AI学习笔记--Tensorflow--TF overfit and underfit

本章主要介绍了使用 Keras API来解决过拟合和欠拟合的处理方式。在前面的两个例子分类电影评论和预测燃油效率，我们看到，我们的模型对验证数据的准确性达到峰值后，出现了一个明显拐点，然后将开始下降。换句话说，我们的模型会与训练数据过度匹配。这也是让模型学习如何处理过度拟合问题变得更加紧要。虽然通常可以在训练集上获得高精度，但我们真正想要的是开发能够很好地部署到测试集的模型（或他们以前从未见过的数据）。

过拟合（overfit）的另一面是欠拟合（underfit）。当一组测试数据输入模型后，得出的结果和我们预期差距较大时，这也体现了模型任然有很大的改进空间。出现这类情况有很多原因，例如：训练合集出现过多条件限制，导致过拟合，或者训练数据不够多，训练时间不够长等情况都会导致模型最终的效果不达预期。

但是，如果喂养数据过长，模型就会开始过度拟合，导致在新的数据上难以出现心理想要的预期。下面就来讨论如何使用适当的条件，产生最大的价值。这也是在机器学习领域十分常用的一种技术。

通常情况下，为了防止过度拟合，我们最简单的方式是使用更多的数据喂养模型，这样产生的模型也可以适配更多的场景决策。但是如果在获取数据比较困难的情况下，我们可以用正则表达式等方式，来解决数据不足的情况。如果一个神经网络在训练过程中只能拥有少量的数据，那么如何使用现有的优化手段来完善整个模型。同时，这类优化手段可以把整个模型更加的泛化，可以适应更复杂的运行环境。

在这里，我们重点介绍了常见正则表达式和权重正则表达式两种优化方案，并且应用他们来优化电影评论的例子。

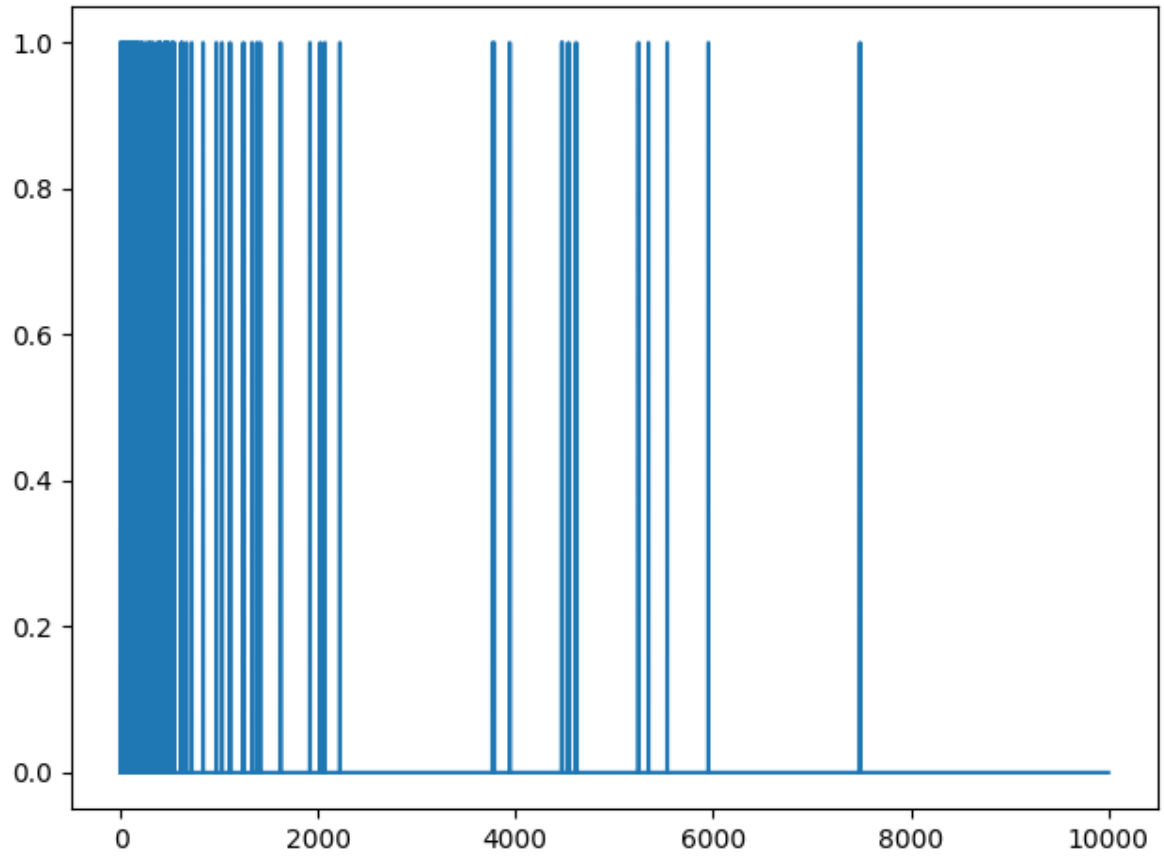
第一步，还是下载 IMDB database 的数据库，用作训练数据和测试数据。并且拆分他们为 5：3。

```
from __future__ import absolute_import, division, print_function,
unicode_literals

import tensorflow as tf
from tensorflow import keras

import numpy as np
import matplotlib.pyplot as plt

print(tf.__version__)

NUM_WORDS = 10000

(train_data, train_labels), (test_data, test_labels) =
keras.datasets.imdb.load_data(num_words=NUM_WORDS)

def multi_hot_sequences(sequences, dimension):
    # Create an all-zero matrix of shape (len(sequences),
dimension)
    results = np.zeros((len(sequences), dimension))
    for i, word_indices in enumerate(sequences):
        results[i, word_indices] = 1.0  # set specific indices of
```

```
    results[i] to 1s
    return results


train_data = multi_hot_sequences(train_data, dimension=NUM_WORDS)
test_data = multi_hot_sequences(test_data, dimension=NUM_WORDS)
```

如果需要看其中之一向量的数据，可以看到单词的索引是有一定规律的。在0-1 之间集中了很多，可以用 plt 绘制图形。

```
plt.plot(train_data[0])
plt.show()
```



- 证明过拟合

防止过度拟合的最简单方法是减小模型的大小，即模型中可学习参数的数量（由层数和每层单元数决定）。在深度学习中，模型中可学习参数的数量通常被称为模型的“容量”。直观地说，一个具有更多参数的模型将具有更多的“记忆容量”，因此能够很容易地学习到一个完美的字典式的训练样本与其目标之间的映射，这会是一个没有任何泛化能力的模型，只对以前看不见的数据进行预测时是用处不大的。

请记住：深度学习模型往往很好地拟合训练数据，但真正的挑战是泛化，而不是拟合。

另一方面，如果网络的记忆资源有限，就无法像学习地图那样容易。为了使损失最小化，它必须学习具有更高预测能力的压缩表示。同时，如果模型太小，则很难拟合训练数据。在“容量过大”和“容量不足”之间存在平衡。

不幸的是，没有神奇的公式来确定模型的正确大小或架构（根据层的数量，或每个层的正确大小）。你将不得不尝试使用一系列不同的架构。要找到合适的模型大小，最好从相对较少的层和参数开始，然后开始增加层的大小或添加新的层，直到看到验证损失的回报递减。让我们在我们的影评分类网上试试这个。

我们将创建一个仅使用密集层作为基线的简单模型，然后创建越来越小的版本，并对它们进行比较。

- 创建模型

```
baseline_model = keras.Sequential([
    # `input_shape` is only required here so that `.summary` works.
    keras.layers.Dense(16, activation='relu', input_shape=(NUM_WORDS,)),
    keras.layers.Dense(16, activation='relu'),
    keras.layers.Dense(1, activation='sigmoid')
])

baseline_model.compile(optimizer='adam',
                       loss='binary_crossentropy',
                       metrics=['accuracy', 'binary_crossentropy'])

baseline_model.summary()

baseline_history = baseline_model.fit(train_data,
                                      train_labels,
                                      epochs=20,
                                      batch_size=512,
                                      validation_data=(test_data,
test_labels),
                                      verbose=2)
```

- Create a smaller model

```
# 创建一个容量较小的模型 迭代 4 层
smaller_model = keras.Sequential([
    keras.layers.Dense(4, activation='relu', input_shape=(NUM_WORDS,)),
    keras.layers.Dense(4, activation='relu'),
    keras.layers.Dense(1, activation='sigmoid')
])
```

```
smaller_model.compile(optimizer='adam',
                      loss='binary_crossentropy',
                      metrics=['accuracy', 'binary_crossentropy'])

smaller_model.summary()

smaller_history = smaller_model.fit(train_data,
                                    train_labels,
                                    epochs=20,
                                    batch_size=512,
                                    validation_data=(test_data,
test_labels),
                                    verbose=2)
```

Create a bigger model

As an exercise, you can create an even larger model, and see how quickly it begins overfitting. Next, let's add to this benchmark a network that has much more capacity, far more than the problem would warrant:

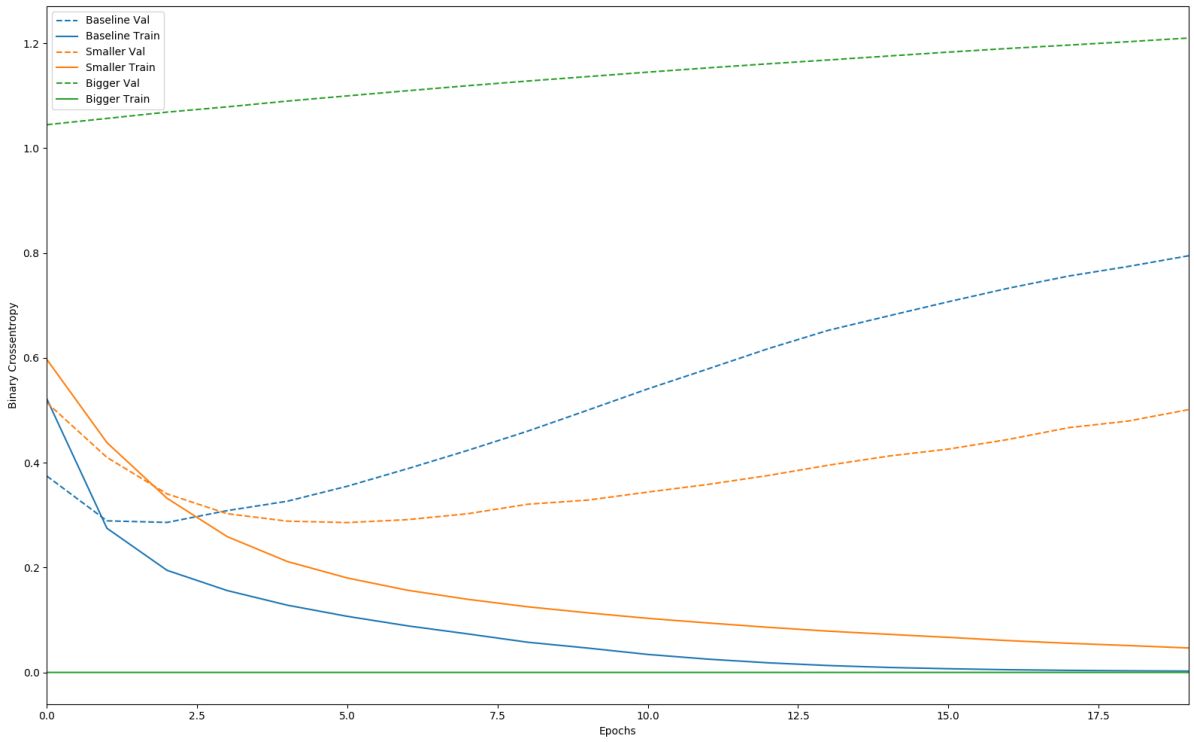
```
bigger_model = keras.models.Sequential([
    keras.layers.Dense(512, activation='relu', input_shape=(NUM_WORDS,)),
    keras.layers.Dense(512, activation='relu'),
    keras.layers.Dense(1, activation='sigmoid')
])

bigger_model.compile(optimizer='adam',
                     loss='binary_crossentropy',
                     metrics=['accuracy','binary_crossentropy'])

bigger_model.summary()

bigger_history = bigger_model.fit(train_data, train_labels,
                                  epochs=20,
                                  batch_size=512,
                                  validation_data=(test_data, test_labels),
                                  verbose=2)
bigger_history = bigger_model.fit(train_data, train_labels,
                                  epochs=20,
                                  batch_size=512,
                                  validation_data=(test_data, test_labels),
                                  verbose=2)
```

绘制培训和验证损失图

实线表示训练损失，虚线表示验证损失（请记住：验证损失越低表示模型越好）。在这里，较小的网络比基线模型开始过度拟合的时间晚（在6个阶段之后而不是4个阶段之后），一旦开始过度拟合，其性能下降的速度就会慢得多。

```
def plot_history(histories, key='binary_crossentropy'):
  plt.figure(figsize=(16,10))

  for name, history in histories:
    val = plt.plot(history.epoch, history.history['val_'+key],
                   '--', label=name.title()+' Val')
    plt.plot(history.epoch, history.history[key], color=val[0].get_color(),
             label=name.title()+' Train')

  plt.xlabel('Epochs')
  plt.ylabel(key.replace('_',' ').title())
  plt.legend()

  plt.xlim([0,max(history.epoch)])


plot_history([('baseline', baseline_history),
              ('smaller', smaller_history),
              ('bigger', bigger_history)])
```

结果：



注意，过大的网络几乎在一个拐点之后立即开始过度拟合，并且过度拟合的程度要严

重得多。可以看到网络的容量越大，它能够更快地对训练数据进行建模（导致较低的训练损失），但越容易过度拟合（导致训练和验证损失之间的巨大差异）。

- 防止过度拟合的策略

加权正则化

Occam's Razor 原理，任何事都会有两种原因，往往最简单的原因就是事件发生最本质的原因，即用最少的假设解释一件事物发生的缘由。这种思维也适用于机器学习领域。给定一些训练数据和网络结构，用多组权重值（多模型）的形式来总结数据走势的规律，这样可以降低模型的复杂程度，防止出现过拟合的现象。

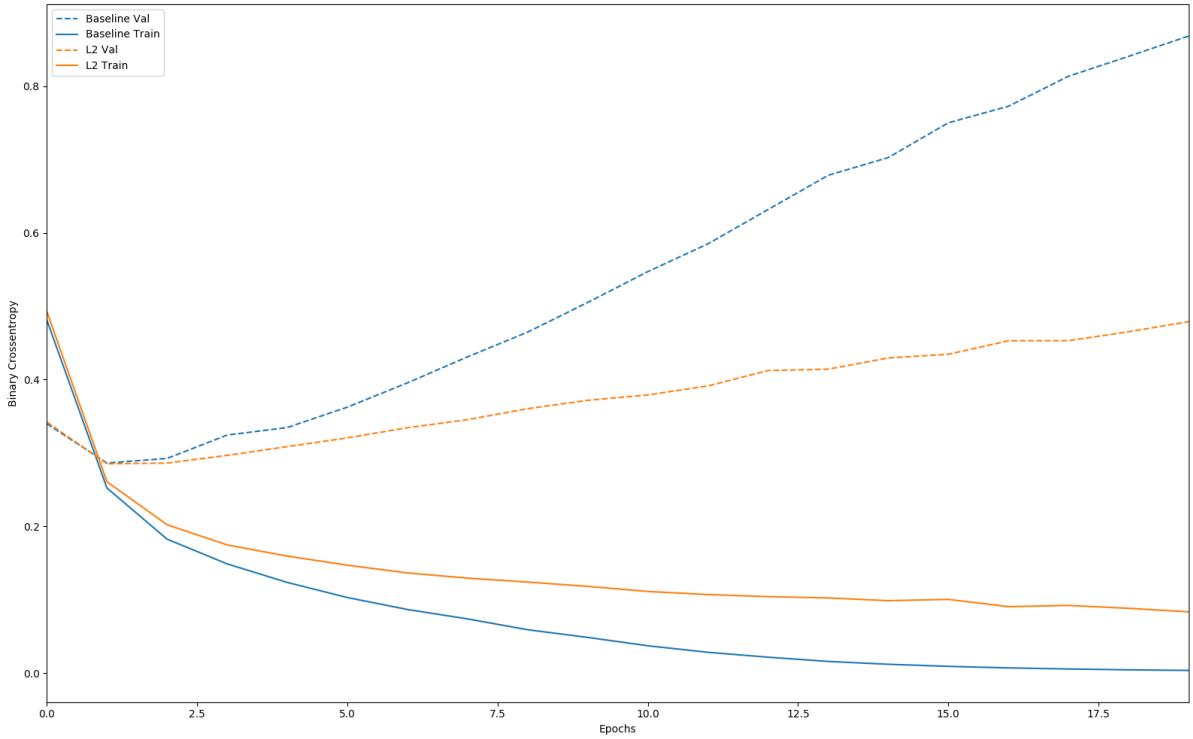
在这种情况下，简单模型是一个数据分布熵值较小的模型（或者是参数较少）、如上面所看到的的例子。如果用一种强制对某些输入项目加权的方式来限制总体模型的复杂度就可以让整个模型避免过拟合，这样的权值也满足一定的规则，也可以叫作是'权重正则法则'。他通过在网络损失函数中加入具有较大权重关联的方式达到目的，一般而言，具有两种方式：

- L1 正则化，其中增加的比重与权重系数的绝对值成比例（即，与权重的"L1范数"成比例）。
- L2正则化，其中所增加的比重与权重系数值的平方成正比（即，与权重的平方"L2范数"成正比）。L2正则化在神经网络中也称为权值衰减。不要让不同的名字迷惑你：权重衰减在数学上与L2正则化完全相同。

L1正则化引入稀疏性使一些权重参数为零。L2正则化将惩罚权重参数，而不会使其稀疏，这是L2更常见的一个原因。在tf.keras中，权重正则化是通过将权重正则化实例作为关键字参数传递给层来添加的。现在让我们添加L2权重正则化。

```
l2_model = keras.models.Sequential([
    keras.layers.Dense(16, kernel_regularizer=keras.regularizers.l2(0.001),
                       activation='relu', input_shape=(NUM_WORDS,)),
    keras.layers.Dense(16, kernel_regularizer=keras.regularizers.l2(0.001),
                       activation='relu'),
    keras.layers.Dense(1, activation='sigmoid')
])

l2_model.compile(optimizer='adam',
                 loss='binary_crossentropy',
                 metrics=['accuracy', 'binary_crossentropy'])

l2_model_history = l2_model.fit(train_data, train_labels,
                                epochs=20,
                                batch_size=512,
                                validation_data=(test_data, test_labels),
                                verbose=2)
```

对比结果：

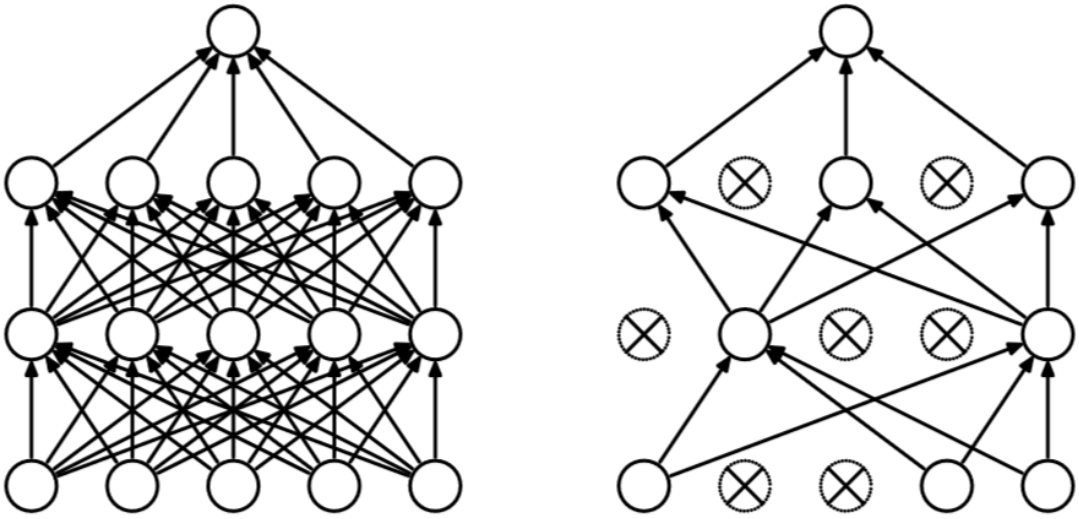


如您所见，L2正则化模型比基础模型更能抵抗过度拟合，即使两个模型的参数数量相同。

Add dropout

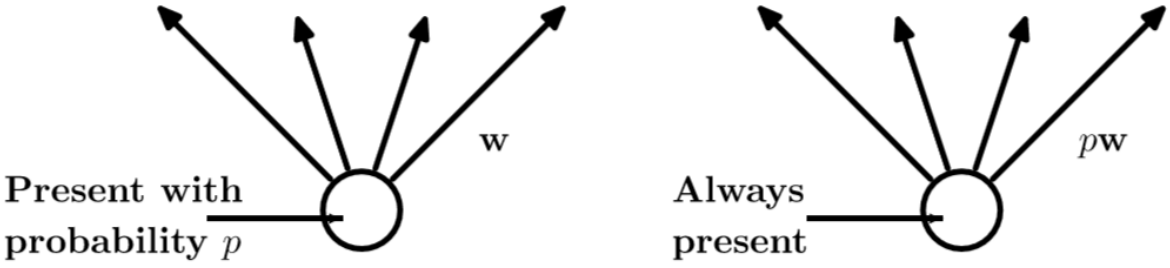
Dropout 是最有效和最常用的神经网络正则化技术之一，由辛顿和他的学生在多伦多大学开发。Dropout应用于层的包括在训练期间随机“Dropout out”（即设置为零）层的许多输出特征。假设一个给定的层通常会在训练期间为给定的输入样本返回一个向量[0.2，0.5，1.3，0.8，1.1]；应用退出后，这个向量将有几个随机分布的零项，例如[0，0.5，1.3，0，1.1]。“退出率”是被归零的特征的分数；它通常设置在0.2到0.5之间。在测试时，没有单元被丢弃，相反，层的输出值被一个等于丢弃率的因子缩小，以便平衡比训练时更多的单元处于活动状态的事实。

在tf.keras中，可以通过dropout层在网络中引入dropout，该层应用于之前层的输出。Drapout 方法原理图：



(a) Standard Neural Net　　(b) After applying dropout.

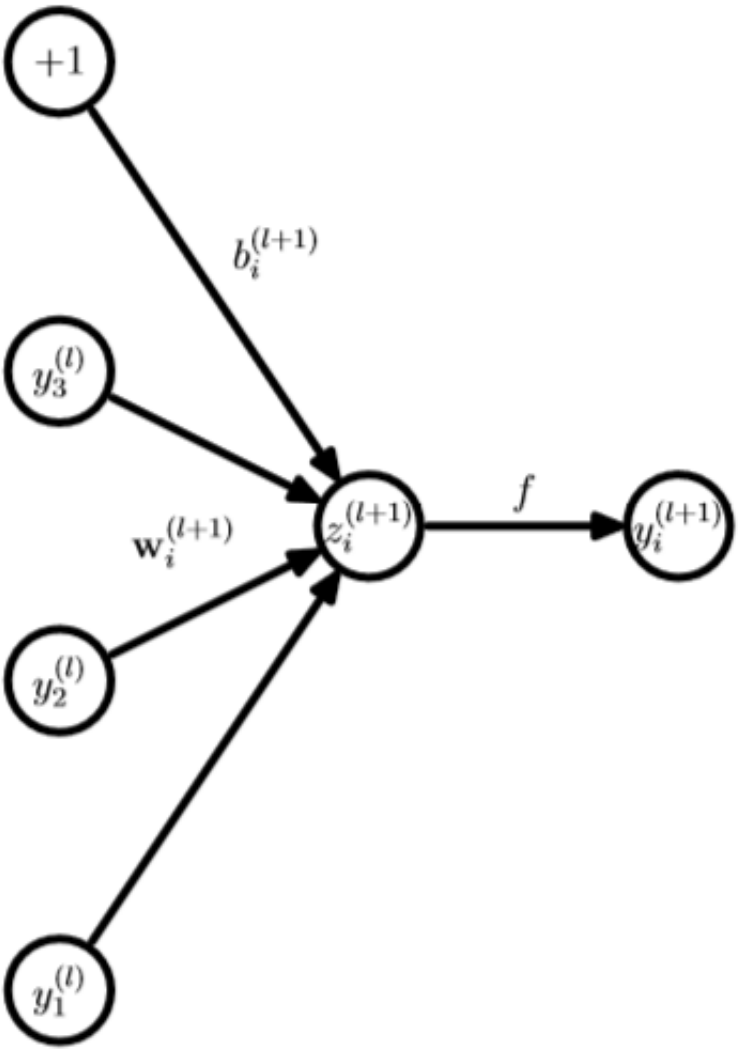
左边是标准神经网络，右边是使用Dropout的神经网络，可见只是连接度少了一些，并不影响模型继续训练。人类通过成千上万年进化，依然保持着这样的繁衍方式：男人贡献一半基因序列，女人贡献一半基因序列，最后组成后代的完整基因序列。当我们观察男人基因序列中的一个基因片段，它不仅要和男人基因序列很好地组合与配合，在繁衍后代时，也要和女人的那一半基因序列组合和配合，这个较好的一个基因片段一定要在两个情况下都很好的适应才行，这正像神经网络中的一个神经元，它要在各个情况下都很好地适应训练，所有，我们要Dropout一些神经元啊！



(a) At training time　　(b) At test time

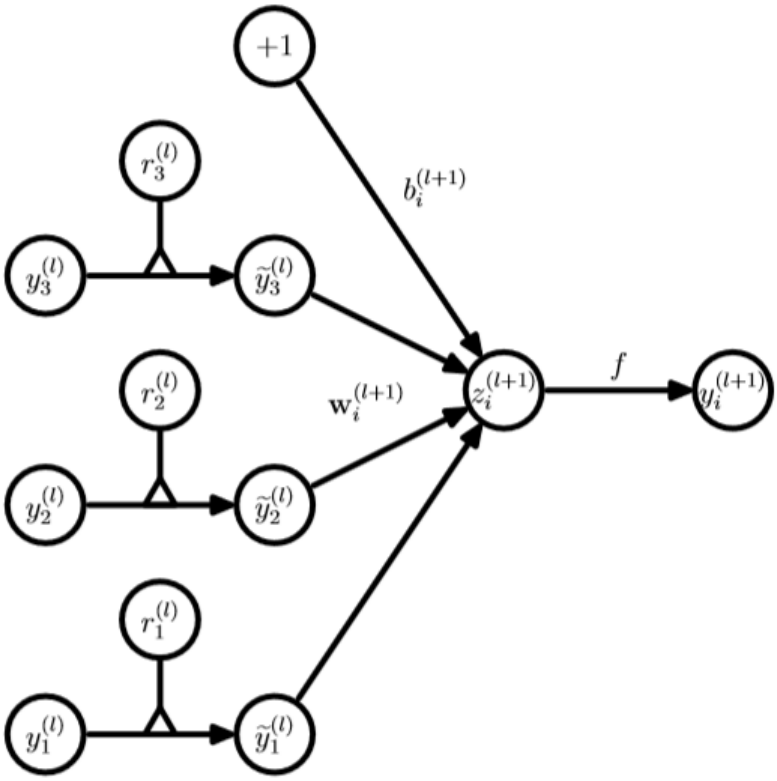
需要指出的是，在训练时，权重参数是共享的。就是说，只要连接权重的神经单元不dropout，那么每次调参时，权重接着调整参数值。换句话说，权重参数个数和不用Dropout的神经网络参数个数是一样的。不同的是，见上图，在训练时，每个神经单元都可能以概率去除；在测试阶段，每个神经单元都是存在的，权重参数要与之相乘：。

每层Dropout网络和传统网络计算的不同之处：



(a) Standard network

$$
\begin{aligned}
z_i^{(l+1)} &= \mathbf{w}_i^{(l+1)} \mathbf{y}^l + b_i^{(l+1)}, \\
y_i^{(l+1)} &= f(z_i^{(l+1)}),
\end{aligned}
$$



(b) Dropout network

$$
\begin{aligned}
r_j^{(l)} &\sim \text{Bernoulli}(p), \\
\tilde{\mathbf{y}}^{(l)} &= \mathbf{r}^{(l)} * \mathbf{y}^{(l)}, \\
z_i^{(l+1)} &= \mathbf{w}_i^{(l+1)} \tilde{\mathbf{y}}^{l} + b_i^{(l+1)}, \\
y_i^{(l+1)} &= f(z_i^{(l+1)}).
\end{aligned}
$$

让我们在IMDB网络中添加两个Dropout层，看看它们在减少过度拟合方面有多好：

接下来看 L2 正则和 Dropout 还有基础方法对数据模型的运行结果：

总结一下：以下是防止神经网络过度拟合的最常用方法：

- 获取更多训练数据。
- 减少网络层数和复杂度。
- 增加权重正则化。
- Dropout 方法。