Trabalho Prático 3 - Redes Neurais Artificiais

Hugo Araujo de Sousa

Computação Natural (2017/2)
Departamento de Ciência da Computação
Universidade Federal de Minas Gerais (UFMG)

hugosousa@dcc.ufmg.br

Resumo. *Nesse trabalho são explorados conceitos relacionados a redes neurais, colocando-os em prática através da utilização da biblioteca Keras com Tensorflow, que nos permite abordar um problema de classificação.*

1. INTRODUÇÃO

Dentro da área de Computação Natural, o campo de Redes Neurais Artificiais tem como objetivo a criação de modelos computacionais inspirados pelo conhecimento que temos sobre como funciona o sistema nervoso, mais especificamente, na estrutura e função dos neurônios no cérebro [Brownlee 2011].

Dessa forma, uma Rede Neural Artificial é uma coleção de neurônios artificiais que são conectados a fim de se realizar alguma computação em padrões de entrada para gerar padrões de saída. Esses neurônios, então, se adaptam, modificando sua estrutura interna ao longo do tempo. Geralmente, essa modificação se dá através da atualização dos pesos das conexões entre os neurônios da rede. Esse processo define o aprendizado da rede neural, que então se torna, com o tempo, cada vez mais adaptada na tarefa de gerar o padrão de saída correto de acordo com a entrada.

No trabalho em questão, usaremos a biblioteca Keras [1] com Tensorflow [2], que juntas fornecem implementações de redes neurais já prontas para uso. A partir dessas duas bibliotecas, o problema a ser resolvido será o de classificação de um conjunto de dados específico.

Esse conjunto de dados reúne informações de 1429 proteínas, descritas por 8 atributos (números reais). Para cada uma dessas proteínas, a sua classe se refere à parte da célula em que a proteína se encontra. Ao todo, existem 7 classes possíveis, descritas na Tabela 1.

Portanto, a rede neural criada será alimentada com os dados dessa base e, ao longo do tempo, tentará aprender os padrões que determinam a saída da mesma, isto é, dado um conjunto de 8 atributos de uma proteína, a rede deve dizer qual é a posição que essa proteína ocupa na célula (representada pela classe, dentre as 7 descritas acima).

2. MODELAGEM

A modelagem do problema de classificação em questão, utilizando redes neurais implementadas através da biblioteca Keras com Tensorflow é mostrada nessa seção.

[1] https://keras.io/
[2] https://www.tensorflow.org/

Classe	Descrição
CYT	Citoplasma
MIT	Mitocôndria
ME1	Uma membrana específica da célula
ME2	Uma membrana específica da célula
ME3	Uma membrana específica da célula
EXC	Exterior da célula
NUC	Núcleo da célula

Tabela 1. Classes que descrevem a posição das proteínas em uma célula.

2.1. Arquitetura

Para modelar esse problema, o primeiro passo é determinar qual o tipo da rede neural a ser construída - existem vários tipos de arquitetura de redes neurais diferentes.

Para o trabalho em questão, optou-se por simplicidade em termos de representação, o que, por sua vez, acarreta em maior controle sobre a estrutura da rede e de seu funcionamento. Dessa forma, a arquitetura escolhida foi a *Multilayer Perceptron - MLP* (Perceptron de múltiplas camadas) -, que é uma versão da rede *Perceptron* generalizada para conter múltiplas camadas escondidas de neurônios. A Figura 1 mostra a estrutura da rede MLP.

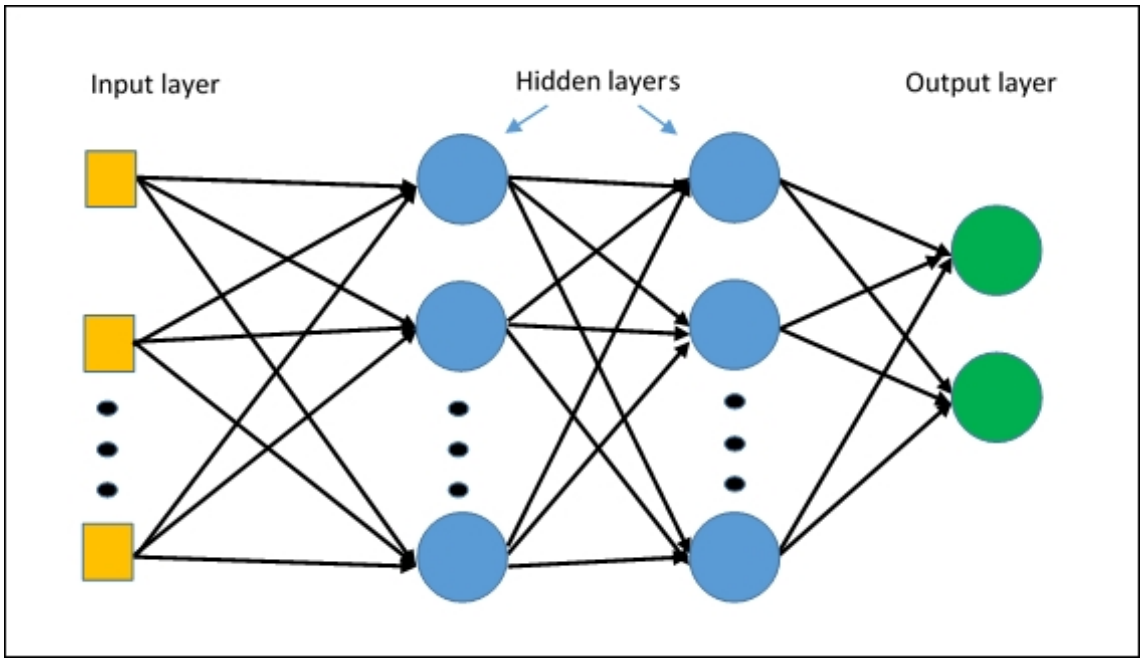

Figura 1. Estrutura da rede *Multilayer Perceptron*. Vemos que, além das camadas de entrada e saída, a rede também pode apresentar um número arbitrário de camadas escondidas de neurônios. O número de neurônios em cada camada também pode ser escolhido arbitrariamente.

O número de camadas escondidas e número de neurônios em cada camada utilizados no trabalho foram definidos experimentalmente, como mostrado na Seção 4.

De forma geral, temos que a saída que um neurônio produz consiste na soma ponderada (pelos pesos) de suas entradas, além de um valor de *bias*.

$$Saida = \sum(Pesos * Entradas) + Bias$$

Além disso, essa saída passa por uma função de ativação antes de servir como entrada para os neurônios da camada seguinte. Essa função de ativação determina se e quanto um neurônio deve contribuir na rede, de acordo com os valores que gera.

2.2. Inicialização de Pesos

Uma decisão importante é a de como inicializar os pesos na rede. Sabemos que se uma camada escondida tem um grande número de entradas, pequenas mudanças nos pesos podem causar grandes alterações nas saídas dessa camada. Uma forma de contornar isso é tornar os pesos iniciais da rede proporcionais ao número de entradas. Para esse trabalho, optou-se por inicializar os pesos proporcionalmente à raiz quadrada do número de entradas nas camadas. Na biblioteca Keras, pode-se fazer isso adicionando o argumento *kernel_initializer='lecun_uniform'* ao construtor de uma nova camada.

2.3. Funções de Ativação

Como mencionado, o processo de ativar um neurônio é fundamental para o funcionamento correto da rede neural artificial. Dessa forma, é uma decisão importante escolher qual função de ativação utilizar nas camadas da rede. Existem muitas opções e a escolha deve levar em consideração o tipo de problema a ser resolvido, além dos conhecimentos sobre os padrões de entrada[Principe et al. 2000]. Para o trabalho em questão, duas funções foram escolhidas, uma para as camadas escondidas e uma para a camada de saída.

Para camadas escondidas, a função ReLU foi escolhida, uma vez que preserva algumas propriedades interessantes de funções lineares e faz com que a otimização com descida do gradiente seja fácil [Goodfellow et al. 2016].

$$RELU(x) = \begin{cases} 0, & \text{if } x < 0 \\ 1, & \text{if } x \geq 0 \end{cases} \tag{1}$$

Já para a camada de saída, a função escolhida foi a Sotfmax, que apresenta resultados geralmente bons para problemas de classificação multi classe. Essa função transforma as saídas de cada neurônio para um valor entre 0 e 1, como uma função sigmoide. Além disso, ela também faz com que a soma total das saídas seja igual a 1.

2.4. Função de Custo

A rede implementada no trabalho utiliza o conceito de *back-propagation*, com o qual os erros da saída da camada de saída são passados de volta para os neurônios internos, que usam esses erros para calcular seus próprios erros e atualizar seus pesos. Esse algoritmo é baseado na otimização de uma função de custo através do método de descida do gradiente. Dessa forma, é também uma decisão de implementação escolher a função de custo do algoritmo. Para esse trabalho, a função escolhida foi a *cross-entropy*

2.5. Validação Cruzada

A fim de garantir que os resultados de generalização da rede não estejam sendo obtidos ao acaso, foi implementado um sistema de validação cruzada de 3 partições. Nesse sistema, o conjunto de dados de entrada é separado em 3 partes, sendo que, a cada momento, duas delas são usadas para treino e a parte restante para teste.

2.6. Codificação da Saída

Tendo em vista que os dados de entrada apresentam as classes de saída como strings, foi necessário transformá-las para alimentar a rede neural. Essas strings que representam as classes foram codificadas utilizando o sistema de variáveis categóricas distribuídas. Dessa forma, cada uma delas foi transformada em um vetor de zeros e uns.

3. ESTRUTURA DO PROJETO E EXECUÇÃO

O trabalho foi implementado utilizando-se a linguagem Python 3. Como já mencionado, as redes neurais artificiais foram criadas através da biblioteca Keras com Tensorflow.

O projeto está estruturado em 4 diretórios, como definidos abaixo:

- **doc:** Contém a documentação do trabalho.
- **src:** Arquivo executável que lê os dados de entrada e cria a rede neural artificial com base nos argumentos passados na linha de comando para resolver o problema de classificação.
- **input:** Base de dados de entrada.
- **tests:** Contém os resultados dos testes executados durante a fase de experimentação.

Para executar o programa, basta executar o seguinte comando no diretório raiz do projeto:

tp3.py [-h] [-e EPOCHS] [-l HLAYERS] [-n NEURONS] [-b BATCHSIZE] [-s RSEED]
[-r LRATE] [-d LRDECAY] arquivo_entrada arquivo_saida

- **Argumentos posicionais:**
  - input_filename: Nome do arquivo de entrada.
  - output_filename: Nome do arquivo de saída.
- **Argumentos opcionais:**
  - -h, –help: Mostra uma mensagem de ajuda e termina execução.
  - -e EPOCHS: Executa o aprendizado da rede neural artificial por um número EPOCHS de épocas.
  - -l HLAYERS: Define o número de camadas escondidas como HLAYERS.
  - -n NEURONS: Faz com que cada camada escondida tenha um número NEURONS de neurônios.
  - -b BATCHSIZE: Define BATCHSIZE como o tamanho dos batches usados no treinamento da rede.
  - -r LRATE: Define a taxa de aprendizado da rede como LRATE.
  - -d LRDECAY: Define a taxa de decaimento da taxa de aprendizado como LRDECAY.
  - -s RSEED: Define RSEED como a semente para geração de números aleatórios durante a execução do programa.

4. EXPERIMENTOS

A fim de verificar o comportamento da rede de acordo com os diversos parâmetros do programa, uma série de experimentos foi realizada. Para cada um deles, um parâmetro foi variável (aquele que buscava-se determinar), enquanto os outros foram mantidos fixos. Além disso, para cada configuração de parâmetros, o programa foi executado com 10 sementes diferentes, obtendo-se assim, ao final de cada experimento, a média dos valores de cada execução. Todos os testes foram executados em um computador i7-5500U CPU @ 2.40 GHz x 4, com Ubuntu 14.04 e 7.7 GB de memória RAM.

4.1. Experimento 1 - Número de neurônios

O primeiro experimento serviu para determinar o número de neurônios a ser utilizado em cada camada escondida da rede. Para isso, os parâmetros foram utilizados como descrito na Tabela 2 e o resultado do experimento é mostrado na Figura 2.

Parâmetro	Valor(es)
e	100
l	1
n	8, 16, 32, 64, 128
b	10
r	0.05
d	0
s	10 para cada n

Tabela 2. Valores de parâmetros para o Experimento 1.

Vemos então que o aumento no número de neurônios sempre acarreta em uma convergência mais rápida, além da rede conseguir alcançar valores mais altos de acurácia para o mesmo número de épocas. Nessa configuração, uma boa escolha para número de neurônios é então o valor 32 que, quando comparado com os outros valores, possui uma velocidade de convergência razoável, além de não perder muito em acurácia em relação aos valores de 64 e 128 neurônios.

4.2. Experimento 2 - Número de camadas escondidas

Uma vez determinado o número de neurônios a utilizar, precisamos determinar quantas camadas escondidas usar na rede. De forma análoga ao Experimento 1, os parâmetros utilizados nesse teste são mostrados na Tabela 3 e o resultado do experimento é apresentado na Figura 3.

Para esse experimento podemos verificar que com 4 camadas escondidas, a rede consegue o maior valor de acurácia ao final das épocas, além de não apresentar convergência imatura, quando comparada com os valores de 1 e 2 camadas.

4.3. Experimento 3 - Taxa de aprendizado

Com o número de camadas escondidas e número de neurônios em cada camada determinados, o próximo parâmetro a definir é a taxa de aprendizado da rede. Os parâmetros do experimento são mostrados na Tabela 4 e os resultados do mesmo são apresentados na Figura 4.

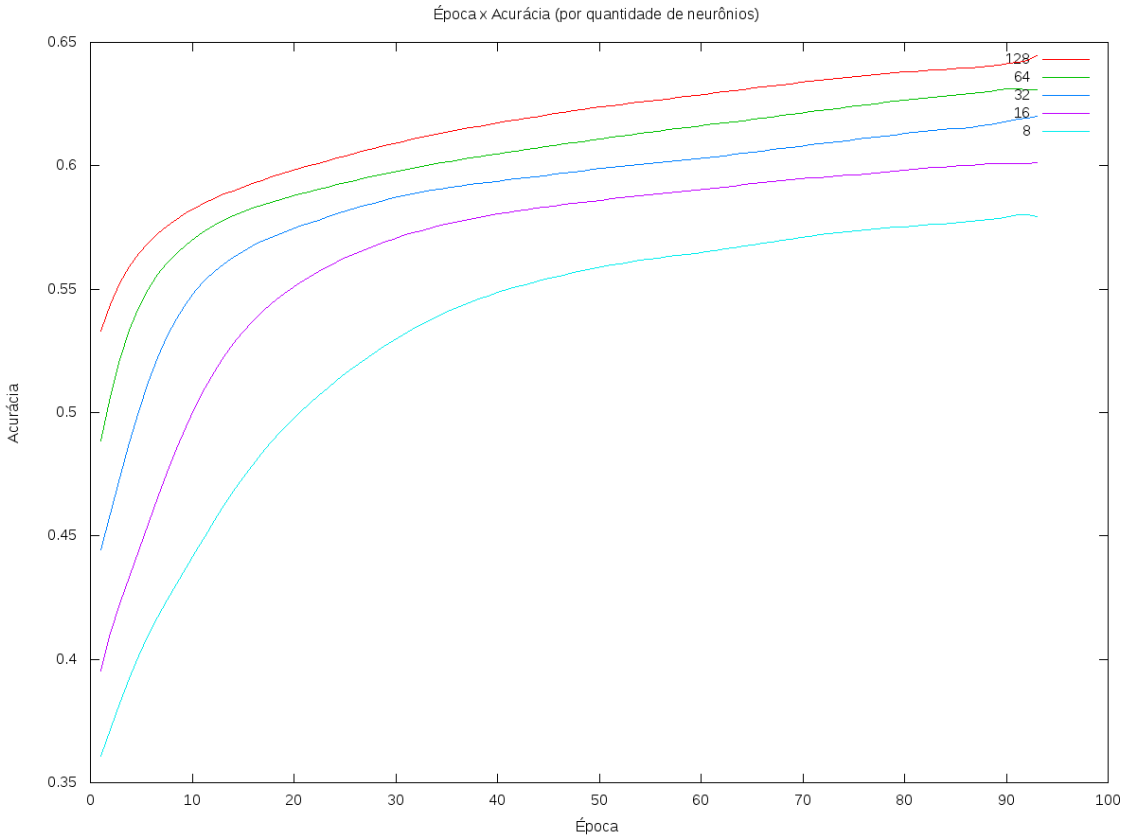

Figura 2. Experimento 1 - Época x Acurácia da rede para diferentes números de neurônios na camada escondida.

Com o resultado desse teste podemos verificar uma curiosa anomalia: para valores a partir de 0.4 o desempenho da rede se torna extremamente ruim, sendo a mesma incapaz de aprender. A escolha fica então entre os valores de 0.05 e 0.1, que atingem valores muito próximos de acurácia. O desempate fica então pelo fator de convergência, escolhendo-se assim o valor 0.05, para o qual a convergência não ocorre tão rapidamente quanto com taxa de aprendizado igual a 0.1.

4.4. Experimento 4 - Taxa de decaimento da taxa de aprendizado

Além de fixar a taxa de aprendizado, podemos verificar o que acontece quando a fazemos variar com o tempo. Para isso, é possível especificar uma taxa de decaimento para a taxa de aprendizado, que é então reduzida automaticamente ao longo das épocas de treino. No experimento 4 isso foi feito com os parâmetros mostrados na Tabela 5, obtendo os resultados mostrado na Figura 5.

Aqui percebemos que, com uma taxa de decaimento para a taxa de aprendizado, somente retardamos a convergência da rede. Nessa situação optou-se por não utilizar uma taxa de decaimento, isto é, usar $d = 0$.

Parâmetro	Valor(es)
e	200
l	1, 2, 4, 6, 8
n	32
b	10
r	0.05
d	0
s	10 para cada l

Tabela 3. Valores de parâmetros para o Experimento 2.

Parâmetro	Valor(es)
e	200
l	4
n	32
b	10
r	0.05, 0.1, 0.2, 0.4, 0.6, 0.9
d	0
s	10 para cada r

Tabela 4. Valores de parâmetros para o Experimento 3.

4.5. Experimento 5 - Tamanho de batch

5. CONCLUSÃO

6. REFERÊNCIAS

[Brownlee 2011] Brownlee, J. (2011). *Clever Algorithms: Nature-Inspired Programming Recipes*. Lulu.com, 1st edition.

[Goodfellow et al. 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

[Principe et al. 2000] Principe, J., Euliano, N., and Lefebvre, W. (2000). *Neural and Adaptive Systems: Fundamentals Through Simulations*. Wiley.

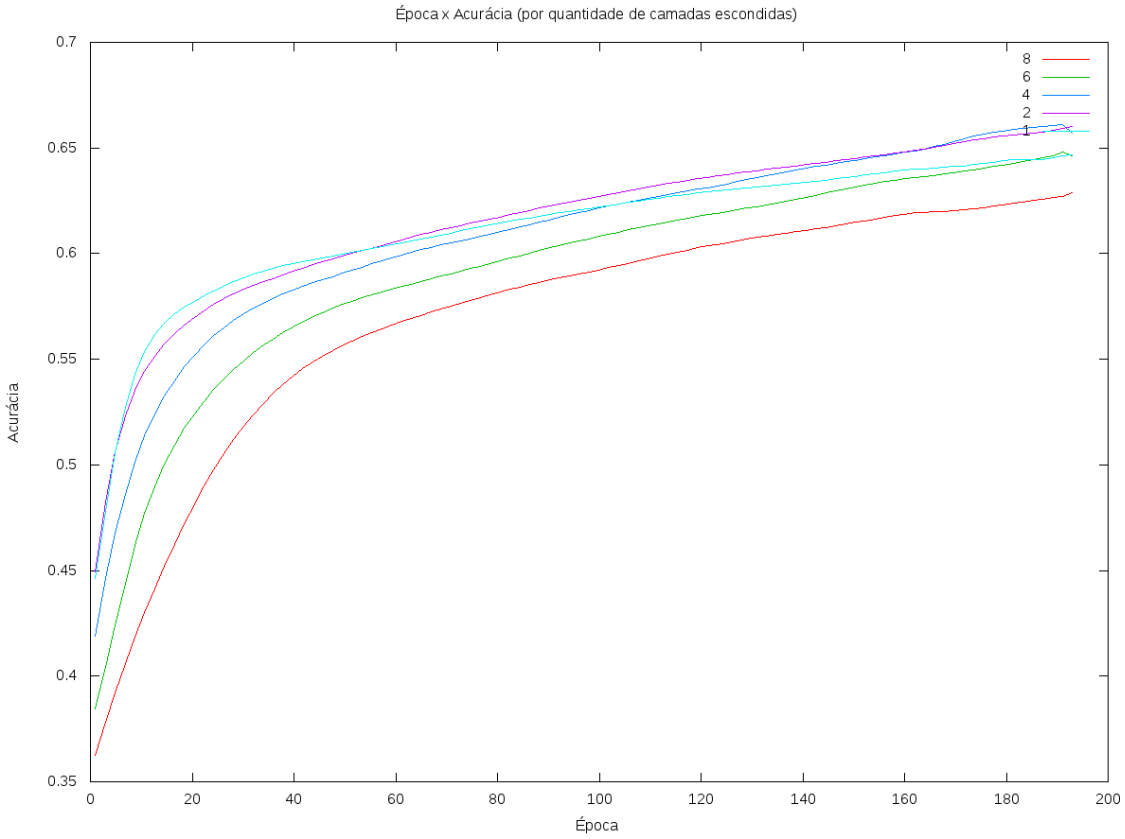

Figura 3. Experimento 2 - Época x Acurácia da rede para diferentes números de camadas escondidas.

Parâmetro	Valor(es)
e	200
l	4
n	32
b	10
r	0.05
d	0.01, 0.02, 0.04, 0.08, 0.1
s	10 para cada d

Tabela 5. Valores de parâmetros para o Experimento 4.

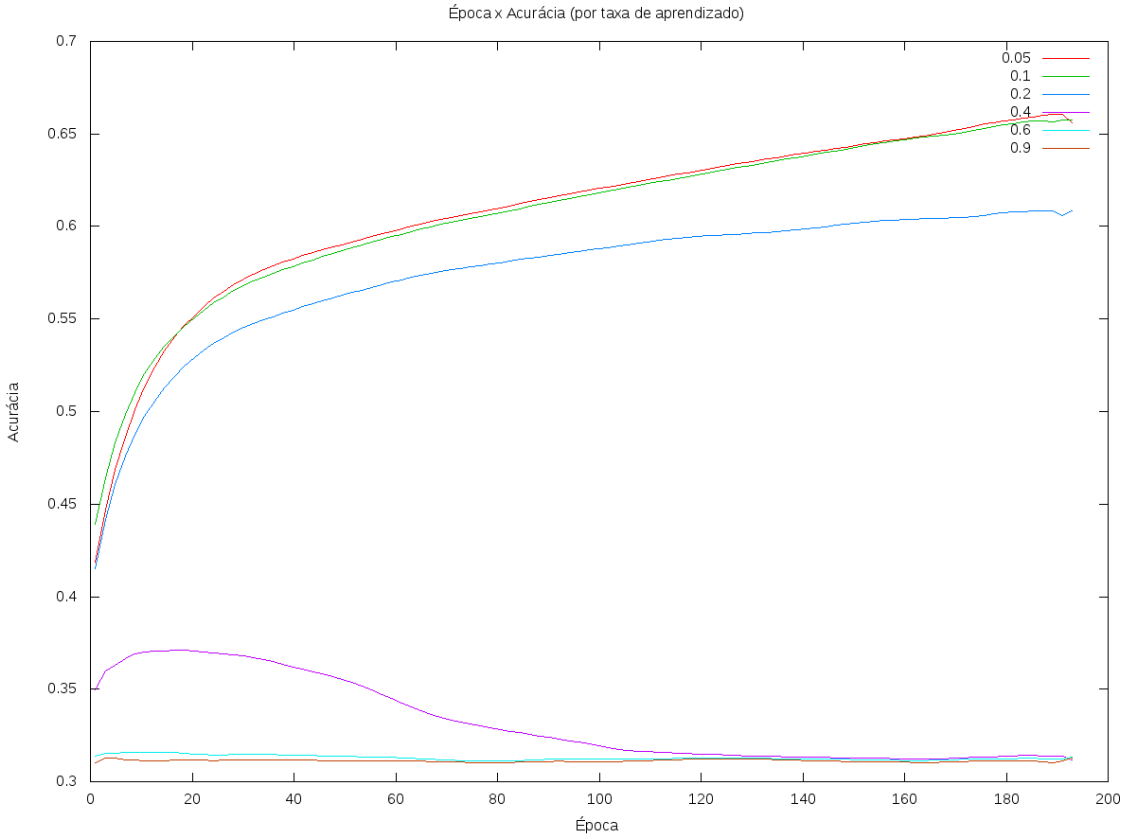

Figura 4. Experimento 3 - Época x Acurácia da rede para diferentes valores de taxa de aprendizado.

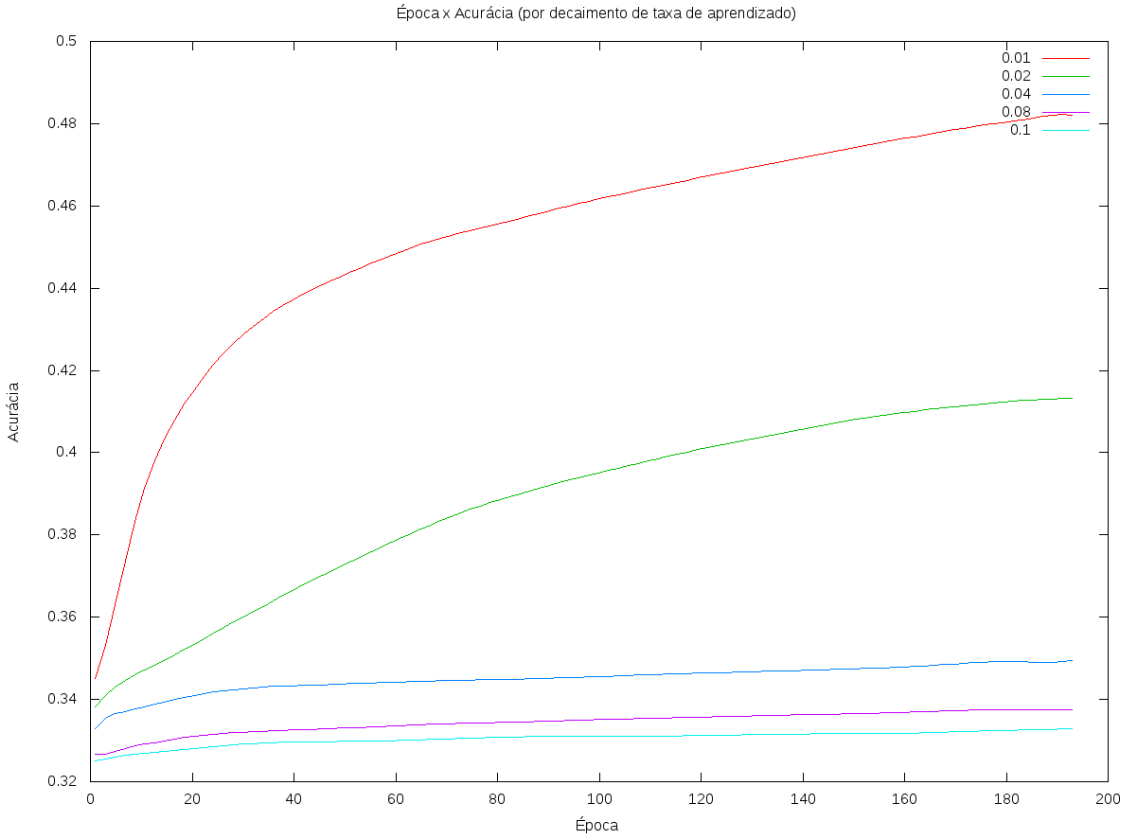

Figura 5. Experimento 4 - Época x Acurácia da rede para diferentes valores de decaimento taxa de aprendizado.