**Assignment 5**

CBOW : usupervised learning, for NLPs – word embedding & text classification

It works by predicting target word given its surrounding words

* Matplotlib : python plotting library
* Seaborn : library bulid top of matplot , for more informative and concise way to make statistical plots
* Pylab : combines functions of numpy and matplot
* Numpy : for scientific calculations
* Re : provides regular expression operations in Python. Regular expressions are a powerful tool for searching, editing, and manipulating text.
* “”” “”” : allows to write multi line string
* % matplotlib inline : called ad magic command

tells the Jupyter notebook to display Matplotlib plots directly in the notebook cells Otherwise, the plots will be displayed in separate windows.

* Re.sub(): sub is for substitute
* sentences = re.sub('[^A-Za-z0-9]+', ' ', sentences) : ^ mathches beginning of string

$ matches end of string , + matches one or more occurrences of the preceding character class

* above re matches any sequence of one or more characters that are not letters, numbers, or spaces.
* re.sub(r'(?:^| )\w(?:$| )', ' ', sentences).strip() : removes all 1-letter words from the string `sentences` and strips the leading and trailing whitespace from the string.
* vocab\_size: The vocabulary size is the number of unique words in the vocabulary. It is an important parameter to consider when training a word embedding model, as it determines the size of the embedding matrix.
* embed\_dim: The embedding dimension is the dimensionality of the word embedding vectors. It determines the complexity of the word embedding model and the amount of information that can be captured in the embedding vectors.
* context\_size: The context size is the number of words to consider on either side of the target word when generating the word embedding vector. A larger context size will allow the model to learn more complex word relationships, but it will also require more training data and computational resources.
* Embedding model :
* An embedding model is a machine learning model that learns to represent words or other entities as vectors of numbers. These vectors, called embeddings, capture the semantic and syntactic relationships between words and entities. Embedding models are used in a variety of natural language processing (NLP) tasks, such as machine translation, text summarization, and question answering. Embedding models are typically trained using a technique called word2vec (word to vector).
* word\_to\_ix: This dictionary maps words to their corresponding indices in the vocabulary.
* ix\_to\_word: This dictionary maps indices to their corresponding words in the vocabulary.
* Enumerate():used to iterate over a sequence and return a tuple containing the index and the value of each item in the sequence.
* The for i in range(2, len(words) - 2) loop iterates over the words list, starting at index 2 and ending at index len(words) - 2. This ensures that the context and target are always valid.
* The context = [words[i - 2], words[i - 1], words[i + 1], words[i + 2]] line creates a context list for the current word. The context list contains the four words that precede and follow the current word.
* The target = words[i] line sets the target to the current word.
* The data.append((context, target)) line appends the tuple (context, target) to the data list.
* embeddings = np.random.random\_sample((vocab\_size, embed\_dim)) : creates a NumPy array called embeddings that contains random embedding vectors for each word in the vocabulary.function generates random floating-point numbers in the half-open interval [0.0, 1.0)
* def linear(m, theta): w = theta return m.dot(w) : dot returns dot produnct of 2,

m = A NumPy array representing the input features.

theta: A NumPy array representing the model parameters.

* NLLLoss: negative log likehood loss function , returns the mean value, loss function that measures how well a model's predictions match the ground truth labels.
* Logs: logits
* Log\_softmax : return exponential of all logits, activation function that converts logits to probabilities.
* The out = np.zeros\_like(logits) line creates a NumPy array of zeros with the same shape as the logits array.
* The out[np.arange(len(logits)),target] = 1 line sets the element at index (i, target) of the out array to 1 for each sample i. This creates a one-hot encoding of the target labels.
* The softmax = np.exp(logits) / np.exp(logits).sum(axis=-1,keepdims=True) line calculates the softmax of the logits. The keepdims=True argument ensures that the output of the sum() function has the same shape as the input array.
* The return (- out + softmax) / logits.shape[0] line calculates the log softmax cross-entropy loss and returns it. The logits.shape[0] term is used to normalize the loss.
* def forward(context\_idxs, theta): used to predict the target word for each sample
* backward pass function : to train the word embedding model using a gradient descent optimizer. The optimizer will use the gradient to update the model parameters in order to reduce the loss function.
* Grad : gradient of the loss function with respect to the model parameters.
* lr: The learning rate.
* theta = np.random.uniform(-1, 1, (2 \* context\_size \* embed\_dim, vocab\_size)) :

initializes the embedding matrix with random values between -1 and 1. To ensures embedding vectors are not all zero and that they are evenly distributed in the embedding space. The embedding matrix is also initialized with a size of (2 \*---). This is because the embedding vectors for the context words are concatenated together before being used to calculate the pre-softmax logits.

* Epoch : one complete pass through training data
* Cross entropy: measure of the difference between two probability distributions. It is often used in machine learning as a loss function to measure how well a model's predictions match the ground truth labels.
* Gradient descent algo : finds minimum of function, Gradient descent works by iteratively moving in the direction of the negative gradient of the function. The gradient of a function is a vector that points in the direction of the steepest ascent of the function
* Optimizer : updates the parameters of a model to minimize or maximize a specific objective function, allows the model to learn from the data and improve its performance over time.
* Ex: GD, SGD, mini batch GD

**ASSIGNMNET 6**

Transfer learning : ML technique where a model developed for one task is reused as the starting point for a model on a second task, avoids computationally expensive & time consuming model training

Adv : reduce risk of overfitting, improved performance, training time and data Is reduced

* VGG16 model : model of 16 layers

-13 conv layer, 3 fully connected layers, and a op layer

-uses small filter of 3\*3 , stride of 1 and padding of 1

-maxpooling with kernel size 2\*2 and stride =2

-activation function = relu

Uses ImageNet dataset -> for visual object recognition,contain over 14 million handlabelled images

Adv :availability , efficiency, high accuracy

Dis : large size, prone to overfit, computational cost

Applications : object detection, facial – scene recognition, medical image analysis

* Stride : controls how far the filter moves across the input image when performing a convolution operation. A stride of 1 means that the filter moves one pixel at a time
* Padding : controls how many pixels are added to the borders of the input image before the convolution operation is performed. Padding can be used to keep the output image the same size as the input image, or to control the amount of information that is lost at the borders of the image.
* Kernel : to extract features, small matrix of weights
* decode\_predictions() : function converts the probabilities to class labels
* image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))

1 : batch size set to one (number of images that are processed by the model)

Next three params are height, width and channels for image(like RGB)

* (label[1], label[2]\*100)) = The label[2] value is the probability of the predicted class. This value is typically between 0 and 1. Multiplying the probability by 100 converts it to a percentage. This makes the probability easier to understand and interpret.