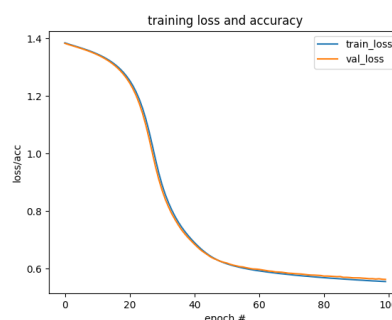
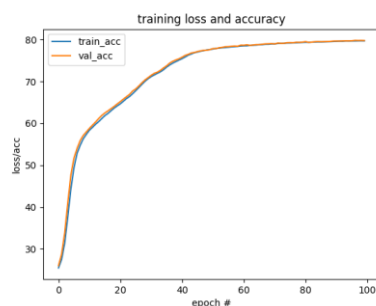


Laboratório 4 - Pipeline de NLP
Tarefa de Classificação de Notícias

Diário de bordo

- 1) Inicialmente testamos 13 configurações/ajustes, sempre com a variável `use_glove = True`, no código fornecido pelo professor, obtendo os resultados abaixo:

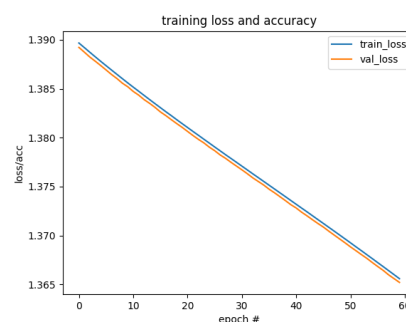
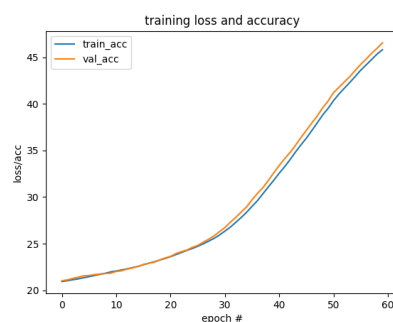
- embedding_size = 100
 hidden_dim = 100
 batch_size = 128
 learning_rate = 0.001
 dropout = (desabilitado)
 num_epochs = 100
 optimizer = SGD



Comentários: depois de 60 épocas, os valores ficaram praticamente estáveis.

Vamos diminuir o nº de épocas e baixar a taxa de aprendizagem.

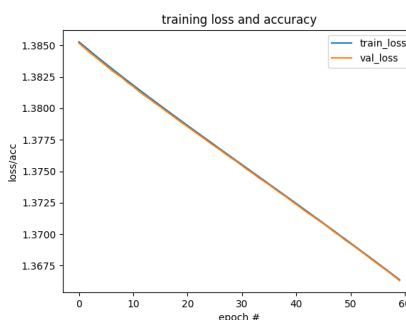
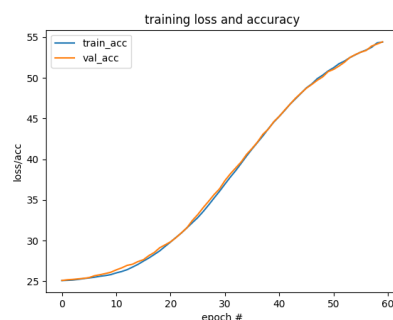
- embedding_size = 100
 hidden_dim = 100
 batch_size = 128
 learning_rate = **0.0001**
 dropout = (desabilitado)
 num_epochs = **60**
 optimizer = SGD



Comentários: acurácia diminuiu muito em relação ao 1º modelo.

Vamos acrescentar poder (nodos) ao modelo.

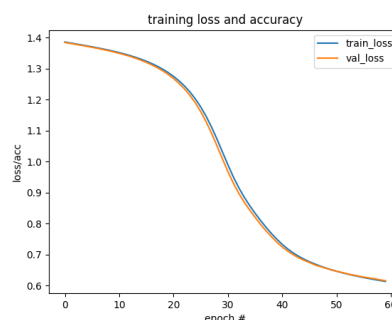
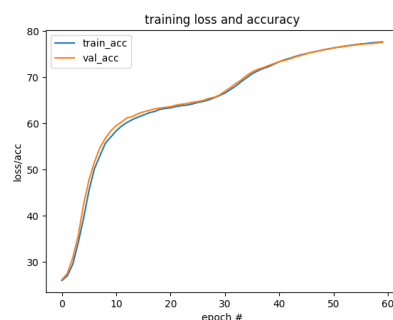
- embedding_size = 100
 hidden_dim = **200**
 batch_size = 128
 learning_rate = 0.0001
 dropout = (desabilitado)
 num_epochs = 60
 optimizer = SGD



Comentários: acurácia melhorou em relação ao modelo anterior, mas ficou muito aquém do 1ª modelo.

Vamos aumentar a taxa de aprendizagem.

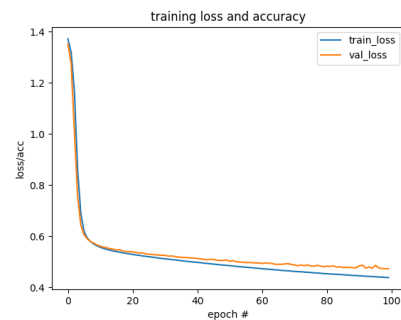
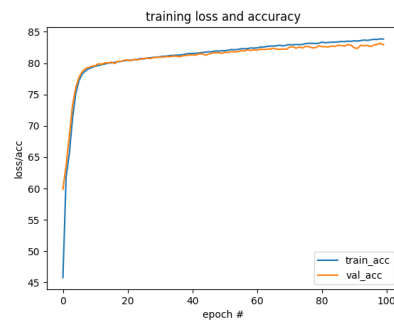
- embedding_size = 100
 hidden_dim = 200
 batch_size = 128
 learning_rate = **0.001**
 dropout = (desabilitado)
 num_epochs = 60
 optimizer = SGD



Comentários: praticamente os mesmos resultados do 1ª modelo apesar de mais poder computacional.

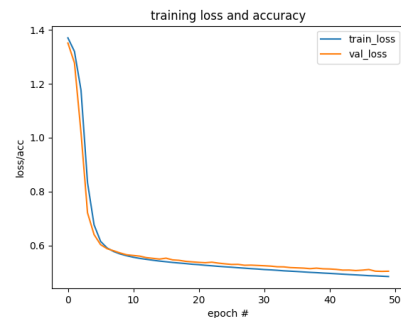
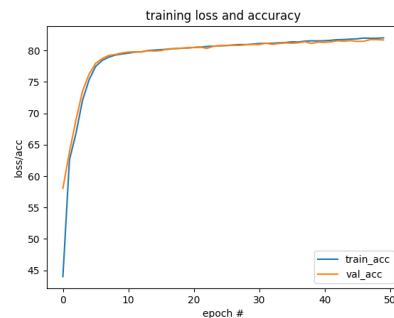
Vamos aumentar mais a taxa de aprendizagem.

5 embedding_size = 100
 hidden_dim = 200
 batch_size = 128
 learning_rate = **0.01**
 dropout = (desabilitado)
 num_epochs = 100
 optimizer = SGD



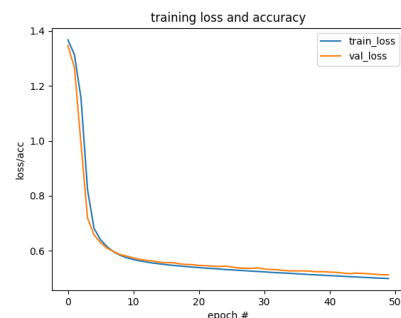
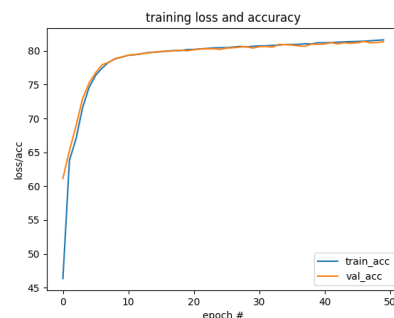
Comentários: praticamente os mesmos resultados anteriores.
 Vamos diminuir o nº de épocas.

6 embedding_size = 100
 hidden_dim = 200
 batch_size = 128
 learning_rate = 0.01
 dropout = (desabilitado)
 num_epochs = **50**
 optimizer = SGD



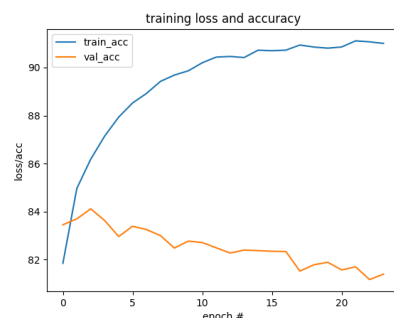
Comentários: praticamente os mesmos resultados anteriores.
 Vamos diminuir aumentar o poder da rede.

7 mbedding_size = 100
 hidden_dim = **400**
 batch_size = 128
 learning_rate = 0.01
 dropout = (desabilitado)
 num_epochs = 50
 optimizer = SGD



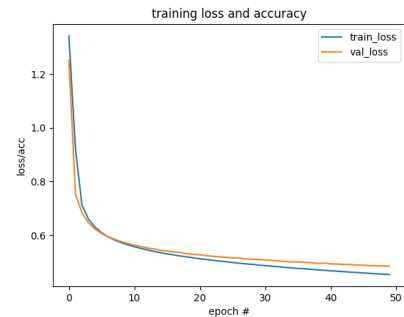
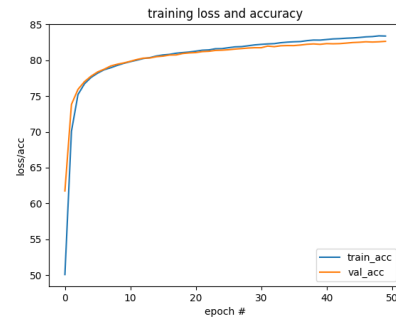
Comentários: praticamente em efeitos na acurácia.
 Vamos alterar o otimizador e diminuir o poder da rede.

8 embedding_size = 100
 hidden_dim = **200**
 batch_size = 128
 learning_rate = 0.01
 dropout = (desabilitado)
 num_epochs = 50 (20)
 optimizer = **ADAM**



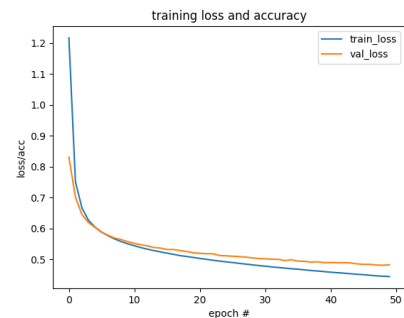
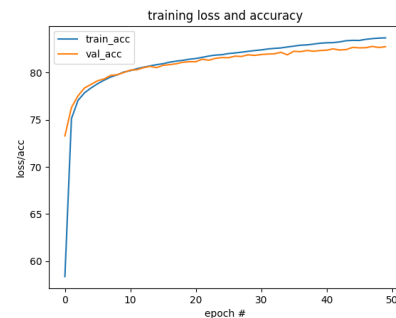
Comentários: de longe a maior acurácia, mas com overfitting. Paramos o treinamento com 20 épocas.
 Vamos diminuir a taxa de aprendizagem em 3 ordens de grandeza.

9 embedding_size = 100
 hidden_dim = 200
 batch_size = 128
 learning_rate = **0.00001**
 dropout = (desabilitado)
 num_epochs = 50
 optimizer = ADAM



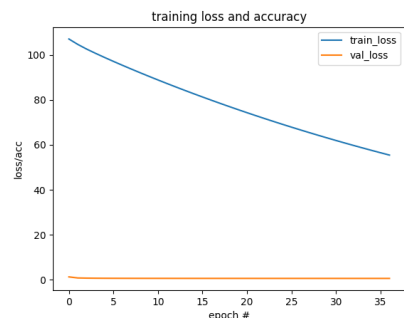
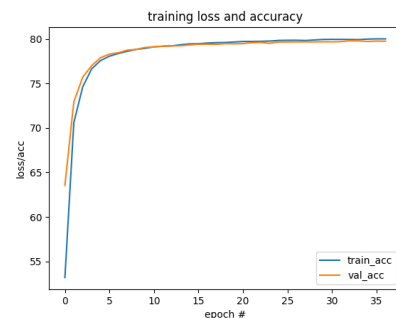
Comentários: o melhor resultado até aqui.
 Vamos aumentar o poder da rede.

10 embedding_size = 100
 hidden_dim = **400**
 batch_size = 128
 learning_rate = 0.00001
 dropout = (desabilitado)
 num_epochs = 50
 optimizer = ADAM



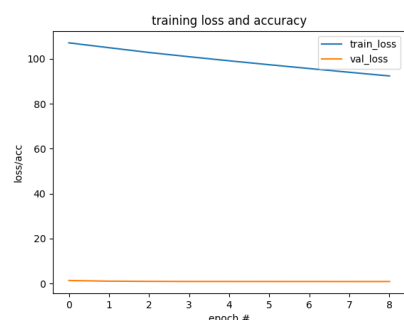
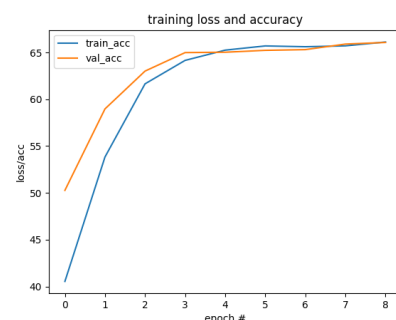
Comentários: praticamente os mesmos resultados que a rede anterior,
 apesar do maior poder (nodos).
 Vamos fazer regularização L2.

11 embedding_size = 100
 hidden_dim = 400
 batch_size = 128
 learning_rate = 0.00001
 dropout = (desabilitado)
 num_epochs = 50 (35)
 optimizer = ADAM
l2_regulariz. = 0.001



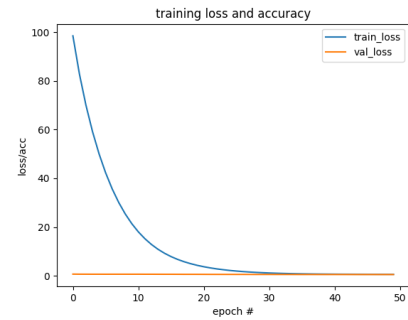
Comentários: Sem melhorar na acurácia, contudo a perda de validade fica
 zerada.
 Vamos habilitar dropout.

12 embedding_size = 100
 hidden_dim = 400
 batch_size = 128
 learning_rate = 0.00001
dropout = 0.5
 num_epochs = 50 (8)
 optimizer = ADAM
 l2_regularization = 0.001



Comentários: além de piorar a acurácia, a perda de validade continuou
 zerada.
 Vamos diminuir o poder da rede e aumentar a taxa de aprendizagem.

13 embedding_size = 100
 hidden_dim = **200**
 batch_size = 128
 learning_rate = **0.0001**
 dropout = 0.5
 num_epochs = 50
 optimizer = ADAM
 l2_regularization = 0.001



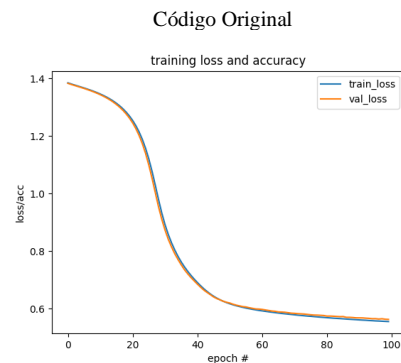
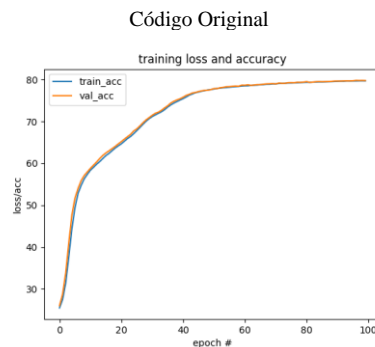
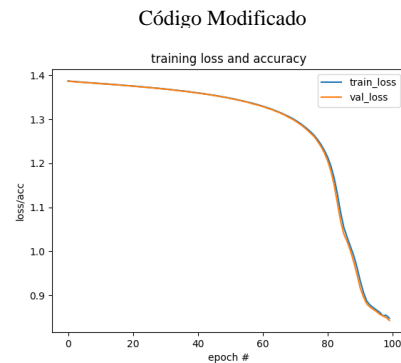
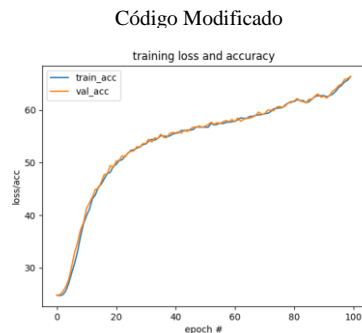
Comentários: Depois que utilizamos a regularização L2, a perda de validação ficou sempre zerada.

2) Implementados a modificação no código da etapa anterior – conforme solicitado no item 1 do laboratório, e escolhemos 5 configurações/ajustes realizados no item 1, de forma a permitir comparações dos resultados. Também calculamos as métricas de desempenho para os dados de testes dos modelos modificados.

embedding_size = 100
 1 hidden_dim = 100
 batch_size = 128
 learning_rate = 0.001
 dropout = (desabilitado)
 num_epochs = 100
 optimizer = SGD

Modificação:

features =
 torch.mean(output,
 dim=1)



Comentários: Conforme gráficos acima, a acurácia do código original foi melhor que a do código modificado.

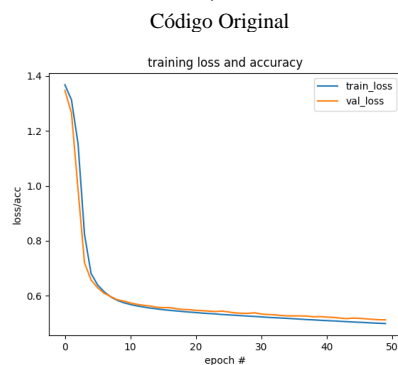
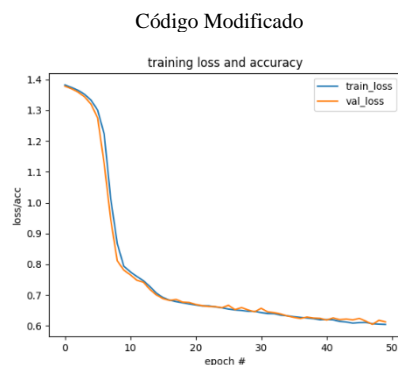
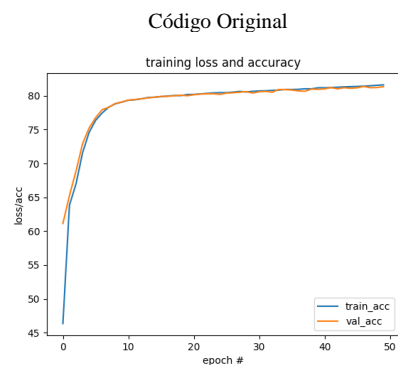
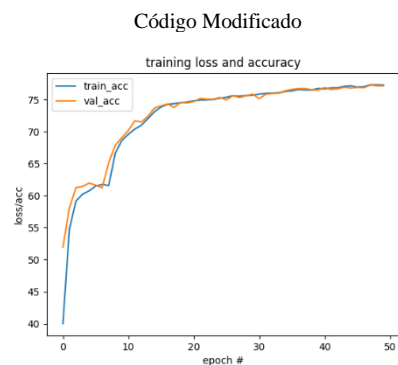
Test loss: 0.845932878766741

Test Accuracy: 66.64620535714283

7 embedding_size = 100
 hidden_dim = **400**
 batch_size = 128
 learning_rate = 0.01
 dropout = (desabilitado)
 num_epochs = 50
 optimizer = SGD

Modificação:

features =
 torch.mean(output,
 dim=1)



Comentários: Novamente, a acurácia do código original foi melhor que a do código modificado.

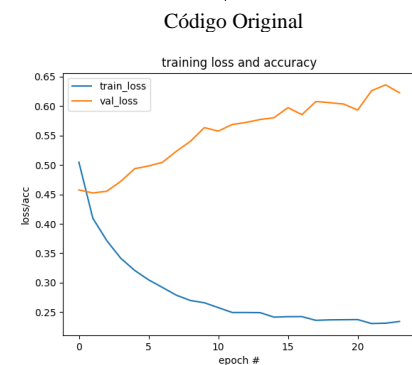
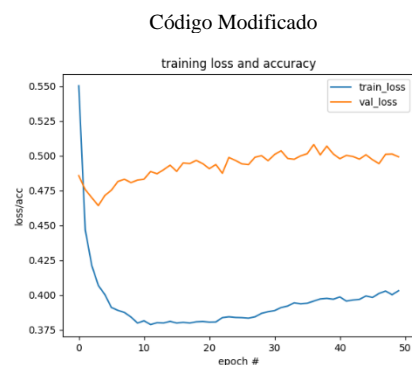
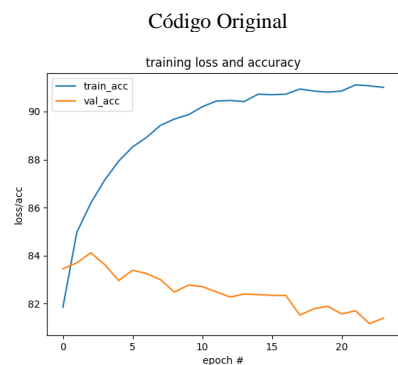
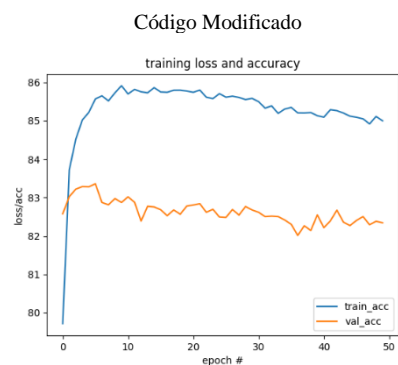
Test loss: 0.6029862405998364

Test Accuracy: 77.23214285714286

8 embedding_size = 100
 hidden_dim = **200**
 batch_size = 128
 learning_rate = 0.01
 dropout = (desabilitado)
 num_epochs = 50 (20)
 optimizer = **ADAM**

Modificação:

features =
 torch.mean(output,
 dim=1)



Comentários: Apesar da acurácia do modelo modificar ter ficado ligeiramente menor que o modelo original, o overfitting diminuiu.

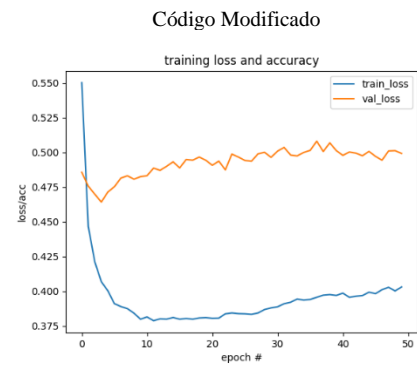
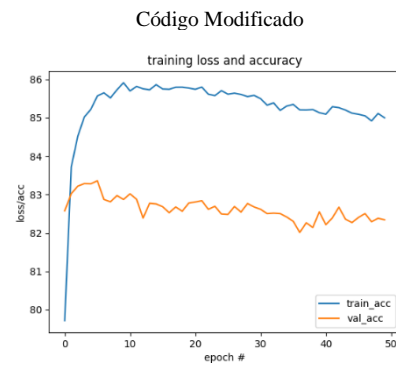
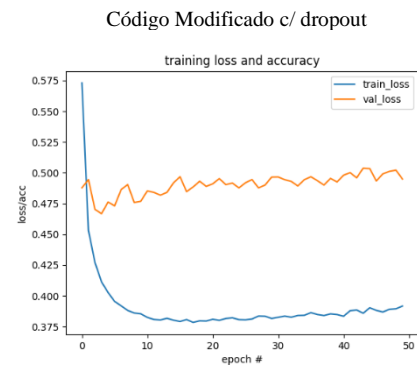
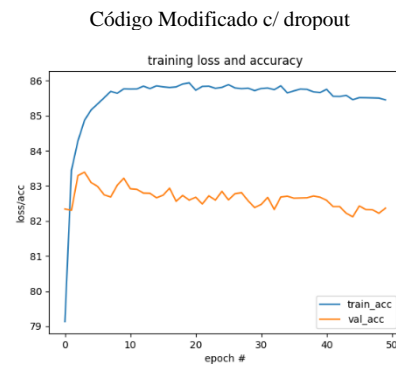
Test loss: 0.4548148612890925

Test Accuracy: 83.57142857142854

Para este modelo habilitamos o dropout para tentar diminuir o overfitting.

8.1 embedding_size = 100
 hidden_dim = 200
 batch_size = 128
 learning_rate = 0.01
 dropout = 0.5
 num_epochs = 50
 optimizer = ADAM

Modificação:
 features =
 torch.mean(output,
 dim=1)

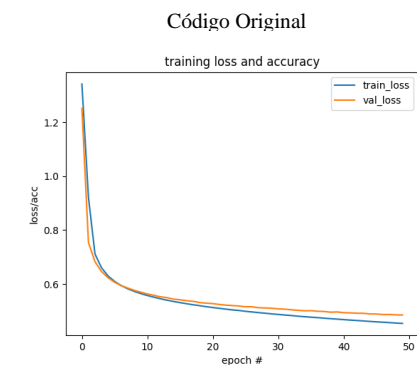
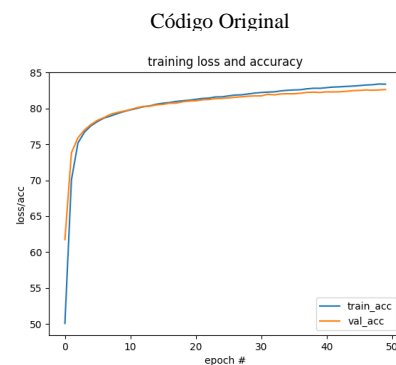
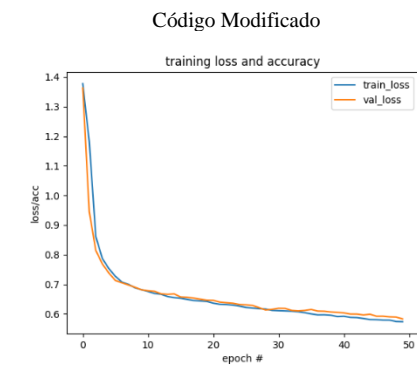
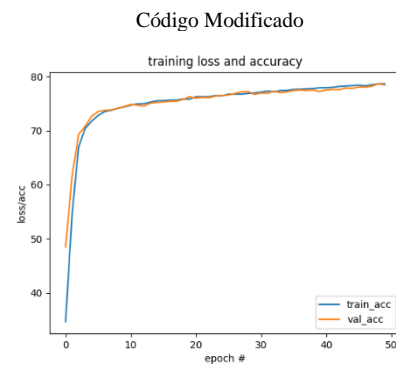


Comentários: A habilitação do dropout praticamente não trouxe melhorias no modelo.

Test loss: 0.46459209663527373

Test Accuracy: 83.27566964285717

9 embedding_size = 100
 hidden_dim = 200
 batch_size = 128
 learning_rate = **0.00001**
 dropout = (desabilitado)
 num_epochs = 50
 optimizer = ADAM



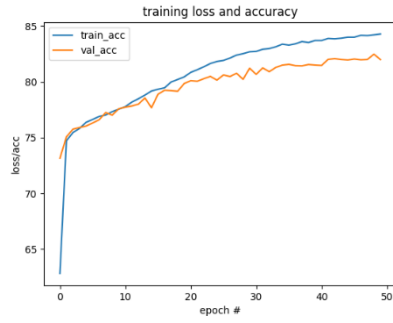
Comentários: Os resultados de ambos os modelos foram praticamente os mesmos.

Test loss: 0.5759246660130368

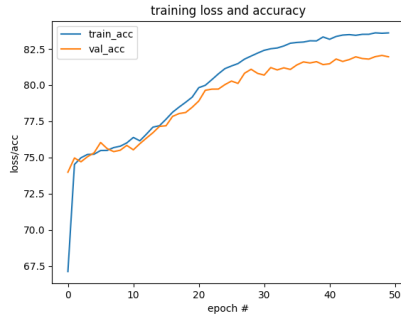
Test Accuracy: 78.60491071428571

```
13 embedding_size = 100
    hidden_dim = 200
    batch_size = 128
    learning_rate = 0.0001
    dropout = 0.5
    num_epochs = 50
    optimizer = ADAM
    l2_regularization = 0.001
```

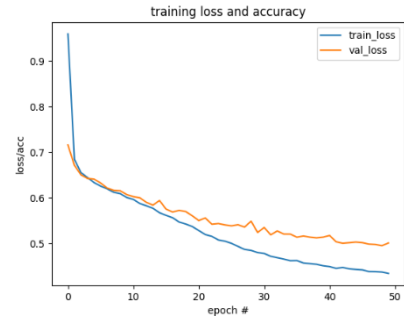
Código Modificado



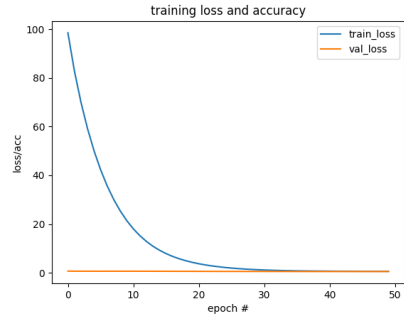
Código Original



Código Modificado



Código Original



Comentários: Com o modelo modificado, a perda de validação deixou de ficar sempre zerada. Este foi o melhor resultado obtido.

Test loss: 0.48753383862120775

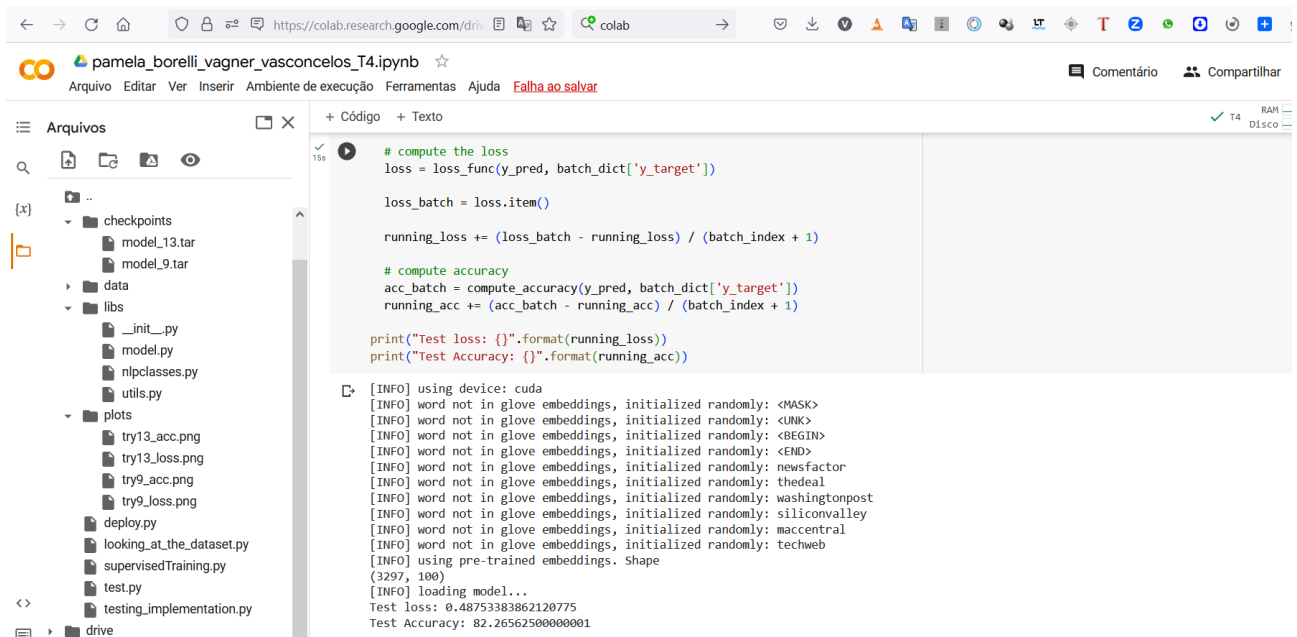
Test Accuracy: 82.26562500000001

Observações:

Executamos o treinamento dos modelos em máquinas locais, utilizando o VSCode.

[illegible]

Também utilizamos o Google Colab, conseguindo utilizar basicamente a mesma estrutura de pasta do projeto.



The screenshot displays the Google Colab interface for a notebook titled 'pamela_borelli_vagner_vasconcelos_T4.ipynb'. The left sidebar shows a file explorer with a directory structure including 'checkpoints', 'data', 'libs', 'plots', and 'drive'. The main area contains a code cell with the following Python code:

```
# compute the loss
loss = loss_func(y_pred, batch_dict['y_target'])

loss_batch = loss.item()

running_loss += (loss_batch - running_loss) / (batch_index + 1)

# compute accuracy
acc_batch = compute_accuracy(y_pred, batch_dict['y_target'])
running_acc += (acc_batch - running_acc) / (batch_index + 1)

print("Test loss: {}".format(running_loss))
print("Test Accuracy: {}".format(running_acc))
```

Below the code, the output shows a series of INFO messages indicating word embeddings are being loaded from GloVe, followed by the final test results:

```
[INFO] using device: cuda
[INFO] word not in glove embeddings, initialized randomly: <MASK>
[INFO] word not in glove embeddings, initialized randomly: <UNK>
[INFO] word not in glove embeddings, initialized randomly: <BEGIN>
[INFO] word not in glove embeddings, initialized randomly: <END>
[INFO] word not in glove embeddings, initialized randomly: newsfactor
[INFO] word not in glove embeddings, initialized randomly: thedeal
[INFO] word not in glove embeddings, initialized randomly: washingtonpost
[INFO] word not in glove embeddings, initialized randomly: siliconvalley
[INFO] word not in glove embeddings, initialized randomly: maccentral
[INFO] word not in glove embeddings, initialized randomly: techweb
[INFO] using pre-trained embeddings. Shape
(3297, 100)
[INFO] loading model...
Test loss: 0.48753383862120775
Test Accuracy: 82.26562500000001
```