Lista1

Silvaneo Viera dos Santos Junior

Questão 1

a)

Primeiramente, observe que:

$$\lim_{\Delta\xi \to 0^-} \frac{h_\xi(\xi + \Delta\xi) - h_\xi(\xi)}{\Delta\xi} = \lim_{\Delta\xi \to 0^-} \frac{\max(0, \cancel{\xi} + \Delta\xi - \cancel{\xi})^3 - \max(0, \cancel{\xi} - \cancel{\xi})^3}{\Delta\xi} = \lim_{\Delta\xi \to 0^-} \frac{\max(0, \Delta\xi)^3}{\Delta\xi} = 0$$

Sendo que a última igualdade vale, pois estamos tomando o limite à esquerda, i.e., $\Delta\xi < 0$.

Vejamos agora que:

$$\lim_{\Delta\xi \to 0^+} \frac{h_\xi(\xi + \Delta\xi) - h_\xi(\xi)}{\Delta\xi} = \lim_{\Delta\xi \to 0^+} \frac{\max(0, \Delta\xi)^3}{\Delta\xi} = \lim_{\Delta\xi \to 0^+} \frac{\Delta\xi^3}{\Delta\xi} = \lim_{\Delta\xi \to 0^+} \Delta\xi^2 = 0$$

Logo, para todo nó ξ, o limite $\lim_{\Delta\xi \to 0^+} \frac{h_\xi(\xi+\Delta\xi) - h_\xi(\xi)}{\Delta\xi}$ existe e é igual a 0, ou seja, h é diferenciável em todos os nós ξ.

b)

É fácil perceber que:

$$h'_\xi(x) = \begin{cases} 3(x - \xi)^2, \text{ se } x > \xi \\ 0, \text{ caso contrário.} \end{cases}$$

Observe que:

$$\lim_{\Delta\xi \to 0^-} \frac{h'_\xi(\xi + \Delta\xi) - h'_\xi(\xi)}{\Delta\xi} = \lim_{\Delta\xi \to 0^-} \frac{0}{\Delta\xi} = 0$$

Sendo que, novamente, como estamos tomando o limite à esquerda, vale que $\xi + \Delta\xi < \xi$, logo $h'_\xi(\xi + \Delta\xi) = 0$.

Ademais, podemos verificar que:

$$\lim_{\Delta\xi \to 0^+} \frac{h'_\xi(\xi + \Delta\xi) - h'_\xi(\xi)}{\Delta\xi} = \lim_{\Delta\xi \to 0^+} \frac{3(\xi + \Delta\xi - \xi)^2 - 0}{\Delta\xi} = \lim_{\Delta\xi \to 0^+} \frac{3\Delta\xi^2}{\Delta\xi} = \lim_{\Delta\xi \to 0^+} 3\Delta\xi = 0$$

Concluímos assim que o limite $\lim_{\Delta\xi \to 0} \frac{h'_\xi(\xi+\Delta\xi) - h'_\xi(\xi)}{\Delta\xi}$ existe e é igual a 0, ou seja, h é duplamente diferenciável em qualquer nó ξ.

c)

É fácil ver que

$$h_\xi''(x) = \begin{cases} 6(x-\xi), \text{ se } x > \xi \\ \quad\quad 0, \text{ caso contrário.} \end{cases}$$

Trivialmente, $6(x-\xi)$ é contínua em $\mathbb{R}$, assim como a função identicamente nula, logo, resta apenas mostrar que h'' é contínua em 0 (para os outros pontos de $\mathbb{R}$, a continuidade é "herdada" das funções identicamente nula e $6(x-\xi)$).

Veja que:

$$\lim_{\Delta\xi \to 0} 6(\cancel{\xi} + \Delta\xi - \cancel{\xi}) = \lim_{\Delta\xi \to 0} 6\Delta\xi = 0$$

Daí:

$$\lim_{\Delta\xi \to 0^-} h_\xi''(\xi + \Delta\xi) = \lim_{\Delta\xi \to 0^+} h_\xi''(\xi + \Delta\xi) = \lim_{x \to 0^+} 6\Delta\xi = 0 = h_\xi''(\xi)$$

Isto é, h'' é contínua em todo nó ξ.

d)

Veja que $\beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3$ é um polinômio, portanto, é infinitamente diferenciável, ademais, sabemos que o conjunto das funções de classe C^2 é fechado para a operação de soma finita, daí, como K é um valor pré-fixado, vale que $\sum_{k=1}^{K} \beta_{k+3} h_{\xi_k}(x)$ é de classe C^2, com isto podemos concluir que $s(x) = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \sum_{k=1}^{K} \beta_{k+3} h_{\xi_k}(x)$ é de classe C^2.

Questão 2

a)

$$\begin{aligned} & y \sim \mathcal{N}(\mu, \sigma^2) \\ & \mu = g(X\beta) \Rightarrow \text{ função de ligação } g(x) = x \\ & \text{A } i\text{-ésima linha de } \boldsymbol{X} \text{ é igual a } \boldsymbol{x_i} = [1, \ln x_i] \end{aligned}$$

b)

$$\begin{aligned} & y \sim \mathbf{Bin}(n, p) \\ & p = g(X\beta) \Rightarrow \text{ função de ligação } g(x) = \frac{1}{1+e^{-x}} \\ & \text{A } i\text{-ésima linha de } \boldsymbol{X} \text{ é igual a } \boldsymbol{x_i} = \begin{bmatrix} 1 \\ x_i \\ x_i^2 \\ x_i^3 \\ \max\{0, (x_i - 0.2)\}^3 \\ \max\{0, (x_i - 1.3)\}^3 \\ \max\{0, (x_i - 2)\}^3 \\ \max\{0, (x_i - 3.2)\}^3 \end{bmatrix}^t \end{aligned}$$

c)

Primeiro, vamos definir algumas funções auxiliares:

$$\begin{aligned}\sigma(x) =& (1+e^{-x})^{-1} \\ \text{softmax}(\vec{x}) =& \frac{\sigma(\vec{x})}{||\sigma(\vec{x})||_1}\end{aligned}$$

Onde consideramos, para $f : \mathbb{R} \to \mathbb{R}$, que:

$$f(\vec{x}) = \begin{bmatrix} f(x_1) \\ \vdots \\ f(x_n) \end{bmatrix}$$

Ademais, definimos a norma $||\vec{x}||_1$ como:

$$||\vec{x}||_1 = \sum_{i=1}^{n} |x_i|$$

No caso particular onde $\vec{x}$ possui todas as entradas positivas, temos que $||\vec{x}||_1 = \sum_{i=1}^{n} x_i$.

Vale destacar que:

$$||\text{softmax}(\vec{x})||_1 = 1$$

Seguiremos agora com a descrição do modelo:

$$\begin{aligned} & y \sim \textbf{Categorica}(\vec{p}) \\ & \vec{p} = g(X\beta) \Rightarrow \text{ função de ligação } g(\vec{x}) = \text{softmax}(\vec{x}) \\ & \text{A } i\text{-ésima linha de } \boldsymbol{X} \text{ é igual a } \boldsymbol{x_i} = \begin{bmatrix} 1 \\ x_i \\ x_i^2 \\ x_i^3 \\ \max\{0, (x_i - 0)\}^3 \\ \max\{0, (x_i - 1)\}^3 \\ \max\{0, (x_i - 2)\}^3 \end{bmatrix}^t \end{aligned}$$

Questão 3

Nas questões a seguir (3 e 4), alternamos entre o uso das linguagens *R* (para gráficos) e *Python* (para cálculos um pouco mais intesivos), sendo que o pacote *reticulate* foi utilizado para intermediá-las, desta forma, a variável *py* no *R* armazena as variáveis do *Python* (ou seja, o comando *py$exemplo* acessa a variável *exemplo* do *python* no *R*) e, de forma análoga, a variável *r* no *Python* armazena as variáveis do *R* (ou seja, o comando *r.exemplo* acessa a variável *exemplo* do *R* no *Python*). Para evitar confusões entre as linguagens, colocamos um indicativo de qual a linguagem usada no início de cada bloco de código.

Vale destacar também que usamos a versão 3.8.10 do *Python* e os pacotes *Numpy* e *Tensorflow*, ademais usamos a versão 4.1.0 do *R* com os pacotes *MASS*, *ggplot2*, *latex2exp*, *tidyr* e *reticulate*.

```r
knitr::opts_chunk$set(echo = TRUE)

# Este código deve ser usado no RStudio integrado a algum Kernel do Python.
```

```r
# A versão do RStudio usada é 1.4.1717.
# A versão do Python usada é 3.8.10.
# A versão do knitr usada é 1.33.
library(MASS)       # Versão 7.3-54
library(ggplot2)    # Versão 3.3.5
library(latex2exp)  # Versão 0.5.0
library(tidyr)      # Versão 1.1.3
library(reticulate) # Versão 1.20

sessionInfo()
```

```
## R version 4.1.0 (2021-05-18)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 19043)
##
## Matrix products: default
##
## locale:
## [1] LC_COLLATE=English_United States.1252
## [2] LC_CTYPE=English_United States.1252
## [3] LC_MONETARY=English_United States.1252
## [4] LC_NUMERIC=C
## [5] LC_TIME=English_United States.1252
## system code page: 65001
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] reticulate_1.20 tidyr_1.1.3     latex2exp_0.5.0 ggplot2_3.3.5
## [5] MASS_7.3-54
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_1.0.7        bslib_0.2.5.1     compiler_4.1.0    pillar_1.6.2
##  [5] jquerylib_0.1.4   tools_4.1.0       digest_0.6.27     lattice_0.20-44
##  [9] jsonlite_1.7.2    evaluate_0.14     lifecycle_1.0.0   tibble_3.1.3
## [13] gtable_0.3.0      png_0.1-7         pkgconfig_2.0.3   rlang_0.4.11
## [17] Matrix_1.3-3      yaml_2.2.1        xfun_0.24         withr_2.4.2
## [21] stringr_1.4.0     dplyr_1.0.7       knitr_1.33        generics_0.1.0
## [25] fs_1.5.0          sass_0.4.0        vctrs_0.3.8       grid_4.1.0
## [29] tidyselect_1.1.1  glue_1.4.2        R6_2.5.0          fansi_0.5.0
## [33] rmarkdown_2.10    purrr_0.3.4       magrittr_2.0.1    scales_1.1.1
## [37] htmltools_0.5.1.1 ellipsis_0.3.2    colorspace_2.0-2  utf8_1.2.2
## [41] stringi_1.7.3     munsell_0.5.0     crayon_1.4.1
```

```python
import tensorflow as tf # Versão 2.5.0
import numpy as np      # Versão 1.19.5
```

a)

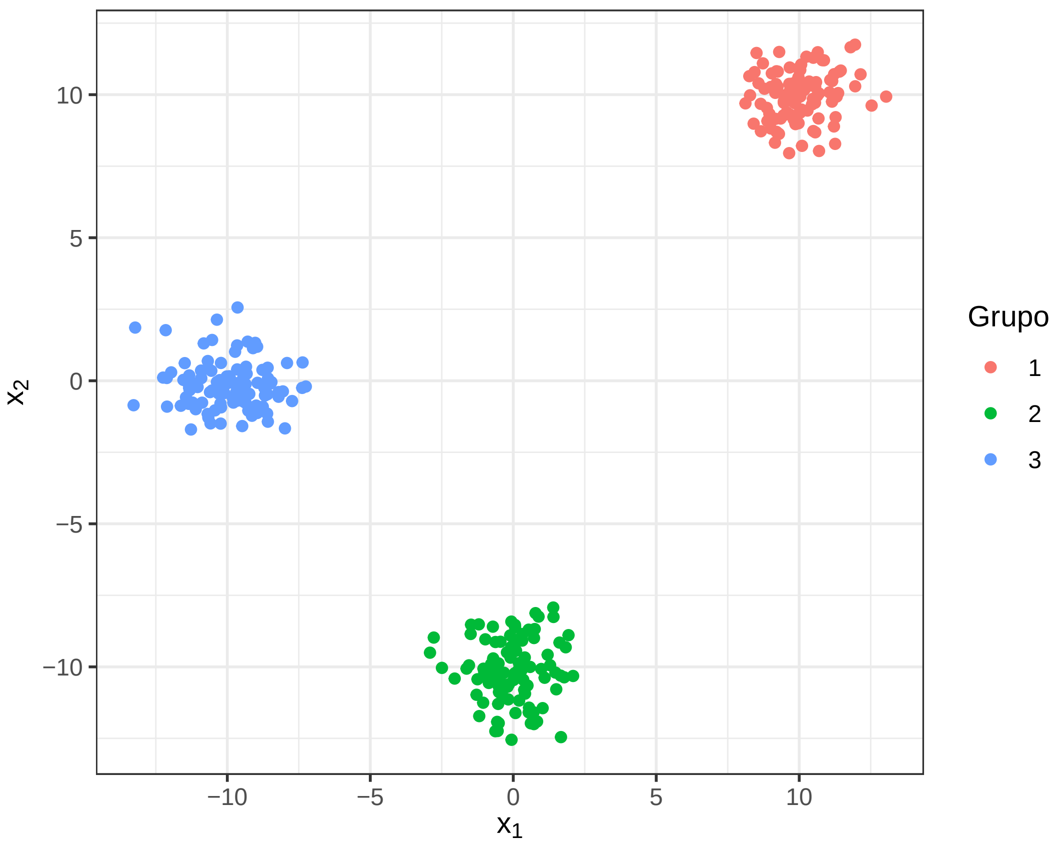
Faremos a simulação tomando $\mu_1 = [10, 10]^t$, $\mu_2 = [0, -10]^t$ e $\mu_3 = [-10, 0]^t$, ademais, usaremos a semente 13031998.

```r
#### R ####
set.seed(13031998)
```

```r
# Usando o pacote MASS para a amostragem
dataset=mvrnorm(300,c(0,0),diag(c(1,1)))
dataset=dataset+matrix(c(rep(c(10,10),100),
                         rep(c(0,-10),100),
                         rep(c(-10,0),100)),
                       300,
                       2,
                       byrow = T)

dataset=as.data.frame(cbind(dataset,c(rep(1,100),
                                      rep(2,100),
                                      rep(3,100))))
names(dataset)=c('x_1','x_2','Grupo')
dataset$Grupo=as.factor(dataset$Grupo)
dataset$Bias=1

ggplot(dataset)+
  geom_point(aes(x=x_1,y=x_2,color=Grupo))+
  scale_x_continuous(TeX('x_1'))+
  scale_y_continuous(TeX('x_2'))+
  theme_bw()+ coord_fixed()
```



b)

Para cada x_i observado, definiremos as variáveis dummy $\bar{1}_i$, $\bar{2}_i$ e $\bar{3}_i$ como:

$$\overline{N}_i = \begin{cases} 1, \text{ se } y_i = N \\ 0, \text{ caso contrário.} \end{cases}$$

Com isto, temos que, para um vetor de probabilidades $\vec{p_i} = [p_{i1}, p_{i2}, p_{i3}]$:

$$\mathbb{P}(y_i = k|\vec{p_i}) = p_{ik}^{\bar{k}}$$

Daí, podemos obter que:

$$\ln(\mathbb{P}(y_i = k|\vec{p_i})) = \bar{k}\ln(p_{ik})$$

Vamos supor que $\vec{p_i}$ pode ser representada pelo seguinte modelo:

$$p_{ij} = \frac{\exp\{x_i^t\vec{\beta}_j\}}{\sum_{l=1}^3 \exp\{x_i^t\vec{\beta}_l\}}$$

Temos que:

$$\begin{aligned}
\frac{d}{dp_{ij}}\ln(\mathbb{P}(y_i = k|\vec{p_i})) &= \begin{cases} \frac{1}{p_{ik}}, \text{ se } j = k \\ 0, \text{ caso contrário.} \end{cases} \\
\nabla_{\vec{\beta}_j} p_{ij} =& \nabla_{\vec{\beta}_j} \frac{\exp\{x_i^t\vec{\beta}_j\}}{\sum_{l=1}^3 \exp\{x_i^t\vec{\beta}_l\}} = \frac{x_i\exp\{x_i^t\vec{\beta}_j\}\sum_{l=1}^3 \exp\{x_i^t\vec{\beta}_l\} - x_i\exp\{x_i^t\vec{\beta}_j\}^2}{\left(\sum_{l=1}^3 \exp\{x_i^t\vec{\beta}_l\}\right)^2} \\
=& \frac{x_i\exp\{x_i^t\vec{\beta}_j\}\cancel{\sum_{l=1}^3 \exp\{x_i^t\vec{\beta}_l\}}}{\left(\sum_{l=1}^3 \exp\{x_i^t\vec{\beta}_l\}\right)^{\cancel{2}}} - \frac{x_i\exp\{x_i^t\vec{\beta}_j\}^2}{\left(\sum_{l=1}^3 \exp\{x_i^t\vec{\beta}_l\}\right)^2} \\
=& x_i\left(\frac{\exp\{x_i^t\vec{\beta}_j\}}{\sum_{l=1}^3 \exp\{x_i^t\vec{\beta}_l\}} - \left(\frac{\exp\{x_i^t\vec{\beta}_j\}}{\sum_{l=1}^3 \exp\{x_i^t\vec{\beta}_l\}}\right)^2\right) \\
=& x_i(p_{ij} - p_{ij}^2) = x_ip_{ij}(1 - p_{ij})
\end{aligned}$$

Agora (caso o leitor tenha sobrevivido as contas acima), vamos obter $\nabla_{\vec{\beta}_j} p_{ik}$ para $j \neq k$:

$$\begin{aligned}
\nabla_{\vec{\beta}_j} p_{ik} =& \nabla_{\vec{\beta}_j} \frac{\exp\{x_i^t\vec{\beta}_k\}}{\sum_{l=1}^3 \exp\{x_i^t\vec{\beta}_l\}} = -x_i\frac{\exp\{x_i^t\vec{\beta}_j\}\exp\{x_i^t\vec{\beta}_k\}}{\left(\sum_{l=1}^3 \exp\{x_i^t\vec{\beta}_l\}\right)^2} \\
=& -x_ip_{ij}p_{ik}
\end{aligned}$$

Usando a regra da cadeia, obtemos que:

$$\nabla_{\vec{\beta}_j}\ln(\mathbb{P}(y_i = k|x_i, \boldsymbol{\beta}) = \begin{cases} \frac{x_i\cancel{p_{ij}}(1 - p_{ij})}{\cancel{p_{ik}}} = x_i(1 - p_{ij}), \text{ se } j = k. \\ \frac{-x_ip_{ij}\cancel{p_{ik}}}{\cancel{p_{ik}}} = -x_ip_{ij}, \text{ se } j \neq k. \end{cases}$$

A expressão acima pode ser reescrita como:

$$\nabla_{\vec{\beta}_j}\ln(\mathbb{P}(y_i = k|x_i, \boldsymbol{\beta}) = x_i(\mathbb{1}(y_i = j) - p_{ij})$$

Agora, ao levarmos em conta todos os dados, obtemos:

$$\nabla_{\vec{\beta}_j} l(\beta_k; \boldsymbol{y}, \boldsymbol{x}) = \sum_{i=1}^{n} x_i(\mathbb{1}(y_i = j) - p_{ij})$$

A equação de atualização para os parâmetros do modelo pode ser escrita como:

$$\beta_k^{i+1} = \beta_k^i + \lambda \nabla_{\vec{\beta}_j} l(\beta_k^i; \boldsymbol{y}, \boldsymbol{x}) = \beta_k^i + \lambda \sum_{i=1}^{n} x_i(\mathbb{1}(y_i = j) - p_{ij})$$

Onde β_k^i é o valor do vetor β_k na etapa i do algoritmo e λ é uma constante arbitrária que regula a velocidade com a qual os parâmetros são atualizados.

Por último, observe que $p_{ij} = 1 - \sum_{k \neq j} p_{ik}$, donde Concluímos que a equação de atualização do modelo depende da probabilidade estimada de cada classe.

c)

Trivialmente, escrevendo $\vec{\beta}_j + c_i = \vec{\beta}_j + \vec{1}c_i$, onde $\vec{1}$ é um vetor de 1's com dimensão igual a de $\vec{\beta}_j$, obtemos que:

$$\begin{aligned}
\mathbb{P}(y_i = j | X, \beta_1 + c_i, \beta_2 + c_i, \beta_3 + c_i) =& \frac{\exp\{x_i^t(\vec{\beta}_j + \vec{1}c_i)\}}{\sum_{l=1}^{3} \exp\{x_i^t(\vec{\beta}_l + \vec{1}c_i)\}} \\
=& \frac{\exp\{x_i^t\vec{\beta}_j + x_i^t\vec{1}c_i\}}{\sum_{l=1}^{3} \exp\{x_i^t\vec{\beta}_l + x_i^t\vec{1}c_i\}} \\
=& \frac{\exp\{x_i^t\vec{\beta}_j\}\cancel{\exp\{x_i^t\vec{1}c_i\}}}{\sum_{l=1}^{3} \exp\{x_i^t(\vec{\beta}_l\}\cancel{\exp\{x_i^t\vec{1}c_i\}}} \\
=& \frac{\exp\{x_i^t\vec{\beta}_j\}}{\sum_{l=1}^{3} \exp\{x_i^t(\vec{\beta}_l\}} \\
=& \mathbb{P}(y_i = j | X, \beta_1, \beta_2, \beta_3)
\end{aligned}$$

A sentença acima vale para todo $c_i \in \mathbb{R}$.

d)

Seja n o número de iterações e λ a taxa de aprendizado, então o pseudo-código para o algoritmo é:

1 - Inicializemos o algoritmo com $\beta_j^0 = \vec{0}$, para todo j, $k = 0$ e $c_i^0 = 0$.

2 - Computamos $p_{ij} = \frac{\exp\{x_i^t(\vec{\beta}_j^k + c_i^k)\}}{\sum_{l=1}^{3} \exp\{x_i^t(\vec{\beta}_l^k + c_i^k)\}}$ para todo $i = 1, ..., n$ e $j = 1, 2, 3$.

3 - Para cada $j = 2, 3$, tomamos:

$$\beta_j^{k+1} = \beta_j^k + \lambda \sum_{i=1}^{n} x_i(\mathbb{1}(y_i = j) - p_{ij})$$

4 - Para cada $i = 1, ..., n$, tomamos:

$$c_i^{k+1} = -\max_j \{x_i^t \beta_j^{k+1}\}$$

5 - Tomamos $k = k + 1$.

6 - Se $k > n$, encerramos o algoritmo, do contrário, retornamos ao passo 2.

e)

```
#### Python ####

def multinom_pred_train(beta,X):
  pre_ativ=X @ beta
  c=-tf.math.reduce_max(pre_ativ,axis=1,keepdims=True)
  ativ=tf.math.exp(pre_ativ+c)
  ativ=ativ/(tf.math.reduce_sum(ativ,axis=1,keepdims=True))
  return tf.squeeze(ativ)

multinom_pred=tf.function(multinom_pred_train)

@tf.function
def multinom_train_step(beta_0,X,Y,n_labels,lamb=10**-3):
  pred=multinom_pred_train(beta_0,X)
  pre_grad= tf.one_hot(Y,n_labels)-pred
  grad=tf.transpose(X) @ pre_grad
  # Atualizando beta_0
  # A parte (1-tf.one_hot(tf.zeros(beta_0.shape[0],dtype='int32'),n_labels))
  # serve para fixar a beta_1=0.
  beta_0.assign(
    beta_0+(1-tf.one_hot(tf.zeros(beta_0.shape[0],dtype='int32'),n_labels))*lamb*grad
    )

def multinom_train(beta_0,X,Y,lamb=10**-3,n=10,batch_size=10):
  erros=[]
  n_labels=tf.cast(tf.math.reduce_max(Y)+1,'int32')
  for step in range(n):
    for i in range(X.shape[0]//batch_size+1):
      X_batched=X[i*batch_size:(i+1)*batch_size]
      Y_batched=Y[i*batch_size:(i+1)*batch_size]
      if tf.math.reduce_sum(X_batched**2)==0:
        break
      if tf.math.reduce_any(tf.math.logical_not(tf.math.is_finite(beta_0))):
        raise ValueError('NAN no pesos')
      multinom_train_step(beta_0,X_batched,Y_batched,n_labels,lamb)

    pred=tf.cast(tf.argmax(multinom_pred(beta_0,X),axis=1),dtype='int32')
    score=tf.cast(tf.equal(pred,data_y),dtype='float32')
    erros.append(tf.math.reduce_mean(score).numpy())
  return erros
```

f)

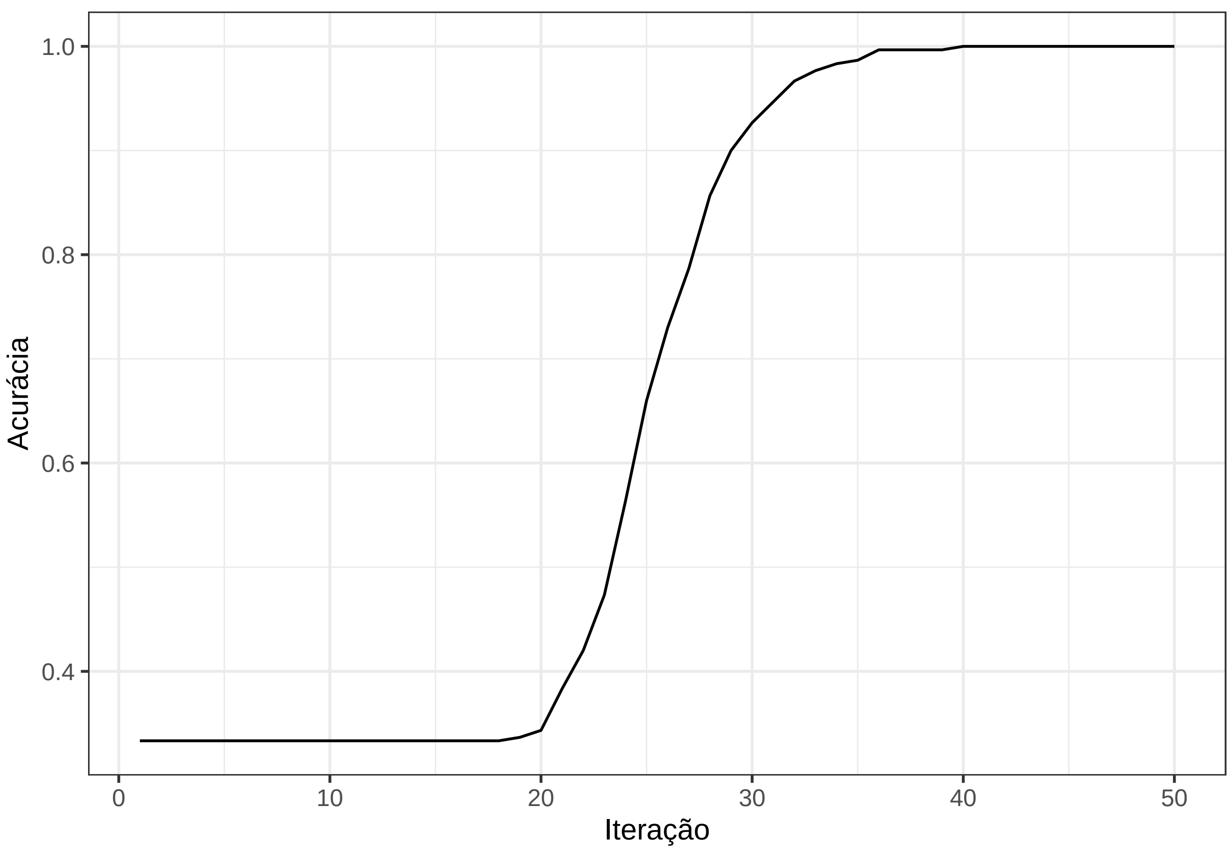
Para este item, tomamos o tamanho dos pacotes $m = 300$, a taxa de aprendizado $\lambda = \frac{1}{m}10^{-10}$ e a quantidade de iterações $n = 100$. Adiante temos o gráfico da acurácia do modelo ao longo das iterações. O valor da taxa de aprendizado foi escolhido propositalmente baixo, de modo que o gráfico ficasse mais interessante (para outros valores de λ o algoritmo obtem 100% de precisão na primeira iteração).

```python
#### Python ####

# Embaralhando a ordem dos dados
index=tf.range(300)
tf.random.set_seed(13031998)
index=tf.random.shuffle(index)
data_x=tf.gather(tf.constant(np.asarray(r.dataset,dtype='float32')[:,[3,0,1]]),index,axis=0)
data_y=tf.gather(tf.constant(np.asarray(r.dataset,dtype='int32')[:,2]-1),index,axis=0)
beta=tf.Variable(tf.zeros([3,3]))
n=50
m=300

erros=multinom_train(beta,
                     data_x,
                     data_y,
                     lamb=tf.constant(10**-10)/m,
                     n=n,
                     batch_size=tf.constant(m,dtype='int32'))
erros=np.asarray(erros)
```

```r
#### R ####
ggplot()+
  #geom_point(aes(x=c(1:py$n),y=py$erros))+
  geom_line(aes(x=c(1:py$n),y=py$erros))+
  scale_x_continuous('Iteração')+
  scale_y_continuous('Acurácia')+
  theme_bw()
```



Questão 4

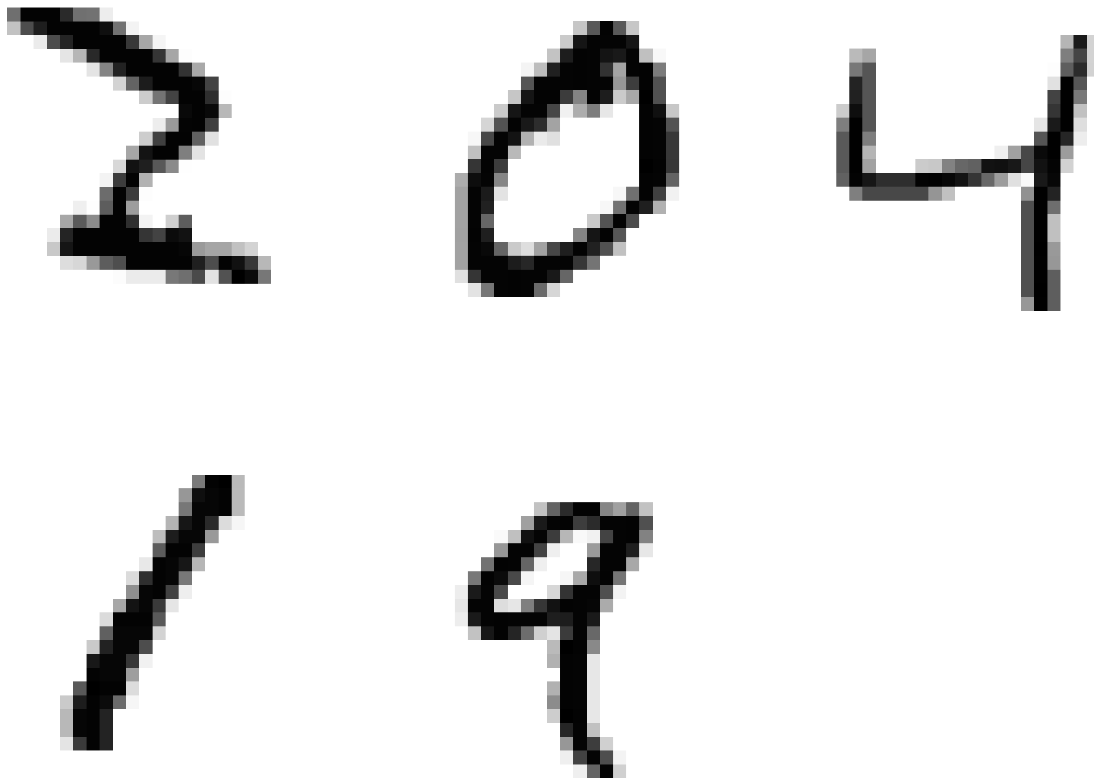
```r
#### R ####
data_train=read.csv('mnist_train.csv',header=F)
train_x=data_train[1:55000,2:785]
train_y=data_train[1:55000,1]

val_x=data_train[55001:60000,2:785]
val_y=data_train[55001:60000,1]

data_test=read.csv('mnist_test.csv',header=F)
test_x=data_test[,2:785]
test_y=data_test[,1]
```

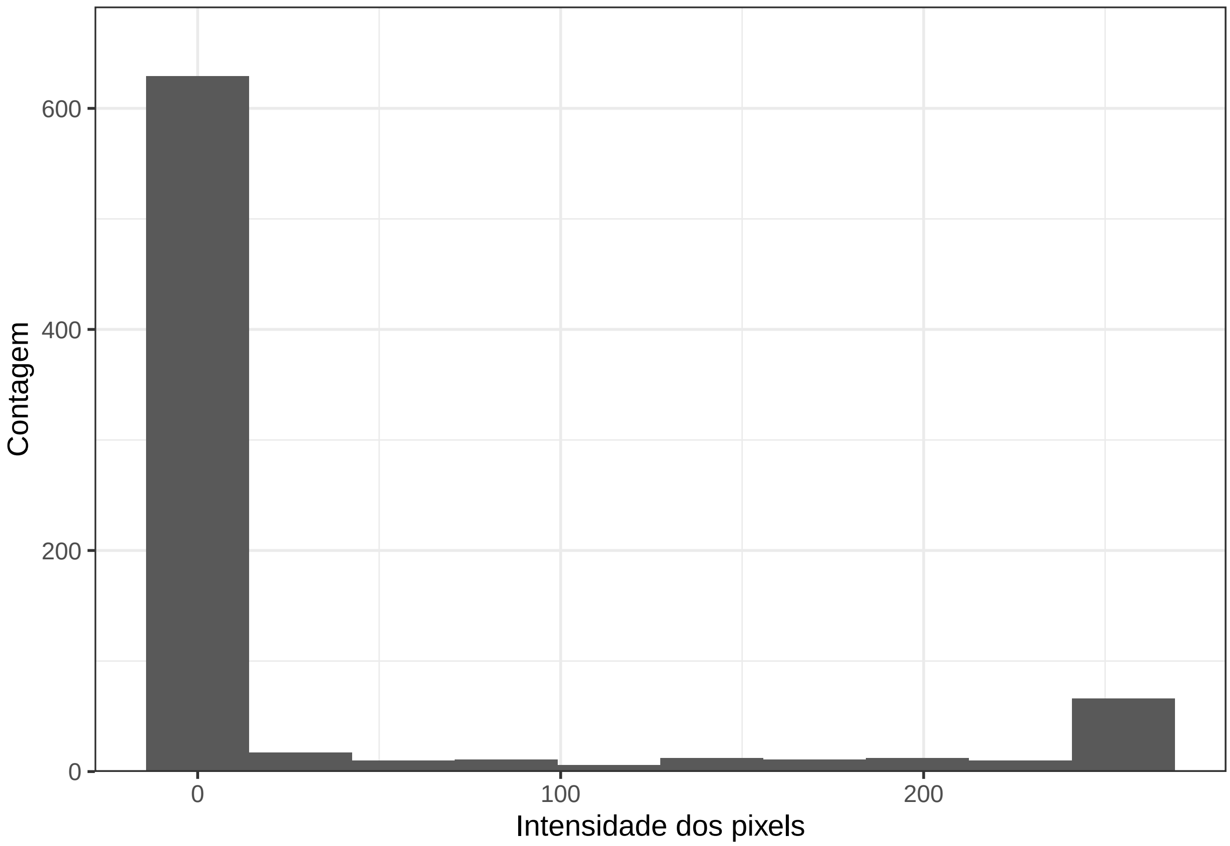
```r
#### R ####
mat=as.data.frame(matrix(train_x[1,],28,28))
names(mat)=c(1:28)
mat$Row=c(1:28)
mat$Index=1
data=as.data.frame(pivot_longer(mat,c(1:28)))
for(i in c(2:5)){
  mat=as.data.frame(matrix(train_x[i,],28,28))
  names(mat)=c(28:1)
  mat$Row=c(1:28)
  mat$Index=i
  data=rbind(data,as.data.frame(pivot_longer(mat,c(1:28))))
}

data$name=as.numeric(data$name)
data$value=as.numeric(data$value)
data$Index=as.factor(data$Index)
ggplot(data)+
  geom_tile(aes(x=Row,y=name,fill=value))+
  scale_fill_gradient(high='#000000',low='#ffffff')+
  guides(fill='none')+
  facet_wrap(~Index)+
  theme_void()+
  theme(strip.background = element_blank(),
        strip.text.x = element_blank())
```



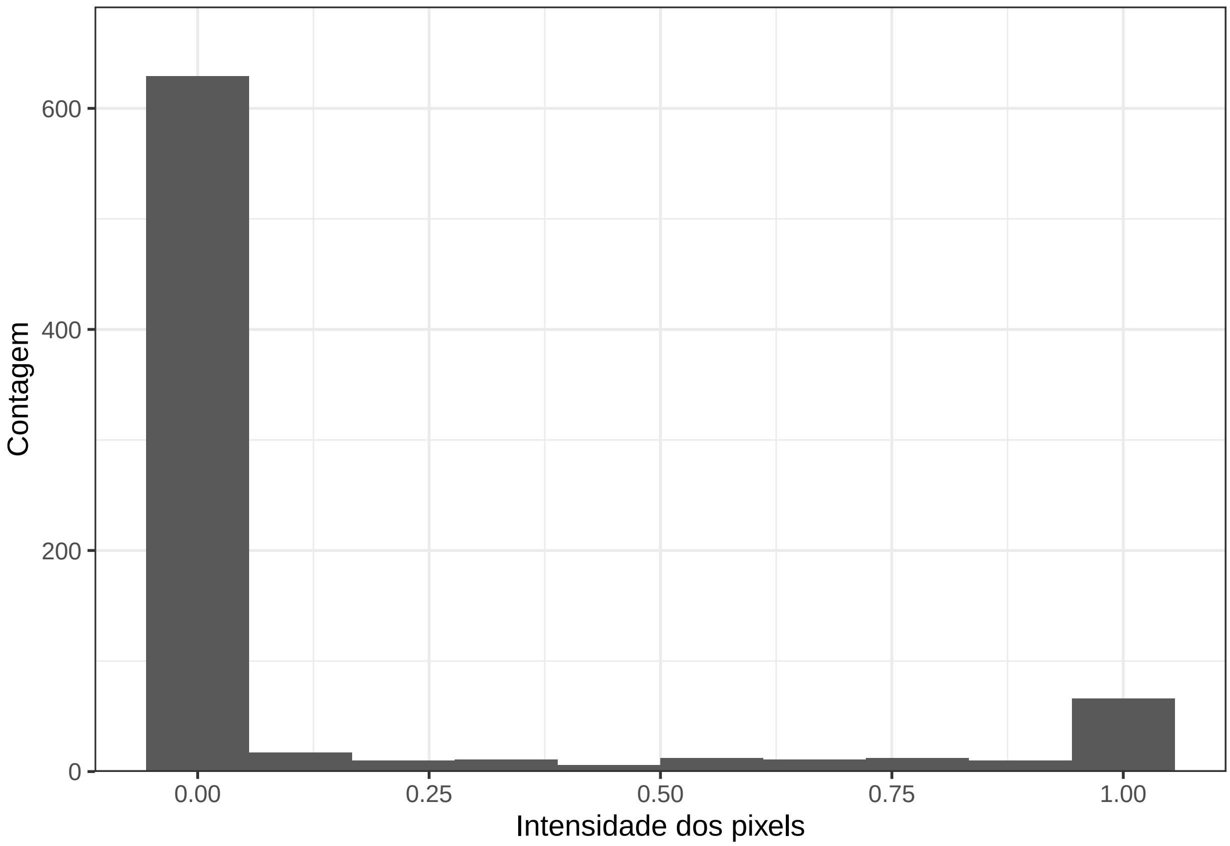
b)

```r
#### R ####
data=data.frame(index=c(1:784),values=as.vector(t(train_x[1,])))
ggplot(data,aes(values))+
  geom_histogram(bins=10)+
  scale_x_continuous('Intensidade dos pixels')+
  scale_y_continuous('Contagem',expand=c(0,0,0.1,0))+
  theme_bw()
```



Com base no histograma acima e observando que a intensidade dos pixels varia no intervalo $[0, 255]$, proponho dividir a intensidade dos pixels por 255. Segue o histograma depois da transformação proposta:

```
#### R ####
data=data.frame(index=c(1:784),values=as.vector(t(train_x[1,])))
ggplot(data,aes(values/255))+
  geom_histogram(bins=10)+
  scale_x_continuous('Intensidade dos pixels')+
  scale_y_continuous('Contagem',expand=c(0,0,0.1,0))+
  theme_bw()
```



c)

```python
#### Python ####
def multinom_train(beta_0,X,Y,X_val=None,Y_val=None,lamb=10**-3,n=10,batch_size=10):
  erros=[]
  n_labels=tf.cast(tf.math.reduce_max(Y)+1,'int32')
  for step in range(n):
    for i in range(X.shape[0]//batch_size+1):
      X_batched=X[i*batch_size:(i+1)*batch_size]
      Y_batched=Y[i*batch_size:(i+1)*batch_size]
      if tf.math.reduce_sum(X_batched**2)==0:
        break
      if tf.math.reduce_any(tf.math.logical_not(tf.math.is_finite(beta_0))):
        raise ValueError('NAN no pesos')
      multinom_train_step(beta_0,X_batched,Y_batched,n_labels,lamb)
    if X_val is not None and Y_val is not None:
      pred=tf.cast(tf.argmax(multinom_pred(beta_0,X_val),axis=1),dtype='int32')
      score=tf.cast(tf.equal(pred,Y_val),dtype='float32')
      score=tf.math.reduce_mean(score)
      erros.append(score)
      tf.print(score)

  return erros
```

d)

Observamos que o tamanho dos batch's estava causando instabilidade, pois a equação de atualização é um somatório nos elementos da amostra, assim, para batch's muito grandes, tínhamos um gradiente igualmente grande em módulo, dificultando a convergência do algoritmo. Para fazer com que a velocidade de treino não dependesse do tamanho do pacote, tomamos como taxa de aprendizado $\lambda = \frac{1}{m}\gamma$, onde γ é a taxa de aprendizado proposta inicialmente e m é a quantidade de elementos em cada batch.

Com a modificações acima e tomando o tamanho dos pacotes $m = 100$, obtemos o seguinte resultado.

```python
#### Python ####

# Adicionando coluna de bias aos dados.
data_x=tf.concat([tf.ones([55000,1],dtype='float32'),
                  tf.constant(r.train_x,dtype='float32')/255],
                axis=1)
val_x=tf.concat([tf.ones([5000,1],dtype='float32'),
                 tf.constant(r.val_x,dtype='float32')/255],
                axis=1)
test_x=tf.concat([tf.ones([10000,1],dtype='float32'),
                  tf.constant(r.test_x,dtype='float32')/255],
                axis=1)

# Salvando conjunto de validação
val_y=tf.constant(r.val_y)
data_y=tf.constant(r.train_y)
test_y=tf.constant(r.test_y)

batch_size=500
n=200

# Treino para gamma=2
beta=tf.Variable(tf.zeros([785,10]))
erros_l1=multinom_train(beta,
                     data_x,
                     data_y,
                     val_x,
                     val_y,
                     lamb=tf.constant((2*10**-0)/batch_size,dtype='float32'),
                     n=n,
                     batch_size=tf.constant(batch_size,dtype='int32'))

# Treino para gamma=1
beta=tf.Variable(tf.zeros([785,10]))
erros_l2=multinom_train(beta,
                     data_x,
                     data_y,
                     val_x,
                     val_y,
                     lamb=tf.constant((10**-0)/batch_size,dtype='float32'),
                     n=n,
                     batch_size=tf.constant(batch_size,dtype='int32'))

# Treino para gamma=0.1
beta=tf.Variable(tf.zeros([785,10]))
```

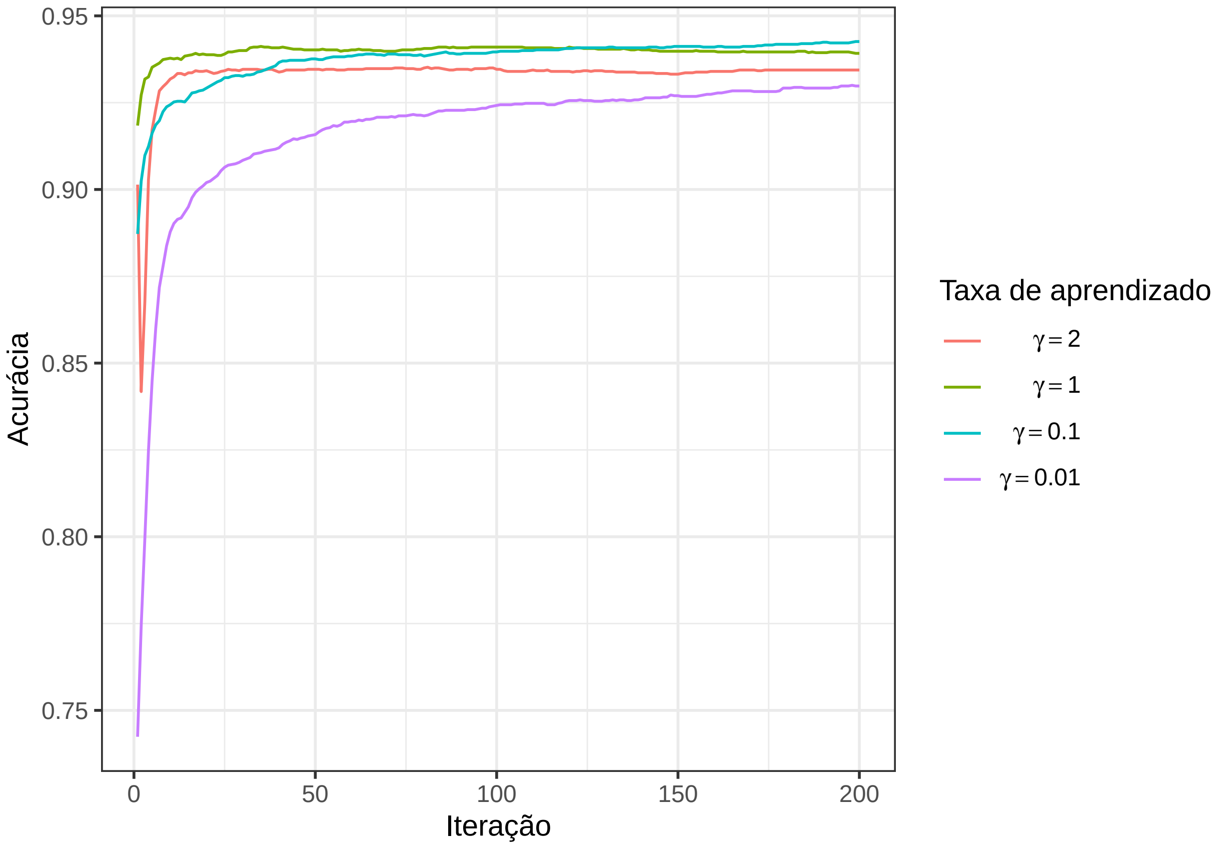
```
erros_l3=multinom_train(beta,
                     data_x,
                     data_y,
                     val_x,
                     val_y,
                     lamb=tf.constant((10**-1)/batch_size,dtype='float32'),
                     n=n,
                     batch_size=tf.constant(batch_size,dtype='int32'))

# Treino para gamma=0.01
beta=tf.Variable(tf.zeros([785,10]))
erros_l4=multinom_train(beta,
                     data_x,
                     data_y,
                     val_x,
                     val_y,
                     lamb=tf.constant((10**-2)/batch_size,dtype='float32'),
                     n=n,
                     batch_size=tf.constant(batch_size,dtype='int32'))

# Criando matriz com as acurácias ao longo do treino
erros=np.asarray([erros_l1,erros_l2,erros_l3,erros_l4])
```

```
#### R ####
indices=c(1:py$n)
precisao=cbind(indices,as.data.frame(t(py$erros)))
names(precisao)=c('Tempo','lambda_1','lambda_2','lambda_3','lambda_4')
precisao=as.data.frame(pivot_longer(precisao,c(2:5)))
names(precisao)=c('Iteração','Taxa_de_aprendizado','Acuracia')
precisao$Taxa_de_aprendizado=as.factor(precisao$Taxa_de_aprendizado)

ggplot(precisao)+
  #geom_point(aes(x=Iteração,y=Acuracia,color=Taxa_de_aprendizado))+
  geom_line(aes(x=Iteração,y=Acuracia,color=Taxa_de_aprendizado))+
  scale_color_hue('Taxa de aprendizado',
                labels=c('lambda_1'=parse(text = TeX('$\\gamma==2$')),
                         'lambda_2'=parse(text = TeX('$\\gamma==1$')),
                         'lambda_3'=parse(text = TeX('$\\gamma==0.1$')),
                         'lambda_4'=parse(text = TeX('$\\gamma==0.01$'))))+
  scale_x_continuous('Iteração')+
  scale_y_continuous('Acurácia')+
  theme_bw()
```



Podemos observar que a taxa de aprendizado $\gamma = 0.1$ trouxe o melhor resultado.

e)

Tomando a taxa de aprendizado $\gamma = 0.1$, obtermos uma precisão de 92.66% no conjunto de testes, que um valor significativamente menor do que o que foi obtido no item anterior (94.26%) com a mesma configuração de treino. Isto pode indicar que há overfitting através dos hiper-parâmetros, porém, mais estudos seriam necessários para chegar a uma conclusão. Adiante, temos o código usado para treino e o gráfico da evolução da acurácia ao longo do treino:

```
#### Python ####

# Adicionando coluna de bias aos dados.
data_x=tf.concat([tf.ones([55000,1],dtype='float32'),
                  tf.constant(r.train_x,dtype='float32')/255],
                axis=1)
val_x=tf.concat([tf.ones([5000,1],dtype='float32'),
                 tf.constant(r.val_x,dtype='float32')/255],
                axis=1)
data_x=tf.concat([data_x,val_x],axis=0)
test_x=tf.concat([tf.ones([10000,1],dtype='float32'),
                  tf.constant(r.test_x,dtype='float32')/255],
                axis=1)

# Salvando conjunto de validação
data_y=tf.constant(r.train_y)
val_y=tf.constant(r.val_y)
data_y=tf.concat([data_y,val_y],axis=0)
```

```python
test_y=tf.constant(r.test_y)

batch_size=500
n=500

# Treino para gamma=0.1
beta=tf.Variable(tf.zeros([785,10]))
erros=multinom_train(beta,
                     data_x,
                     data_y,
                     test_x,
                     test_y,
                     lamb=tf.constant((10**-1)/batch_size,dtype='float32'),
                     n=n,
                     batch_size=tf.constant(batch_size,dtype='int32'))

# Criando matriz com as acurácias ao longo do treino
erros=np.asarray(erros)
```

```r
#### R ####
ggplot()+
  #geom_point(aes(x=c(1:py$n),y=py$erros))+
  geom_line(aes(x=c(1:py$n),y=py$erros))+
  scale_x_continuous('Iteração')+
  scale_y_continuous('Acurácia')+
  theme_bw()
```