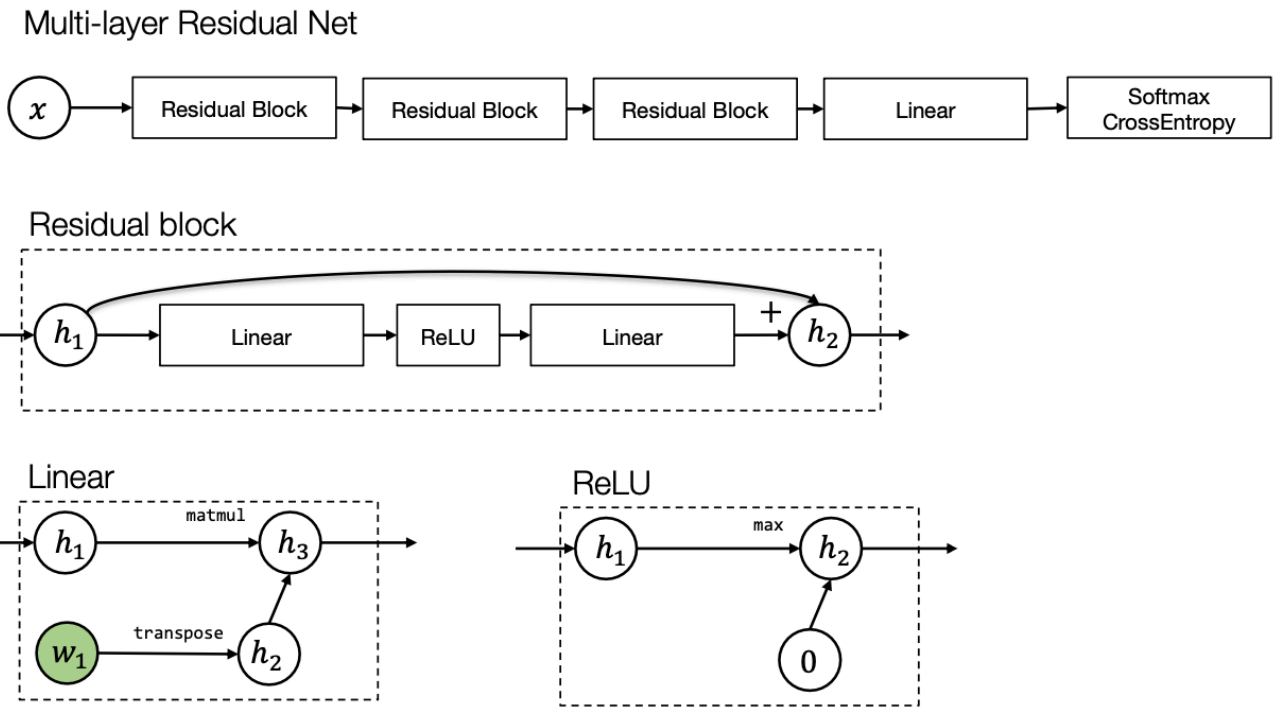
JIT 编译等技术可以弥补优化能力的不足，因此这种方式更受欢迎

High Level Modular Library Components

机器学习模型是模块化的，深度学习更是如此

Hypothesis class 本身也是模块化的

最基本的是 tensor 和对应的操作



nn.Module: Compose Things Together

Tensor in, tensor out

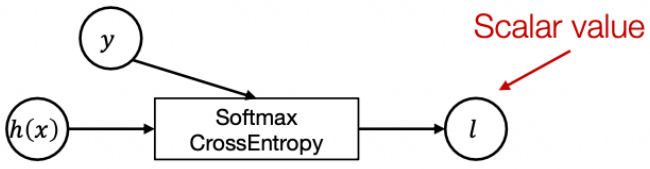
关键点：

- 对于给定输入，如何计算输出
- 获取训练参数以进行更新
- 初始化参数的方法

Loss Functions as a Special Kind of Module

Tensor in, scalar out

计算 loss



Optimizer

执行参数优化

以权重和额外状态 (e.g. u_i in momentum) 为输入，并且要在内部维护额外状态

SGD

$$w_i \leftarrow w_i - \alpha g_i$$

SGD with momentum

$$u_i \leftarrow \beta u_i + (1-\beta) g_i$$
$$w_i \leftarrow w_i - \alpha u_i$$

Adam

$$u_i \leftarrow \beta_1 u_i + (1-\beta_1) g_i$$
$$v_i \leftarrow \beta_2 v_i + (1-\beta_2) g_i^2$$
$$w_i \leftarrow w_i - \alpha u_i / (v_i^{1/2} + \epsilon)$$

Regularization and Optimizer

两种引入正则化的方法：

- 作为 loss function 的一部分实现
- 直接融入 optimizer 更新的一部分

SGD with weight decay (l_2 regularization) $w_i \leftarrow (1-\alpha\lambda)\, w_i - \alpha g_i$

Initialization

权重初始化过小会导致在传播过程中接近 0, 初始化过大会导致在传播过程中变得更大

初始化取决于模型本身和参数类型

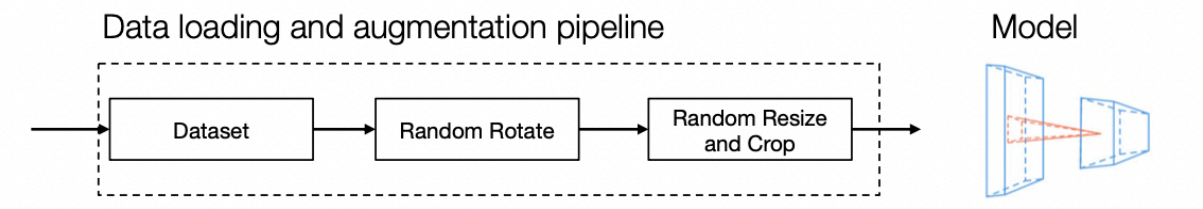
大部分神经网络库都有常用的初始化方法

- 权重：数量级取决于输入/输出，通常采用正态分布/高斯分布
- bias: 初始化为 0
- Running sum of variance: 1

初始化可以融入 nn.module 的构建阶段

Data Loader and Preprocessing

加载数据并做预处理 (augment) 以达到高效训练

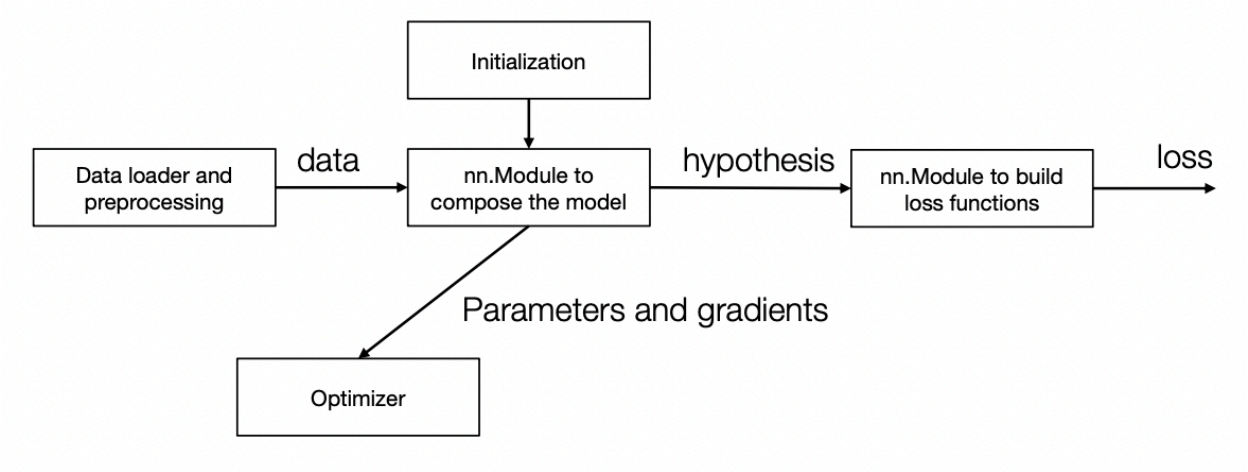


预处理通常包括 randomly shuffle 和 transform the input

不同的 data augmentation 对预测正确率的加速影响很大，通常占据模型的一大部分

Summary of nn.module

nn.module 将上述各个部分组合在一起


Initialization
Data loader and preprocessing
data
nn.Module to compose the model
hypothesis
nn.Module to build loss functions
loss
Parameters and gradients
Optimizer