

# 1.LSTM实验部分

## \* 数据集介绍

本次实验是一个简单的情感分类问题，给定一句话，判断是否有恶意。数据集来源([https://drive.google.com/uc?export=download&id=1dPHl8ZnfDz\\_fxNd2ZeBYedTat2lfxco](https://drive.google.com/uc?export=download&id=1dPHl8ZnfDz_fxNd2ZeBYedTat2lfxco)),其中包含有标注训练集、未标注训练集以及测试集。

## \* 数据处理

首先，将模型中出现的单词都映射成具有对应(feature)的vector，具体的方法可以使用Skip-grams(SG).

```
def train_word2vec(x):  
    # 训练 word to vector 的 word embedding  
    model = word2vec.Word2Vec(x, size=250, window=5, min_count=5, workers=12, iter=10, sg=1)  
    return model
```

然后再将这个model保存下来。

## \* LSTM模型搭建

通过上一步训练得到的模型，将train\_x转换为我们需要的数据形式，并且令每个词向量的长度是固定的。以训练集为例，得到的结果如下：

```
loading data ...  
Get embedding ...  
loading word to vec model ...  
get words #16551  
total words: 16553  
embedding.shape: torch.Size([16553, 250])  
train_x shape: torch.Size([200000, 20])
```

统计得到的总次数为16553，最后的输出结果的维度为250；训练集中的20000条数据，令每句话的词数为20；总取隐含层的最后一层，作为词向量的feature，输出的维度为150,作为LSTM网络的inputs。

根据embedding矩阵的参数，搭建如下的网络：

```

class LSTM_Net(nn.Module):
    def __init__(self, embedding, embedding_dim, hidden_dim, num_layers, dropout=0.5, fix_embedding=False):
        super(LSTM_Net, self).__init__()
        # 製作 embedding layer
        self.embedding = torch.nn.Embedding(embedding.size(0), embedding.size(1))
        self.embedding.weight = torch.nn.Parameter(embedding)
        # 是否將 embedding fix 住, 如果 fix_embedding 為 False, 在訓練過程中, embedding 也會跟著被訓練
        self.embedding.weight.requires_grad = False if fix_embedding else True
        self.embedding_dim = embedding.size(1)
        self.hidden_dim = hidden_dim
        self.num_layers = num_layers
        self.dropout = dropout
        self.lstm = nn.LSTM(embedding_dim, hidden_dim, num_layers=num_layers, batch_first=True)
        self.classifier = nn.Sequential( nn.Dropout(dropout),
                                          nn.Linear(hidden_dim, 1),
                                          nn.Sigmoid() )

```

关于最后的DNN，在全连接层先调用dropout()函数，防止模型过拟合；然后调用linear函数，输入为hidden\_dim，也就是feature的维度。最后对模型进行训练，得到的模型在验证集上的准确率有80.339%。

```

Train | Loss:0.50074 Acc: 74.793
Valid | Loss:0.45170 Acc: 78.254
saving model with acc 78.254
-----

```

```

Train | Loss:0.44363 Acc: 79.062
Valid | Loss:0.43624 Acc: 79.274
saving model with acc 79.274
-----

```

```

Train | Loss:0.42666 Acc: 80.148
Valid | Loss:0.43122 Acc: 79.668
saving model with acc 79.668
-----

```

```

Train | Loss:0.41471 Acc: 80.879
Valid | Loss:0.42220 Acc: 80.051
saving model with acc 80.051
-----

```

```

Train | Loss:0.40300 Acc: 81.423
Valid | Loss:0.42075 Acc: 80.339
saving model with acc 80.339
-----

```

## 2.Explainable/Interpretable ML

### \* why Explainable/Interpretable ML

深度学习中的模型对我们来讲，一般都是黑箱模型，大多时候我们并不知道它为什么会得到这样的结论，所以需要设计这样的ML模型，使得它能够告诉我们真的有学到东西。线性模型是可解释的，所以我们可以通过线性模型，尝试去解释神经网络模型。

### \* Explain the Decision

假设有一个图像分类的模型，当我们想要知道为什么这个分类器会做出这样的决策，比如判断图片中有一只猫，或者机器为什么认为这个特征是重要的，可以通过一种比较基础的方法：

$$\text{object } x \rightarrow \text{Components} : \{x_1, x_2, \dots, x_n\}$$

这里的 $x_i$ 可以是像素；在文本分类问题中， $x_i$ 可以是一个word。我们可以选择改变其中一个 $x$ ，然后观察对最终的决策是否有很大的影响，如果是就说明这是很重要的特征，也是机器判断的标准之一，这样我们也就部分解释了这个模型。

### \* Limitation of Gradient based Approaches

如果只是单纯考虑目标函数对特征的导数的大小，那么会遇到很大的局限。在处理现实问题中，一般会遇到梯度饱和(Gradient Saturation)的问题，所以不能单纯地考虑梯度值的大小。为了解决这个问题，可以考虑参考Integrated gradient、DeepLIFT。

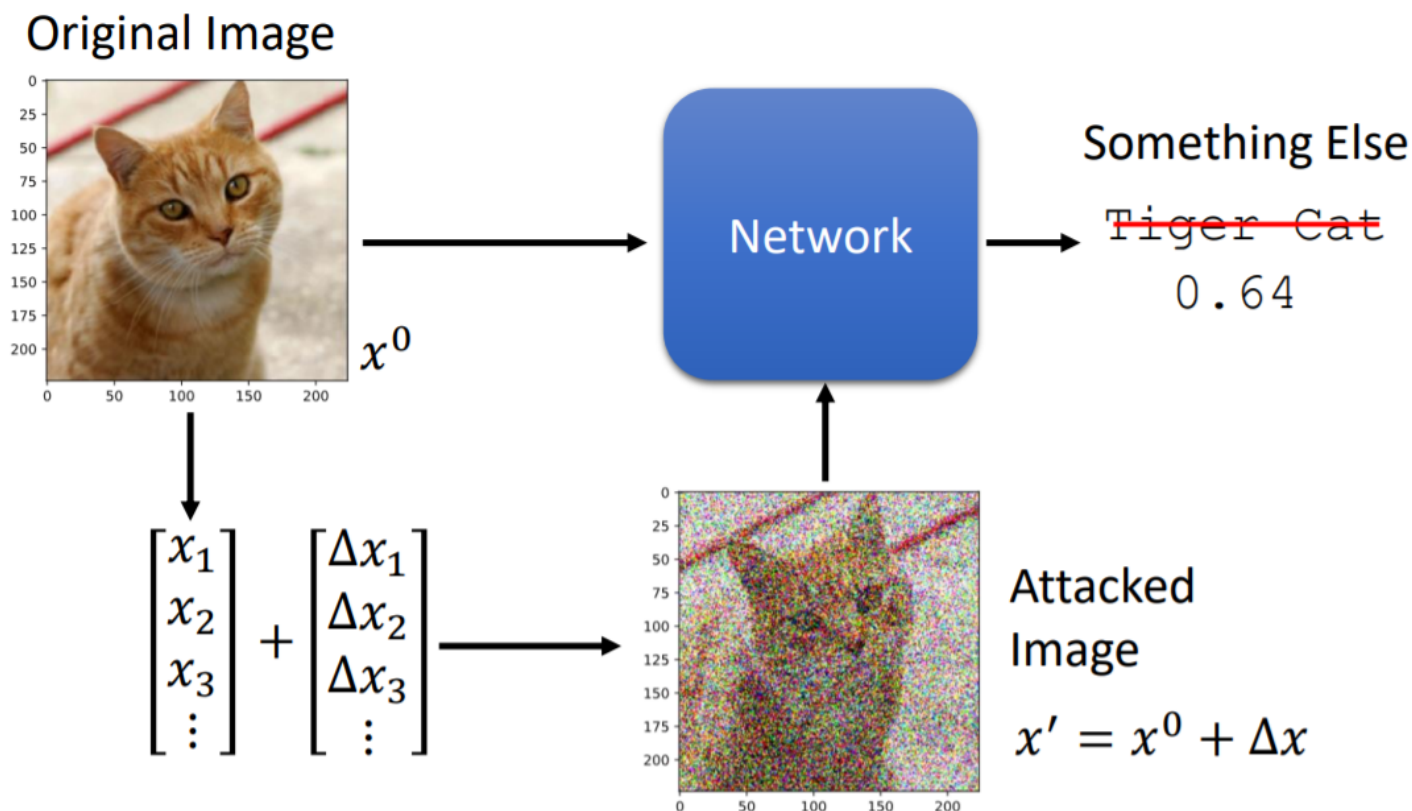
### \* Using a model to explain another

通常我们没法用线性模型完整地解释一个神经网络模型，但是，可以去模仿神经网络的局部。比如我们可以给出神经网络模型计算的损失函数值函数图像，然后用线性模型去模仿局部的图像，从而尝试去解释。

## 3.Adversarial Attack

### \* 动机

设计深度学习的模型，不仅仅需要考虑它是否有效，而且还要考虑它能否抵御来自第三方的恶意攻击。假设有一张图片，图片中有一只小猫，在正常的情况下，分类器可以很好得判断出图片中的动物，但是当外界给图片加上一些噪音时，分类器就不能正确判断了。



经过修改的图片是可以很明显的看出添加过噪音的。

## \* Loss Function for Attack

一般为了不让设计者发现训练数据是经过修改的，所以会增加一个限制： $d(x^0, x') < \epsilon$ , 假设模型训练的目标函数是这样的：

$$L_{train}(\theta) = C(y^0, y^{true})$$

那么可以设计这样的目标函数：

$$L(x') = -C(y', y^{true}) + C(y', y^{false})$$

其中， $-C(y', y^{true})$ 的绝对值需要尽量变大， $C(y', y^{false})$ 的值需要尽量小。关于 $d(x^0, x')$ ,主要有两种计算方式，取决于应用的领域。

## \* How to Attack

原理上和原来的Gradient decent方法是一样的，只是将原来的参数 $\theta$ 替换为精心修改过的 $x'$ ：

$$x^* = \arg \min L(x')$$

设计如下的算法：

```
Start from origin image  $x^0$ 
For  $t = 1$  to  $T$ 
   $x^t \leftarrow x^{t-1} - \eta \nabla L(x^{t-1})$ 
  if  $d(x^0, d^t) > \epsilon$ 
     $x^t \leftarrow \text{fix}(x^t)$ 
```

其中， $\text{fix}()$ 函数的定义如下：

```
def  $\text{fix}(x^t)$ :
  For all  $x$  fulfill  $d(x^0, d^1) < \epsilon$ 
  return the one closest to  $x^t$ 
```

## \* Attack Approaches

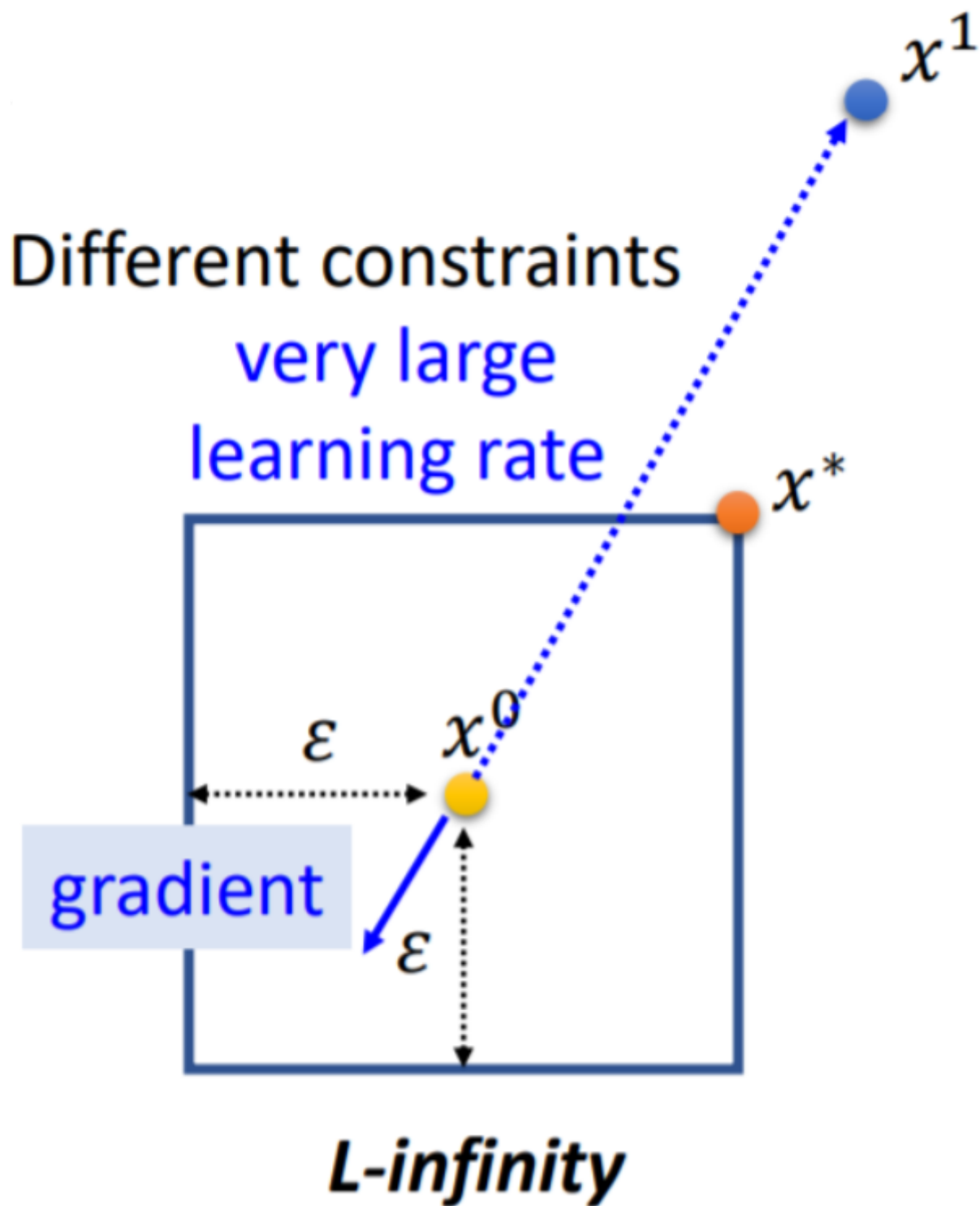
要设计出能够攻击深度学习模型的方法，主要是针对两个方面进行思考：一方面是考虑设计不同优化参数的方法，去最小化 $\min L(x')$ 的；另一方面是去考虑设计不同的限制方法。下面给出一种具体的攻击方法。

- Fast Gradient Sign Method (FGSM)

$$x^* \leftarrow x^0 - \epsilon \Delta x$$

$$\Delta x = \begin{Bmatrix} \text{sign}(\partial L / \partial x_1) \\ \text{sign}(\partial L / \partial x_2) \\ \text{sign}(\partial L / \partial x_3) \\ \vdots \end{Bmatrix}$$

其中， $\Delta x$ 只有1或-1。假设原来的算法找到的梯度方向为 $\text{gradient}$ ，那么需要更新的梯度方向是 $-\text{gradient}$ ，并且让学习率特别大，最后更新的参数落在限制范围外即可。



如上图所示，我们计算得到了 $gradient$ ，当前的学习率为 $\epsilon$ ，那么我们将梯度取反，然后给一个非常大的学习率，那么就会得到 $x^1$ ，又因为 $x^1$ 是不满足限制条件的，所以 $def(x^t)$ 会自动让参数落在 $x^*$ 。

## \* Defense

- Passive defense: 以图片分类为例，被动的防御模式就是模型出错时，尽可能得找到出错的数据，不去修改自己的模型。如果是非常特殊的数据，一般是可以检查出来的。

- Proactive defense: 通过训练一个模型, 使得它足够强大, 可以自己找出漏洞, 让模型自己尽可能去抵御。

假设有一组训练数据:  $\{(x_1, \hat{y}_1), (x_2, \hat{y}_2), \dots, (x_n, \hat{y}_n)\}$ , 然后使用  $X$  去训练你的模型。

*For  $t = 1$  to  $T$  :*

*For  $n = 1$  to  $N$  :*

*Find adversarial input  $\tilde{x}_n$  given by an attack algorithm.*

*We have new training data :*

$$X' = \{(\tilde{x}_1, \hat{y}_1), (\tilde{x}_2, \hat{y}_2), \dots, (\tilde{x}_n, \hat{y}_n)\}$$

*Using both  $X'$  to update your model.*