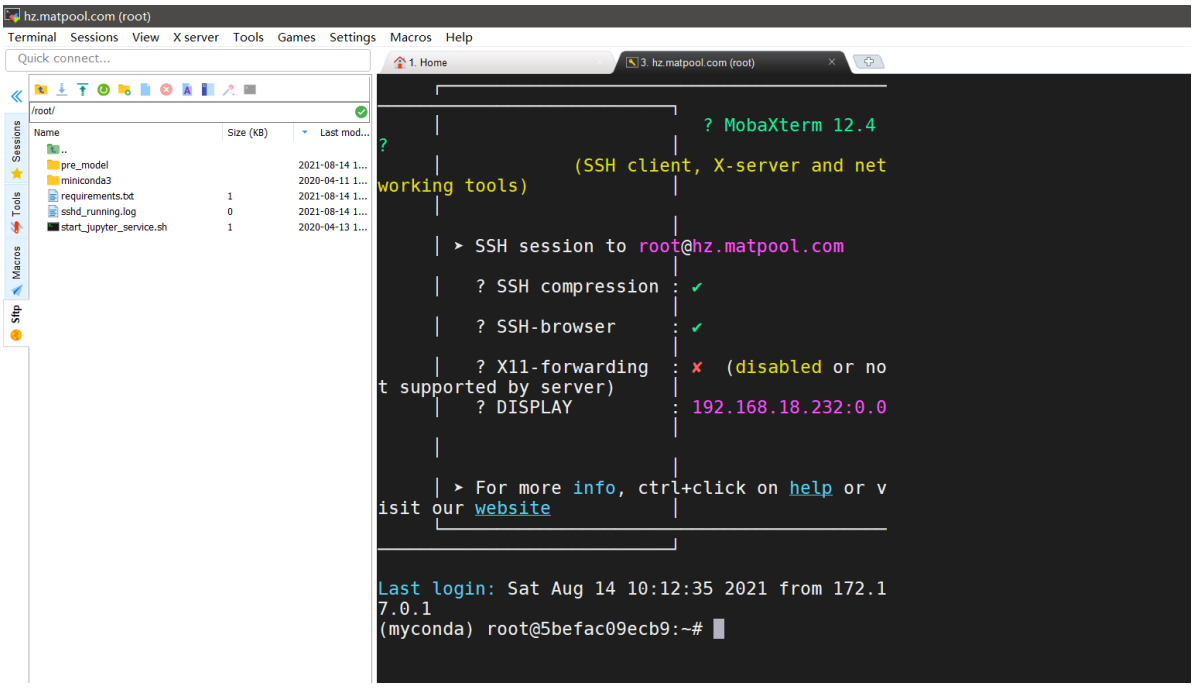
mobaxterm、Git、Keras与BP神经网络

- 远程连接服务器神器：mobaxterm
- https://mobaxterm.mobatek.net/download.html
- 使用指南：https://zhuanlan.zhihu.com/p/344035406
- win ssh远程连接mac电脑：https://blog.csdn.net/phunxm/article/details/44758753



Git

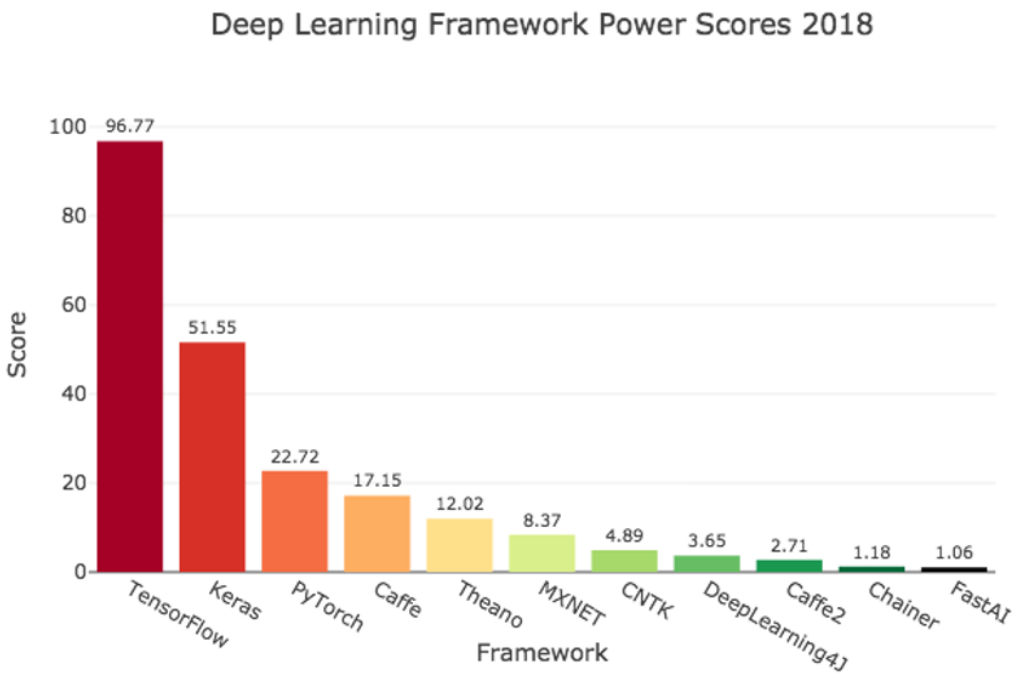
- https://git-scm.com/downloads
- 三大区域
  - 暂存区：stage or index， .git目录下的index文件
  - 工作区：电脑所看到的目录
  - 版本库：.git目录
- 创建github账户：https://github.com/

```
git clone https://github.com/ChileWang0228/alg-chile.git # 从现有远程仓库中拷贝项目

git status  # 查看当前代码仓库状态
git add . # 添加新的修改到暂存区
git commit -m "new commit"  # 将暂存区内容添加到版本库中
git push # 将本地仓库的内容推送的远程仓库并合并
```

Keras

- 当前三大主流框架
  - tensorflow
  - keras
  - pytorch



- keras的优点
  - 允许简单快速的原型设计（用户友好性，模块化和可扩展性）。
  - 支持卷积网络和循环网络，以及两者的组合。
  - 在CPU和GPU上无缝运行。
- keras的缺点
  - Keras比较注重网络层次，然而并非所有网络都是层层堆叠的
  - Keras在设计新的网络方面会比Tensorflow差一些

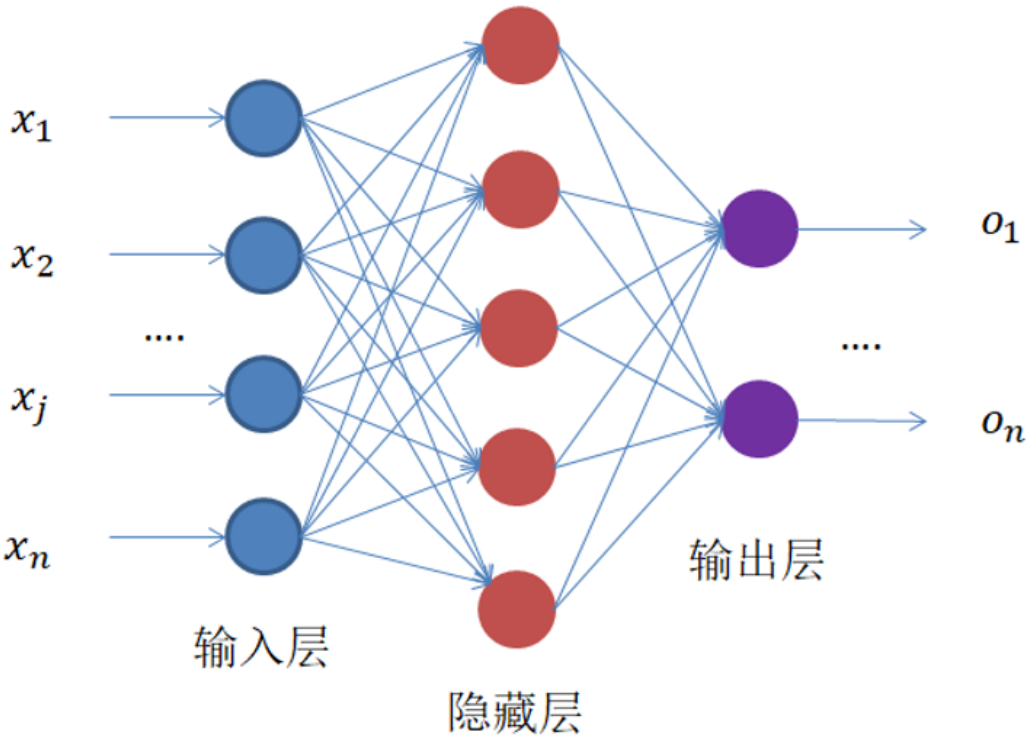
```
conda install keras  # CPU版本
conda install keras-gpu # GPU版本

pip install keras
```

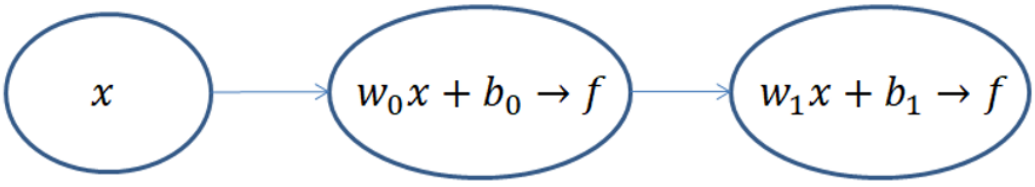
- https://www.anaconda.com/products/individual#Downloads

BP神经网络（back propagation）

- 注意点
  - 输入：x
  - 输出：o
  - 隐藏：h



- 简单的BP神经网络

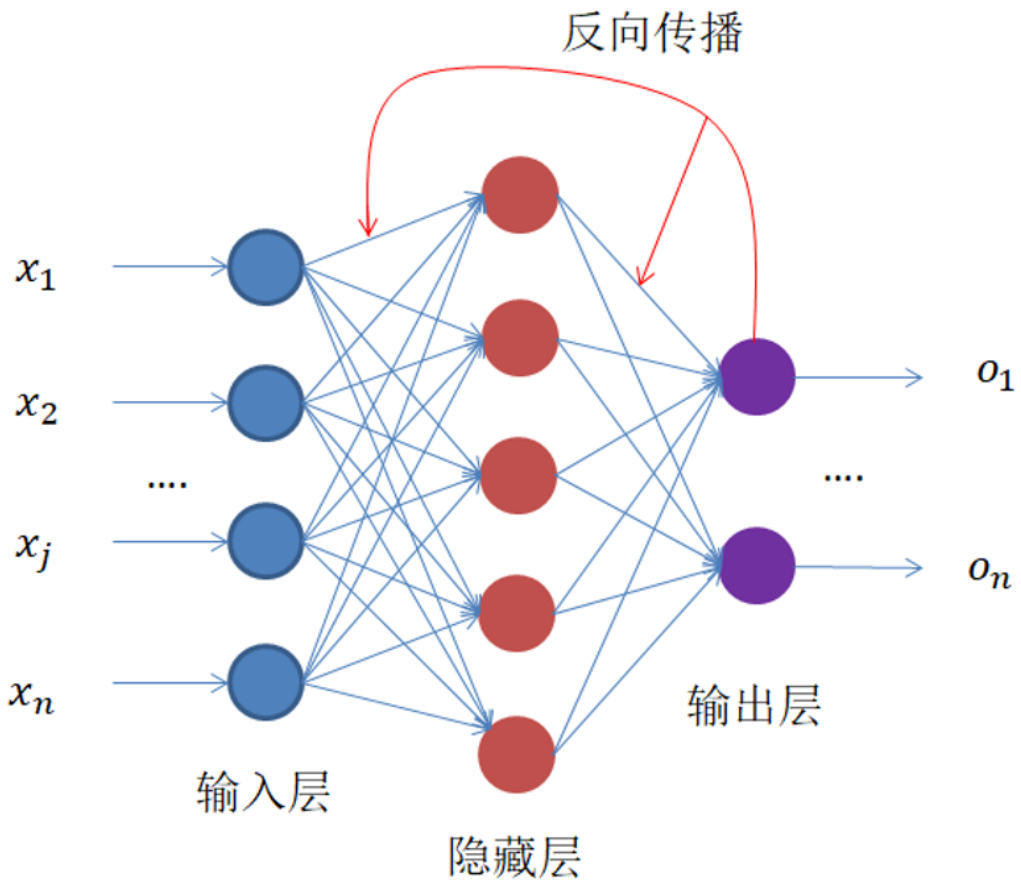


- 不算输入层，上面的网络结构总共有两层，隐藏层和输出层，它们"圆圈"里的计算

$$z = wx + b$$

$$f(z) = \frac{1}{e^{-z}}$$

- 每一级都是利用前一级的输出做输入，再经过圆圈内的组合计算，输出到下一级
- *f*(*z*)的目的
  - 将输出的值域压缩到（0，1），也就是所谓的归一化，因为每一级输出的值都将作为下一级的输入，只有将输入归一化了，才会避免某个输入无穷大，导致其他输入无效，变成“一家之言”，最终网络训练效果非常不好。
- 反向传播在哪?



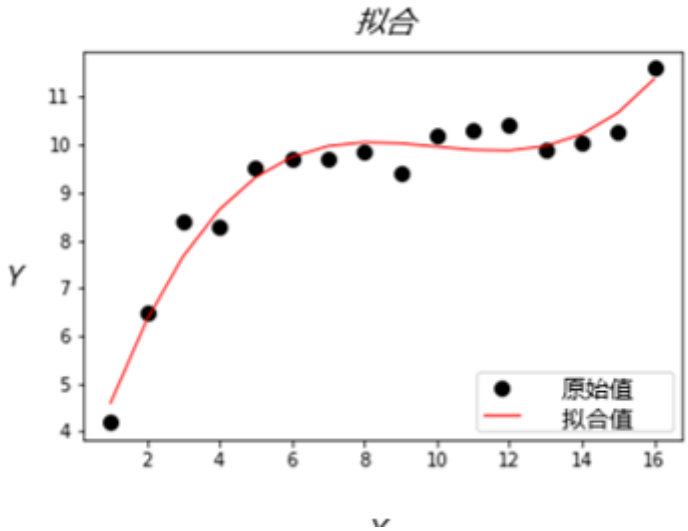
- 反向传播的目的
  - 神经网络的训练是有监督的学习，也就是输入*X*有着与之对应的真实值Y 。
  - 神经网络的输出Y 与真实值Y 之间的损失Loss 就是网络反向传播的东西。
  - 整个网络的训练过程就是不断缩小损失Loss 的过程。

$$Loss = \sum_{i=1}^{n}(y_i - (wx_i + b))^2$$

$$Loss = \sum_{i=1}^{n}\left(x_i^2 w^2 + b^2 + 2x_i wb - 2y_i b - 2x_i y_i w + y_i^2\right)$$

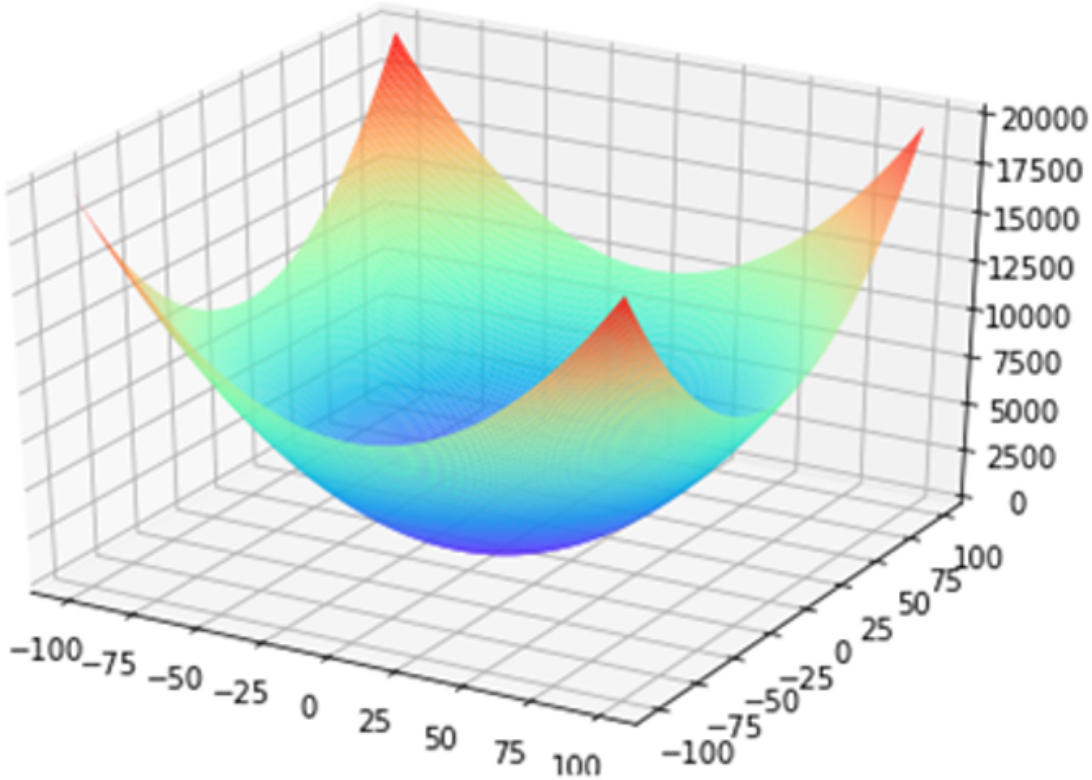
$$Loss = \sum_{i=1}^{n}\left(Aw^2 + Bb^2 + Cwb - Db - Ew + F\right)$$

- 上述的公式经过化简，我们可以看到*A*、*B*、*C*、*D*、*E*、*F*都是常系数，未知数就是*w* 和*b* ，也就是为了让Loss 最小，我们要求解出最佳的*w* 和*b* 。
- 我们稍微想象一下，如果这是个二维空间，那么我们相当于要找一条曲线，让它与坐标轴上所有样本点距离最小。



- 其实LOSS方程为三维空间，那就可以转化成一个三维空间的求最优解的过程。
  - 三维图像就像一个“碗”，它和二维空间的抛物线一样，存在极值，那我们只要将极值求出，那就保证了我们能求出最优的（*w*，*b*）也就是这个“碗底”的坐标，使Loss 最小

$$Z = X^2 + Y^2$$



- 如何求解最优的"碗底"坐标（w，b）
  - 高等数学中的偏导：简单来讲，也就是对X，Y分别求导，在求导过程中，把其他的未知量当成常数即可
  - 求偏导的过程如同下山
    - 想象自己在一座山上，要想从山上最快地去到谷底，那就要沿着最陡峭的地方往下走。
    - 这个最陡峭的地方，我们叫作梯度，像不像我们对上面那个"碗"做切线，找出最陡的那条切线，事实上我们做的就是这个。

$$\text{梯度}: \nabla = \left(\frac{\partial f(x,y)}{\partial x}, \frac{\partial f(x,y)}{\partial y} \right)$$

  - 我们每走一步，坐标就会更新

$$x_{n+1} = x_n - \alpha \frac{\partial f(x,y)}{\partial x}$$

$$y_{n+1} = y_n - \alpha \frac{\partial f(x,y)}{\partial y}$$

- 这是三维空间中的，假如我们在多维空间漫步呢，其实也是一样的，也就是对各个维度求偏导，更新自己的坐标

$$\nabla = \left(\frac{\partial f(w)}{\partial w_1}, \frac{\partial f(w)}{\partial w_2}, \frac{\partial f(w)}{\partial w_3}, \frac{\partial f(w)}{\partial w_4}, \cdots, \frac{\partial f(w)}{\partial w_n} \right)$$

$$w_{i+1} = w - \alpha \frac{\partial f(w)}{\partial w^i}$$

- 其中， w的上标 表示第几个w ， w的下标表示第几步， α是学习率，后面会介绍的作用。
- 所以，我们可以将整个求解过程看做下山（求偏导过程），为此，我们先初始化自己的初始位置

$$(w_0, b_0)$$

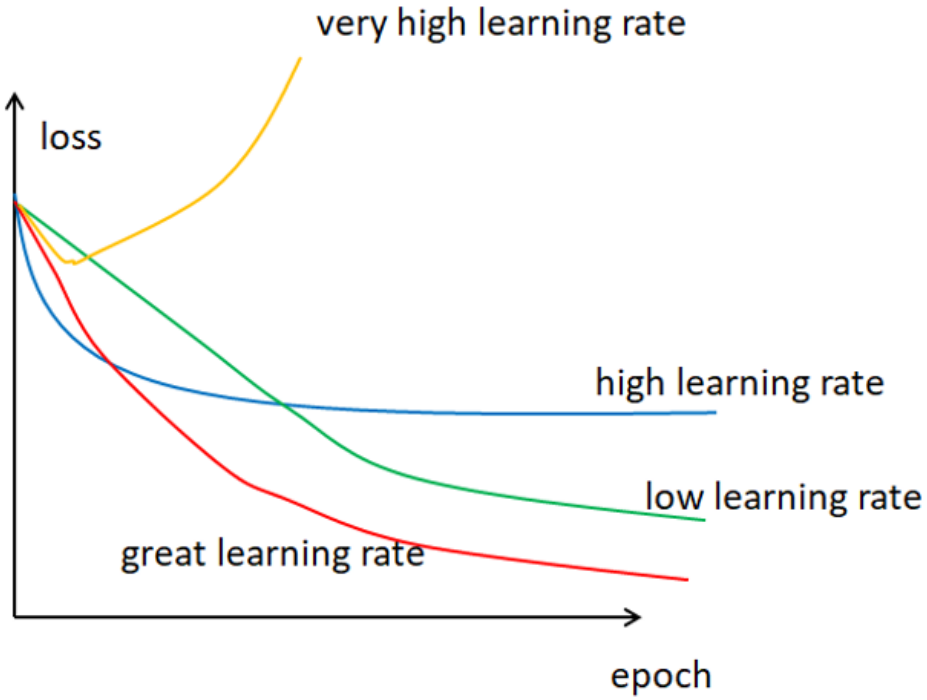
$$w_1 = w_0 - \alpha \frac{\partial Loss(w,b)}{\partial w}$$

$$b_1 = b_0 - \alpha \frac{\partial Loss(w,b)}{\partial b}$$

> * 这样我们不断地往下走（迭代），当我们逐渐接近山底的时候，每次更新的步伐也就越来越小，损失值也就越来越小，直到达到某个阈值或迭代次数时，停止训练，这样找到（*w*，*b*）就是我们要求的解。
> * 我们将整个求解过程称为梯度下降求解法。

- 学习率α
  - 什么是学习率?

> * 通常来说，学习率是可以随意设置，你可以根据过去的经验或书本资料选择一个最佳值，或凭直觉估计一个合适值，一般在（0，1）之间。
> * 这样做可行，但并非永远可行。事实上选择学习率是一件比较困难的事
> * epoch为使用训练集全部样本训练一次的单位，loss表示损失



    - 学习率的目的
        - 直接影响我们的模型能够以多快的速度收敛到局部最小值（也就是达到最好的精度）。
        - 一般来说，学习率越大，神经网络学习速度越快。
        - 如果学习率太小，网络很可能会陷入局部最优；
        - 但是如果太大，超过了极值，损失就会停止下降，在某一位置反复震荡。
        - 也就是说，如果我们选择了一个合适的学习率，我们不仅可以在更短的时间内训练好模型，还可以节省各种运算资源的花费。
        - 如何选择?
            - 业界并没有特别硬性的定论，总的来说就是试出来的，看哪个学习率能让Loss收敛得更快，Loss最小，就选哪个。
- 作业
    - git
    - 总结
- 下一节课
    - CNN卷积神经网络