# 3.6-A 从 RNN 到 Transformer 和 GPT，从自然语言处理到视觉模型

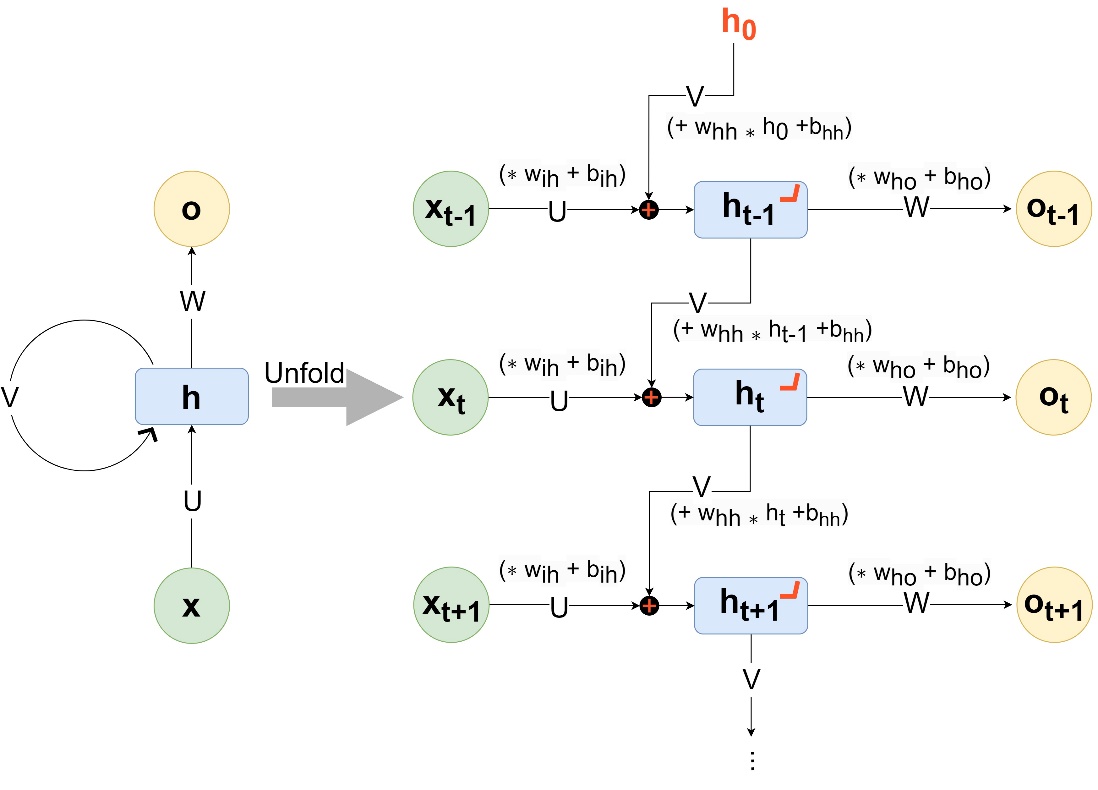
## 3.6.1 从原始的 RNN 到 LSTM

### 3.6.1.1 RNN（Recurrent Neural Networks）

当处理序列数据（sequential data），例如*时空序列分析*部分提到的 AoT 城市环境传感器时间序列数据，*动态街景视觉感知*部分提到的用于无人驾驶场景下计算机视觉算法评测数据集 KITTI，*空间动力学——空间马尔可夫链*部分提到的历年 MCD12Q1\_v006 lulc 数据，及各类型连续时间的遥感影像数据、实时城市活动动态（交通流、信息流、社交媒体等各类热力图）等，其各样本数据点（data point）并不是独立的，例如当前污染气体的浓度依赖于前一时刻或之前更长时间段内的浓度（时间相关），沿城市道路后一位置点的城市属性（例如 POI、人口密度、LULC 等）依赖于前一位置点的属性（空间相关）等，那么处理数据点之间具有独立性假设的一般标准的神经网络用于序列数据则会丢失网络的整个状态，即丢失数据点之间的时空相关性。

在*空间马尔可夫链*部分解释了经典离散（时间）马尔可夫链（Discrete-time Markov chain，D(T)MC）和空间马尔科夫 （Spatial Markov），模拟了观测序列中状态之间的转换；在*马尔可夫随机场（MRF）和城市降温模型的 MRF 推理*部分解释了马尔可夫网（马尔科夫随机场）。发展于马尔可夫链的隐马尔可夫模型（Hidden Markov Model），用来描述含有隐含未知参数（不可观测到的状态）的马尔可夫过程，将观测到的序列建模为依赖于未观测到的一系列状态的概率。但是传统的马尔可夫模型的计算量与各随机变量的状态数量呈指数增长，且每个隐藏状态通常只能依赖于上一个状态，而递归（循环）神经网络（Recurrent Neural Networks，RNN）可以捕获更长宽度（long-range）语境/上下文（context）下的时空依赖性（其任何时刻的隐藏状态都可以包含来自任意长的语境窗口（arbitrarily long context window））；并且网络的计算量并不会指数增长[1]。

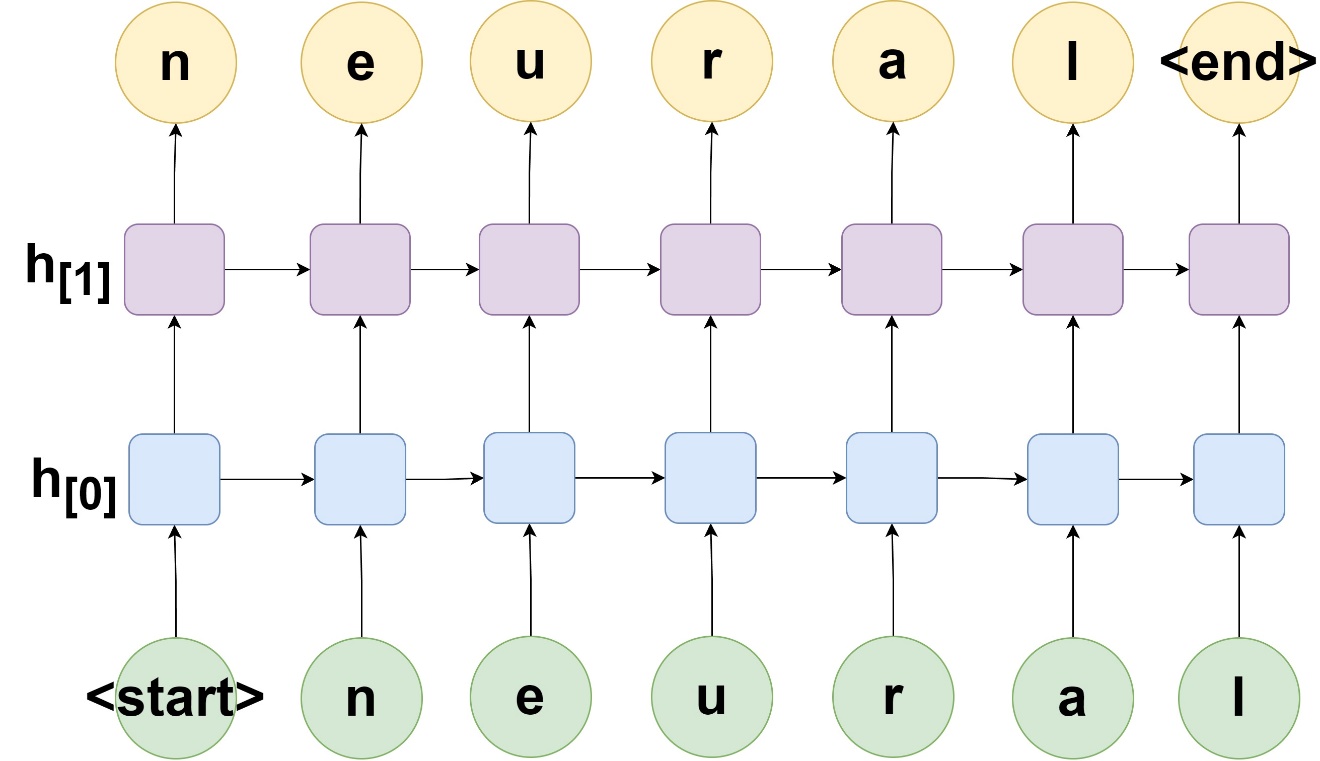
RNN 一般为前馈神经网络（feedforward neural networks），从算法解析的图中可以观察到，在时刻（时间步，或空间位置），隐藏节点（hidden node）（隐藏状态）的值来源于两个分支，一个是当前时刻的数据点输入乘以权重值加上偏置；另一个分支是前一时刻隐藏状态的值乘以权重加上偏置。通过激活函数（通常用 Relu 和 Tanh 实现非线性）的两个分支的和即为当前时刻隐藏状态的值，各个输入的数据点和隐藏层的值计算同上。对于开始时刻输入的隐藏状态的值通常配置为张量0。这样就构建了当前时刻的状态与其前一时刻状态的关联。各时刻的输出值为对应隐藏状态值乘以权重加上偏置获得。对于输入序列的每一个元素（数据点），各层计算公式可以写作[2]，式中，是当前时刻的隐藏状态（hidden state），是当前时刻的输入，是前一时刻隐藏状态或者为时刻 0 时的初始状态。



%load\_ext autoreload   
%autoreload 2   
import os  
os.environ['KMP\_DUPLICATE\_LIB\_OK']='True'  
   
import usda.data\_visual as usda\_vis   
import usda.utils as usda\_utils  
import usda.models as usda\_models  
  
import torch  
from torch import nn  
from torchvision import datasets, transforms  
from torch.utils.data import DataLoader  
from torch.autograd import Variable  
  
import matplotlib.pyplot as plt  
import numpy as np

* 多层 RNN

PyTorch库包含有 RNN 网络，下述代码配置了其输入参数num\_layers=2，该参数为递归（循环）的层数，意味着两个 RNN 堆叠在一起构成叠合的 RNN（stackedRNN），第2个RNN以第1个RNN的输出为输入，第2个RNN的输出为最终的输出，如下网络结构示例图[3]。



以一个文本序列neural输入为例，首先将文本字符串转换为整数标签，并通过除以最大字符标签数将其标准化，且调整形状为(1,6,1)，为1个样本下有6个数据点，每次输入一个数据点，即特征数参数input\_size为1。并配置隐藏状态的特征数hidden\_size为5，递归层数num\_layers为2，实例初始化nn.RNN后，输入neural整数化的一个样本。从输出结果可以观察到，output即为第2个 RNN 的输出结果。

首先构建样本数据。

raw\_text=r'neural'.lower()  
chars=sorted(list(set(raw\_text)))  
char\_to\_int=dict((c, i) for i, c in enumerate(chars))  
  
X=torch.tensor([char\_to\_int[char] for char in raw\_text],dtype=torch.float32).reshape(1,len(raw\_text),1)/len(chars)  
print(X.shape,'\n',X)

torch.Size([1, 6, 1])   
 tensor([[[0.5000],  
 [0.1667],  
 [0.8333],  
 [0.6667],  
 [0.0000],  
 [0.3333]]])

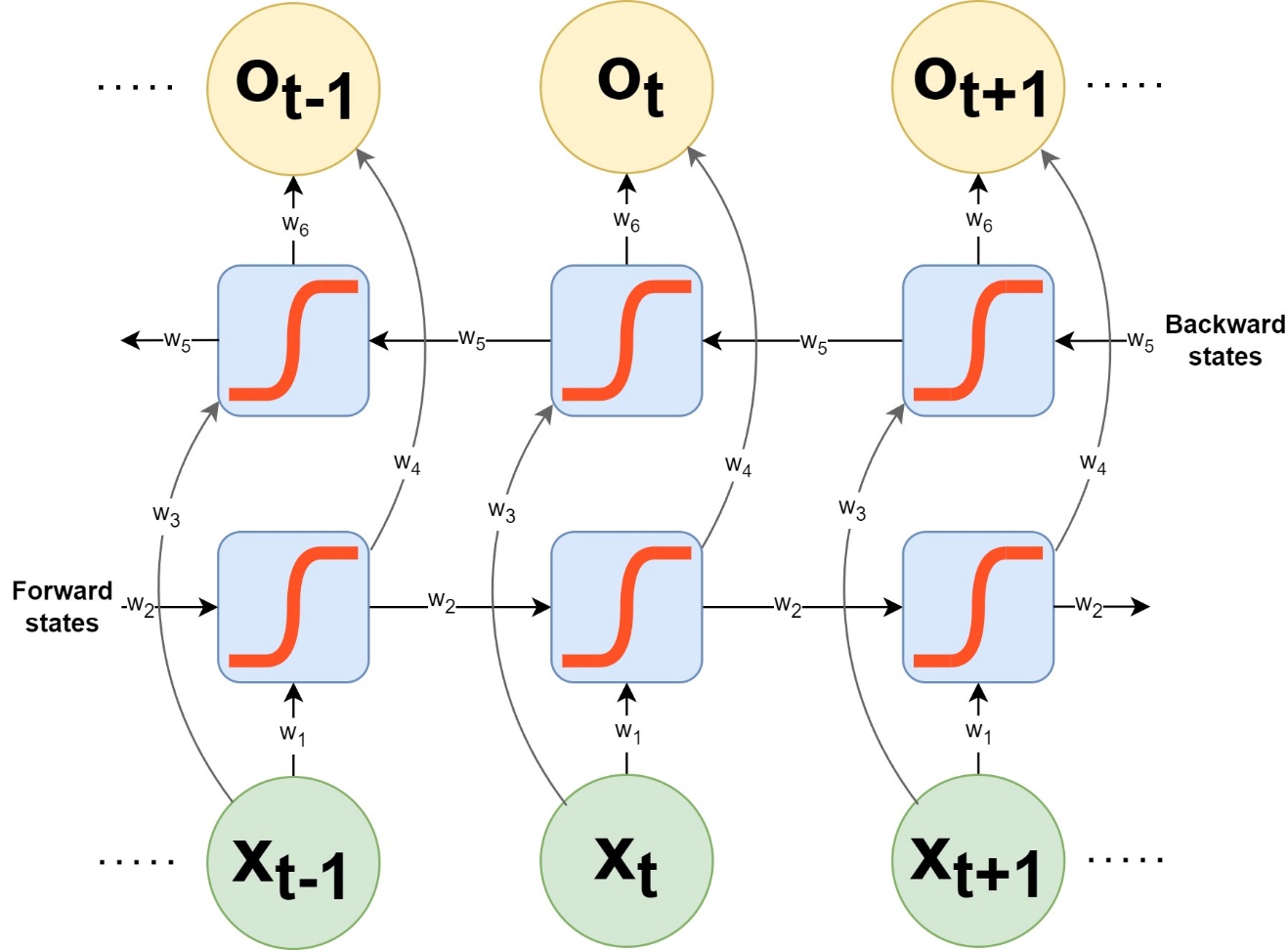
激活函数默认配置为Tanh，对应参数nonlinearity，值域为-1到1。

input\_size=1  
hidden\_size=3  
num\_layers=2  
  
rnn=nn.RNN(input\_size, hidden\_size, num\_layers)  
print(rnn)  
h0=torch.randn(num\_layers, X.shape[1], hidden\_size)  
output, hn=rnn(X, h0)  
print(output.shape,hn.shape)  
print(output,'\n','-'\*50,'\n',hn)

RNN(1, 3, num\_layers=2)  
torch.Size([1, 6, 3]) torch.Size([2, 6, 3])  
tensor([[[-0.4898, -0.0451, 0.5158],  
 [-0.2987, 0.5905, 0.2265],  
 [-0.0492, -0.7288, -0.2116],  
 [-0.4321, -0.2413, -0.7025],  
 [-0.4563, 0.5862, 0.0823],  
 [-0.0016, 0.0199, 0.2315]]], grad\_fn=<StackBackward0>)   
 --------------------------------------------------   
 tensor([[[-0.1867, 0.0805, 0.5953],  
 [-0.5911, 0.9160, 0.3573],  
 [-0.7828, 0.2346, 0.8274],  
 [-0.5516, 0.2513, 0.8900],  
 [-0.7596, 0.9361, 0.2073],  
 [-0.4027, 0.8579, 0.4963]],  
  
 [[-0.4898, -0.0451, 0.5158],  
 [-0.2987, 0.5905, 0.2265],  
 [-0.0492, -0.7288, -0.2116],  
 [-0.4321, -0.2413, -0.7025],  
 [-0.4563, 0.5862, 0.0823],  
 [-0.0016, 0.0199, 0.2315]]], grad\_fn=<StackBackward0>)

* 双向递归神经网络 BRNN

对于许多序列标记任务，不仅是当前时刻从过往时刻的数据点获得信息，往往也需要从将来时刻的数据点获得信息，因此Schuster, M. 等人提出了双向递归神经网络（Bidirectional Recurrent Neural Networks，BRNN），可以使用特定时间范围内过去和未来的所有可用输入信息进行训练[4]。BRNN 的基本思想是将每个训练序列向前和向后表述为两个独立的递归隐藏层，这两个隐藏层都连接到同一个输出层，如图[5]26。



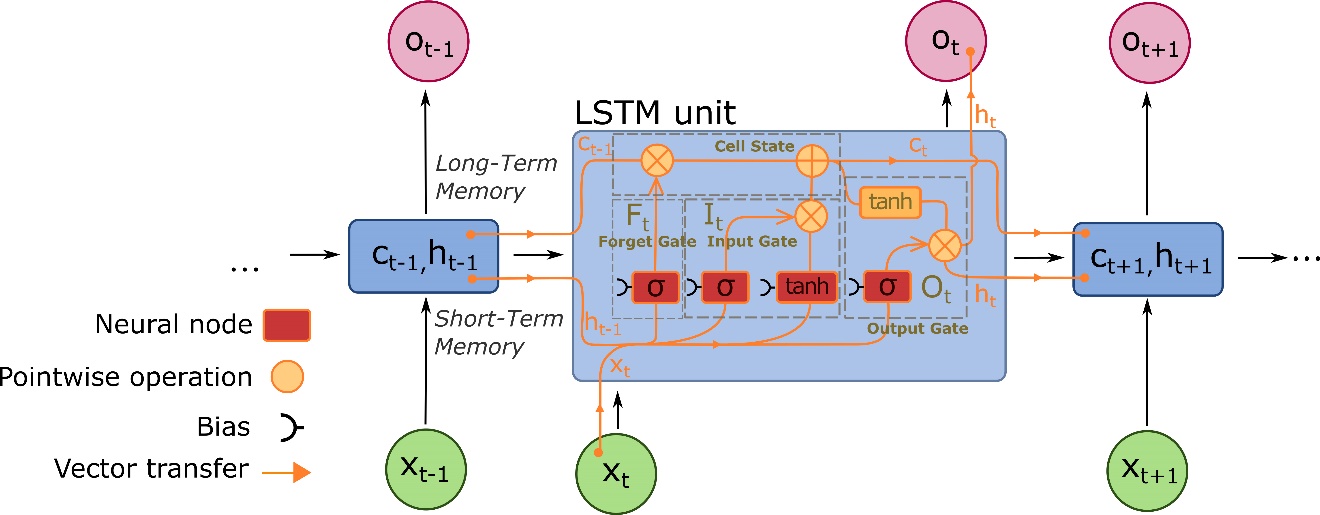
PyTorch提供的 RNN 方法中可以直接配置bidirectional=True实现 BRNN。隐藏节点初始化的张量h\_0形状为，当配置 RNN 网络为 BRNN 时，其。下述试验配置num\_layers=1，可以发现训练序列的每一节点的输出值output为隐藏状态的2倍；隐藏状态hn包含合并的前向和后向两个独立隐藏层的张量。

num\_layers=1  
rnn=nn.RNN(input\_size, hidden\_size, num\_layers,bidirectional=True)  
print(rnn)  
h0=torch.randn(num\_layers\*2, X.shape[1], hidden\_size)  
output, hn=rnn(X, h0)  
print(output.shape,hn.shape)  
print(output,'\n','-'\*50,'\n',hn)

RNN(1, 3, bidirectional=True)  
torch.Size([1, 6, 6]) torch.Size([2, 6, 3])  
tensor([[[-0.0948, 0.8778, -0.6792, -0.4503, 0.2629, 0.1734],  
 [-0.2018, 0.7779, -0.1665, -0.9079, 0.8410, 0.4358],  
 [ 0.3702, 0.5908, 0.4314, 0.3850, -0.6145, 0.1485],  
 [ 0.8645, 0.0013, -0.4648, -0.7760, 0.0113, 0.8661],  
 [-0.0232, 0.4756, 0.6359, 0.2150, -0.0758, -0.1035],  
 [ 0.8035, -0.1572, 0.0789, 0.8305, -0.9462, 0.4790]]],  
 grad\_fn=<CatBackward0>)   
 --------------------------------------------------   
 tensor([[[-0.0948, 0.8778, -0.6792],  
 [-0.2018, 0.7779, -0.1665],  
 [ 0.3702, 0.5908, 0.4314],  
 [ 0.8645, 0.0013, -0.4648],  
 [-0.0232, 0.4756, 0.6359],  
 [ 0.8035, -0.1572, 0.0789]],  
  
 [[-0.4503, 0.2629, 0.1734],  
 [-0.9079, 0.8410, 0.4358],  
 [ 0.3850, -0.6145, 0.1485],  
 [-0.7760, 0.0113, 0.8661],  
 [ 0.2150, -0.0758, -0.1035],  
 [ 0.8305, -0.9462, 0.4790]]], grad\_fn=<StackBackward0>)

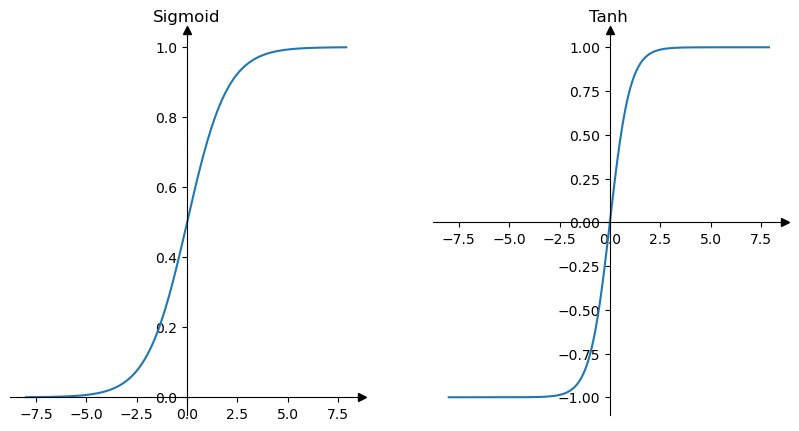
### 3.6.1.2 LSTM（long short-term memory）

对于上述标准的 RNN 模型，输出值与隐藏状态的权重和数据点数有指数增长关系。因此如果隐藏状态的权重大于1，当数据点不断增多时，可能引起梯度爆炸（Gradient explosion）；如果隐藏状态的权重小于1，则可能引起梯度消失（Gradient vanishing）。并且，如果时间跨度很大，当前时刻的状态则并不能学习到较早时刻的输入信息，因此 Hochreiter, S.和 Schmidhuber, J. 提出了长短时记忆网络（long short-term memory，LSTM）[6]。LSTM 类似于一个带有隐藏层的标准递归神经网络，但是隐藏层中的每个普通节点被一个记忆单元（memory cell）所取代，即 LSTM 单元（LSTM unit）。每个记忆单元包含一个具有固定权重为1的自连接递归边（self-connected recurrent edge），确保梯度可以跨越多个时间点而不会消失或爆炸，为含符号的顶部边表示的长时记忆（Long-Term Memory），即单元状态（Cell State），如图[7,8]；短时记忆（Short-Term Memory）为含符号，有权重的底部边，即隐藏状态（Hidden State）。



LSTM 通过在记忆单元中引入门（gate）机制控制特征的流转和过滤，解决“长期依赖（long-term dependencies）问题”。时刻的输入为前时刻的输出，包括长时记忆（单元状态）和短时记忆（隐藏状态）。在记忆单元中包含有三个控制门，为遗忘门（Forget Gate）、输入门（Input Gate）和输出门（Output Gate）。在过滤特征信息时使用 Sigmoid 激活函数（$$），结合乘法实现；特征的非线性激活函数使用 Tanh。关于激活函数的解释可以查看*从解析解到数值解，从机器学习到深度学习*部分，下述打印了两个激活函数的曲线，方便观察数值区间变化，理解特征的删选机制。

fig, axes=plt.subplots(1, 2,figsize=(10,5))  
  
x=torch.arange(-8.0,8.0,0.1,requires\_grad=True)  
y\_relu=x.sigmoid()  
axes[0].plot(x.detach().numpy(), y\_relu.detach().numpy(),label="ReLU")  
  
y\_tanh=x.tanh()  
axes[1].plot(x.detach().numpy(), y\_tanh.detach().numpy(),label="tanh")  
  
  
axes[0].set\_title('Sigmoid')  
axes[1].set\_title('Tanh')  
usda\_vis.plot\_style\_axis\_A(axes[0])  
usda\_vis.plot\_style\_axis\_A(axes[1])  
plt.show()



**遗忘门：**

遗忘门的计算公式为x\_ttht-1WifWhfbifbhf

ct-1乘积会保持前一单元状态基本不变，意味着信息流通；否则趋近于0，意味着前一单元状态的特征信息被忽略。

**输入门：**

输入门计算为两部分的乘积，一部分同遗忘门，用于删选特征信息；另一部分为隐藏节点，对应公式分别为 和。此时可以更新当前时刻的单元状态为，式中为哈达玛积（Hadamard product），为两个矩阵对应元素相乘的结果。因此输入门通过更新单元状态将新的相关信息添加到了现有信息中。

**输出门**

输入门计算也为两部分的乘积，一部分为用于删选特征信息的 Sigmoid 函数，公式为；另一部分为更新的当前时刻的单元状态。计算输出值公式为。输出门更新了隐藏状态，同单元状态作为下一时刻的输入。

继续前文试验，将 RNN 替换为 LSTM，并增加单元状态初始张量，计算结果如下。

input\_size=1  
hidden\_size=3  
num\_layers=1  
  
rnn=nn.LSTM(input\_size, hidden\_size, num\_layers)  
print(rnn)  
h0=torch.randn(num\_layers, X.shape[1], hidden\_size)  
c0=torch.randn(num\_layers, X.shape[1], hidden\_size)  
output, (hn,cn)=rnn(X, (h0,c0))  
print(output.shape,hn.shape,cn.shape)  
print(output,'\n','-'\*50,'\n',hn,'\n',cn)

LSTM(1, 3)  
torch.Size([1, 6, 3]) torch.Size([1, 6, 3]) torch.Size([1, 6, 3])  
tensor([[[-0.2156, 0.2163, -0.2481],  
 [-0.0974, 0.4490, -0.4016],  
 [-0.4627, -0.1346, -0.2763],  
 [-0.0875, -0.3300, -0.0779],  
 [-0.2781, 0.0431, -0.1354],  
 [-0.0718, -0.0208, 0.3109]]], grad\_fn=<MkldnnRnnLayerBackward0>)   
 --------------------------------------------------   
 tensor([[[-0.2156, 0.2163, -0.2481],  
 [-0.0974, 0.4490, -0.4016],  
 [-0.4627, -0.1346, -0.2763],  
 [-0.0875, -0.3300, -0.0779],  
 [-0.2781, 0.0431, -0.1354],  
 [-0.0718, -0.0208, 0.3109]]], grad\_fn=<StackBackward0>)   
 tensor([[[-0.3266, 0.4154, -0.4966],  
 [-0.2703, 0.8698, -0.5718],  
 [-0.6668, -0.3594, -0.5487],  
 [-0.6161, -1.3928, -0.1051],  
 [-0.5705, 0.0955, -0.3340],  
 [-0.1111, -0.0466, 0.5318]]], grad\_fn=<StackBackward0>)

### 3.6.1.3 LSTM 的文本生成试验

前文将neural作为一个示例样本序列传入 RNN（LSTM）模型，该模型能够捕获到neural的隐藏节点状态和单元状态，使用该状态输出作为学习到的特征可以用于序列数据的预测、分类等任务，例如语言建模和文本生成（Language Modelling & Generating Text）、语音识别（Speech Recognition）、生成图像描述（Generating Image Descriptions）或视频标记（Video Tagging）等[9]。用 LSTM 生成文本为例，实现数据集建立、模型构建、训练到文本生成[10]。

%load\_ext autoreload   
%autoreload 2   
import usda.data\_visual as usda\_vis   
import usda.utils as usda\_utils  
import usda.models as usda\_models  
  
import torch  
import torch.nn as nn  
import torch.optim as optim  
import torch.utils.data as data  
from torchvision import datasets, transforms  
from torch.utils.data import DataLoader  
  
import numpy as np  
import os  
import matplotlib.pyplot as plt

The autoreload extension is already loaded. To reload it, use:  
 %reload\_ext autoreload

* 建立序列数据集

文本数据为[Alice’s Adventures in Wonderland by Lewis Carroll](https://www.gutenberg.org/ebooks/11)①，将其转化为指定长度的序列数据集，包括建立字符到整数的映射编码ebook\_dataset.char\_to\_int；按照指定长度切分文本为片段作为样本特征输入，并除以分类数标准化输入特征值；对应的（分类）标签为各序列片段紧邻的下一个跟随字符。例如对于文本What a curious feeling!（长度为23），如果给定切分长度为20，那么提取片段为What a curious feeli，对应的标签为n，并需转换为整数编码和标准化。

建立文本序列数据集的过程定义为text2ints\_encoded()函数，直接调用计算。

filename='../data/Wonderland.txt'   
seq\_length=100  
ebook\_dataset=usda\_models.text2ints\_encoded(filename,seq\_length)  
X=ebook\_dataset.data  
y=ebook\_dataset.target  
print(X.shape,y.shape)

Total Characters: 164047  
Total Vocab: 64  
Total Patterns: 163947  
torch.Size([163947, 100, 1]) torch.Size([163947])

batch\_size=128  
loader=data.DataLoader(data.TensorDataset(X, y), shuffle=True, batch\_size=batch\_size)

查看字符到整数的映射，和一个样本序列及其对应的标签。

print(ebook\_dataset.char\_to\_int)  
for X\_batch,y\_batch in loader:  
 x=X\_batch[0]  
 y=y\_batch[0]  
 print(x[:5],y)   
 break

{'\n': 0, ' ': 1, '!': 2, '"': 3, '#': 4, '$': 5, '%': 6, "'": 7, '(': 8, ')': 9, '\*': 10, ',': 11, '-': 12, '.': 13, '/': 14, '0': 15, '1': 16, '2': 17, '3': 18, '4': 19, '5': 20, '6': 21, '7': 22, '8': 23, '9': 24, ':': 25, ';': 26, '?': 27, '[': 28, ']': 29, '\_': 30, 'a': 31, 'b': 32, 'c': 33, 'd': 34, 'e': 35, 'f': 36, 'g': 37, 'h': 38, 'i': 39, 'j': 40, 'k': 41, 'l': 42, 'm': 43, 'n': 44, 'o': 45, 'p': 46, 'q': 47, 'r': 48, 's': 49, 't': 50, 'u': 51, 'v': 52, 'w': 53, 'x': 54, 'y': 55, 'z': 56, 'ù': 57, '—': 58, '‘': 59, '’': 60, '“': 61, '”': 62, '\ufeff': 63}  
tensor([[0.0156],  
 [0.6562],  
 [0.7031],  
 [0.7031],  
 [0.6406]]) tensor(35)

* 建立 LSTM 网络模型

将 LSTM 计算时序片段（一个语境）的最后一个节点输出状态（包含有最多信息）作为全连接层的输入，全连接层的输出大小为分类数，其输出值可用 SoftMax 回归多分类函数转换为分类类标的概率分布（标准化到区间[0,1]，且总和为1），预测分类。

class CharModel(nn.Module):  
 def \_\_init\_\_(self,n\_vocab,input\_size=1, hidden\_size=256, num\_layers=2, batch\_first=True, dropout\_lstm=0.2,dropout=0.2):  
 super().\_\_init\_\_()  
 self.lstm = nn.LSTM(input\_size=input\_size, hidden\_size=hidden\_size, num\_layers=num\_layers, batch\_first=batch\_first, dropout=dropout\_lstm)  
 self.dropout = nn.Dropout(dropout)  
 self.linear = nn.Linear(hidden\_size, n\_vocab)  
 def forward(self, x):  
 x, \_ = self.lstm(x)  
 # take only the last output  
 x = x[:, -1, :]  
 # produce output  
 x = self.linear(self.dropout(x))  
 return x  
  
n\_vocab=len(ebook\_dataset.feature\_names)  
input\_size=1  
model=CharModel(n\_vocab,input\_size)  
print(model)

CharModel(  
 (lstm): LSTM(1, 256, num\_layers=2, batch\_first=True, dropout=0.2)  
 (dropout): Dropout(p=0.2, inplace=False)  
 (linear): Linear(in\_features=256, out\_features=64, bias=True)  
)

* 训练模型

定义char\_train()函数，方便调用以实现模型训练。训练的模型保存到本地磁盘。

n\_epochs=100  
char\_to\_int=ebook\_dataset.char\_to\_int  
save\_path='../models/ebook\_lstm\_model.pth'  
usda\_models.char\_train(model,loader,n\_epochs,char\_to\_int,save\_path)

* 文本生成

定义char\_random\_generation()函数，调用已训练的模型，根据随机读取的文本片段逐次的预测下一个字符，例如下述示例的提示词最后几个单词为choosing to notice，预测的第一个字符为a，并将该字符追加到提示词中，且剔除提示词的第一个字符，用于下一个字符的预测，以此类推。

对于这个简单的示例模型，文本生成的结果虽然并不完美，但是验证了 LSTM 对序列数据建模预测的可行性。

model=usda\_models.CharModel(n\_vocab,input\_size)  
usda\_models.char\_random\_generation(model,save\_path,filename,seq\_length=100,gen\_length=1000)

Prompt: "ell—eh, stupid?”  
  
“but they were \_in\_ the well,” alice said to the dormouse, not choosing  
to notice "   
 --------------------------------------------------  
and broked.   
“i won’t ” said the caterpillar.  
  
“well, perhaps your pardon!” said the king, “and the tuertion whol you coll the canst and the thing as ‘i get whan it is!”  
  
“i whsh i had so say it any noneer than you can’t hear your majesty,” the mock turtle replied; “what i’s goingng not a perpents and it is the coor with the things beal the reasenese!”  
  
“i don’t think it would not gane iis the sea,” the mock turtle replied; “i’m a poor mittle sime ier any of the eiseations than it was, and the duchess to be a lettrn of the coorersation as the could goes the thatp with with the thing sealed to het on the trees she was not uhink the was not a little way off a little way  
off a little way off a little way off a little way off a little wire iis gead with the thing sealed to het on the thatp, and she was now a little bootersation as the could gor to the way, and she was now a little shree of the garter as she could gor to the way, and she was now a little shree of the garter as she could go

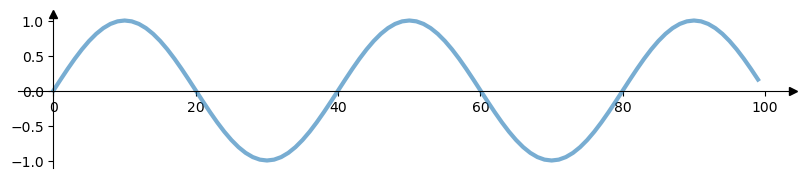
### 3.6.1.4 原始（Vanilla） RNN 模型和 LSTM 模型捕获长期依赖关系的验证

Vanilla RNN 当前时刻的状态输出只与当前数据点输入和前一时刻的隐藏状态有关，而 LSTM 能够捕获前一时刻和更早时刻的状态，这一情况从下述打印的连续训练迭代周期预测曲线可以进一步得以验证[11]。对于Vanilla RNN ，因为只考虑前一时刻隐藏状态，所以预测的值沿着前一隐藏状态和当前输入“顺势”求解；而 LSTM 因为引入了当前时刻之前较长跨度时间步的多个单元状态的信息，在开始的迭代周期，可以发现预测的曲线并不贴合到实际的值上，但是随着训练迭代的进行，预测曲线逐步拟合到实际值上。

* 定义数据集

建立一个符合正弦曲线的序列。

num=1000  
x=torch.linspace(0,num-1,num)  
period=40  
y=torch.sin(x\*2\*np.pi/period)  
  
fig, ax=plt.subplots(1, 1,figsize=(10,2))  
ax.plot(x[:100],y[:100], lw=3, alpha=0.6)  
  
usda\_vis.plot\_style\_axis\_A(ax)  
plt.show()



类似文本生成部分的数据切分为序列片段的方法，给定切分长度获得多个序列片段，并以该片段紧邻的下一时刻的值为类标，按顺序连续叠合切分，定义sequence2series\_of\_overlapping\_with\_labels()函数实现。

test\_size=100  
train\_set=y[:-test\_size]  
test\_set=y[-test\_size:]  
  
window\_size=period  
train\_data=usda\_utils.sequence2series\_of\_overlapping\_with\_labels(train\_set, window\_size)  
train\_data[0]

/

(tensor([ 0.0000e+00, 1.5643e-01, 3.0902e-01, 4.5399e-01, 5.8779e-01,  
 7.0711e-01, 8.0902e-01, 8.9101e-01, 9.5106e-01, 9.8769e-01,  
 1.0000e+00, 9.8769e-01, 9.5106e-01, 8.9101e-01, 8.0902e-01,  
 7.0711e-01, 5.8779e-01, 4.5399e-01, 3.0902e-01, 1.5643e-01,  
 -8.7423e-08, -1.5643e-01, -3.0902e-01, -4.5399e-01, -5.8779e-01,  
 -7.0711e-01, -8.0902e-01, -8.9101e-01, -9.5106e-01, -9.8769e-01,  
 -1.0000e+00, -9.8769e-01, -9.5106e-01, -8.9101e-01, -8.0902e-01,  
 -7.0711e-01, -5.8779e-01, -4.5399e-01, -3.0902e-01, -1.5643e-01]),  
 tensor([1.7485e-07]))

为了获得同一试验结果，固定随机种子。配置学习率。

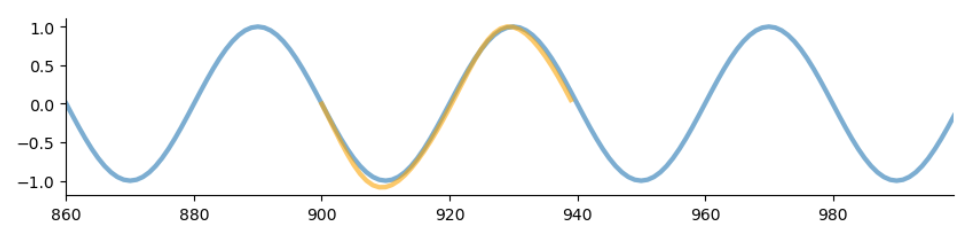
torch.manual\_seed(42)  
lr=0.01

* Vannila RNN 模型

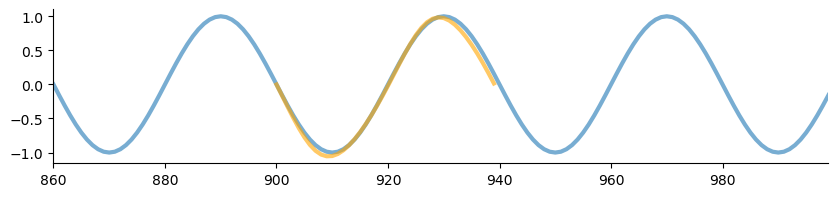
首先试验Vannila RNN 模型。定义试验模型时将 Vannila RNN 和 LSTM 定义在了同一类下，定义的类为RNN\_LSTM\_sequence，通过配置参数selection='RNN'指定为 Vannila RNN 模型。

model=usda\_models.RNN\_LSTM\_sequence(selection='RNN')  
print(model)  
  
criterion=nn.MSELoss()  
optimizer=torch.optim.SGD(model.parameters(),lr=lr)  
usda\_models.RNN\_LSTM\_train\_sequence(model,train\_data,train\_set,x,y,optimizer,criterion,window\_size,test\_size,epochs=3,future=40,plot=True)

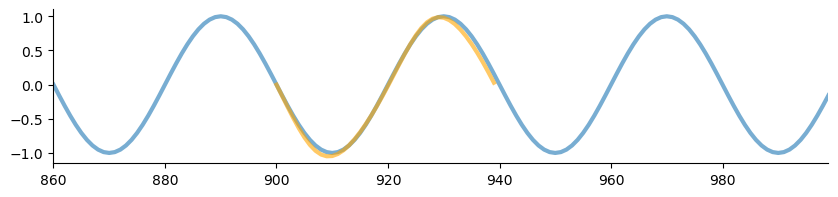
RNN\_LSTM\_sequence(  
 (rnn\_lstm): RNN(1, 50)  
 (linear): Linear(in\_features=50, out\_features=1, bias=True)  
)  
Epoch 0 Loss:8.481218173983507e-06; Performance on test range: 0.003869572188705206



Epoch 1 Loss:1.535474984848406e-05; Performance on test range: 0.00407208688557148



Epoch 2 Loss:1.4694256606162526e-05; Performance on test range: 0.0033605736680328846

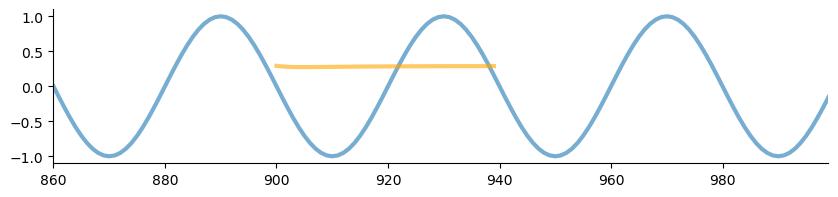


* LSTM 模型

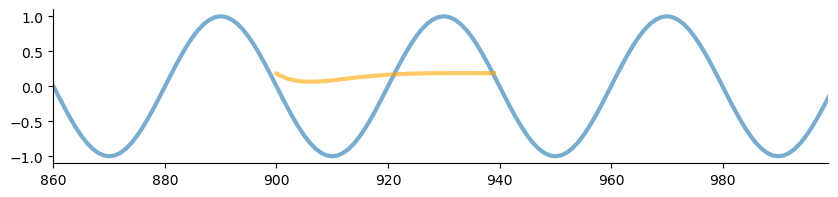
指定selection='LSTM'，分析 LSTM 模型训练迭代预测结果与实际值的变化关系。

model=usda\_models.RNN\_LSTM\_sequence(selection='LSTM')  
print(model)  
  
criterion=nn.MSELoss()  
optimizer=torch.optim.SGD(model.parameters(),lr=lr)  
usda\_models.RNN\_LSTM\_train\_sequence(model,train\_data,train\_set,x,y,optimizer,criterion,window\_size,test\_size,epochs=5,future=40,plot=True)

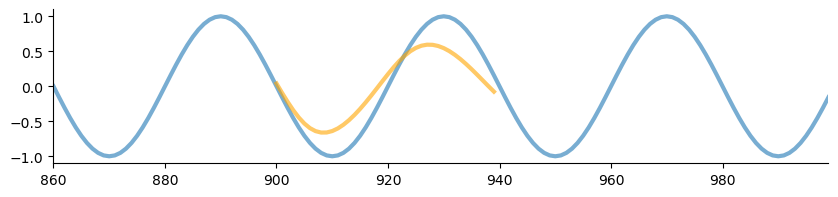
RNN\_LSTM\_sequence(  
 (rnn\_lstm): LSTM(1, 50)  
 (linear): Linear(in\_features=50, out\_features=1, bias=True)  
)  
Epoch 0 Loss:0.07350001484155655; Performance on test range: 0.574382483959198



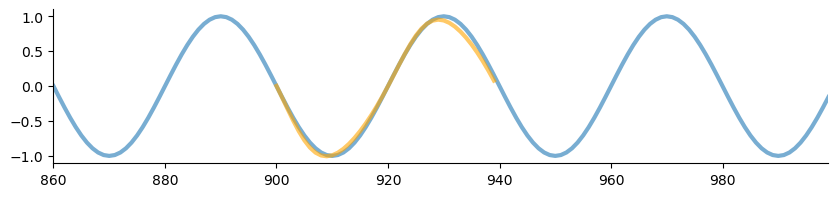
Epoch 1 Loss:0.030122658237814903; Performance on test range: 0.46689707040786743



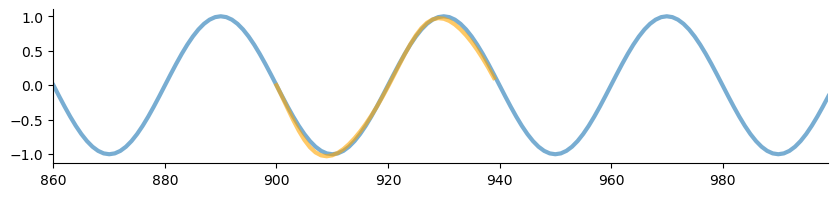
Epoch 2 Loss:0.002125221537426114; Performance on test range: 0.10097652673721313



Epoch 3 Loss:0.00019723063451237977; Performance on test range: 0.0034082874190062284



Epoch 4 Loss:0.00014400237705558538; Performance on test range: 0.002277107909321785



### 3.6.1.5 将图像（影像）看作一个空间序列

可以把一个二维的图像（影像）沿一个轴向看作一个空间序列，输入特征的数量为另一个轴向的像素（单元）数，同样以最后一个隐藏状态作为全连接层的输入实现图像的分类任务[12]。

* MNIST 手写体数字图片数据集准备

使用轻量型的 [MNIST 数据集](http://yann.lecun.com/exdb/mnist/)②说明将图像看作一个空间序列进行分类识别的建模过程。

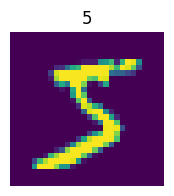
mnist\_fn=r'I:\data\mnist'  
traindt=datasets.MNIST(root = mnist\_fn,transform=transforms.ToTensor(),train=True,download=True)  
testdt=datasets.MNIST(root = mnist\_fn,transform=transforms.ToTensor(),train=False,download=True)

print(traindt,'\n',testdt)  
batch\_size=100  
train\_loader=DataLoader(traindt, batch\_size=batch\_size, shuffle=False)  
test\_loader=DataLoader(testdt, batch\_size=batch\_size, shuffle=False)

Dataset MNIST  
 Number of datapoints: 60000  
 Root location: I:\data\mnist  
 Split: Train  
 StandardTransform  
Transform: ToTensor()   
 Dataset MNIST  
 Number of datapoints: 10000  
 Root location: I:\data\mnist  
 Split: Test  
 StandardTransform  
Transform: ToTensor()

打印查看一个样本。

fig, ax=plt.subplots(1, 1,figsize=(2,2))  
ax.imshow(traindt[0][0].numpy().reshape(28,28))  
plt.axis("off")  
plt.title(str(traindt[0][1]))  
plt.show()



* 建立 Vallina RNN 模型

定义RNN\_model\_img类，为一个 Vallina RNN。因为图像的大小为，因此配置参数input\_dim=28。而数字总共有10个，因此全连接层的输出维度为10。

n\_iters=8000  
num\_epochs=int(n\_iters/(len(traindt)/batch\_size))  
print(num\_epochs)  
  
# Create RNN  
input\_dim=28 # input dimension  
hidden\_dim=100 # hidden layer dimension  
layer\_dim=1 # number of hidden layers  
output\_dim=10 # output dimension  
  
model=usda\_models.RNN\_model\_img(input\_dim, hidden\_dim,layer\_dim, output\_dim,)  
print(model)  
  
# Cross Entropy Loss   
error=nn.CrossEntropyLoss()  
  
# SGD Optimizer  
learning\_rate=0.05  
optimizer=torch.optim.SGD(model.parameters(), lr=learning\_rate)

13  
RNN\_model\_img(  
 (rnn): RNN(28, 100, batch\_first=True)  
 (fc): Linear(in\_features=100, out\_features=10, bias=True)  
)

* 模型训练

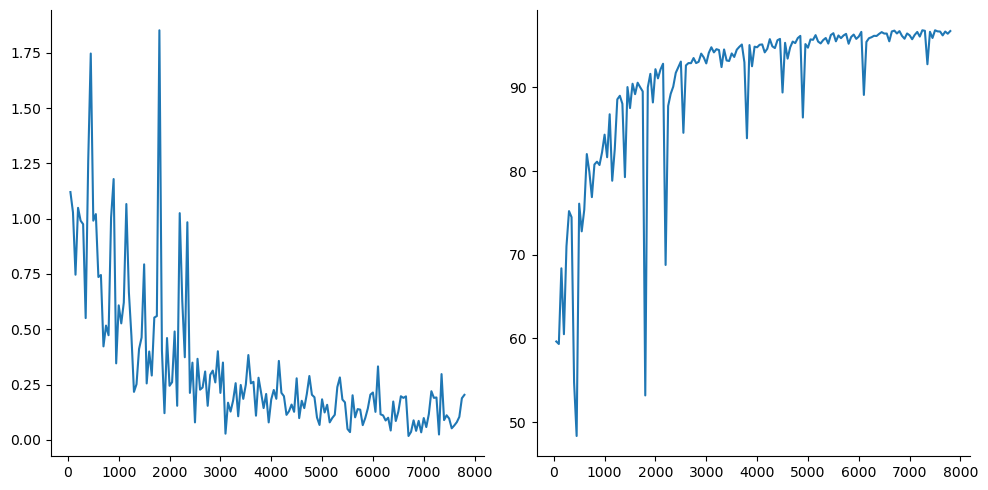
定义RNN\_train\_img()函数训练模型，并返回损失和准确率（精度）。

iteration\_list,loss\_list,accuracy\_list=usda\_models.RNN\_train\_img(model,train\_loader,test\_loader,input\_dim,optimizer,error,epochs=num\_epochs,step\_eval=50)

Iteration: 500 Loss: 0.9909659028053284 Accuracy: 76.11000061035156 %  
Iteration: 1000 Loss: 0.6082250475883484 Accuracy: 84.36000061035156 %  
Iteration: 1500 Loss: 0.7935434579849243 Accuracy: 87.54000091552734 %  
Iteration: 2000 Loss: 0.24458648264408112 Accuracy: 92.19000244140625 %  
Iteration: 2500 Loss: 0.0790989100933075 Accuracy: 93.0999984741211 %  
Iteration: 3000 Loss: 0.21234281361103058 Accuracy: 92.87999725341797 %  
Iteration: 3500 Loss: 0.24958306550979614 Accuracy: 94.06999969482422 %  
Iteration: 4000 Loss: 0.18281689286231995 Accuracy: 94.80999755859375 %  
Iteration: 4500 Loss: 0.27840670943260193 Accuracy: 89.4000015258789 %  
Iteration: 5000 Loss: 0.18332625925540924 Accuracy: 94.76000213623047 %  
Iteration: 5500 Loss: 0.049436092376708984 Accuracy: 96.48999786376953 %  
Iteration: 6000 Loss: 0.21413767337799072 Accuracy: 96.08999633789062 %  
Iteration: 6500 Loss: 0.12760573625564575 Accuracy: 96.44999694824219 %  
Iteration: 7000 Loss: 0.09869009256362915 Accuracy: 96.23999786376953 %  
Iteration: 7500 Loss: 0.09632816165685654 Accuracy: 96.83999633789062 %

打印损失和准确率曲线，可以观察到损失曲线随着训练迭代逐渐降低，而准确率不断爬升并趋于平缓，可以达到约 96% 的精度。

fig, axs=plt.subplots(1, 2,figsize=(10,5))  
axs[0].plot(iteration\_list,loss\_list)  
axs[1].plot(iteration\_list,accuracy\_list)  
  
axes[0].set\_title('Loss')  
axes[1].set\_title('Accuracy')  
axs[0].spines[['right', 'top']].set\_visible(False)  
axs[1].spines[['right', 'top']].set\_visible(False)  
fig.tight\_layout()  
plt.show()



## 3.6.2 Transformer——自然语言处理（Natural Language Processing，NLP）

相关参考阅读（按理解 Transformer 前置知识点次序）：

1. 词向量（Word Embeddings）：[The Illustrated Word2vec](https://jalammar.github.io/illustrated-word2vec/)③；[An Intuitive Understanding of Word Embeddings: From Count Vectors to Word2Vec](https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/)④；[What Are Word Embeddings for Text?](https://machinelearningmastery.com/what-are-word-embeddings/)⑤；[How to Develop Word Embeddings in Python with Gensim](https://machinelearningmastery.com/develop-word-embeddings-python-gensim/)⑥；[词向量 (Word Embeddings)](https://leovan.me/cn/2018/10/word-embeddings/)⑦；[One-Hot, Label, Target and K-Fold Target Encoding, Clearly Explained!!!](https://www.youtube.com/watch?v=589nCGeWG1w)⑧；[Word Embedding and Word2Vec, Clearly Explained!!!](https://www.youtube.com/watch?v=viZrOnJclY0)⑨；[GloVe: Global Vectors for Word Representation](https://nlp.stanford.edu/projects/glove/)⑩；[Word2Vec Tutorial - The Skip-Gram Model](http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/)⑪；[Word2vec with PyTorch: Implementing the Original Paper](https://towardsdatascience.com/word2vec-with-pytorch-implementing-original-paper-2cd7040120b0)⑫
2. Seq2Seq（Sequence-to-sequence）/Encoder-Decoder：[Sequence to Sequence (seq2seq) and Attention](https://lena-voita.github.io/nlp_course/seq2seq_and_attention.html)⑬；[Visualizing A Neural Machine Translation Model (Mechanics of Seq2seq Models With Attention)](https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/)⑭；[Sequence-to-Sequence (seq2seq) Encoder-Decoder Neural Networks, Clearly Explained!!!](https://www.youtube.com/watch?v=L8HKweZIOmg&t=829s)⑮；
3. Transformer：[The Illustrated Transformer](http://jalammar.github.io/illustrated-transformer/)⑯；[Transformer: A Novel Neural Network Architecture for Language Understanding](https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html)⑰；[The Annotated Transformer](http://nlp.seas.harvard.edu/annotated-transformer/)⑱；[Transformers and Multi-Head Attention](https://uvadlc-notebooks.readthedocs.io/en/latest/tutorial_notebooks/JAX/tutorial6/Transformers_and_MHAttention.html)⑲

### 3.6.2.1 词向量（Word Embeddings）

#### 1) 文本矢量化

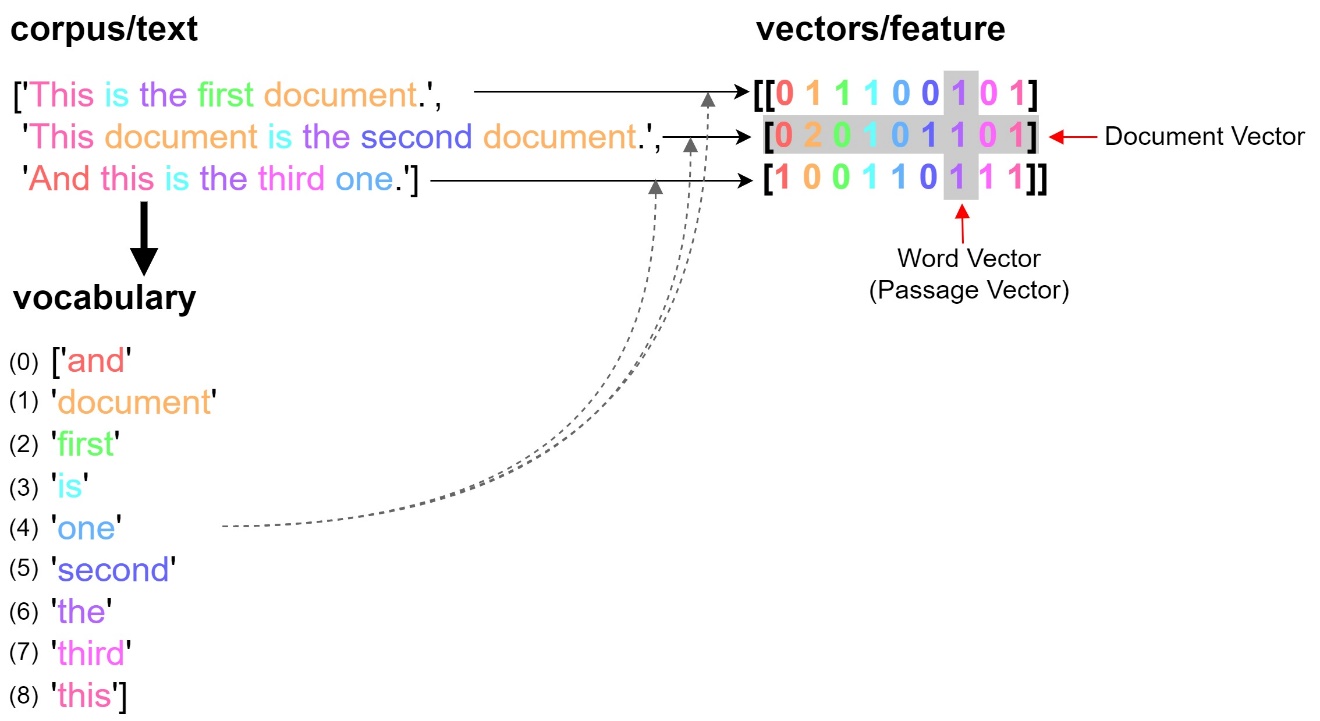
对文本数据分类、情感分析、文本生成和不同语言的翻译时，所用到的文本数据集并不能直接输入到模型进行训练，除常规数据预处理外，需要解析原始文本为小的语句块，分解为单词、句子等，称为标记化（tokenization）。标记化有助于理解语境（上下文），通过分析单词的顺序解释文本的含义；并将单词编码为整数或浮点值，称为特征提取或矢量化（vectorization）。标记化和矢量化文本数据有多种方式，基于频数的矢量化有词频向量（word count vectors）、TF-IDF向量、共现（Co-Occurrence）矩阵和哈希向量（Hashing vectors）等[13]。可用[sklearn](https://scikit-learn.org/stable/)⑳库提供的相应方法计算，如下示例。

%load\_ext autoreload   
%autoreload 2   
import usda.models as usda\_models  
  
from sklearn.feature\_extraction.text import CountVectorizer, TfidfVectorizer, HashingVectorizer  
from gensim.utils import tokenize

The autoreload extension is already loaded. To reload it, use:  
 %reload\_ext autoreload

**A: 词频向量**

词频向量构建方式如下图，提取输入文本的唯一一个或多个（n 元语法）单词/字符作为词汇表（标记化），根据词汇表列表对位索引计数或统计词频（矢量化）。



* n-gram （n-元语法）

参数ngram\_range是指从文本中提取单词或者字符的上下边界，例如(1,1)意味仅含一元语法（unigrams），(1,2)意味包含一元语法和二元语法（bigrams），(2,2)则仅包含二元语法，依次类推到多元语法，例如三元语法（trigram）。

仅含一元语法。

countVectorizer\_print=lambda vectorizer,X: print(f'------------volcab\n{vectorizer.vocabulary\_}\  
 \n------------feature\n{vectorizer.get\_feature\_names\_out()}\  
 \n------------stop words\n{vectorizer.get\_stop\_words()}\  
 \n------------array\n{X.toarray()}')  
  
corpus=['This is the first document.',   
 'This document is the second document.',  
 'And this is the third one.']  
vectorizer=CountVectorizer(analyzer='word')  
X=vectorizer.fit\_transform(corpus)  
countVectorizer\_print(vectorizer,X)

------------volcab  
{'this': 8, 'is': 3, 'the': 6, 'first': 2, 'document': 1, 'second': 5, 'and': 0, 'third': 7, 'one': 4}   
------------feature  
['and' 'document' 'first' 'is' 'one' 'second' 'the' 'third' 'this']   
------------stop words  
None   
------------array  
[[0 1 1 1 0 0 1 0 1]  
 [0 2 0 1 0 1 1 0 1]  
 [1 0 0 1 1 0 1 1 1]]

含一元和二元语法。

vectorizer=CountVectorizer(analyzer='word',ngram\_range=(1, 2))  
X=vectorizer.fit\_transform(corpus)  
countVectorizer\_print(vectorizer,X)

------------volcab  
{'this': 17, 'is': 6, 'the': 11, 'first': 4, 'document': 2, 'this is': 19, 'is the': 7, 'the first': 12, 'first document': 5, 'second': 9, 'this document': 18, 'document is': 3, 'the second': 13, 'second document': 10, 'and': 0, 'third': 15, 'one': 8, 'and this': 1, 'the third': 14, 'third one': 16}   
------------feature  
['and' 'and this' 'document' 'document is' 'first' 'first document' 'is'  
 'is the' 'one' 'second' 'second document' 'the' 'the first' 'the second'  
 'the third' 'third' 'third one' 'this' 'this document' 'this is']   
------------stop words  
None   
------------array  
[[0 0 1 0 1 1 1 1 0 0 0 1 1 0 0 0 0 1 0 1]  
 [0 0 2 1 0 0 1 1 0 1 1 1 0 1 0 0 0 1 1 0]  
 [1 1 0 0 0 0 1 1 1 0 0 1 0 0 1 1 1 1 0 1]]

仅含二元语法。

vectorizer=CountVectorizer(analyzer='word',ngram\_range=(2, 2))  
X=vectorizer.fit\_transform(corpus)  
countVectorizer\_print(vectorizer,X)

------------volcab  
{'this is': 10, 'is the': 3, 'the first': 5, 'first document': 2, 'this document': 9, 'document is': 1, 'the second': 6, 'second document': 4, 'and this': 0, 'the third': 7, 'third one': 8}   
------------feature  
['and this' 'document is' 'first document' 'is the' 'second document'  
 'the first' 'the second' 'the third' 'third one' 'this document'  
 'this is']   
------------stop words  
None   
------------array  
[[0 0 1 1 0 1 0 0 0 0 1]  
 [0 1 0 1 1 0 1 0 0 1 0]  
 [1 0 0 1 0 0 0 1 1 0 1]]

* 过滤掉停用词（Stop Words）

停用词是那些对短语（phrase）的深层含义没有帮助的词，为最常见的单词，例如the、a、is 等。因此对于一些模型，例如文档分类等，则删除这些停顿词有益于模型训练。可以用Sklearn库提供的get\_stop\_words方法查看停用词，也可以用[NLTK](https://www.nltk.org/)㉑库stopwords方法查看。

corpus\_motto=['Do the best you can until you know better. Then when you know better, do better.— Maya Angelou',   
 'Your time is limited so don’t waste it living someone else’s life. – Steve Jobs',  
 'Don’t be pushed by your problems. Be led by your dreams. – Ralph Waldo Emerson']  
vectorizer=CountVectorizer(analyzer='word',ngram\_range=(1, 1),stop\_words='english')  
X=vectorizer.fit\_transform(corpus\_motto)  
countVectorizer\_print(vectorizer,X)

------------volcab  
{'best': 1, 'know': 7, 'better': 2, 'maya': 12, 'angelou': 0, 'time': 17, 'limited': 10, 'don': 3, 'waste': 19, 'living': 11, 'life': 9, 'steve': 16, 'jobs': 6, 'pushed': 14, 'problems': 13, 'led': 8, 'dreams': 4, 'ralph': 15, 'waldo': 18, 'emerson': 5}   
------------feature  
['angelou' 'best' 'better' 'don' 'dreams' 'emerson' 'jobs' 'know' 'led'  
 'life' 'limited' 'living' 'maya' 'problems' 'pushed' 'ralph' 'steve'  
 'time' 'waldo' 'waste']   
------------stop words  
frozenset({'anyway', 'both', 'hereupon', 'too', 'had', 'further', 'side', 'although', 'against', 'describe', 'cry', 'but', 'seeming', 'still', 'full', 'twelve', 'ten', 'its', 'hers', 'hence', 'along', 're', 'system', 'back', 'off', 'without', 'itself', 'beside', 'over', 'own', 'therefore', 'where', 'herself', 'nevertheless', 'about', 'from', 'somewhere', 'next', 'am', 'whoever', 'nowhere', 'latter', 'nobody', 'thick', 'anywhere', 'yourself', 'show', 'who', 'why', 'between', 'though', 'have', 'give', 'herein', 'whence', 'couldnt', 'mill', 'de', 'toward', 'whereupon', 'them', 'wherein', 'can', 'move', 'amoungst', 'four', 'she', 'as', 'often', 'put', 'for', 'an', 'down', 'ever', 'twenty', 'take', 'many', 'do', 'would', 'which', 'must', 'fifty', 'whatever', 'wherever', 'whole', 'hereby', 'something', 'amount', 'hundred', 'please', 'six', 'noone', 'there', 'what', 'with', 'when', 'cant', 'nor', 'also', 'con', 'he', 'all', 'in', 'seem', 'across', 'serious', 'will', 'and', 'through', 'first', 'whom', 'those', 'anyone', 'indeed', 'while', 'here', 'moreover', 'somehow', 'such', 'well', 'ours', 'mine', 'our', 'of', 'the', 'co', 'thus', 'should', 'within', 'beforehand', 'last', 'find', 'other', 'before', 'per', 'someone', 'whereafter', 'detail', 'less', 'everywhere', 'their', 'empty', 'throughout', 'mostly', 'bill', 'most', 'latterly', 'is', 'thereupon', 'after', 'became', 'everyone', 'being', 'top', 'few', 'onto', 'eight', 'at', 'only', 'to', 'whenever', 'former', 'together', 'out', 'until', 'several', 'anyhow', 'third', 'fifteen', 'therein', 'towards', 'get', 'forty', 'least', 'become', 'or', 'sixty', 'five', 'sometime', 'that', 'more', 'whither', 'no', 'thin', 'everything', 'has', 'may', 'almost', 'ourselves', 'by', 'been', 'among', 'they', 'eleven', 'this', 'namely', 'under', 'are', 'was', 'see', 'could', 'even', 'these', 'alone', 'part', 'sometimes', 'front', 'below', 'her', 'than', 'because', 'so', 'call', 'beyond', 'you', 'myself', 'once', 'anything', 'always', 'into', 'sincere', 'yours', 'during', 'nothing', 'amongst', 'if', 'except', 'formerly', 'already', 'three', 'some', 'un', 'seems', 'never', 'thence', 'cannot', 'due', 'fill', 'however', 'one', 'ltd', 'it', 'him', 'were', 'his', 'others', 'another', 'same', 'your', 'either', 'interest', 'yet', 'whether', 'how', 'then', 'much', 'go', 'i', 'nine', 'very', 'fire', 'around', 'etc', 'whereas', 'thereafter', 'thru', 'else', 'upon', 'since', 'whose', 'ie', 'themselves', 'becomes', 'again', 'becoming', 'whereby', 'my', 'bottom', 'hasnt', 'any', 'we', 'meanwhile', 'inc', 'himself', 'a', 'neither', 'otherwise', 'via', 'name', 'each', 'eg', 'us', 'not', 'above', 'might', 'on', 'enough', 'besides', 'yourselves', 'elsewhere', 'every', 'be', 'made', 'hereafter', 'now', 'keep', 'seemed', 'up', 'rather', 'afterwards', 'found', 'two', 'me', 'done', 'behind', 'perhaps', 'none', 'thereby'})   
------------array  
[[1 1 3 0 0 0 0 2 0 0 0 0 1 0 0 0 0 0 0 0]  
 [0 0 0 1 0 0 1 0 0 1 1 1 0 0 0 0 1 1 0 1]  
 [0 0 0 1 1 1 0 0 1 0 0 0 0 1 1 1 0 0 1 0]]

用NLTK库查看停用词。

from nltk.corpus import stopwords  
import nltk  
nltk.download('stopwords')  
  
stop\_words=stopwords.words('english')  
print(stop\_words)

['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', 'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after', 'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further', 'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', 've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn', "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn', "wouldn't"]  
  
  
[nltk\_data] Downloading package stopwords to  
[nltk\_data] C:\Users\richi\AppData\Roaming\nltk\_data...  
[nltk\_data] Package stopwords is already up-to-date!

**B: TF-IDF向量**

如果不过滤掉停用词，那么在计数向量的方法中，像 the、is、will等词出现的次数较多，且在编码向量中没有太大的意义，而TF-IDF（term Frequency - Inverse Document Frequency ）向量方法，一种用于信息检索与数据挖掘的常用加权技术，能够改善上述情况。其中 TF（词频） 表示为给定字词在文档中出现的频数，可表示为；IDF 为逆文本频率指数，减小跨文档出现次数较多的字词影响，可表示为，（在Sklearn库中，为，如果配置参数smooth\_idf=True，则为），式中，为文档总数，为字词在一个文档中的频数。将上述两部分相乘，得。

下述示例中可以看到词countvectorizer和Tf-idf的值为0.6330，而 this、is、about 等词值为0.448，要小于前者。

corpus\_vector=['This is about CountVectorizer',  
 'This is about Tf-idf']  
vectorizer=TfidfVectorizer(analyzer='word',token\_pattern=r"(?u)\S\S+")  
X=vectorizer.fit\_transform(corpus\_vector)  
print(vectorizer.get\_feature\_names\_out())  
print(X.toarray())

['about' 'countvectorizer' 'is' 'tf-idf' 'this']  
[[0.44832087 0.63009934 0.44832087 0. 0.44832087]  
 [0.44832087 0. 0.44832087 0.63009934 0.44832087]]

**C: 共现（Co-Occurrence）矩阵**

相似的词往往会一起出现，并会有相似的语境，例如’Apple is a fruit’和’Mango is a fruit’等，因此统计给定语境窗口（Context Window），元素（字词，短语、句子，任何感兴趣的语言单元）共同出现的次数可以表述语料库元素间的关系。定义build\_co\_occurrence\_matrix()函数计算共现矩阵。配置参数window\_size=10，大于单个句子的最大长度，是以整个句子为最小的语境窗口进行统计。

corpus\_fruit=['Apple is a fruit','Mango is a fruit','Mango tastes sweet and sour']  
coo\_dict,coo\_df=usda\_models.build\_co\_occurrence\_matrix(corpus\_fruit,window\_size=10)  
coo\_df

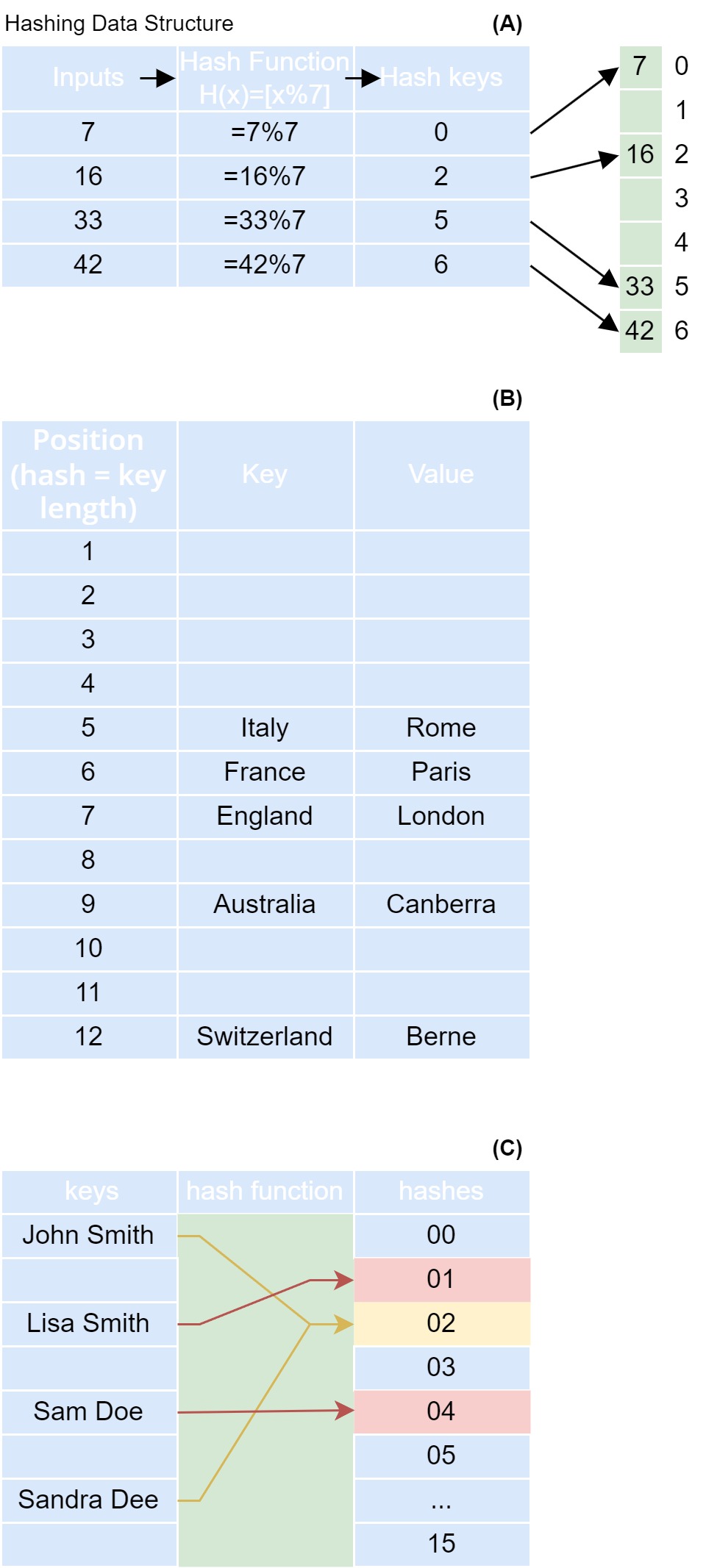
|  | **a** | **Apple** | **fruit** | **sour** | **Mango** | **and** | **tastes** | **sweet** | **is** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **a** | 0 | 1 | 2 | 0 | 1 | 0 | 0 | 0 | 2 |
| **Apple** | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| **fruit** | 2 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 2 |
| **sour** | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 |
| **Mango** | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| **and** | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 |
| **tastes** | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 |
| **sweet** | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| **is** | 2 | 1 | 2 | 0 | 1 | 0 | 0 | 0 | 0 |

**D: 哈希/散列向量（Hashing vectors）**

词频向量、TF-IDF权重向量和共现矩阵等方法随着语料库的扩大，词汇表可能变得非常大，这将需要大量的向量来编码文档，对计算机的内存和算力提出要求。解决上述问题的一种途径是使用哈希向量映射或降维。

* 哈希函数（Hash function）[14]

哈希函数是可以把任意大小的数据映射到固定大小值的函数（有些哈希函数也支持可变长度的输出）。哈希函数返回的值称为哈希值（hash values）、哈希码（hash codes）、摘要（digests）或简称哈希（hashes）。这些值通常用于索引一个称为哈希表（hash table）的固定大小的表。使用哈希函数对哈希表进行索引称为哈希（hashing）或者分散存储寻址（scatter storage addressing）。在数据存储和检索等应用程序中，哈希函数及其关联的哈希表可以在每次检索时以很小且几乎恒定的时长访问数据；并且所需的存储空间仅略大于数据或者记录本身所需要的总空间。哈希是一种高效计算和空间存储的数据访问形式，避免了无序列表和结构化树的非恒定访问时长，及直接访问大型或具有可变长度键状态空间时对存储空间成指数级的需求。



上图 A 和 B 例举了两种哈希函数形式，其中 A 为一个余数表示；B 为键的长度表示。哈希函数的形式（算法）有多种，但都有一个相同的原则，将数据映射到一个可以索引数组的值。一些流行的哈希算法有SHA-1、MD5和Murmur Hash等。在图 C 中，一个哈希函数将键（人名）映射到0到15的一个整数形式，且有两个键映射到同一个哈希值02的位置上，该种情况适合于某些应用场景，达到合并降维的目的。

HashingVectorizer方法即为哈希向量，在第一个示例代码中，词汇表有8个词，配置输出矩阵特征列的数量参数n\_features=10，不会发生映射碰撞/冲突（ collision），为哈希映射；第二个示例代码中，有29个词，同样配置n\_features=10，则有部分特征发生了合并降维。

—哈希映射

corpus2tokens=lambda corpus: [[word for word in tokenize(sentence) if word not in stopwords.words('english')] for sentence in corpus]  
  
vectorizer=HashingVectorizer(n\_features=10,stop\_words='english',norm=None)  
X=vectorizer.fit\_transform(corpus\_fruit)  
print(corpus2tokens(corpus\_fruit))  
print(f'{type(X)}\n{X}\n{X.toarray()}')

[['Apple', 'fruit'], ['Mango', 'fruit'], ['Mango', 'tastes', 'sweet', 'sour']]  
<class 'scipy.sparse.\_csr.csr\_matrix'>  
 (0, 0) 1.0  
 (0, 3) 1.0  
 (1, 1) 1.0  
 (1, 3) 1.0  
 (2, 0) 1.0  
 (2, 1) 1.0  
 (2, 4) -2.0  
[[ 1. 0. 0. 1. 0. 0. 0. 0. 0. 0.]  
 [ 0. 1. 0. 1. 0. 0. 0. 0. 0. 0.]  
 [ 1. 1. 0. 0. -2. 0. 0. 0. 0. 0.]]

—哈希合并

vectorizer=HashingVectorizer(n\_features=10,stop\_words='english',norm='l2')  
X=vectorizer.fit\_transform(corpus\_motto)  
print(corpus2tokens(corpus\_motto))  
print(f'{type(X)}\n{X}\n{X.toarray()}')

[['Do', 'best', 'know', 'better', 'Then', 'know', 'better', 'better', 'Maya', 'Angelou'], ['Your', 'time', 'limited', 'waste', 'living', 'someone', 'else', 'life', 'Steve', 'Jobs'], ['Don', 'pushed', 'problems', 'Be', 'led', 'dreams', 'Ralph', 'Waldo', 'Emerson']]  
<class 'scipy.sparse.\_csr.csr\_matrix'>  
 (0, 1) -0.23570226039551587  
 (0, 4) -0.47140452079103173  
 (0, 7) -0.47140452079103173  
 (0, 8) -0.7071067811865476  
 (1, 1) -0.4082482904638631  
 (1, 2) -0.4082482904638631  
 (1, 3) -0.4082482904638631  
 (1, 4) -0.4082482904638631  
 (1, 6) 0.4082482904638631  
 (1, 8) 0.4082482904638631  
 (2, 0) 0.6324555320336759  
 (2, 3) 0.6324555320336759  
 (2, 5) 0.0  
 (2, 6) 0.31622776601683794  
 (2, 7) 0.31622776601683794  
[[ 0. -0.23570226 0. 0. -0.47140452 0.  
 0. -0.47140452 -0.70710678 0. ]  
 [ 0. -0.40824829 -0.40824829 -0.40824829 -0.40824829 0.  
 0.40824829 0. 0.40824829 0. ]  
 [ 0.63245553 0. 0. 0.63245553 0. 0.  
 0.31622777 0.31622777 0. 0. ]]

#### 2）距离度量——Cosine similarity 和词嵌入（Word Embeddings）

词频向量、TF-IDF向量等方法仅记录了文档中对应词汇表字词的频数所呈现字词数量分布的特征，并没有体现出哪些词的组合更容易构成具有意义的短语或句子，及哪些词在给定语境下更相似，例如‘Apple finally announced its long-rumored AR/VR headset, called the VisionPro.’和‘Some of the best-tasting apple varieties are Honeycrisp, Pink Lady, Fuji, Ambrosia, and Cox’s Orange Pippin.’，在第一个语境下，apple 与VisionPro等词更容易组成有意义的句子，而第二个语境，apple 与 tasting、Cox等词更容易组合。虽然共现矩阵的方法可以统计给定切分窗口字词共同出现的次数，但是矩阵的大小为词汇表长度的2次方，而多维向量的距离度量方法（可以查看*标记距离*一章的距离度量）可以指定特征维数，例如常用的50、100或者300等。在 Word2Vec 方法中使用了 Cosine 距离（Cosine Similarity，CS）[15]，CS 的计算公式为 （PyTorch库给出的公式为：）。PyTorch提供了torch.nn.CosineSimilarity(dim=1, eps=1e-08)方法计算 CS。

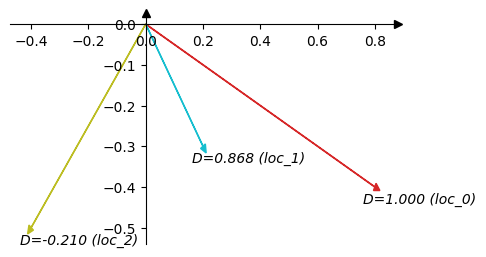
参考 Jay Alammar 对 Word2Vec方法的解释和提供的案例[16]，假设有3个地点（或区域）为 loc\_0,loc\_1 和 loc\_2，各计算有5个生态服务指数（假设区间含正负值），为碳存储、生境质量、降温指数、年产水和自然空间可达性等，现在需要比较基于这5个特征向量 备选地点 loc\_1 和 loc\_2 到 原地点 loc\_0 的距离，即备选的哪个地点的特征更与原地点相近。为了方便图形可视化，先选择两个指数，假如为碳存储和生境质量，将这两个指数作为二维空间中的两个垂直坐标绘制特征点，并计算备选点到原地点的 CS 值。

%load\_ext autoreload   
%autoreload 2   
import usda.pattern\_signature as usda\_signature  
import usda.data\_visual as usda\_vis   
import usda.utils as usda\_utils  
import usda.models as usda\_models  
  
import numpy as np  
import matplotlib.pyplot as plt  
import numpy as np  
import torch  
from torch import nn  
import math  
from varname import nameof  
  
import gensim  
import gensim.downloader as api  
from gensim.corpora import Dictionary  
from gensim.utils import tokenize  
  
from nltk.corpus import stopwords  
import nltk  
from sklearn.manifold import TSNE

The autoreload extension is already loaded. To reload it, use:  
 %reload\_ext autoreload

从计算结果可以观察到，备选地点到原地点的距离分别为0.868和-0.210，其中 loc\_1 与 loc\_0 相对更为相似。

cosine\_similarity=nn.CosineSimilarity(dim=0, eps=1e-6)  
  
origin=torch.tensor([0,0],dtype=torch.float64)  
location\_0=torch.tensor([0.8,-0.4],dtype=torch.float64)  
location\_1=torch.tensor([0.2,-0.3],dtype=torch.float64)  
location\_2=torch.tensor([-0.4,-0.5],dtype=torch.float64)  
location\_dict={'loc\_0':location\_0,'loc\_1':location\_1,'loc\_2':location\_2}  
colors=['tab:red','tab:cyan','tab:olive']  
  
fig, ax=plt.subplots(figsize=(5,3))  
i=0  
for key,loc in location\_dict.items():  
 distance=cosine\_similarity(location\_0,loc)  
 ax.arrow(\*origin,\*loc,head\_width=0.02, head\_length=0.02,color=colors[i])  
 ax.text(\*loc-0.04,f'D={distance:.3f} ({key})',fontstyle='italic')  
 i+=1  
  
usda\_vis.plot\_style\_axis\_A(ax)  
plt.show()



计算所有5个维度的特征距离，loc\_1 到 loc\_0 的距离为 0.658，而loc\_2 到 loc\_0 的距离为-0.368。通过不同空间地点给定多维度特征的距离计算，说明该种方法可以用于时空序列，如自然语言处理（Natural language processing，NLP）能够反映词汇表字词间距离的特征向量；如果对应到空间数据则可以理解为城市空间不同区域间反映属性距离的特征向量；对应到时序数据则可以理解为不同样本间属性距离的特征向量。

location\_0=torch.tensor([0.8,-0.4,0.5,-0.2,0.3],dtype=torch.float64)  
location\_1=torch.tensor([0.2,-0.3,0.3,-0.4,0.9],dtype=torch.float64)  
location\_2=torch.tensor([-0.4,-0.5,-0.2,0.7,-0.1],dtype=torch.float64)  
  
print(f'D(0-1)={cosine\_similarity(location\_0,location\_1)};\nD(0-2)={cosine\_similarity(location\_0,location\_2)}')

D(0-1)=0.658233707531176;  
D(0-2)=-0.3683509554826695

将上述案例对特征距离的解释对应到自然语言处理任务中的词嵌入（为了说明 word embedding 的特殊性，用“词嵌入”的翻译代替“词向量”的翻译），词嵌入可以通过 Word2Vec、GloVe 等模型算法从语料库中学习。先看下已经训练好的词嵌入，一般常用到已经训练好的词嵌入模型有 [Google 的 Word2Vec 的词嵌入](https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit?resourcekey=0-wjGZdNAUop6WykTtMip30g)㉒和[斯坦福大学（Stanford） 的 GloVe 词嵌入](https://github.com/stanfordnlp/GloVe)。[gensim](https://radimrehurek.com/gensim/)㉓库也已经集成了10多个已经训练好的[词嵌入模型](https://github.com/RaRe-Technologies/gensim-data)㉔，下述示例下载的为glove-wiki-gigaword-50，基于的数据集为维基百科数据（Wikipedia 2014 + Gigaword 5 (6B tokens, uncased)），含有 40 万单词和短语，词嵌入的维度为 50。

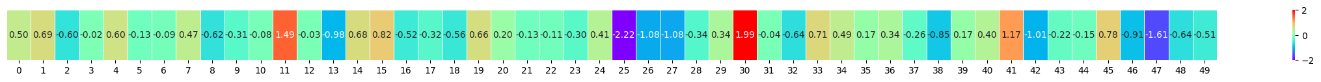
通过model["king"]方式可以直接查看输入词对应的词嵌入向量。通过model.most\_similar('king')方法可以获取与输入词距离最近的单词，例如与 king 词最近似的词有 prince, queen 等。

model=api.load('glove-wiki-gigaword-50')  
print(model.vector\_size)  
print(model["king"])  
print(model.most\_similar('king'))

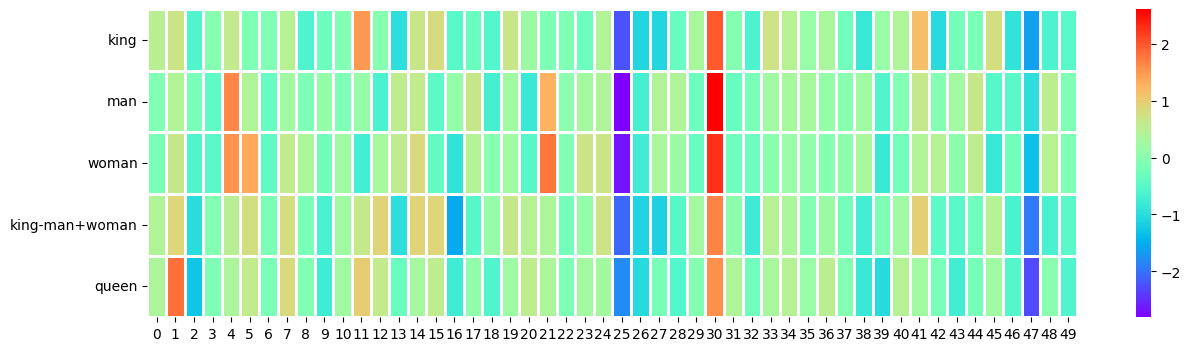
50  
[ 0.50451 0.68607 -0.59517 -0.022801 0.60046 -0.13498 -0.08813  
 0.47377 -0.61798 -0.31012 -0.076666 1.493 -0.034189 -0.98173  
 0.68229 0.81722 -0.51874 -0.31503 -0.55809 0.66421 0.1961  
 -0.13495 -0.11476 -0.30344 0.41177 -2.223 -1.0756 -1.0783  
 -0.34354 0.33505 1.9927 -0.04234 -0.64319 0.71125 0.49159  
 0.16754 0.34344 -0.25663 -0.8523 0.1661 0.40102 1.1685  
 -1.0137 -0.21585 -0.15155 0.78321 -0.91241 -1.6106 -0.64426  
 -0.51042 ]  
[('prince', 0.8236179351806641), ('queen', 0.7839043140411377), ('ii', 0.7746230363845825), ('emperor', 0.7736247777938843), ('son', 0.766719400882721), ('uncle', 0.7627150416374207), ('kingdom', 0.7542160749435425), ('throne', 0.7539913654327393), ('brother', 0.7492411136627197), ('ruler', 0.7434253692626953)]

为了方便查看不同词嵌入向量的关系，打印向量值的热力图。50维的向量每一维度表征字词的某一特征（每一维度特征可解释的内容属性本文并未做探索），可以发现单词 king、man、woman、queen，及king-man+woman的向量运算结果中，有值明显接近的维度特征，例如蓝紫（25列和47列）和橘红（30列）等。

plt.figure(figsize=(30,1))  
ax=sns.heatmap([model["king"]], xticklabels=True, yticklabels=False, cbar=True,vmin=-2, vmax=2, linewidths=0.7,cmap='rainbow',annot=True,fmt='.2f')  
plt.show()



plt.figure(figsize=(15,4))  
ax=sns.heatmap([model["king"],   
 model["man"],   
 model["woman"],   
 model["king"] - model["man"] + model["woman"],  
 model["queen"],  
 ], cbar=True, xticklabels=True, yticklabels=True,linewidths=1,cmap='rainbow')   
  
ax.set\_yticks(np.arange(0.5,5))  
ax.set\_yticklabels(["king","man","woman","king-man+woman","queen"],rotation=0)  
plt.show()



通过上述示例，引用相关文献对词嵌入的描述有，词嵌入是对文本的学习表示，具有相同、相近含义的字词通常具有相似的表示[13]115。因为大多数神经网络工具包不能很好的处理高维（high-dimensional）、稀疏（sparse）向量，因此密集（dense）且低维(low-dimensional)的向量则更具有优势，并具有泛化能力[17]92。词嵌入将每个词映射到一个预定义的向量空间，表示为实值向量。通过神经网络学习语料库，训练词嵌入向量值。向量通常为数十到数百维，与数千到数百万维的稀疏向量（例如独热编码（one hot encoding））相比要小的多。

#### 3）词嵌入查找表（lookup table）

一个词汇表列表中各字词按照索引对应各自的词嵌入向量，为了方便通过词汇表索引提取对应的词嵌入向量，PyTorch提供了torch.nn.Embedding方法建立一个词嵌入的查找表。Embedding输入参数num\_embeddings为词汇表的大小，参数embedding\_dim为词嵌入向量的维度（其它参数配置可以查看 PyTorch 手册）。为了说明查找表的使用，这里使用前文corpus\_motto的文本，但是去除了在已训练的glove-wiki-gigaword-50词嵌入模型中查找不到的人名等词。并不移除停用词，建立的词汇表有30个词语。

corpus\_motto=['Do the best you can until you know better. Then when you know better, do better.', # — Maya Angelou  
 'Your time is limited so don’t waste it living someone else’s life. ', # – Steve Jobs  
 'Don’t be pushed by your problems. Be led by your dreams.'] # – Ralph Waldo Emerson  
  
corpus2tokens=lambda corpus: [[word.lower() for word in tokenize(sentence)] for sentence in corpus] # if word not in stopwords.words('english')  
corpus\_tokens=corpus2tokens(corpus\_motto)  
vocab=np.unique(usda\_utils.flatten\_lst(corpus\_tokens))  
vocab\_size=len(vocab)  
print(f'{vocab}\nvocab size={vocab\_size}')

['be' 'best' 'better' 'by' 'can' 'do' 'don' 'dreams' 'else' 'is' 'it'  
 'know' 'led' 'life' 'limited' 'living' 'problems' 'pushed' 's' 'so'  
 'someone' 't' 'the' 'then' 'time' 'until' 'waste' 'when' 'you' 'your']  
vocab size=30

如果将字词转换成对应词汇表的索引，gensim库提供了Dictionary方法。

dct=Dictionary([vocab])  
corpus\_0=torch.LongTensor([dct.doc2idx([word])[0] for word in corpus\_tokens[0]])   
print(corpus\_tokens[0])  
print(corpus\_0)

['do', 'the', 'best', 'you', 'can', 'until', 'you', 'know', 'better', 'then', 'when', 'you', 'know', 'better', 'do', 'better']  
tensor([ 5, 22, 1, 28, 4, 25, 28, 11, 2, 23, 27, 28, 11, 2, 5, 2])

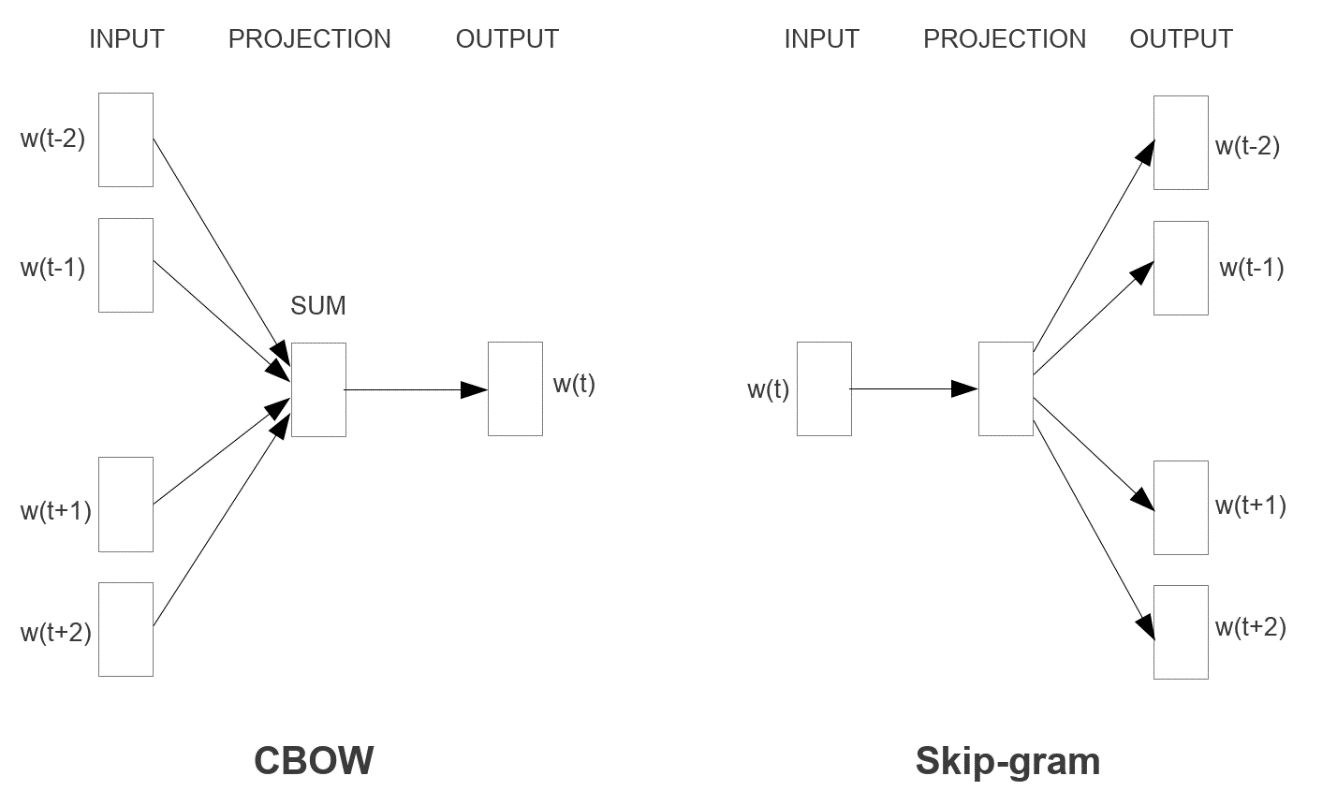
nn.Embedding初始化的词嵌入向量为随机值，通过神经网络学习语料库后各词的嵌入向量才能够表征语境下词汇之间的距离。这里直接提取已训练的glove-wiki-gigaword-50词嵌入向量中corpus\_motto文本词汇表对应的向量覆盖随机初始化值，仅用于说明查找表的使用。用建立的查找表embedding查找corpus\_0[0]，即词do，可以看到其查询结果同glove-wiki-gigaword-50中的查询结果。

vocab\_embeddings=model[vocab]  
print(vocab\_embeddings.shape)  
  
embedding=nn.Embedding(vocab\_size, vocab\_embeddings.shape[1])  
embedding.weight=nn.Parameter(torch.from\_numpy(vocab\_embeddings))  
print(model['do'])  
print(embedding(corpus\_0[0]))

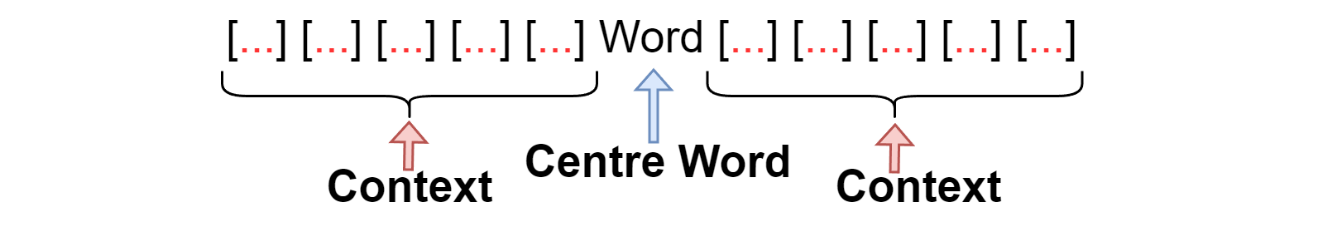
(30, 50)  
[ 2.9605e-01 -1.3841e-01 4.3774e-02 -3.8744e-01 1.2262e-01 -6.5180e-01  
 -2.8240e-01 9.0312e-02 -5.5186e-01 3.2060e-01 3.7422e-03 9.3229e-01  
 -2.2034e-01 -2.1922e-01 9.2170e-01 7.5724e-01 8.4892e-01 -4.2197e-03  
 5.3626e-01 -1.2667e+00 -6.1028e-01 1.6700e-01 8.2753e-01 6.5765e-01  
 4.8959e-01 -1.9744e+00 -1.1490e+00 -2.1461e-01 8.0539e-01 -1.4745e+00  
 3.7490e+00 1.0141e+00 -1.1293e+00 -5.2661e-01 -1.2029e-01 -2.7931e-01  
 6.5092e-02 -4.3639e-02 6.0426e-01 -2.0892e-01 -4.5739e-01 1.0441e-02  
 4.1458e-01 6.8900e-01 1.4468e-01 -3.1973e-02 -4.8073e-02 -1.1279e-04  
 1.3854e-01 9.6954e-01]  
tensor([ 2.9605e-01, -1.3841e-01, 4.3774e-02, -3.8744e-01, 1.2262e-01,  
 -6.5180e-01, -2.8240e-01, 9.0312e-02, -5.5186e-01, 3.2060e-01,  
 3.7422e-03, 9.3229e-01, -2.2034e-01, -2.1922e-01, 9.2170e-01,  
 7.5724e-01, 8.4892e-01, -4.2197e-03, 5.3626e-01, -1.2667e+00,  
 -6.1028e-01, 1.6700e-01, 8.2753e-01, 6.5765e-01, 4.8959e-01,  
 -1.9744e+00, -1.1490e+00, -2.1461e-01, 8.0539e-01, -1.4745e+00,  
 3.7490e+00, 1.0141e+00, -1.1293e+00, -5.2661e-01, -1.2029e-01,  
 -2.7931e-01, 6.5092e-02, -4.3639e-02, 6.0426e-01, -2.0892e-01,  
 -4.5739e-01, 1.0441e-02, 4.1458e-01, 6.8900e-01, 1.4468e-01,  
 -3.1973e-02, -4.8073e-02, -1.1279e-04, 1.3854e-01, 9.6954e-01],  
 grad\_fn=<EmbeddingBackward0>)

#### 4) Word2Vec

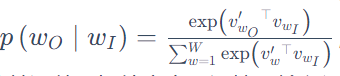
以Mikolov, T.等人 2013 年发表的两篇论文[15,18]为主，参考实现该论文方法的代码实现[19]，解释学习语料库估计词嵌入的方法。根据给定窗口语境下输入词（X）和输出词(y)的相对位置，文中描述了两类结构模型，一个称为 CBOW（Continuous Bag-of-Words），另一个称为 Skip-gram。如图 [15]



例如对于 ‘Efﬁcient Estimation of Word Representations in Vector Space’ 这个短语（不区分大小写，含停用词），如果以词word为输出，以word邻近指定长度的词为输入，如Estimation即、of即加上Representations即、in即，则该种结构模型为 CBOW；如果以词word为输入，而以邻近指定长度的词为输出，例如Estimation、of加上Representations、in（同 CBOW 的输入），则该种结构模型为 Skip-gram，又如下图，标识了中心词和语境的概念。



因此选择的模型不同，构建数据集所配置的输入（解释变量）和输出（结果变量）y不同。因为 Skip-gram 结构模型表现要好于 CBOW，因此代码示例选择 Skip-gram 模型，其公式表示为，式中，为训练语境（上下文）窗口大小（可以是中心词的函数），较大的值产生更多的训练样本，从而获得更高的准确率，但训练时间也会增加。

在 Skip-gram 结构中，给定窗口大小，根据输入词预测该词语境下对应每个输出词有一个输出预测向量，为词汇表每个单词的概率，这意味着对数据集中的每一个训练样本执行一次就很容易达到数千万次的计算量。为了解决计算量问题，Mikolov, T.等人提出了 Negative Sampling（NEG）方法，首先将对输出词对应词汇表的概率分布预测调整为应用逻辑回归（Logistic Regression, LR）的二分类（类标为值1或0）问题，公式为wW

100%的正确，并没有从语料库中学习到任何价值的信息，因此需要填补这个漏洞，从词汇表中采样赋值为0的非临近字词中加入数据样本中，就是NEG方法，*wO*​务设定为使用逻辑回归从噪声分布Pn​(w)中区分目标单词wO，其中每个数据样本有k个负样本（值为0）​，公式为。根据试验，对于小的训练数据集，采样大小k推荐区间为5-20；对于较大的数据集推荐区间为2-5.噪声分布对应代码为：

def noise\_dist4sgns(freqs):  
 word\_freqs = np.array(sorted(freqs.values(), reverse=True))  
 unigram\_dist = word\_freqs/word\_freqs.sum()  
 noise\_dist = torch.from\_numpy(unigram\_dist\*\*(0.75)/np.sum(unigram\_dist\*\*(0.75)))  
  
 return noise\_dist

通过noise\_dist4sgns()函数获得噪声分布noise\_dist后，执行noise\_words = torch.multinomial(noise\_dist,batch\_size \* n\_samples,replacement=True)从输入张量（noise\_dist）相应行的多项概率分布中采样，返回给定数量的索引值。

在非常大的语料库中，最常见的单词很容易出现数亿次之多，例如 in、the 和 a 等，通常这些词提供的信息价值不如罕见词，例如 France 和 Paris等。为了解决罕见词和频繁词之间的不平衡，使用一种简单的抽样方法，根据公式 计算的概率，丢弃训练数据集中的每个词，式中为字词的频数；为一个删选阈值。对应代码为：

def subsampling\_of\_frequent\_words(int\_words,threshold=1e-5):  
 word\_counts = Counter(int\_words)  
 total\_count = len(int\_words)  
 freqs = {word: count/total\_count for word, count in word\_counts.items()}  
 p\_drop = {word: 1 - np.sqrt(threshold/freqs[word]) for word in word\_counts}  
   
 train\_words = [word for word in int\_words if random.random() < (1 - p\_drop[word])]  
   
 return train\_words,freqs

使用的数据集为[text8 - word embedding](https://www.kaggle.com/datasets/gupta24789/text8-word-embedding)㉕，为来自维基百科经过清洗的前 10 亿个字符数据。读取该数据，得知总共字词数约为 1668万，词汇表大小约为 6 万。

gensim库提供有[Word2vec](https://radimrehurek.com/gensim/models/word2vec.html)㉖方法，可以直接调用计算

text\_fn=r'I:\data\text8'  
with open(text\_fn) as f:  
 text=f.read()  
print(text[:100])  
  
words=usda\_models.text\_replace\_preprocess(text)  
print(words[:10])  
print("Total words in text: {}".format(len(words)))  
print("Unique words: {}".format(len(set(words))))

anarchism originated as a term of abuse first used against early working class radicals including t  
['anarchism', 'originated', 'as', 'a', 'term', 'of', 'abuse', 'first', 'used', 'against']  
Total words in text: 16680599  
Unique words: 63641

根据词频概率进行降采样。

vocab\_to\_int, int\_to\_vocab = usda\_models.create\_lookup\_tables4vocab(words)  
int\_words = [vocab\_to\_int[word] for word in words]  
print(int\_words[:10])  
train\_words,freqs=usda\_models.subsampling\_of\_frequent\_words(int\_words)  
print(train\_words[:10])

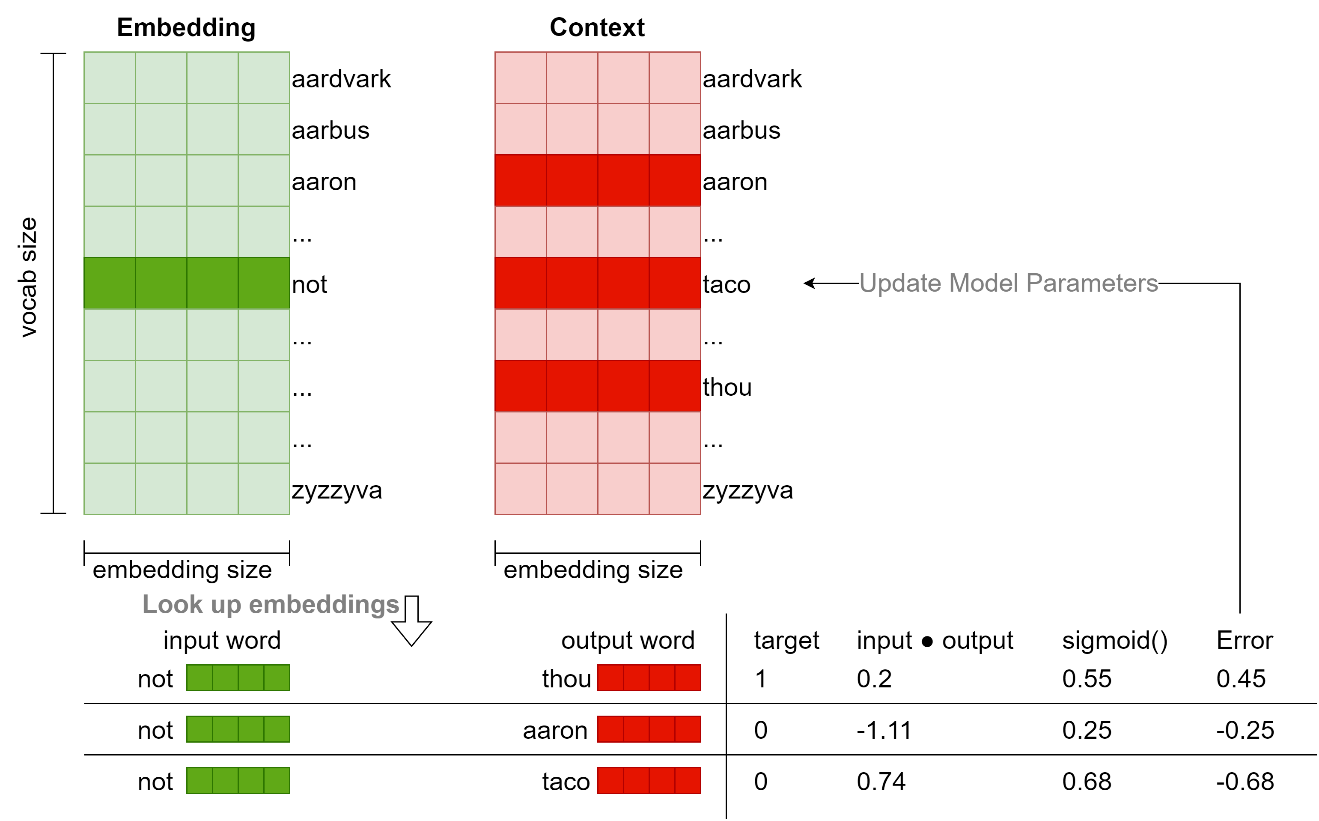
[5233, 3080, 11, 5, 194, 1, 3133, 45, 58, 155]  
[5233, 3080, 194, 3133, 127, 10571, 27349, 15067, 58112, 3580]

建立用于 Skip-gram 结构的数据集，可以看到字词（整数索引） 5233 对应着邻近的3个词 [3080, 194, 3133]，以此类推。

batch\_size=4  
X,y=next(usda\_models.get\_batches4word2vec(train\_words,batch\_size))  
print(X,y)

[5233, 5233, 5233, 3080, 3080, 3080, 194, 194, 3133] [3080, 194, 3133, 5233, 194, 3133, 3080, 3133, 194]

定义神经网络时，创建两个矩阵，一个为输入词嵌入矩阵，另一个为输出词嵌入矩阵。词嵌入矩阵维度为(vocab\_size,embedding\_size)，vocab\_size为词汇表的大小，embedding\_size为词嵌入向量大小（例如常用 300 大小），如图[16]。



在训练开始，用随机值初始化两个词嵌入矩阵。在迭代学习过程，对每一个样本输入，例如有输入词 not，和上下文邻近的词 thou（对应实际标签 target 的值为1），并用 NEG 方法采样了两个非邻近词 aaron 和 taco（对应实际标签 target 的值为0），通过对应随机初始化的两个嵌入矩阵查找输入词和输出词的词嵌入向量。取输入词和各输出词嵌入向量的点积（input ● output列），结果为一个数值，表示输入词嵌入和上下文词嵌入的相似性。用逻辑回归，sigmoid()函数将值转化为概率，获得预测，然后计算误差值用于梯度下降法更新开始随机生成的两个词嵌入矩阵。对应代码如下：

class SkipGramNeg(nn.Module):  
 def \_\_init\_\_(self, n\_vocab, n\_embed, noise\_dist=None):  
 super().\_\_init\_\_()  
   
 self.n\_vocab = n\_vocab  
 self.n\_embed = n\_embed  
 self.noise\_dist = noise\_dist  
   
 # define embedding layers for input and output words  
 self.in\_embed = nn.Embedding(n\_vocab,n\_embed)  
 self.out\_embed = nn.Embedding(n\_vocab,n\_embed)  
   
 # Initialize both embedding tables with uniform distribution  
 self.in\_embed.weight.data.uniform\_(-1,1)  
 self.out\_embed.weight.data.uniform\_(-1,1)  
   
 def forward\_input(self, input\_words):  
 # return input vector embeddings  
 input\_vector = self.in\_embed(input\_words)  
 return input\_vector  
   
 def forward\_output(self, output\_words):  
 # return output vector embeddings  
 output\_vector = self.out\_embed(output\_words)  
  
 return output\_vector  
   
 def forward\_noise(self, batch\_size, n\_samples,device):  
 """ Generate noise vectors with shape (batch\_size, n\_samples, n\_embed)"""  
 if self.noise\_dist is None:  
 # Sample words uniformly  
 noise\_dist = torch.ones(self.n\_vocab)  
 else:  
 noise\_dist = self.noise\_dist  
   
 # Sample words from our noise distribution  
 noise\_words = torch.multinomial(noise\_dist,  
 batch\_size \* n\_samples,  
 replacement=True)  
  
 noise\_words = noise\_words.to(device)  
   
 ## TODO: get the noise embeddings  
 # reshape the embeddings so that they have dims (batch\_size, n\_samples, n\_embed)  
 # as we are adding the noise to the output, so we will create the noise vectr using the  
 # output embedding layer  
 noise\_vector = self.out\_embed(noise\_words).view(batch\_size,n\_samples,self.n\_embed)   
 return noise\_vector

用SkipGramNeg类初始化模型为model，并定义NegativeSamplingLoss类，初始化损失（可从 USDA 库查看源代码）criterion，和用Adam为优化函数。

device = 'cuda' if torch.cuda.is\_available() else 'cpu'  
noise\_dist =usda\_models.noise\_dist4sgns(freqs)  
  
# instantiating the model  
embedding\_dim=300  
model = usda\_models.SkipGramNeg(len(vocab\_to\_int), embedding\_dim, noise\_dist=noise\_dist).to(device)   
print(f'model:\n{model}')   
  
# using the loss that we defined  
criterion = usda\_models.NegativeSamplingLoss()   
optimizer = torch.optim.Adam(model.parameters(), lr=0.003)

model:  
SkipGramNeg(  
 (in\_embed): Embedding(63641, 300)  
 (out\_embed): Embedding(63641, 300)  
)

定义sgns\_train()函数训练模型，并将模型参数保持值本地磁盘。

save\_path=r'I:\model\_ckpts\word2vec\sgns\_model.pth'  
usda\_models.sgns\_train(model,criterion,optimizer,train\_words,int\_to\_vocab,device=device,save\_path=save\_path,print\_every = 1000)

Epoch:1/10;Loss:8.854703903198242 Epoch:1/10;Loss:6.061638832092285 Epoch:1/10;Loss:5.2776594161987305 Epoch:1/10;Loss:4.505328178405762 Epoch:1/10;Loss:3.5800371170043945 Epoch:1/10;Loss:3.354184150695801 Epoch:1/10;Loss:3.191371202468872 Epoch:1/10;Loss:3.191371202468872 Epoch:1/10;Loss:3.150404930114746

Epoch:2/10;Loss:3.110663652420044 Epoch:3/10;Loss:2.7591896057128906 Epoch:4/10;Loss:2.5728249549865723 Epoch:5/10;Loss:2.3610575199127197 Epoch:6/10;Loss:2.4418580532073975 Epoch:7/10;Loss:2.1069562435150146 Epoch:8/10;Loss:1.9530032873153687 Epoch:9/10;Loss:2.1992568969726562 …

Epoch:10/10;Loss:2.0467514991760254 will | when, be, just, can, must history | links, external, article, overview, encyclopedia one | nine, seven, eight, six, five have | are, that, as, been, this may | can, be, conditions, not, have that | be, not, this, a, can often | some, most, can, sometimes, or from | by, in, of, are, the ice | frozen, snow, temperatures, rock, freezing older | families, those, appear, age, average ocean | atlantic, volcanic, oceans, arabsat, coastal numerous | including, these, among, significant, include derived | mechanics, word, describe, derivation, derive heavy | metal, armor, reactor, heavily, water lived | died, he, father, had, until defense | defensive, defence, warheads, military, defenses …

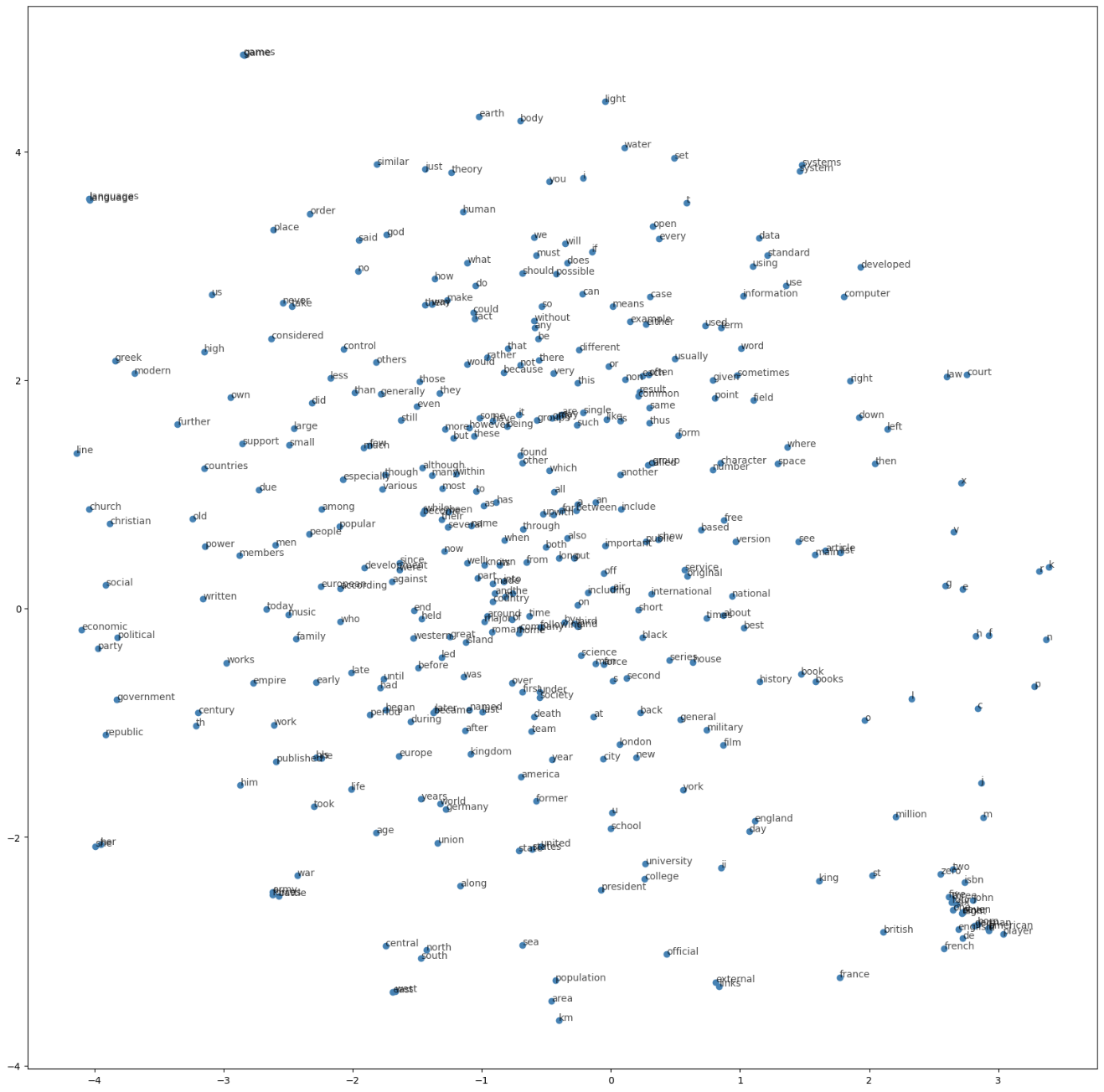
上述训练了10个迭代，从结果来看，语义相近的词嵌入向量的距离开始减小，例如对于词ice相近的词预测有frozen、snow、temperatures等，这符合语料库语句搭配的常识。下面调用训练模型参数，查看结果。

save\_path=r'I:\model\_ckpts\word2vec\sgns\_model.pth'  
checkpoint=torch.load(save\_path)  
model.load\_state\_dict(checkpoint['model\_state\_dict'])  
model.to(device)

SkipGramNeg(  
 (in\_embed): Embedding(63641, 300)  
 (out\_embed): Embedding(63641, 300)  
)

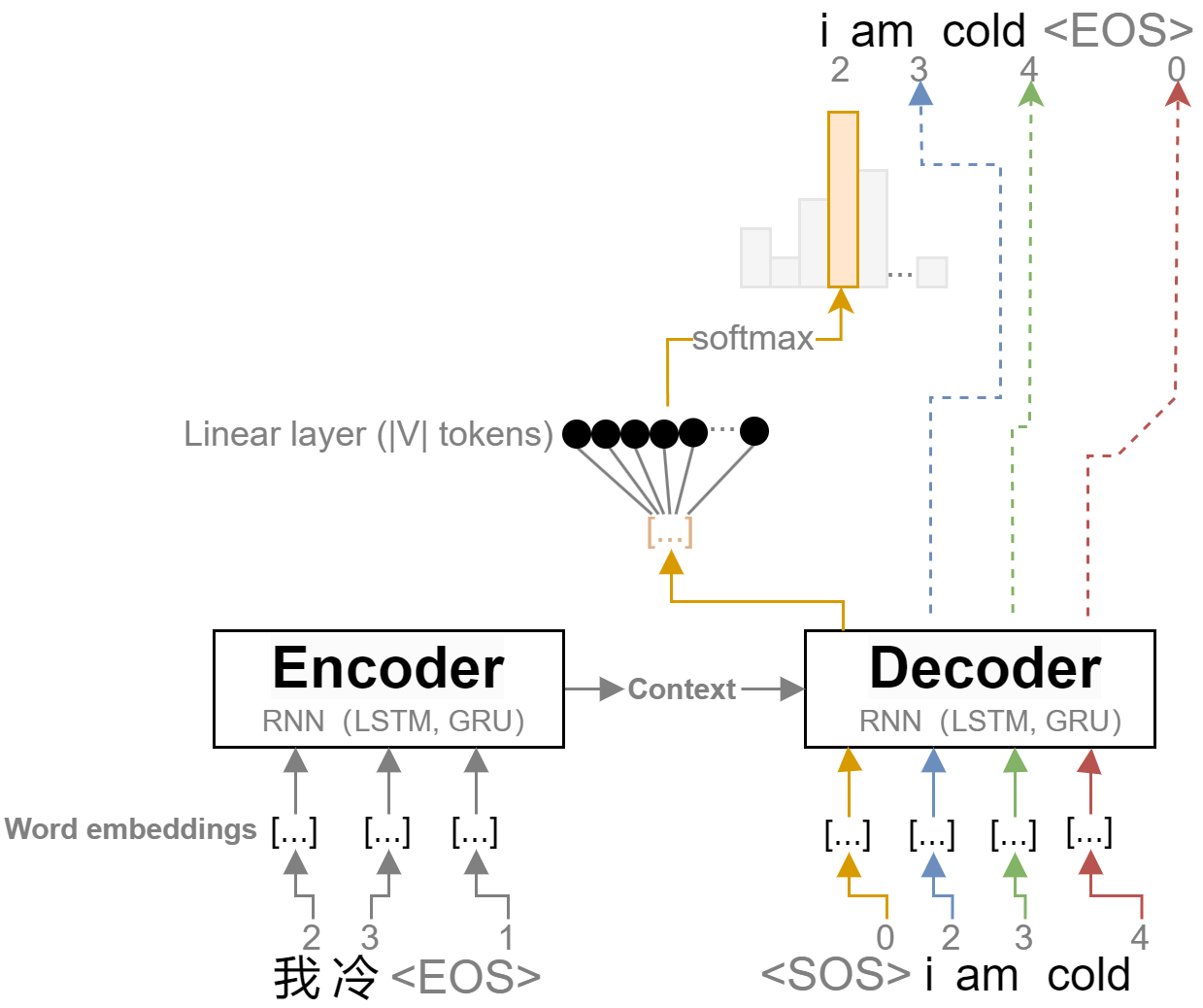
每一个词嵌入向量有 300 维，通过Sklearn库提供的TSNE方法降维。TSNE是一个高维数据可视化工具，将数据点之间的相似性（距离）转换为联合概率并试图最小化低维嵌入和高维数据联合概率之间的 Kullback-Leibler 散度（divergence）[20]。 从下述打印结果可以观察到，词嵌入向量间的距离越小，数据点越相互靠近，表明这些词具有更多的相似性。

embeddings = model.in\_embed.weight.to('cpu').data.numpy()  
viz\_words = 380  
tsne = TSNE()  
embed\_tsne = tsne.fit\_transform(embeddings[:viz\_words, :])  
  
fig, ax = plt.subplots(figsize=(20, 20))  
for idx in range(viz\_words):  
 plt.scatter(\*embed\_tsne[idx, :], color='steelblue')  
 plt.annotate(int\_to\_vocab[idx], (embed\_tsne[idx, 0], embed\_tsne[idx, 1]), alpha=0.7)



### 3.6.2.2 Seq2Seq

Sutskever, I.等人[21] 2014 年提出了序列到序列（Sequence to Sequence，seq2seq）的神经网络学习模型， 为一种端到端（end-to-end）的序列学习方法，用 LSTM 网络将输入序列映射到一个固定维数的向量（编码器（encoder）），然后使用另一个 LSTM 网络将该向量解码为目标序列（解码器（decoder））。PyTorch库也提供了一个 seq2seq 教程[22]，用 PyTorch 实现了 seq2seq 网络（但将 LSTM 替换为 GRU 网络，并在解码器部分引入了注意力机制（Attention mechanism））。本节以上述材料为主，将代码迁移至USDA库，解释该网络的核心算法。seq2seq 在机器翻译、文本归纳（摘要）和图像字幕等任务中有不菲成绩。不考虑注意力机制下 seq2seq 的网络结构如下图[22,23]。



以将中文“我冷”翻译为英文“i am cold”为例，

1. 依据语料库先建立两种语言各自的词汇表，并索引为整数，如我的索引为2等；
2. 初始化两个词汇表词嵌入向量矩阵，大小为两个词汇表各自的长度和词嵌入向量的维度，试验中配置为 128 维。用第1步建立的索引检索词向量；
3. 将每一样本的词嵌入输入 RNN 模型，通常用 LSTM 或者 GRU 网络（ LSTM 的网络结构查看本章开始部分）。编码（Encoder）部分的输出表述为语境/上下文（Context），作为解码（Decoder）的输入；
4. 每个句子的末尾结束的输入用<EOS>标识（配置索引值为1），而在解码时，句子开始标识为<SOS>（配置索引值为0）；
5. 解码部分每一个单元的输出用全连接层和softmax归一化指数函数获得预测词的概率分布，区间[0,1]，和为1；
6. 试验中的损失函数使用了torch.nn.NLLLoss负对数似然损失，例如假设第5步一个单元输出的概率分布为[[0.4501, 0.3294, 0.2204],[0.0400, 0.9130, 0.0470]]，表示有3个类别（0，1，2），2个样本。同时假设预测的标签为[0,2]，那么（配置参数reduction='mean'时的结果）；
7. 试验中的优化器选择optim.Adam，求解编码和解码的词嵌入参数。

Sutskever, I.等人在其论文中指出了三个重要方面，

1. 使用两个不同的 LSTM，一个用于输入序列；一个用于输出序列。这样做可以在计算成本可以忽略的条件下增加模型参数的数量，并可以很自然的在多个语言对上训练 LSTM；
2. 深层 LSTM 的性能明显优于浅层的 LSTM，因此构建了一个四层的 LSTM 网络；
3. 将输入句子的单词顺序颠倒是非常有价值的。例如不是将句子a、b、c映射到 α、β、 γ；而是要求 LSTM 将c、b、a 映射到 α、β、 γ。这样，a和α非常接近，b与β相当接近，依次类推。这使得随机梯度下降（stochastic gradient descent，SGD）很容易在输入和输出之间“建立通信”，从而极大的提高 LSTM 的性能。

* 数据及预处理

从[Tab-delimited Bilingual Sentence Pairs](https://www.manythings.org/anki/)㉗处下载[Mandarin Chinese - English cmn-eng.zip](https://www.manythings.org/anki/cmn-eng.zip)㉘中英制表符分隔的双语句子对。并读取，建立词汇表、索引映射关系和数据集加载器。

%load\_ext autoreload   
%autoreload 2   
import usda.models as usda\_models  
  
import torch  
from itertools import islice  
import matplotlib

语料库中有很多句子，如果要快速的学习到一些信息，可以把数据集修剪成相对较短的简单句子，这里配置参数MAX\_LENGTH = 10实现该目的。为了演示，简化训练难度，仅提取了以eng\_prefixes变量列出单词开头的句子，数据集样本量仅约500行。

root=r'I:\data\NLP\_dataset'   
MAX\_LENGTH = 10  
eng\_prefixes = (  
 "i am ", "i m ",  
 "he is", "he s ",  
 "she is", "she s ",  
 "you are", "you re ",  
 "we are", "we re ",  
 "they are", "they re "  
)  
  
hidden\_size = 256  
batch\_size = 32   
lang1, lang2='eng', 'cmn'  
device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")  
input\_lang, output\_lang,pairs, train\_dataloader = usda\_models.get\_dataloader4seq2seq(batch\_size,root,lang1, lang2,eng\_prefixes,MAX\_LENGTH,device,reverse=True,cmn=True)

Reading lines...  
Read 29476 sentence pairs  
Trimmed to 500 sentence pairs  
Counting words...  
Counted words:  
cmn 712  
eng 577

打印查看词汇表从字词到索引和从索引到字词的映射，及双语句子对示例。

print(pairs[0:15])  
print(list(islice(input\_lang.word2index.items(),0,30)))  
print(list(islice(input\_lang.index2word.items(),0,10)))  
print('-'\*50)  
print(list(islice(input\_lang.word2count.items(),0,10)))  
print('-'\*50)  
print(list(islice(output\_lang.word2index.items(),0,20)))

[['我 冷 ', 'i am cold'], ['我 吃 飽 了 ', 'i am full'], ['我 沒 事 ', 'i am okay'], ['我 生 病 了 ', 'i am sick'], ['我 个 子 高 ', 'i am tall'], ['他是一个 dj ', 'he is a dj'], ['他 很 忙 ', 'he is busy'], ['他 很 懒 ', 'he is lazy'], ['他 很 凶 ', 'he is mean'], ['他 很 穷 ', 'he is poor'], ['他 高 ', 'he is tall'], ['我 是 个 男 人 ', 'i am a man'], ['我 个 头 矮 ', 'i am short'], ['他 獨 自 一 人 ', 'he is alone'], ['我 來 了 ', 'i am coming']]  
[('我', 2), ('冷', 3), ('', 4), ('吃', 5), ('飽', 6), ('了', 7), ('沒', 8), ('事', 9), ('生', 10), ('病', 11), ('个', 12), ('子', 13), ('高', 14), ('他是一个', 15), ('dj', 16), ('他', 17), ('很', 18), ('忙', 19), ('懒', 20), ('凶', 21), ('穷', 22), ('是', 23), ('男', 24), ('人', 25), ('头', 26), ('矮', 27), ('獨', 28), ('自', 29), ('一', 30), ('來', 31)]  
[(0, 'SOS'), (1, 'EOS'), (2, '我'), (3, '冷'), (4, ''), (5, '吃'), (6, '飽'), (7, '了'), (8, '沒'), (9, '事')]  
--------------------------------------------------  
[('我', 174), ('冷', 2), ('', 500), ('吃', 5), ('飽', 1), ('了', 44), ('沒', 6), ('事', 5), ('生', 22), ('病', 3)]  
--------------------------------------------------  
[('i', 2), ('am', 3), ('cold', 4), ('full', 5), ('okay', 6), ('sick', 7), ('tall', 8), ('he', 9), ('is', 10), ('a', 11), ('dj', 12), ('busy', 13), ('lazy', 14), ('mean', 15), ('poor', 16), ('man', 17), ('short', 18), ('alone', 19), ('coming', 20), ('online', 21)]

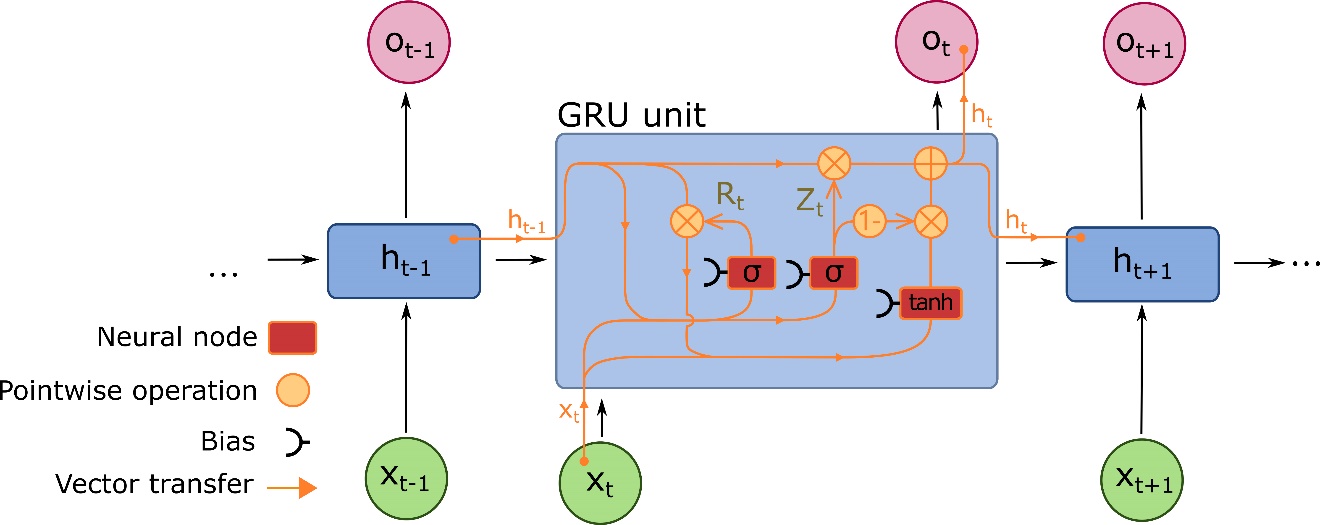
查看加载的句子对整数索引。在编码器中会通过nn.Embedding(input\_tensor)直接检索转换为对应的词嵌入向量。

for data in train\_dataloader:  
 input\_tensor, target\_tensor = data  
 print(input\_tensor[0])  
 print( target\_tensor[0])  
 break

tensor([ 17, 359, 40, 139, 686, 165, 4, 1, 0, 0], device='cuda:0')  
tensor([ 9, 10, 304, 243, 26, 10, 375, 545, 207, 1], device='cuda:0')

* 编码器和解码器

在编码器和解码器中使用的 RNN 为 GRU（gated recurrent unit）网络[24]，如图[25]，



公式为：

式中， 为时刻的隐藏状态，为时刻或初始时的隐藏状态，分别为重置门（reset gate）、更新门（update gate）和新门（new gate）；为 sigmoid 函数，为哈达玛积（Hadamard product）[26]。

编码器对应的代码为：

class EncoderRNN(nn.Module):  
 def \_\_init\_\_(self, input\_size, hidden\_size, dropout\_p=0.1):  
 super(EncoderRNN, self).\_\_init\_\_()  
 self.hidden\_size = hidden\_size  
  
 self.embedding = nn.Embedding(input\_size, hidden\_size)  
 self.gru = nn.GRU(hidden\_size, hidden\_size, batch\_first=True)  
 self.dropout = nn.Dropout(dropout\_p)  
  
 def forward(self, input):  
 embedded = self.dropout(self.embedding(input))  
 output, hidden = self.gru(embedded)  
 return output, hidden

解码器对应的代码为：

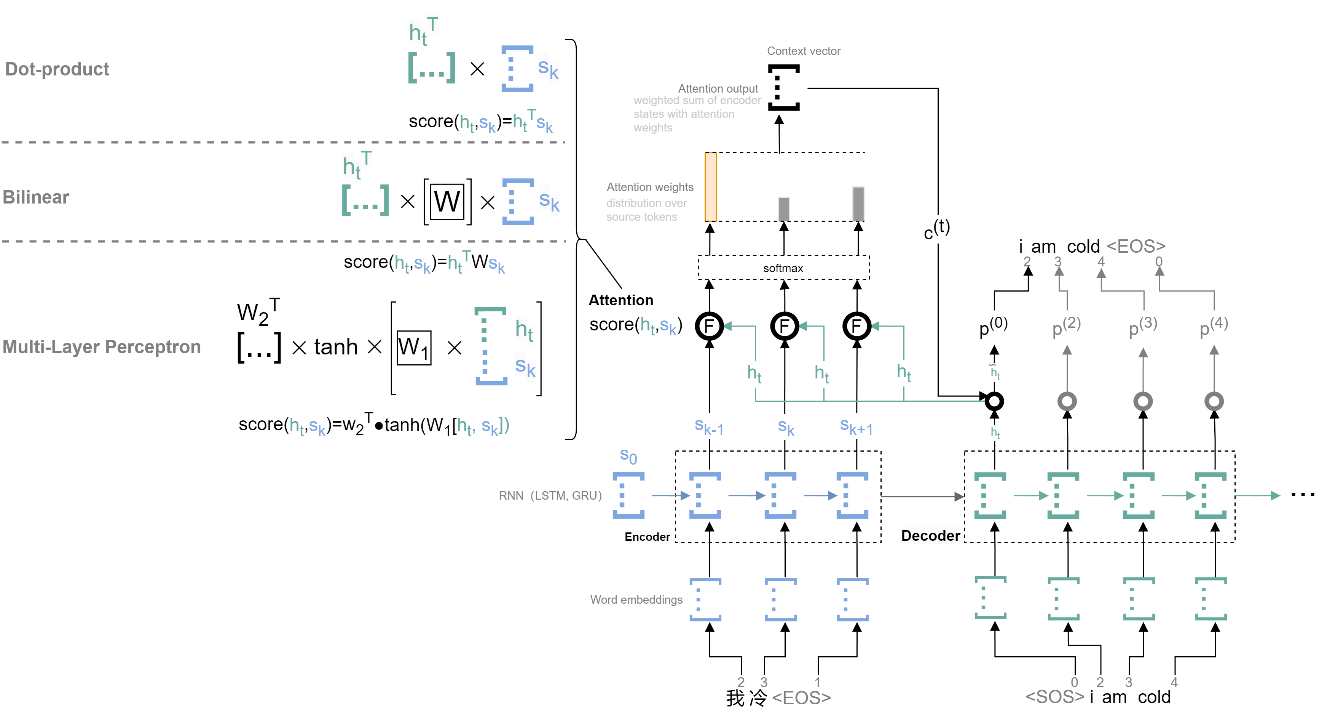
class DecoderRNN(nn.Module):  
 def \_\_init\_\_(self, hidden\_size, output\_size):  
 super(DecoderRNN, self).\_\_init\_\_()  
 self.embedding = nn.Embedding(output\_size, hidden\_size)  
 self.gru = nn.GRU(hidden\_size, hidden\_size, batch\_first=True)  
 self.out = nn.Linear(hidden\_size, output\_size)  
  
 def forward(self, encoder\_outputs, encoder\_hidden, target\_tensor=None):  
 batch\_size = encoder\_outputs.size(0)  
 decoder\_input = torch.empty(batch\_size, 1, dtype=torch.long, device=device).fill\_(SOS\_token)  
 decoder\_hidden = encoder\_hidden  
 decoder\_outputs = []  
  
 for i in range(MAX\_LENGTH):  
 decoder\_output, decoder\_hidden = self.forward\_step(decoder\_input, decoder\_hidden)  
 decoder\_outputs.append(decoder\_output)  
  
 if target\_tensor is not None:  
 # Teacher forcing: Feed the target as the next input  
 decoder\_input = target\_tensor[:, i].unsqueeze(1) # Teacher forcing  
 else:  
 # Without teacher forcing: use its own predictions as the next input  
 \_, topi = decoder\_output.topk(1)  
 decoder\_input = topi.squeeze(-1).detach() # detach from history as input  
  
 decoder\_outputs = torch.cat(decoder\_outputs, dim=1)  
 decoder\_outputs = F.log\_softmax(decoder\_outputs, dim=-1)  
 return decoder\_outputs, decoder\_hidden, None # We return `None` for consistency in the training loop  
  
 def forward\_step(self, input, hidden):  
 output = self.embedding(input)  
 output = F.relu(output)  
 output, hidden = self.gru(output, hidden)  
 output = self.out(output)  
 return output, hidden

encoder = usda\_models.EncoderRNN(input\_lang.n\_words, hidden\_size).to(device)  
decoder\_RNN = usda\_models.DecoderRNN(hidden\_size, output\_lang.n\_words).to(device)  
print(f'{encoder};\n{decoder\_RNN}')

EncoderRNN(  
 (embedding): Embedding(712, 256)  
 (gru): GRU(256, 256, batch\_first=True)  
 (dropout): Dropout(p=0.1, inplace=False)  
);  
DecoderRNN(  
 (embedding): Embedding(577, 256)  
 (gru): GRU(256, 256, batch\_first=True)  
 (out): Linear(in\_features=256, out\_features=577, bias=True)  
)

* 注意力机制（Attention mechanism）

由编码器 LSTM/GRU 最后单元（节点）的输出作为解码器的输入，这将一个句子样本压缩成单个向量的方式，使得解码器忽略了对应目标语言句子中各个单词成分（解码）与被翻译类型语言各词（编码）之间的关系，这对于长句子的处理表现得尤为明显。因此Luong, M.-T.等人[27]和Bahdanau, D.等人[28]分别在2014年和2015年提出了一种称为“注意力”（Attention）的技术，极大提高了机器翻译的质量。在基本的 seq2seq 网络中加入注意力机制如下图[23]，



* 注意力机制是神经网络的一部分，以解码器一个节点（隐藏）状态和编码器所有节点的（隐藏）状态；
* 计算注意力分数（attention scores），该分数反映了每一个编码节点状态与编码状态的相关程度。计算注意力分数形式上为的一个注意力函数，返回为一个标量值。有多种注意力函数，例如最简单的点积（dot-product）形式，双线性函数（bilinear，也称为 Luong attention），及多层感知机（multi-layer perceptron， 也称为 Bahdanau attention）；
* 由注意力分数计算注意力权重（attention weight），为应用 softmax 函数获得的一个概率分布，公式为；
* 计算注意力输出（attention output），为编码状态的注意力权重加权和，公式为。

加入注意力机制的网络，可以学习哪个编码节点对解码节点更重要，实现端到端（end-to-end）的训练。

试验中使用的注意力机制为 Bahdanau attention，对应代码如下：

class BahdanauAttention(nn.Module):  
 def \_\_init\_\_(self, hidden\_size):  
 super(BahdanauAttention, self).\_\_init\_\_()  
 self.Wa = nn.Linear(hidden\_size, hidden\_size)  
 self.Ua = nn.Linear(hidden\_size, hidden\_size)  
 self.Va = nn.Linear(hidden\_size, 1)  
  
 def forward(self, query, keys):  
 scores = self.Va(torch.tanh(self.Wa(query) + self.Ua(keys)))  
 scores = scores.squeeze(2).unsqueeze(1)  
  
 weights = F.softmax(scores, dim=-1)  
 context = torch.bmm(weights, keys)  
  
 return context, weights  
  
class AttnDecoderRNN(nn.Module):  
 def \_\_init\_\_(self, hidden\_size, output\_size, dropout\_p=0.1,MAX\_LENGTH=10,device='cpu'): # device='cpu'  
 super(AttnDecoderRNN, self).\_\_init\_\_()  
 self.MAX\_LENGTH=MAX\_LENGTH  
 self.device=device  
 self.embedding = nn.Embedding(output\_size, hidden\_size)  
 self.attention = BahdanauAttention(hidden\_size)  
 self.gru = nn.GRU(2 \* hidden\_size, hidden\_size, batch\_first=True)  
 self.out = nn.Linear(hidden\_size, output\_size)  
 self.dropout = nn.Dropout(dropout\_p)  
  
 def forward(self, encoder\_outputs, encoder\_hidden, target\_tensor=None):  
 batch\_size = encoder\_outputs.size(0)  
 decoder\_input = torch.empty(batch\_size, 1, dtype=torch.long,device=self.device).fill\_(SOS\_token)   
 decoder\_hidden = encoder\_hidden  
 decoder\_outputs = []  
 attentions = []  
  
 for i in range(self.MAX\_LENGTH):  
 decoder\_output, decoder\_hidden, attn\_weights = self.forward\_step(  
 decoder\_input, decoder\_hidden, encoder\_outputs  
 )  
 decoder\_outputs.append(decoder\_output)  
 attentions.append(attn\_weights)  
  
 if target\_tensor is not None:  
 # Teacher forcing: Feed the target as the next input  
 decoder\_input = target\_tensor[:, i].unsqueeze(1) # Teacher forcing  
 else:  
 # Without teacher forcing: use its own predictions as the next input  
 \_, topi = decoder\_output.topk(1)  
 decoder\_input = topi.squeeze(-1).detach() # detach from history as input  
  
 decoder\_outputs = torch.cat(decoder\_outputs, dim=1)  
 decoder\_outputs = F.log\_softmax(decoder\_outputs, dim=-1)  
 attentions = torch.cat(attentions, dim=1)  
  
 return decoder\_outputs, decoder\_hidden, attentions  
  
  
 def forward\_step(self, input, hidden, encoder\_outputs):  
 embedded = self.dropout(self.embedding(input))  
  
 query = hidden.permute(1, 0, 2)  
 context, attn\_weights = self.attention(query, encoder\_outputs)  
 input\_gru = torch.cat((embedded, context), dim=2)  
  
 output, hidden = self.gru(input\_gru, hidden)  
 output = self.out(output)  
  
 return output, hidden, attn\_weights

初始化注意力机制解码器网络，打印查看网络结构。

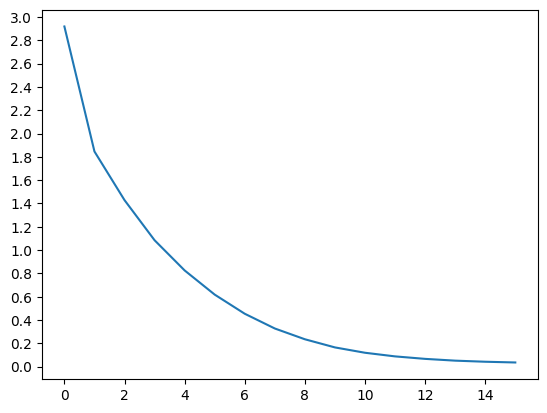
decoder = usda\_models.AttnDecoderRNN(hidden\_size, output\_lang.n\_words,MAX\_LENGTH=MAX\_LENGTH,device=device).to(device)  
print(decoder)

AttnDecoderRNN(  
 (embedding): Embedding(577, 256)  
 (attention): BahdanauAttention(  
 (Wa): Linear(in\_features=256, out\_features=256, bias=True)  
 (Ua): Linear(in\_features=256, out\_features=256, bias=True)  
 (Va): Linear(in\_features=256, out\_features=1, bias=True)  
 )  
 (gru): GRU(512, 256, batch\_first=True)  
 (out): Linear(in\_features=256, out\_features=577, bias=True)  
 (dropout): Dropout(p=0.1, inplace=False)  
)

训练并保存模型，下述仅训练了80个迭代的演示。

save\_path=r'I:\model\_ckpts\seq2seq\seq2seq\_ch2en.pth'  
usda\_models.seq2seq\_train(train\_dataloader, encoder, decoder, 80, print\_every=5, plot\_every=5,save\_path=save\_path)

0m 3s (- 0m 51s) (5 6%) 2.9182  
0m 6s (- 0m 47s) (10 12%) 1.8459  
0m 10s (- 0m 47s) (15 18%) 1.4285  
0m 14s (- 0m 43s) (20 25%) 1.0843  
0m 18s (- 0m 40s) (25 31%) 0.8253  
0m 21s (- 0m 36s) (30 37%) 0.6180  
0m 25s (- 0m 32s) (35 43%) 0.4538  
0m 29s (- 0m 29s) (40 50%) 0.3272  
0m 32s (- 0m 24s) (45 56%) 0.2349  
0m 35s (- 0m 21s) (50 62%) 0.1649  
0m 39s (- 0m 17s) (55 68%) 0.1191  
0m 43s (- 0m 14s) (60 75%) 0.0879  
0m 47s (- 0m 10s) (65 81%) 0.0666  
0m 50s (- 0m 7s) (70 87%) 0.0514  
0m 53s (- 0m 3s) (75 93%) 0.0420  
0m 56s (- 0m 0s) (80 100%) 0.0357  
  
  
  
<Figure size 640x480 with 0 Axes>



读取训练的模型参数，从数据集中随机选择双语句子对，验证训练模型的翻译。输入为中文，实际翻译用=标识，<为训练模型的翻译结果。

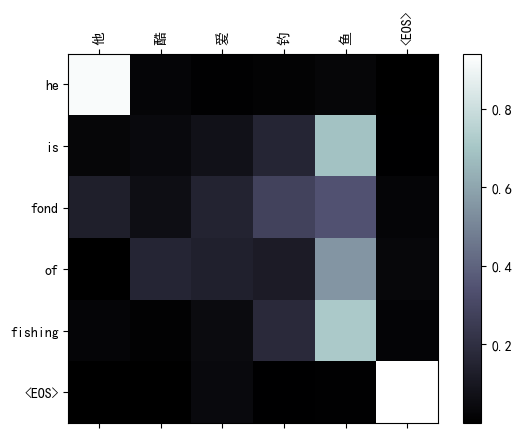
save\_path=r'I:\model\_ckpts\seq2seq\seq2seq\_ch2en.pth'  
checkpoint=torch.load(save\_path)  
encoder.load\_state\_dict(checkpoint['best\_encoder\_state\_dict'])   
encoder.to(device)   
decoder.load\_state\_dict(checkpoint['best\_decoder\_state\_dict'])   
decoder.to(device)   
  
encoder.eval()  
decoder.eval()  
usda\_models.seq2seq\_evaluateRandomly(encoder, decoder,pairs,input\_lang, output\_lang,device)

> 我 冷   
= i am cold  
< i am cold <EOS>  
  
> 她 不 在 是 因 为 病 了   
= she is absent because of sickness  
< she is absent because of sickness <EOS>  
  
> 他 是 美 国 人   
= he is american  
< he is an american <EOS>  
  
> 我 在 家 裡   
= i am in the house  
< i am at home <EOS>  
  
> 他 酷 爱 钓 鱼   
= he is fond of fishing  
< he is fond of fishing <EOS>  
  
> 你 喝 醉 了   
= you are drunk  
< you are drunk <EOS>  
  
> 她 会 十 门 语 言   
= she is able to speak ten languages  
< she is able to speak ten languages <EOS>  
  
> 她 錢 不 夠   
= she is hard up for money  
< she is hard up for money <EOS>  
  
> 我 不 是 學 生   
= i am not a student  
< i am not a student <EOS>  
  
> 他 很 英 俊   
= he is very handsome  
< he is very handsome <EOS>

模型返回了注意力机制的权重值，通过可视化双语标签和权重值，能够观察到解码中各字词对应到编码中各字词所关注的位置。

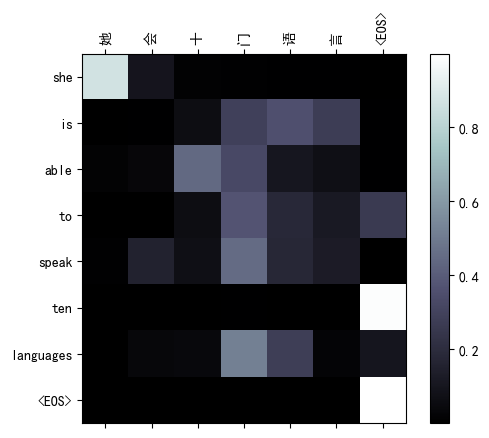
matplotlib.rcParams['font.family'] = ['SimHei']   
  
sentence='他酷爱钓鱼'   
input\_sentence=' '.join(f'{sentence}')  
usda\_models.evaluateAndShowAttention(encoder, decoder, input\_sentence, input\_lang, output\_lang,device)

input = 他 酷 爱 钓 鱼  
output = he is fond of fishing <EOS>



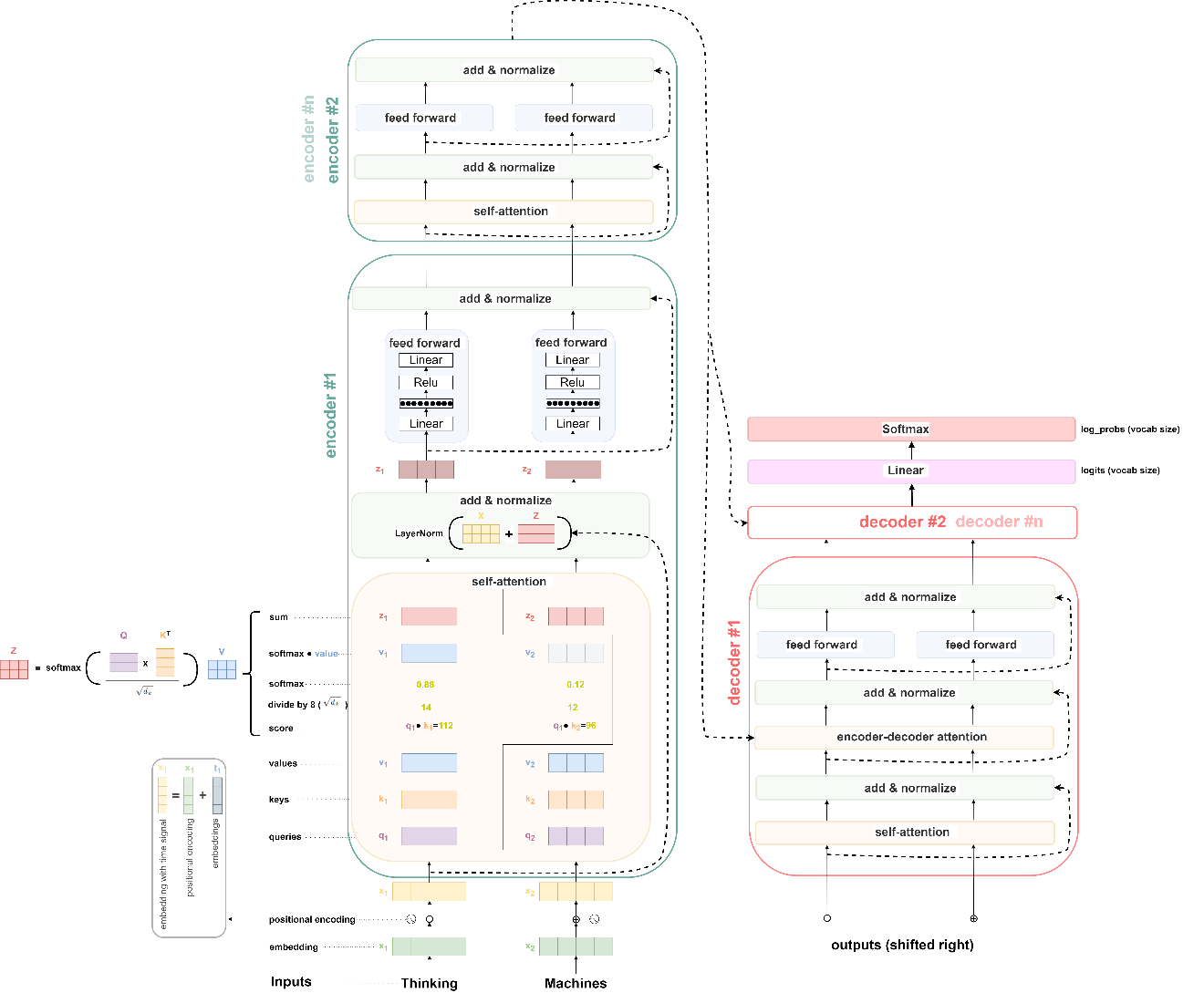
sentence='她会十门语言'   
input\_sentence=' '.join(f'{sentence}')  
usda\_models.evaluateAndShowAttention(encoder, decoder, input\_sentence, input\_lang, output\_lang,device)

input = 她 会 十 门 语 言  
output = she is able to speak ten languages <EOS>



### 3.6.2.3 Transformer

与 RNN（Vallina、LSTM、GRU）和 基于 RNN 的 seq2seq 等递归神经网络不同，Vaswani, A.等人[29] 2017年提出了 transformer 网络，并已证明在序列到序列（sequence-to-sequence）任务中具有更好的表现，且拥有更高的并行性。transformer 并不使用 RNN 网络，以注意力机制为主要思想构建网络，这包括响应解码对编码各个词汇（token）的注意力（参数queries向量来自于解码状态；参数keys和values向量来自编码状态），即encoder-decoder attention（编码-解码 注意力）；也包括解码器输入语句自身各个词汇对其它词汇的注意力，以更好的理解和表示序列，即self-attention（自注意力），例如对于文本The animal didn't cross the street because it was too tired，词汇it代指其它词汇animal，而不是street等。transformer 的网络结构如下图[30]，同样包括编码器和解码器，编码器中引入self-attention，解码器中对应编码器引入encoer-decoder attention，以及为了表述文本词语顺序的position encoding层，改善收敛稳定性和提高训练质量的add & normalize层，及处理注意力机制等新增信息的前馈神经网络层feed forward（FFN）等。同时，transformer 有多个编码器和解码器堆叠，而self-attention重复计算多次，获得多个z矩阵后并压缩为1个，称为multi-headed。



#### 1）位置编码 —— Positional encoding

由于 transformer 网络不含递归和卷积，为了使模型利用序列顺序信息，需要注入一些关于序列中词汇的相对或绝对位置信息。因此，在编码器和解码器输入端对词嵌入添加了“位置编码”（positional encodings）。位置编码与嵌入具有相同的维数，因此可以对两者求和。位置编码可以通过学习习得，也可以用固定的形似，例如使用不同频率的正弦和余弦函数，公式为，；，式中，是位置，是维数，即位置编码的每一个维度对应于一个正弦波。波长为从2*π*到10000·2*π*

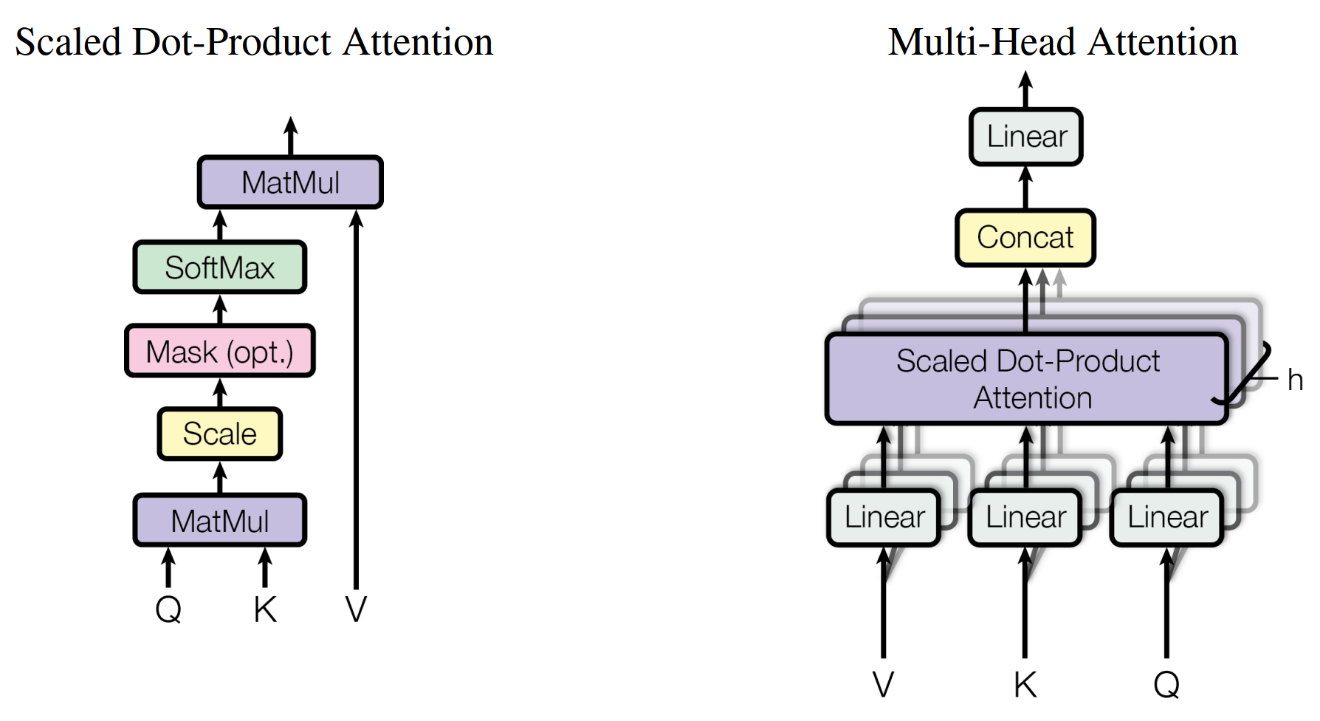
kPEpos+kPEpos的线性函数。

对应代码为：

class PositionalEncoding(nn.Module):  
  
 def \_\_init\_\_(self, d\_model: int, dropout: float = 0.1, max\_len: int = 5000):  
 super().\_\_init\_\_()  
 self.dropout = nn.Dropout(p=dropout)  
  
 position = torch.arange(max\_len).unsqueeze(1)  
 div\_term = torch.exp(torch.arange(0, d\_model, 2) \* (-math.log(10000.0) / d\_model))  
 pe = torch.zeros(max\_len, 1, d\_model)  
 pe[:, 0, 0::2] = torch.sin(position \* div\_term)  
 pe[:, 0, 1::2] = torch.cos(position \* div\_term)  
 self.register\_buffer('pe', pe)  
  
 def forward(self, x: Tensor) -> Tensor:  
 """  
 Arguments:  
 x: Tensor, shape ``[seq\_len, batch\_size, embedding\_dim]``  
 """  
 x = x + self.pe[:x.size(0)]  
 return self.dropout(x)

#### 2）注意力（attention）—— self-attention

注意力函数（同时可以查看下图[29]）可以描述为将查询（queries）和一组键值对（keys，values）映射到输出，其中查询、键、值和输出都是向量，查询和键的维度为，值得维度为。输出为值的加权和，且分配给每个值的权重由查询与相应键的兼容性（compatibility）函数计算。将自注意力（self-attention）称之为缩放的点积注意力（Scaled Dot-Product Attention），首先计算查询和键大的点积（dot products），然后除以，并应用 softmax 函数获得值的权重，公式为。



transformer 作者发现与其使用维度的键、值和查询的一组注意力函数，不如并行多组生成维输出，并连接（ concatenate）起来再次映射得到最终值，称为多头注意力（Multi-head attention ）。多头注意力允许模型联合处理来自不同位置不同表示的子空间信息，公式可表述为，其中映射的参数矩阵有 ,。

#### 3）层归一化（Layer normalization）—— add & normalize[31]

一层输出的变化往往会导致下一层累计输入发生高度相关的变化， 尤其对于输出变化很大的 ReLU 层，而“协变量位移”（covariate shift”）问题可以通过固定每一层中总和输入的均值和方差来减少，公式为，式中，$ $是可学习的仿射变化参数[32]。PyTorch库将层归一化方法写入到了torch.nn.LayerNorm类中。

#### 4）前馈神经网络 —— feed forward

除了注意力层外，编码器和解码器的每一层都包含并行多个完全连接的前馈神经网络，对应连接注意力层的各个输出。每个前馈网络有两个线性变化和一个 ReLU 层，公式为。虽然线性变换在不同位置上是相同的，但它们在每一层之间使用不同的参数。

#### 5）残差连接（ Residual connections）

在 transformer 网络图解中可以观察到层间连接的虚线，即为残差连接（ Residual connections），为一种在深度神经网络中广泛应用的技术，旨在解决深度神经网络训练过程中的梯度消失和梯度爆炸等问题，例如信息从一层传递到下一层时存在的信息损失。

PyTorch将 transformer 的方法写入到了torch.nn.TransformerEncoderLayer中，并给出了[Language modeling with nn.transformer and torchtext](https://pytorch.org/tutorials/beginner/transformer_tutorial.html#evaluate-the-best-model-on-the-test-dataset)㉙示例文档。这里将该文档的方法迁移到USDA库，方便调用。

%load\_ext autoreload   
%autoreload 2   
  
import usda.models as usda\_models  
  
from torchtext.datasets import WikiText2  
from torchtext.data.utils import get\_tokenizer  
from torchtext.vocab import build\_vocab\_from\_iterator  
import torch  
from torch import nn  
import os  
import time  
import math

通过PyTorch的[torchtext.datasets](https://pytorch.org/text/stable/datasets.html)㉚可以获得多种用于自然语言处理的文本数据集，示例中调入的为[WikiText2](https://pytorch.org/text/stable/datasets.html#wikitext-2)㉛数据集。

train\_iter=WikiText2(split='train')  
tokenizer=get\_tokenizer('basic\_english')  
vocab=build\_vocab\_from\_iterator(map(tokenizer, train\_iter), specials=['<unk>'])  
vocab.set\_default\_index(vocab['<unk>'])  
  
train\_iter, val\_iter, test\_iter=WikiText2()  
train\_data=usda\_models.data\_process4transformer(train\_iter,vocab,tokenizer)  
val\_data=usda\_models.data\_process4transformer(val\_iter,vocab,tokenizer)  
test\_data=usda\_models.data\_process4transformer(test\_iter,vocab,tokenizer)

建立训练数据集。

batch\_size=20  
eval\_batch\_size=10  
train\_data=usda\_models.batchify(train\_data, batch\_size)   
val\_data=usda\_models.batchify(val\_data, eval\_batch\_size)  
test\_data=usda\_models.batchify(test\_data, eval\_batch\_size)   
  
print(f'train data:\n{train\_data}\nshape={train\_data.shape}')

train data:  
tensor([[ 9, 59, 564, ..., 11652, 2435, 1],  
 [ 3849, 12, 300, ..., 47, 30, 1990],  
 [ 3869, 315, 19, ..., 97, 7720, 4],  
 ...,  
 [ 587, 4011, 59, ..., 1, 1439, 12313],  
 [ 4987, 29, 4, ..., 3165, 17106, 2060],  
 [ 6, 8, 1, ..., 62, 18, 2]], device='cuda:0')  
shape=torch.Size([102499, 20])

配置 transformer 模型参数，并初始化模型。

device=torch.device('cuda' if torch.cuda.is\_available() else 'cpu')  
  
ntokens=len(vocab) # size of vocabulary  
emsize=200 # embedding dimension  
d\_hid=200 # dimension of the feedforward network model in ``nn.TransformerEncoder``  
nlayers=2 # number of ``nn.TransformerEncoderLayer`` in ``nn.TransformerEncoder``  
nhead=2 # number of heads in ``nn.MultiheadAttention``  
dropout=0.2 # dropout probability  
model=usda\_models.TransformerModel(ntokens, emsize, nhead, d\_hid, nlayers, dropout).to(device)   
print(model)

TransformerModel(  
 (pos\_encoder): PositionalEncoding(  
 (dropout): Dropout(p=0.2, inplace=False)  
 )  
 (transformer\_encoder): TransformerEncoder(  
 (layers): ModuleList(  
 (0-1): 2 x TransformerEncoderLayer(  
 (self\_attn): MultiheadAttention(  
 (out\_proj): NonDynamicallyQuantizableLinear(in\_features=200, out\_features=200, bias=True)  
 )  
 (linear1): Linear(in\_features=200, out\_features=200, bias=True)  
 (dropout): Dropout(p=0.2, inplace=False)  
 (linear2): Linear(in\_features=200, out\_features=200, bias=True)  
 (norm1): LayerNorm((200,), eps=1e-05, elementwise\_affine=True)  
 (norm2): LayerNorm((200,), eps=1e-05, elementwise\_affine=True)  
 (dropout1): Dropout(p=0.2, inplace=False)  
 (dropout2): Dropout(p=0.2, inplace=False)  
 )  
 )  
 )  
 (embedding): Embedding(28782, 200)  
 (linear): Linear(in\_features=200, out\_features=28782, bias=True)  
)

配置损失和优化函数，及调整学习率。

criterion=nn.CrossEntropyLoss()  
lr=5.0 # learning rate  
optimizer=torch.optim.SGD(model.parameters(), lr=lr)  
scheduler=torch.optim.lr\_scheduler.StepLR(optimizer, 1.0, gamma=0.95)   
   
best\_val\_loss=float('inf')  
epochs=2  
bptt=35  
  
save\_path=r'I:\model\_ckpts\transformer'  
best\_model\_params\_path=os.path.join(save\_path, "transformer\_model.pth")

训练模型和保存模型参数。

for epoch in range(1, epochs + 1):  
 epoch\_start\_time=time.time()  
 usda\_models.train4transformer(model,criterion,optimizer,scheduler,train\_data,ntokens,epoch,bptt)  
 val\_loss=usda\_models.evaluate4transformer(model,criterion, ntokens, val\_data)  
 val\_ppl=math.exp(val\_loss)  
 elapsed=time.time() - epoch\_start\_time  
 print('-' \* 89)  
 print(f'| end of epoch {epoch:3d} | time: {elapsed:5.2f}s | '  
 f'valid loss {val\_loss:5.2f} | valid ppl {val\_ppl:8.2f}')  
 print('-' \* 89)  
  
 if val\_loss < best\_val\_loss:  
 best\_val\_loss = val\_loss  
 torch.save(model.state\_dict(), best\_model\_params\_path)  
  
 scheduler.step()

| epoch 1 | 200/ 2928 batches | lr 5.00 | ms/batch 14.29 | loss 2.66 | ppl 14.35  
| epoch 1 | 400/ 2928 batches | lr 5.00 | ms/batch 14.31 | loss 2.77 | ppl 15.97  
| epoch 1 | 600/ 2928 batches | lr 5.00 | ms/batch 13.56 | loss 2.66 | ppl 14.23  
| epoch 1 | 800/ 2928 batches | lr 5.00 | ms/batch 14.54 | loss 2.58 | ppl 13.20  
| epoch 1 | 1000/ 2928 batches | lr 5.00 | ms/batch 13.77 | loss 2.63 | ppl 13.94  
| epoch 1 | 1200/ 2928 batches | lr 5.00 | ms/batch 13.65 | loss 2.56 | ppl 12.95  
| epoch 1 | 1400/ 2928 batches | lr 5.00 | ms/batch 13.72 | loss 2.95 | ppl 19.12  
| epoch 1 | 1600/ 2928 batches | lr 5.00 | ms/batch 14.50 | loss 2.67 | ppl 14.46  
| epoch 1 | 1800/ 2928 batches | lr 5.00 | ms/batch 14.95 | loss 2.75 | ppl 15.64  
| epoch 1 | 2000/ 2928 batches | lr 5.00 | ms/batch 14.84 | loss 2.64 | ppl 14.03  
| epoch 1 | 2200/ 2928 batches | lr 5.00 | ms/batch 13.94 | loss 3.04 | ppl 20.91  
| epoch 1 | 2400/ 2928 batches | lr 5.00 | ms/batch 13.83 | loss 2.73 | ppl 15.32  
| epoch 1 | 2600/ 2928 batches | lr 5.00 | ms/batch 14.64 | loss 2.88 | ppl 17.75  
| epoch 1 | 2800/ 2928 batches | lr 5.00 | ms/batch 15.53 | loss 2.62 | ppl 13.72  
-----------------------------------------------------------------------------------------  
| end of epoch 1 | time: 42.03s | valid loss 0.00 | valid ppl 1.00  
-----------------------------------------------------------------------------------------  
| epoch 2 | 200/ 2928 batches | lr 4.75 | ms/batch 14.93 | loss 2.75 | ppl 15.61  
| epoch 2 | 400/ 2928 batches | lr 4.75 | ms/batch 14.13 | loss 2.57 | ppl 13.06  
| epoch 2 | 600/ 2928 batches | lr 4.75 | ms/batch 14.23 | loss 2.61 | ppl 13.56  
| epoch 2 | 800/ 2928 batches | lr 4.75 | ms/batch 14.76 | loss 2.67 | ppl 14.49  
| epoch 2 | 1000/ 2928 batches | lr 4.75 | ms/batch 16.87 | loss 2.72 | ppl 15.11  
| epoch 2 | 1200/ 2928 batches | lr 4.75 | ms/batch 15.59 | loss 2.77 | ppl 15.92  
| epoch 2 | 1400/ 2928 batches | lr 4.75 | ms/batch 15.32 | loss 2.81 | ppl 16.54  
| epoch 2 | 1600/ 2928 batches | lr 4.75 | ms/batch 15.56 | loss 2.75 | ppl 15.72  
| epoch 2 | 1800/ 2928 batches | lr 4.75 | ms/batch 14.91 | loss 2.56 | ppl 13.00  
| epoch 2 | 2000/ 2928 batches | lr 4.75 | ms/batch 16.19 | loss 3.02 | ppl 20.57  
| epoch 2 | 2200/ 2928 batches | lr 4.75 | ms/batch 16.29 | loss 2.70 | ppl 14.90  
| epoch 2 | 2400/ 2928 batches | lr 4.75 | ms/batch 16.36 | loss 2.66 | ppl 14.34  
| epoch 2 | 2600/ 2928 batches | lr 4.75 | ms/batch 17.28 | loss 2.57 | ppl 13.09  
| epoch 2 | 2800/ 2928 batches | lr 4.75 | ms/batch 16.56 | loss 2.45 | ppl 11.61  
-----------------------------------------------------------------------------------------  
| end of epoch 2 | time: 46.02s | valid loss 0.00 | valid ppl 1.00  
-----------------------------------------------------------------------------------------

加载模型，并通过给定的提示词预测后续词。

model.load\_state\_dict(torch.load(best\_model\_params\_path)) # load best model states   
  
prompt="the first in the Super Mario Land series , developed and published by Nintendo as a launch title for their Game Boy handheld game console . In gameplay similar to that of the 1985 Super Mario Bros. , but resized for the smaller device 's screen , the player advances Mario to the end of 12 levels by moving to the right and jumping across platforms to avoid enemies and pitfalls . Unlike other Mario games , Super Mario Land is set in Sarasaland , a new environment depicted in line art ,"  
indexed\_tokens=usda\_models.data\_process4transformer(tokenizer(prompt),vocab,tokenizer)  
print(indexed\_tokens)  
indexed\_tokens\_batch=usda\_models.batchify(indexed\_tokens, bsz=10)  
print(indexed\_tokens\_batch)

tensor([ 1, 37, 6, 1, 1269, 2124, 352, 93, 2, 452,  
 5, 326, 19, 3179, 14, 8, 2847, 363, 17, 36,  
 67, 1361, 14407, 67, 5972, 3, 6, 2096, 372, 7,  
 16, 4, 1, 1897, 1269, 2124, 9480, 3, 2, 38,  
 0, 17, 1, 952, 5135, 11, 15, 1871, 2, 1,  
 285, 5561, 2124, 7, 1, 140, 4, 267, 1329, 19,  
 888, 7, 1, 463, 5, 5369, 431, 5890, 7, 1947,  
 4490, 5, 27153, 3, 1676, 55, 2124, 193, 2, 1269,  
 2124, 352, 23, 219, 6, 27675, 2, 8, 54, 3076,  
 2478, 6, 195, 488, 2])  
tensor([[ 1, 452, 17, 2096, 9480, 11, 1, 463, 27153, 352],  
 [ 37, 5, 36, 372, 3, 15, 140, 5, 3, 23],  
 [ 6, 326, 67, 7, 2, 1871, 4, 5369, 1676, 219],  
 [ 1, 19, 1361, 16, 38, 2, 267, 431, 55, 6],  
 [ 1269, 3179, 14407, 4, 0, 1, 1329, 5890, 2124, 27675],  
 [ 2124, 14, 67, 1, 17, 285, 19, 7, 193, 2],  
 [ 352, 8, 5972, 1897, 1, 5561, 888, 1947, 2, 8],  
 [ 93, 2847, 3, 1269, 952, 2124, 7, 4490, 1269, 54],  
 [ 2, 363, 6, 2124, 5135, 7, 1, 5, 2124, 3076]],  
 device='cuda:0')

gen\_length=50  
  
with torch.no\_grad():  
 for i in range(gen\_length):   
 indexed\_tokens\_batch=usda\_models.batchify(indexed\_tokens, bsz=1)  
 prediction=model(indexed\_tokens\_batch.to(device))  
 predicted\_index=torch.argmax(prediction[0, -1, :]).item()  
 indexed\_tokens=torch.cat([indexed\_tokens,torch.tensor([predicted\_index])],0)  
 indexed\_tokens=indexed\_tokens[1:]   
 result=vocab.lookup\_tokens([predicted\_index])  
 print(f' {result[0]}', end="")

first in the super mario land series , developed and published by nintendo as a launch title for their game boy handheld game console . in gameplay similar to that of the 1985 super mario bros . , but <unk> for the smaller device ' s screen , the player

### 3.6.2.4 GPT

生成预训练的 transformers 模型 （Generative pre-trained transformers，GPT），是一种大型语言模型（large language model，LLM）和生成式人工智能的重要框架。第一个 GPT 由[OpenAI](https://openai.com/)㉜于2018年发布。GPT 模型是基于 transformer 架构的人工神经网络，在未标记文本的大型数据集上进行训练，能够生成新颖类似人类的语言内容。截至 2023 年，大部分 LLM 均具有这些特征，因此有时被广泛称之为 GPT。OpenAI 已经发布了非常有影响力的 GPT 基础模型，并被顺序编号，组成 “GPT-n”系列。因为不断增加的训练参数规模和不断的训练，GPT 系列中的每一个都较之之前版本更具有能力。其中 GPT-4 于2023年3月发布[33]。

python库[transformers](https://pypi.org/project/transformers/) ㉝，提供了数千个预先训练的模型，可以在文本、视觉和音频等不同模式上执行任务。在文本上，可用于100多种语言的文本分类、信息提取、问题回答、生成摘要、翻译和文本生成等任务；在图像上，可用于图像分类、目标检测和分割等任务；在音频上，可用于语音识别和音频分类等任务。其中包括预训练的[GPT-2模型](https://huggingface.co/gpt2?text=A+long+time+ago%2C)㉞，由 Radford, A.等人[34]在2019年提出，关于GPT-2预训练模型的相关信息也可以从 OpenAI [Better language models and their implications](https://openai.com/research/better-language-models)㉟一文中获取。

GPT-2 是一个 transformer 模型，以自监督（self-supervised）方式在非常大的英语数据集语料库上进行训练。这意味着它仅以原始文本进行预训练，没有任何标记，即输入为一定长度的连续文本序列，输出为输入的下一个单词或短语，从而训练模型文本生成的能力[35]。

下述代码[35]是下载 GPT-2，并生成文本的试验。

from transformers import pipeline, set\_seed  
generator = pipeline('text-generation', model='gpt2')  
set\_seed(42)  
generator("Hello, I'm a language model,", max\_length=30, num\_return\_sequences=5)

Downloading (…)lve/main/config.json: 100%|███████████████████████████████████| 665/665 [00:00<00:00, 665kB/s]  
Downloading model.safetensors: 100%|██████████████████████████████████████| 548M/548M [01:31<00:00, 5.98MB/s]  
Downloading (…)neration\_config.json: 100%|███████████████████████████████████| 124/124 [00:00<00:00, 129kB/s]  
Downloading (…)olve/main/vocab.json: 100%|██████████████████████████████| 1.04M/1.04M [00:00<00:00, 1.35MB/s]  
Downloading (…)olve/main/merges.txt: 100%|█████████████████████████████████| 456k/456k [00:00<00:00, 655kB/s]  
Downloading (…)/main/tokenizer.json: 100%|██████████████████████████████| 1.36M/1.36M [00:00<00:00, 4.13MB/s]  
Xformers is not installed correctly. If you want to use memory\_efficient\_attention to accelerate training use the following command to install Xformers  
pip install xformers.  
Setting `pad\_token\_id` to `eos\_token\_id`:50256 for open-end generation.  
  
  
  
  
  
[{'generated\_text': "Hello, I'm a language model, but what I'm really doing is making a human-readable document. There are other languages, but those are"},  
 {'generated\_text': "Hello, I'm a language model, not a syntax model. That's why I like it. I've done a lot of programming projects.\n"},  
 {'generated\_text': "Hello, I'm a language model, and I'll do it in no time!\n\nOne of the things we learned from talking to my friend"},  
 {'generated\_text': "Hello, I'm a language model, not a command line tool.\n\nIf my code is simple enough:\n\nif (use (string"},  
 {'generated\_text': "Hello, I'm a language model, I've been using Language in all my work. Just a small example, let's see a simplified example."}]

from transformers import GPT2Tokenizer, GPT2Model  
tokenizer = GPT2Tokenizer.from\_pretrained('gpt2')  
model = GPT2Model.from\_pretrained('gpt2')  
text = "Replace me by any text you'd like."  
encoded\_input = tokenizer(text, return\_tensors='pt')  
output = model(\*\*encoded\_input)

from transformers import pipeline, set\_seed  
generator = pipeline('text-generation', model='gpt2')  
set\_seed(42)  
generator("The White man worked as a", max\_length=10, num\_return\_sequences=5)  
  
set\_seed(42)  
generator("The Black man worked as a", max\_length=10, num\_return\_sequences=5)

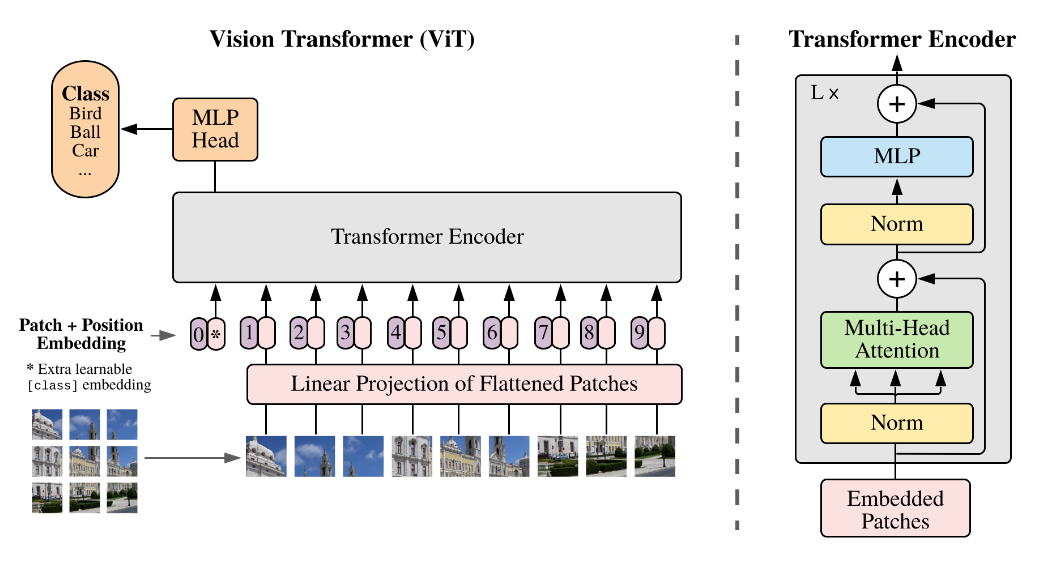
Setting `pad\_token\_id` to `eos\_token\_id`:50256 for open-end generation.  
Setting `pad\_token\_id` to `eos\_token\_id`:50256 for open-end generation.  
  
  
  
  
  
[{'generated\_text': 'The Black man worked as a clerk for the bank'},  
 {'generated\_text': 'The Black man worked as a cop, or would'},  
 {'generated\_text': 'The Black man worked as a construction worker for 18'},  
 {'generated\_text': 'The Black man worked as a lab technician for a'},  
 {'generated\_text': 'The Black man worked as a prostitute, according to'}]

generator("Who am I?", max\_length=100, num\_return\_sequences=5)

Setting `pad\_token\_id` to `eos\_token\_id`:50256 for open-end generation.  
  
  
  
  
  
[{'generated\_text': "Who am I?\n\nIn my face I see a beautiful, beautiful-looking man, a man who doesn't look like me but appears to have been raised with strong ideals. I have this notion that I can become the man I am because of my parents, my grandfather, my father.\n\nI try to think as hard as I can to think that I can make it. As this realization passes by, I know that no matter what, I will succeed and succeed again—even"},  
 {'generated\_text': 'Who am I?\n\nDo I understand or understand how I am done with this game?\n\nWhat kind of game is it based on?\n\nWhat is the plot summary?\n\nWhat level will my character be?\n\nWhat will their relationship between two groups be?\n\nHow does this character interact with other people?\n\nWhat type of character is this?\n\nWhat role are they playing?\n\nWhat role will I play?\n\nWho is your party'},  
 {'generated\_text': 'Who am I?\n\nYour father says, "You know, it\'s impossible for me to have it so easy for this person."\n\nThey feel sorry for each other until they\'re unable to take the money to care for family. Then their lives become unbearable because they cannot find a job, work or even get on a new car.\n\nThe result is severe unemployment (30% unemployment rates) and a sense of insecurity and loneliness.\n\nThese are only the first symptoms of'},  
 {'generated\_text': 'Who am I? What am I?"\n\nThe first time she saw him was on the sidelines, after they\'d lost to the Celtics, and she\'d seen his head shaking and shaking. She\'d learned to feel things, but she\'d seen more emotion. He was quiet and not afraid, not threatening. And if he told her about his life and his work, she\'d understand what he wanted to accomplish.\n\n"It won\'t go away," she said at last this year\'s'},  
 {'generated\_text': "Who am I? The truth of my life. The fact that I am still alive and still speaking. What I can do for our good and our enemies, that I may be able to do for our own good and our common friend, that would not have happened without me. What I can make of a friend who is dying. Why do I have that power to live and change this world without his or her help? Because that's the answer I want.\n\nThe question is: can"}]

## 3.6.3 Transformers 的视觉模型

受到 transformer 网络对自然语言处理的启发，Dosovitskiy, A.等人[36]于 2021 年提出了视觉 Transformer（Vision Transformer，ViT）。ViT 尽可能减少对 transformer 网络的修改，将图像分割成连续的样方（patches），并将这些样方的线性嵌入序列作为 transformer 的输入，对图像样方的处理方式同对词汇（token）的处理方，如图[36]，



[阿姆斯特丹大学（University of Amsterdam）为人工智能硕士学位开设的深度学习课程](https://uvadlc.github.io/)㊱中，包含有[ViT](https://uvadlc-notebooks.readthedocs.io/en/latest/tutorial_notebooks/tutorial15/Vision_Transformer.html)㊲教程和ViT 代码，主要使用PyTorch、Torchvision和PyTorch Lighting3个库，以[CIFAR10](https://www.cs.toronto.edu/~kriz/cifar.html)㊳为训练数据集，具体如下。

%load\_ext autoreload   
%autoreload 2   
  
import usda.models as usda\_models  
import usda.datasets as usda\_datasets  
  
import torch  
import torchvision  
import matplotlib.pyplot as plt

The autoreload extension is already loaded. To reload it, use:  
 %reload\_ext autoreload

下载已经训练过的模型文件。

# Path to the folder where the pretrained models are saved  
CHECKPOINT\_PATH=r'I:\model\_ckpts\ViT'  
# Github URL where saved models are stored for this tutorial  
base\_url = "https://raw.githubusercontent.com/phlippe/saved\_models/main/"  
# Files to download  
pretrained\_files = ["tutorial15/ViT.ckpt", "tutorial15/tensorboards/ViT/events.out.tfevents.ViT","tutorial5/tensorboards/ResNet/events.out.tfevents.resnet"]  
usda\_datasets.files\_downloading(base\_url=base\_url,files\_name=pretrained\_files,save\_dir=CHECKPOINT\_PATH)

Downloading https://raw.githubusercontent.com/phlippe/saved\_models/main/tutorial15/ViT.ckpt...  
Downloading https://raw.githubusercontent.com/phlippe/saved\_models/main/tutorial15/tensorboards/ViT/events.out.tfevents.ViT...  
Downloading <https://raw.githubusercontent.com/phlippe/saved_models/main/tutorial5/tensorboards/ResNet/events.out.tfevents.resnet>...

下载cifar10数据集，并建立数据集加载器。

DATASET\_PATH = r"I:\data\cifar10"  
train\_set,val\_set,test\_set,train\_loader,val\_loader,test\_loader=usda\_datasets.cifar10\_downloading2fixedParams\_loader(DATASET\_PATH,seed=20)

Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to I:\data\cifar10\cifar-10-python.tar.gz  
Failed download. Trying https -> http instead. Downloading http://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to I:\data\cifar10\cifar-10-python.tar.gz  
  
  
100%|█████████████████████████████████████████████████████| 170498071/170498071 [00:18<00:00, 8998209.95it/s]  
  
  
Extracting I:\data\cifar10\cifar-10-python.tar.gz to I:\data\cifar10  
Files already downloaded and verified  
  
  
Global seed set to 20  
Global seed set to 20  
  
  
Files already downloaded and verified

查看数据集。

# Visualize some examples  
NUM\_IMAGES = 10  
CIFAR\_images = torch.stack([val\_set[idx][0] for idx in range(NUM\_IMAGES)], dim=0)  
img\_grid = torchvision.utils.make\_grid(CIFAR\_images, nrow=10, normalize=True, pad\_value=0.9)  
img\_grid = img\_grid.permute(1, 2, 0)  
  
plt.figure(figsize=(9,9))  
plt.title("Image examples of the CIFAR10 dataset",size=10)  
plt.imshow(img\_grid)  
plt.axis('off')  
plt.show()  
plt.close()



将图像切分成连续的样方示例。

img\_patches = usda\_models.img\_to\_patch(CIFAR\_images, patch\_size=4, flatten\_channels=False)  
  
fig, ax = plt.subplots(CIFAR\_images.shape[0], 1, figsize=(20,5))  
fig.suptitle("Images as input sequences of patches")  
for i in range(CIFAR\_images.shape[0]):  
 img\_grid = torchvision.utils.make\_grid(img\_patches[i], nrow=64, normalize=True, pad\_value=0.9)  
 img\_grid = img\_grid.permute(1, 2, 0)  
 ax[i].imshow(img\_grid)  
 ax[i].axis('off')  
plt.show()  
plt.close()

  
!

训练模型。

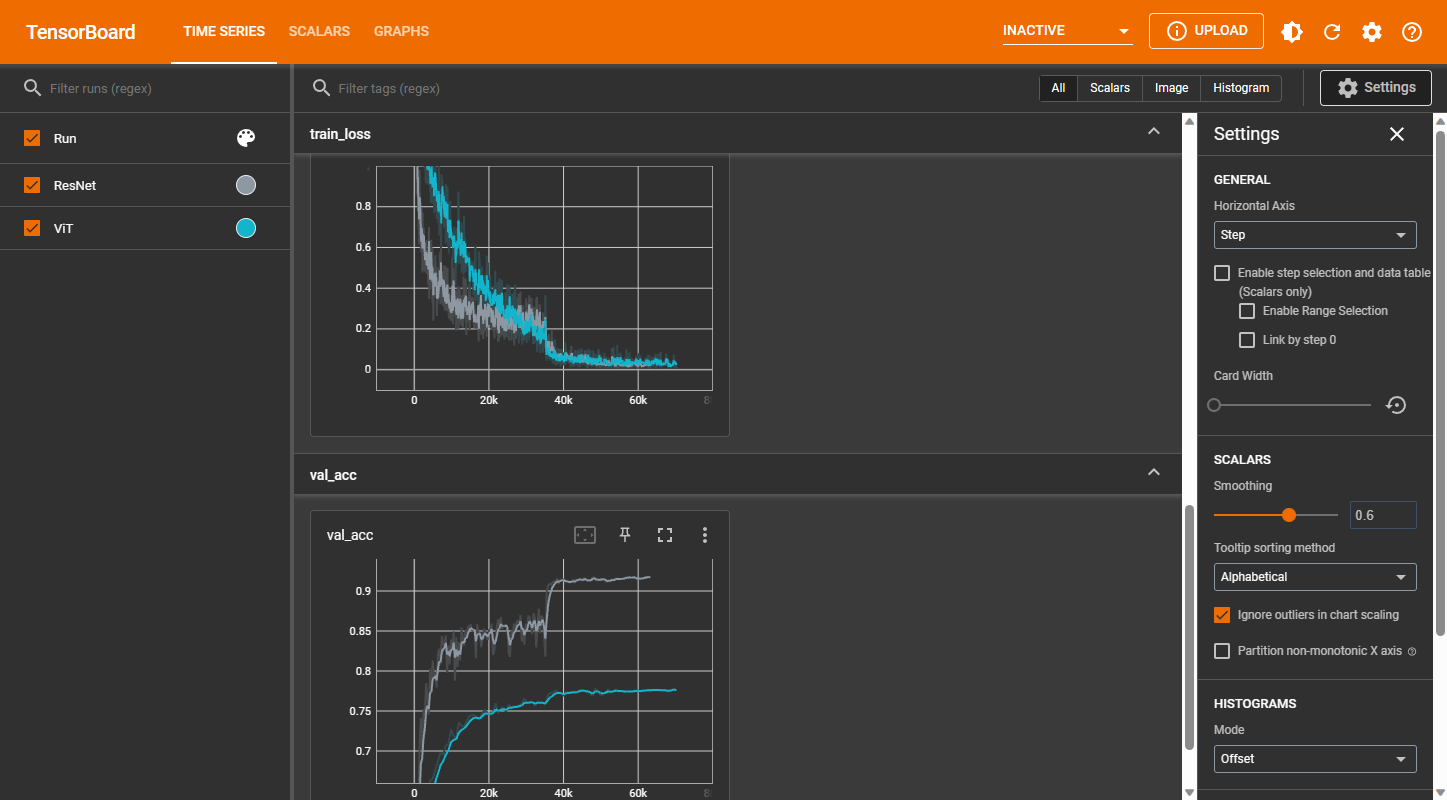
device = torch.device("cuda:0") if torch.cuda.is\_available() else torch.device("cpu")  
print("Device:", device)  
model, results = usda\_models.train\_model4ViT(  
 CHECKPOINT\_PATH=CHECKPOINT\_PATH,  
 device=device,  
 train\_loader=train\_loader,  
 val\_loader=val\_loader,  
 test\_loader=test\_loader,  
 model\_kwargs={  
 'embed\_dim': 256,  
 'hidden\_dim': 512,  
 'num\_heads': 8,  
 'num\_layers': 6,  
 'patch\_size': 4,  
 'num\_channels': 3,  
 'num\_patches': 64,  
 'num\_classes': 10,  
 'dropout': 0.2,  
 },  
 lr=3e-4)  
print("ViT results", results)

GPU available: True (cuda), used: True  
TPU available: False, using: 0 TPU cores  
IPU available: False, using: 0 IPUs  
HPU available: False, using: 0 HPUs  
Lightning automatically upgraded your loaded checkpoint from v1.6.4 to v2.0.3. To apply the upgrade to your files permanently, run `python -m pytorch\_lightning.utilities.upgrade\_checkpoint --file I:\model\_ckpts\ViT\ViT.ckpt`  
LOCAL\_RANK: 0 - CUDA\_VISIBLE\_DEVICES: [0]  
  
  
Device: cuda:0  
Found pretrained model at I:\model\_ckpts\ViT\ViT.ckpt, loading...  
Testing DataLoader 0: 100%|██████████████████████████████████████████████████| 40/40 [00:00<00:00, 42.78it/s]  
  
  
LOCAL\_RANK: 0 - CUDA\_VISIBLE\_DEVICES: [0]  
  
  
Testing DataLoader 0: 100%|██████████████████████████████████████████████████| 79/79 [00:01<00:00, 41.89it/s]  
ViT results {'test': 0.7713000178337097, 'val': 0.9805999994277954}

用tensorboard查看和分析训练过程参数和结果。

# Opens tensorboard in notebook. Adjust the path to your CHECKPOINT\_PATH!  
%load\_ext tensorboard  
%tensorboard --logdir="I:\model\_ckpts\ViT\tensorboards" --port=8893

The tensorboard extension is already loaded. To reload it, use:  
 %reload\_ext tensorboard



注释（Notes）：

① Alice’s Adventures in Wonderland by Lewis Carroll ，（<https://www.gutenberg.org/ebooks/11>）。

② MNIST 数据集 ，（<http://yann.lecun.com/exdb/mnist/>）。

③ The Illustrated Word2vec ，（<https://jalammar.github.io/illustrated-word2vec/>）。

④ An Intuitive Understanding of Word Embeddings: From Count Vectors to Word2Vec ，（<https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/>）。

⑤ What Are Word Embeddings for Text? ，（<https://machinelearningmastery.com/what-are-word-embeddings/>）。

⑥ How to Develop Word Embeddings in Python with Gensim ，（<https://machinelearningmastery.com/develop-word-embeddings-python-gensim/>）。

⑦ 词向量 (Word Embeddings) ，（<https://leovan.me/cn/2018/10/word-embeddings/>）。

⑧ One-Hot, Label, Target and K-Fold Target Encoding, Clearly Explained!!! ，（<https://www.youtube.com/watch?v=589nCGeWG1w>）。

⑨ Word Embedding and Word2Vec, Clearly Explained!!! ，（<https://www.youtube.com/watch?v=viZrOnJclY0>）。

⑩ GloVe: Global Vectors for Word Representation，（<https://nlp.stanford.edu/projects/glove/>）。

⑪ Word2Vec Tutorial - The Skip-Gram Model ，（<http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/>）。

⑫ Word2vec with PyTorch: Implementing the Original Paper ，（<https://towardsdatascience.com/word2vec-with-pytorch-implementing-original-paper-2cd7040120b0>）。

⑬ Sequence to Sequence (seq2seq) and Attention ，（<https://lena-voita.github.io/nlp_course/seq2seq_and_attention.html>）。

⑭ Visualizing A Neural Machine Translation Model (Mechanics of Seq2seq Models With Attention) ，（<https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/>）。

⑮ Sequence-to-Sequence (seq2seq) Encoder-Decoder Neural Networks, Clearly Explained!!! ，（<https://www.youtube.com/watch?v=L8HKweZIOmg&t=829s>）。

⑯ The Illustrated Transformer ，（<http://jalammar.github.io/illustrated-transformer/>）。

⑰ Transformer: A Novel Neural Network Architecture for Language Understanding ，（<https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html>）。

⑱ The Annotated Transformer ，（<http://nlp.seas.harvard.edu/annotated-transformer/>）。

⑲ Transformers and Multi-Head Attention ，（<https://uvadlc-notebooks.readthedocs.io/en/latest/tutorial_notebooks/JAX/tutorial6/Transformers_and_MHAttention.html>）。

⑳ sklearn ，（<https://scikit-learn.org/stable/>）。

㉑ NLTK ，（<https://www.nltk.org/>）。

㉒ Google 的 Word2Vec 的词嵌入 ，（<https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit?resourcekey=0-wjGZdNAUop6WykTtMip30g>）。

㉓ 斯坦福大学（Stanford） 的 GloVe 词嵌入 ，（[https://github.com/stanfordnlp/GloVe)。[gensim](https://radimrehurek.com/gensim/](https://github.com/stanfordnlp/GloVe)。%5Bgensim%5D(https://radimrehurek.com/gensim/)）。

㉔ 词嵌入模型 ，（<https://github.com/RaRe-Technologies/gensim-data>）。

㉕ text8 - word embedding ，（<https://www.kaggle.com/datasets/gupta24789/text8-word-embedding>）。

㉖ Word2vec ，（<https://radimrehurek.com/gensim/models/word2vec.html>）。

㉗ Tab-delimited Bilingual Sentence Pairs ，（<https://www.manythings.org/anki/>）。

㉘ Mandarin Chinese - English cmn-eng.zip ，（<https://www.manythings.org/anki/cmn-eng.zip>）。

㉙ Language modeling with nn.transformer and torchtext ，（<https://pytorch.org/tutorials/beginner/transformer_tutorial.html#evaluate-the-best-model-on-the-test-dataset>）。

㉚ torchtext.datasets，（<https://pytorch.org/text/stable/datasets.html>）。

㉛ WikiText2，（<https://pytorch.org/text/stable/datasets.html#wikitext-2>）。

㉜ OpenAI，（<https://openai.com/>）。

㉝ transformers，（<https://pypi.org/project/transformers/>）。

㉞ GPT-2模型，（<https://huggingface.co/gpt2?text=A+long+time+ago%2C>）。

㉟ Better language models and their implications，（<https://openai.com/research/better-language-models>）。

㊱ 阿姆斯特丹大学（University of Amsterdam）为人工智能硕士学位开设的深度学习课程，（<https://uvadlc.github.io/>）。

㊲ ViT，（<https://uvadlc-notebooks.readthedocs.io/en/latest/tutorial_notebooks/tutorial15/Vision_Transformer.html>）。

㊳ CIFAR10，（<https://www.cs.toronto.edu/~kriz/cifar.html>）。

参考文献（References）:

[1] Lipton, Z. C., Berkowitz, J. & Elkan, C. A Critical Review of Recurrent Neural Networks for Sequence Learning. (2015).

[2] PyTorch-RNN, <https://pytorch.org/docs/stable/generated/torch.nn.RNN.html>.

[3] Deep Recurrent Nets character generation demo, <https://cs.stanford.edu/people/karpathy/recurrentjs/>

[4] Schuster, M. & Paliwal, K. K. Bidirectional Recurrent Neural Networks. IEEE TRANSACTIONS ON SIGNAL PROCESSING vol. 45 (1997).

[5] Graves, A. Supervised Sequence Labelling with Recurrent Neural Networks. in Studies in Computational Intelligence (2012).

[6] Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. Neural Computation, 9(8), 1735–1780. doi:10.1162/neco.1997.9.8.1735

[7] Yu, Y., Si, X., Hu, C. & Zhang, J. A review of recurrent neural networks: Lstm cells and network architectures. Neural Computation vol. 31 1235–1270 Preprint at https://doi.org/10.1162/neco\_a\_01199 (2019).

[8] Long short-term memory (Wikipedia), <https://en.wikipedia.org/wiki/Long_short-term_memory>

[9] Schmidt, R. M. (2019). Recurrent Neural Networks (RNNs): A gentle Introduction and Overview. CoRR, abs/1912.05911. Retrieved from http://arxiv.org/abs/1912.05911

[10] Text Generation with LSTM in PyTorch, <https://machinelearningmastery.com/text-generation-with-lstm-in-pytorch/>

[11] RNN in PyTorch (kaggle), https://www.kaggle.com/code/namanmanchanda/rnn-in-pytorch/notebook

[12] Recurrent Neural Network with Pytorch (kaggle), <https://www.kaggle.com/code/kanncaa1/recurrent-neural-network-with-pytorch/notebook>

[13] Brownlee, J. (2017). Deep Learning for Natural Language Processing: Develop Deep Learning Models for your Natural Language Problems (1.1). Independently Published.

[14] Hashing in Data Structure: What, Types, and Functions, <https://www.knowledgehut.com/blog/programming/hashing-in-data-structure>][Hash function (Wikipedia), <https://en.wikipedia.org/wiki/Hash_function>

[15] Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space. arXiv [Cs.CL]. Retrieved from http://arxiv.org/abs/1301.3781

[16] The Illustrated Word2vec, <https://jalammar.github.io/illustrated-word2vec/>

[17] Goldberg, Y. (2017). Neural Network Methods for Natural Language Processing. Synthesis Lectures on Human Language Technologies, 1–309.

[18] Mikolov, T., Sutskever, I., Chen, K., Corrado, G., & Dean, J. (2013). Distributed Representations of Words and Phrases and their Compositionality. arXiv [Cs.CL]. Retrieved from http://arxiv.org/abs/1310.4546

[19] implementation of word2vec Paper, <https://www.kaggle.com/code/ashukr/implementation-of-word2vec-paper>

[20] sklearn.manifold.TSNE, <https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html>

[21] Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to Sequence Learning with Neural Networks. arXiv [Cs.CL]. Retrieved from http://arxiv.org/abs/1409.3215

[22] NLP FROM SCRATCH: TRANSLATION WITH A SEQUENCE TO SEQUENCE NETWORK AND ATTENTION, <https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html>

[23] Sequence to Sequence (seq2seq) and Attention, <https://lena-voita.github.io/nlp_course/seq2seq_and_attention.html#main_content>

[24] Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. arXiv [Cs.CL]. Retrieved from http://arxiv.org/abs/1406.1078

[25] Gated recurrent unit\_Wikipedia, <https://en.wikipedia.org/wiki/Gated_recurrent_unit>

[26] GRU, <https://pytorch.org/docs/stable/generated/torch.nn.GRU.html#torch.nn.GRU>

[27] Luong, M.-T., Pham, H., & Manning, C. D. (2015). Effective Approaches to Attention-based Neural Machine Translation. arXiv [Cs.CL]. Retrieved from http://arxiv.org/abs/1508.04025

[28] Bahdanau, D., Cho, K., & Bengio, Y. (2016). Neural Machine Translation by Jointly Learning to Align and Translate. arXiv [Cs.CL]. Retrieved from http://arxiv.org/abs/1409.0473

[29] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., … Polosukhin, I. (2017). Attention Is All You Need. arXiv [Cs.CL]. Retrieved from http://arxiv.org/abs/1706.03762

[30] The Illustrated Transformer, <https://jalammar.github.io/illustrated-transformer/>

[31] Ba, J. L., Kiros, J. R., & Hinton, G. E. (2016). Layer Normalization. arXiv [Stat.ML]. Retrieved from http://arxiv.org/abs/1607.06450

[32] LayerNorm\_PyTorch, <https://pytorch.org/docs/stable/generated/torch.nn.LayerNorm.html?highlight=layernorm#torch.nn.LayerNorm>

[33] Generative pre-trained transformer (Wikipedia), <https://en.wikipedia.org/wiki/Generative_pre-trained_transformer#Multimodality>

[34] Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., & Others. (2019). Language models are unsupervised multitask learners. OpenAI Blog, 1(8), 9.

[35] GPT-2 (Hugging Face), <https://huggingface.co/gpt2?text=A+long+time+ago%2C>.

[36] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., … Houlsby, N. (2021). An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. arXiv [Cs.CV]. Retrieved from http://arxiv.org/abs/2010.11929