Q. We used negative log-likelihood in the class to compute the loss. Figure out how to compute the loss using cross entropy and run the same experiments. Report your loss and comment on how this compares to what we did in the class. Provide a snippet of your code where you compute the loss and give your explanation in a few (2-4) sentences. [3 points]

I used PyTorch's cross-entropy function to calculate the loss from my bigram model. The result was 2.162, which is very close to the average negative log-likelihood as computed in class. This shows that both methods measure the same thing, but cross-entropy is easier to use. It also saves time and avoids manually taking logs and averaging.

```
# Create bigram training data
xs, ys = [], []
for w in words:
    str = ['.'] + list(w) + ['.']
    for ch1, ch2 in zip(str, str[1:]):
        index1 = ctoi[ch1]
        index2 = ctoi[ch2]
        xs.append(index1)
        ys.append(index2)

xs = torch.tensor(xs)
ys = torch.tensor(ys)

x_encoded = F.one_hot(xs, num_classes=27).float()

W = torch.log(N + 1).float()  #avoid log(0)
logits = x_encoded @ W
loss = F.cross_entropy(logits, ys)
print(f'Cross-Entropy Loss: {loss.item()}')
```

```
Cross-Entropy Loss: 2.1626980304718018
```

Q. In the class, we used a set of toy examples to train our autoencoder model. Instead, use data from IMDB. Report your losses at different epochs and also a few examples of reconstructed reviews like we did in the class. [7 points]

For this task, I used 1000 IMDB movie reviews to train an autoencoder. I cleaned each review by lowercasing, removing HTML and non-alphanumeric characters, and truncating to a maximum of 15 words. This helped keep the input size fixed and simplified the training process.

I trained the model for 500 epochs. The loss started at 0.7037 and dropped to around 0.0206, showing that the model was learning the reconstruction task over time. However, the outputs mostly repeated common words like "the", "movie", and "film", meaning the model captured frequent patterns but not the full semantic content.

```
Epoch[1/1000], Loss:  0.6958
Epoch[21/1000], Loss:  0.0265
Epoch[41/1000], Loss:  0.0251
Epoch[61/1000], Loss:  0.0233
Epoch[81/1000], Loss:  0.0218
Epoch[101/1000], Loss:  0.0210
Epoch[121/1000], Loss:  0.0206
Epoch[141/1000], Loss:  0.0204
Epoch[161/1000], Loss:  0.0202
Epoch[941/1000], Loss:  0.0111
Epoch[961/1000], Loss:  0.0109
Epoch[981/1000], Loss:  0.0107
```

I also experimented with different encoding dimensions. With just 5 dimensions, the loss plateaued earlier and reconstructions were poorer. Increasing to 50 dimensions led to noticeably better results, with a lower final loss and slightly more meaningful reconstructions. It showed that higher encoding capacity allowed the model to retain more information from the input.

Number of encoding dimensions: 20

Examples:

```
Original: i am curious yellow is a risible and pretentious steaming pile it doesnt matter what
Reconstructed words: and, steaming, risible, yellow, is, pretentious, what, curious, matter, pile, film, it, in

Original: if only to avoid making this type of film in the future this film is
Reconstructed words: this, the, is, to, future, of, type, avoid, making, film, into, has, in

Original: this film was probably inspired by godards masculin fminin and i urge you to see
Reconstructed words: this, to, was, urge, masculin, fminin, inspired, godards, probably, and, see, be, in, dont
```

My understanding tells me maybe this model is beginning to overfit, as it is correctly recalling misspelt words.