Folha de Dicas: Modelos de IA para PLN

Pacote/Método	Descrição	Exemplo de código

PyTorch/Embedding e EmbeddingBag	Embedding é uma classe que representa uma camada de incorporação. Aceita índices de tokens e produz vetores de incorporação. EmbeddingBag é uma classe que agrega incorporações usando operações de média ou soma. Embedding e EmbeddingBag fazem parte do módulo torch.nn. O exemplo de código mostra como você pode usar Embedding e EmbeddingBag no PyTorch.	`# Definindo um conjunto de dados`
		`dataset = [`
		`"Gosto de gatos",`
		`"Odeio cães",`
		`"Sou imparcial em relação a hipopótamos"`
		`]`
		`#Inicializando o tokenizador, iterador do conjunto de dados e vocabulário`
		`tokenizer = get_tokenizer('spacy', language='en_core_web_sm')`
		`def yield_tokens(data_iter):`
		`for data_sample in data_iter:`
		`yield tokenizer(data_sample)`
		`data_iter = iter(dataset)`
		`vocab = build_vocab_from_iterator(yield_tokens(data_iter))`
		`#Tokenizando e gerando índices`
		`input_ids=lambda x:[torch.tensor(vocab(tokenizer(data_sample))) for data_sample in dataset]`
		`index=input_ids(dataset)`
		`print(index)`
		`#Iniciando a camada de incorporação, especificando o tamanho da dimensão para as incorporações,`
		`#determinando a contagem de tokens únicos presentes no vocabulário e criando a camada de incorporação`
		`embedding_dim = 3`
		`n_embedding = len(vocab)`
		`n_embedding:9`
		`embeds = nn.Embedding(n_embedding, embedding_dim)`
		`#Aplicando o objeto de incorporação`
		`i_like_cats=embeds(index[0])`
		`i_like_cats`
		`impartial_to_hippos=embeds(index[-1])`
		`impartial_to_hippos`
		`#Inicializando a camada de bolsa de incorporações`
		`embedding_dim = 3`
		`n_embedding = len(vocab)`
		`n_embedding:9`
		`embedding_bag = nn.EmbeddingBag(n_embedding, embedding_dim)`
		
		Saída da bolsa de embeddings
		
		`dataset = [“Eu gosto de gatos”,”Eu odeio cães”,”Sou imparcial em relação a hipopótamos”]`
		`index:[tensor([0, 7, 2]), tensor([0, 4, 3]), tensor([0, 1, 6, 8, 5])]`
		`i_like_cats=embedding_bag(index[0],offsets=torch.tensor([0]))`
		`i_like_cats`

Função de lote	Define o número de amostras que serão propagadas através da rede.	`def collate_batch(batch):`
		`target_list, context_list, offsets = [], [], [0]`
		`for _context, _target in batch:`
		`target_list.append(vocab[_target])`
		`processed_context = torch.tensor(text_pipeline(_context), dtype=torch.int64)`
		`context_list.append(processed_context)`
		`offsets.append(processed_context.size(0))`
		`target_list = torch.tensor(target_list, dtype=torch.int64)`
		`offsets = torch.tensor(offsets[:-1]).cumsum(dim=0)`

Pacote/Método	Descrição	Exemplo de código

		context_list = torch.cat(context_list)
		return target_list.to(device), context_list.to(device), offsets.to(device)
		TAMANHO_DO_LOTE = 64 # tamanho do lote para treinamento
		dataloader_cbow = DataLoader(cobw_data, batch_size=TAMANHO_DO_LOTE, shuffle=True, collate_fn=collate_batch)

Passagem para frente	Refere-se ao cálculo e armazenamento de variáveis intermediárias (incluindo saídas) para uma rede neural na ordem da camada de entrada até a camada de saída.	def forward(self, text):
GloVe pré-treinado de Stanford	Utiliza dados em larga escala para embeddings de palavras. Pode ser integrado ao PyTorch para melhorar tarefas de PLN, como classificação.	from torchtext.vocab import GloVe,vocab **Criando uma instância da versão 6B do modelo GloVe()** glove_vectors_6B = GloVe(name =’6B’) # você pode especificar o modelo com o seguinte formato: GloVe(name=’840B’, dim=300) **Construir vocabulário a partir de glove_vectors** vocab = vocab(glove_vectors_6B.stoi, 0,specials=(‘<unk>‘, ‘<pad>‘)) vocab.set_default_index(vocab[“<unk>“])
vocab	O objeto vocab é parte da biblioteca torchtext do PyTorch. Ele mapeia tokens para índices. O exemplo de código mostra	# Recebe um iterador como entrada e extrai a próxima frase tokenizada. Cria uma lista de índices de tokens usando o dicionário vocab para cada token. def get_tokenized_sentence_and_indices(iterator): tokenized_sentence = next(iterator) token_indices = [vocab[token] for token in tokenized_sentence] return tokenized_sentence, token_indices **Retorna as sentenças tokenizadas e os índices de token correspondentes. Repete**

Pacote/Método	Descrição	Exemplo de código

	como você pode aplicar o objeto vocab diretamente aos tokens.	`tokenized_sentence, token_indices = get_tokenized_sentence_and_indices(my_iterator)`
		`next(my_iterator)`
		Imprime a frase tokenizada e seus índices de token correspondentes.
		`print("Frase Tokenizada:", tokenized_sentence)`
		`print("Índices de Token:", token_indices)`

Tokens especiais no PyTorch: <eos> e <bos>	Tokens introduzidos em sequências de entrada para transmitir informações específicas ou servir a um propósito particular durante o treinamento. O exemplo de código mostra o uso de <bos> e <eos> durante a tokenização. O token <bos> denota o início da sequência de entrada, e o token <eos> denota o fim.	`# Adiciona <bos> no início e <eos> no final das frases tokenizadas`
		usando um loop que itera sobre as sentenças nos dados de entrada
		`tokenizer_en = get_tokenizer('spacy', language='en_core_web_sm')`
		`tokens = []`
		`max_length = 0`
		`for line in lines:`
		`  tokenized_line = tokenizer_en(line)`
		`  tokenized_line = ['<bos>'] + tokenized_line + ['<eos>']`
		`  tokens.append(tokenized_line)`
		`  max_length = max(max_length, len(tokenized_line))`

Tokens especiais no PyTorch: <pad>	Tokens introduzidos nas sequências de entrada para transmitir informações específicas ou servir a um propósito particular durante o treinamento. O exemplo de código mostra o uso do token <pad> para garantir que	`# Preenche as linhas tokenizadas`
		`for i in range(len(tokens)):`
		`  tokens[i] = tokens[i] + ['<pad>'] * (max_length - len(tokens[i]))`

Pacote/Método	Descrição	Exemplo de código
	todas as sentenças tenham o mesmo comprimento.	

Perda de entropia cruzada	Uma métrica usada em aprendizado de máquina (ML) para avaliar o desempenho de um modelo de classificação. A perda é medida como o valor de probabilidade entre 0 (modelo perfeito) e 1. Normalmente, o objetivo é aproximar o modelo o máximo possível de 0.	`from torch.nn import CrossEntropyLoss`
		`model = TextClassificationModel(vocab_size,emsize,num_class)`
		`loss_fn = CrossEntropyLoss()`
		`predicted_label = model(text, offsets)`
		`loss = criterion(predicted_label, label)`

Otimização	Método para reduzir perdas em um modelo.	`# Cria um objeto iterador`
		`optimizer = torch.optim.SGD(model.parameters(), lr=0.1)`
		`scheduler = torch.optim.lr_scheduler.StepLR(optimizer, 1.0, gamma=0.1)`
		`optimizer.zero_grad()`
		`predicted_label = model(text, offsets)`
		`loss = criterion(predicted_label, label)`
		`loss.backward()`
		`torch.nn.utils.clip_grad_norm_(model.parameters(), 0.1)`
		`optimizer.step()`

sentence_bleu()	NLTK (ou Natural Language Toolkit) fornece esta função para avaliar uma sentença hipótese em relação a uma ou mais sentenças de referência. As sentenças de referência	`from nltk.translate.bleu_score import sentence_bleu`
		`def calculate_bleu_score(generated_translation, reference_translations):`
		
		`# Converter as traduções geradas e as traduções de referência no formato esperado`
		
		`referencias = [referencia.split() for referencia in traducoes_referencia]`
		`hipotese = traducao_gerada.split()`
		
		`# Calcular a pontuação BLEU`
		
		`bleu_score = sentence_bleu(references, hypothesis)`
		`return bleu_score`

Pacote/Método	Descrição	Exemplo de código
	devem ser apresentadas como uma lista de sentenças onde cada referência é uma lista de tokens.	`reference_translations = [“Homem asiático varrendo a calçada.”, “Um homem asiático varrendo a calçada.”, “Um homem asiático varre a calçada.”, “Um homem asiát` `bleu_score = calculate_bleu_score(generated_translation, reference_translations)`

Modelo RNN Encoder	O modelo seq2seq encoder-decoder trabalha em conjunto para transformar uma sequência de entrada em uma sequência de saída. O encoder é uma série de RNNs que processam a sequência de entrada individualmente, passando seus estados ocultos para sua próxima RNN.	`class Encoder(nn.Module):`
		`def __init__(self, vocab_len, emb_dim, hid_dim, n_layers, dropout_prob):`
		`super().__init__()`
		`self.hid_dim = hid_dim`
		`self.n_layers = n_layers`
		`self.embedding = nn.Embedding(vocab_len, emb_dim)`
		`self.lstm = nn.LSTM(emb_dim, hid_dim, n_layers, dropout = dropout_prob)`
		`self.dropout = nn.Dropout(dropout_prob)`
		`def forward(self, input_batch):`
		`embed = self.dropout(self.embedding(input_batch))`
		`embed = embed.to(device)`
		`outputs, (hidden, cell) = self.lstm(embed)`
		`return hidden, cell`

Modelo RNN Decoder	O modelo seq2seq encoder-decoder trabalha em conjunto para transformar uma sequência de entrada em uma sequência de saída. O módulo decoder é uma série de RNNs que gera a tradução de forma autoregressiva, um token de cada vez. Cada token gerado retorna para a próxima RNN junto com o estado oculto para gerar o próximo token	`class Decoder(nn.Module):`
		`def __init__(self, output_dim, emb_dim, hid_dim, n_layers, dropout):`
		`super().__init__()`
		`self.output_dim = output_dim`
		`self.hid_dim = hid_dim`
		`self.n_layers = n_layers`
		`self.embedding = nn.Embedding(output_dim, emb_dim)`
		`self.lstm = nn.LSTM(emb_dim, hid_dim, n_layers, dropout = dropout)`
		`self.fc_out = nn.Linear(hid_dim, output_dim)`
		`self.softmax = nn.LogSoftmax(dim=1)`
		`self.dropout = nn.Dropout(dropout)`
		`def forward(self, input, hidden, cell):`
		`input = input.unsqueeze(0)`
		`embedded = self.dropout(self.embedding(input))`
		`output, (hidden, cell) = self.lstm(embedded, (hidden, cell))`
		`prediction_logit = self.fc_out(output.squeeze(0))`
		`prediction = self.softmax(prediction_logit)`
		`return prediction, hidden, cell`

Pacote/Método	Descrição	Exemplo de código
	da sequência de saída até que o token final seja gerado.	

Modelo Skip-gram	Prediz palavras de contexto ao redor de uma palavra-alvo específica. Ele prediz uma palavra de contexto por vez a partir de uma palavra-alvo.	class SkipGram_Model(nn.Module):
		def __init__(self, vocab_size, embed_dim):
		super(SkipGram_Model, self).__init__()
		
		## Defina a camada de embeddings
		
		self.embeddings = nn.Embedding(num_embeddings=vocab_size, embedding_dim=embed_dim)
		
		## Defina a camada totalmente conectada
		
		self.fc = nn.Linear(in_features=embed_dim, out_features=vocab_size)
		
		## Realizar a passagem para frente
		
		def forward(self, text):
		
		## Passe o texto de entrada pela camada de embeddings
		
		out = self.embeddings(text)
		
		## Passe a saída da camada de embeddings pela camada totalmente conectada
		
		## Aplique a função de ativação ReLU
		
		out = torch.relu(out)
		out = self.fc(out)
		return out
		model_sg = SkipGram_Model(vocab_size, emsize).to(device)
		
		## Função de geração de sequência
		
		CONTEXT_SIZE = 2
		skip_data = []
		for i in range(CONTEXT_SIZE, len(tokenized_toy_data) - CONTEXT_SIZE):
		context = (
		[tokenized_toy_data[i - j - 1] for j in range(CONTEXT_SIZE)] # Palavras precedentes
		+ [tokenized_toy_data[i + j + 1] for j in range(CONTEXT_SIZE)] # Palavras subsequentes)
		target = tokenized_toy_data[i]
		skip_data.append((target, context))
		skip_data=[(‘i’, [‘wish’, ‘i’, ‘was’, ‘little’]), (‘was’, [‘i’, ‘wish’, ‘little’, ‘bit’])],..

Pacote/Método	Descrição	Exemplo de código

collate_fn	Processa a lista de amostras para formar um lote. O argumento `batch` é uma lista de todas as suas amostras.	`def collate_fn(batch):`
		`  target_list, context_list = [], []`
		`  for _context, _target in batch:`
		`  target_list.append(vocab[_target])`
		`  context_list.append(vocab[_context])`
		`  target_list = torch.tensor(target_list, dtype=torch.int64)`
		`  context_list = torch.tensor(context_list, dtype=torch.int64)`
		`  return target_list.to(device), context_list.to(device)`

Função de treinamento	Treina o modelo por um número especificado de épocas. Também inclui uma condição para verificar se a entrada é para skip-gram ou CBOW. A saída desta função inclui o modelo treinado e uma lista de perdas médias para cada época.	`def train_model(model, dataloader, criterion, optimizer, num_epochs=1000):`
		
		`# Lista para armazenar a perda em execução para cada época`
		
		`epoch_losses = []`
		`for epoch in tqdm(range(num_epochs)):`
		`  # Armazenando os valores de perda em execução para a época atual`
		`  running_loss = 0.0`
		
		`# Usando tqdm para uma barra de progresso`
		
		`for idx, samples in enumerate(dataloader):`
		`optimizer.zero_grad()`
		
		`# Verifique a camada EmbeddingBag no modelo CBOW`
		
		`if any(isinstance(module, nn.EmbeddingBag) for _, module in model.named_modules()):`
		`target, context, offsets = samples`
		`predicted = model(context, offsets)`
		
		`# Verifique se há camada de Embedding no modelo skip gram`
		
		`elif any(isinstance(module, nn.Embedding) for , module in model.named_modules()):`
		`target, context = samples`
		`predicted = model(context)`
		`loss = criterion(predicted, target)`
		`loss.backward()`
		`torch.nn.utils.clip_grad_norm`(model.parameters(), 0.1)
		`optimizer.step()`
		`running_loss += loss.item()`
		
		`# Adicionar perda média para a época`
		
		`epoch_losses.append(running_loss / len(dataloader))`
		`return model, epoch_losses`

Pacote/Método	Descrição	Exemplo de código

Modelo CBOW	Utiliza palavras de contexto para prever uma palavra-alvo e gerar sua representação.	class CBOW(nn.Module):
		
		Inicializar o modelo CBOW
		
		def **init**(self, vocab_size, embed_dim, num_class):
		super(CBOW, self).**init**()
		
		Defina a camada de incorporação usando nn.EmbeddingBag
		
		self.embedding = nn.EmbeddingBag(vocab_size, embed_dim, sparse=False)
		
		Defina a camada totalmente conectada
		
		self.fc = nn.Linear(embed_dim, vocab_size)
		def forward(self, text, offsets):
		
		Passe o texto de entrada e os deslocamentos pela camada de incorporação
		
		out = self.embedding(text, offsets)
		
		Aplique a função de ativação ReLU à saída da primeira camada linear
		
		out = torch.relu(out)
		
		Passe a saída da ativação ReLU pela camada totalmente conectada
		
		return self.fc(out)
		vocab_size = len(vocab)
		emsize = 24
		model_cbow = CBOW(vocab_size, emsize, vocab_size).to(device)

Loop de treinamento	Enumera os dados do DataLoader e, em cada passagem do loop, obtém um lote de dados de treinamento do DataLoader,	for epoch in tqdm(range(1, EPOCHS + 1)):
		model.train()
		cum_loss=0
		for idx, (label, text, offsets) in enumerate(train_dataloader):
		optimizer.zero_grad()
		predicted_label = model(text, offsets)
		loss = criterion(predicted_label, label)
		loss.backward()
		torch.nn.utils.clip_grad_norm_(model.parameters(), 0.1)
		optimizer.step()
		cum_loss+=loss.item()

Pacote/Método	Descrição	Exemplo de código

	zera os gradientes do otimizador e realiza uma inferência (obtém previsões do modelo para um lote de entrada).	`cum_loss_list.append(cum_loss)`
		`accu_val = evaluate(valid_dataloader)`
		`acc_epoch.append(accu_val)`
		`if accu_val > acc_old:`
		`  acc_old= accu_val`
		`  torch.save(model.state_dict(), 'my_model.pth')`