Projeto II de MS960

Redes Neurais

Gianlucca Valente Ribeiro Tesseroli
g173047@dac.unicamp.br

Otávio Moreira Paiano
o185284@dac.unicamp.br

Instituto de Matemática, Estatística e Computação Científica
Universidade Estadual de Campinas

20 de Novembro de 2020

Resumo

Redes neurais são algoritmos de aprendizado de máquina usados, principalmente, para classificação. Nesse trabalho, escrevemos um algorítmo para gerar e treinar uma rede neural de qualquer topologia (desde que todos os neuronios da camada anterior tenham conexão com todos os neurônios da camada seguinte) e para qualquer número de exemplos de treino. O problema de classificação resolvido foi a detecção de dígitos manuscritos. Para treinar e testar o algorítmo, usamos o banco de dados de dígitos manuscritos (MNIST, ref).

Estudamos a perfomance e a evolução da rede variando parâmetros como: *a*) número de exemplos para treino (o que nos permite avaliar a curva de aprendizado da rede); *b*) constante de regularização (λ); *c*) topologia da rede (número de camadas e neurônios por camada); *d*) número de iterações (é importante observar o custo sobre o conjunto de teste enquanto a rede é treinada para evitar overfitting).

Também calculamos numericamente as derivadas (através do método da secante) da função custo para comparar com os valores obtidos pelo método "backpropagation". Os resultados foram idênticos, mas o segundo possui uma eficiência muito superior, visto que a maioria dos valores são reaproveitados (são obtidos ao calcular a função de custo). Também criamos um processo de determinar o melhor λ através do uso de um conjunto de validação. Por fim, discutimos a evolução da taxa de acerto da rede por algarismo e comparamos o resultado com a distribuição dos algarismos nos conjuntos de treino e teste.

Sumário

I Introdução **2**
I.i Função de custo e minimização . . . 2

II Backpropagation **2**
II.i Cálculo do gradiente pelo método da secante . . . 3

III Algoritmos de minimização **3**
III.i Gradiente Descendente . . . 3

IV Conjuntos de treino, validação e teste **6**
IV.i Curva de aprendizado . . . 6
IV.ii Uso do conjunto de validação para determinar a melhor constante de regularização . . . 6
IV.ii.1 Conjunto de treino com duzentos exemplos . . . 6
IV.ii.2 Conjunto de treino com três mil exemplos . . . 7

V Estatística da evolução **7**

VI Conclusão **9**

Referências **9**

I Introdução

Uma rede neural é uma função usada para classificar dados desconhecidos após ser "treinada"com exemplos conhecidos. Para treinar a rede, ou seja, determinar os parâmetros dela, é preciso minimizar a função de custo dada pela soma do "erro"de cada exemplo.

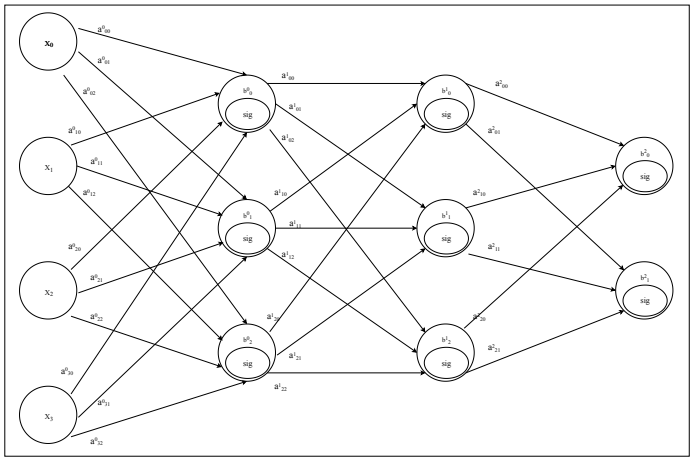


Figura 1: exemplo de rede neural com 4 camadas. X_i são os dados de entrada da rede; $a^l_{i,j}$ são os pesos, e b^l_i são o termo de "bias"da rede. O termo "sig"dentro dos neurônios representa a função sigmoide.

No caso da rede exemplificada na Figura (1), a função que descreve a rede neural seria:

$$\vec{F}(\vec{x}) = \sigma(\ \sigma(\ \sigma(\ \vec{x}\cdot\mathbf{A^{[0]}} + \vec{b}^{[0]})\cdot\mathbf{A^{[1]}} + \vec{b}^{[1]})\cdot\mathbf{A^{[2]}} + \vec{b}^{[2]}) \quad (1)$$

onde os símbolos com → são vetores e os símbolos em negrito são matrizes. A função σ é a sigmoide, definida por

$$\sigma(x) = \frac{1}{1+\exp(-x)} \quad (2)$$

Assim, escrevendo por extenso a passagem da primeira camada para a segunda temos:

$$(x_0,\ x_1,\ x_2,\ x_3)\begin{pmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \\ a_{30} & a_{31} & a_{32} \end{pmatrix}^{[0]} + (b_0,\ b_1,\ b_2)^{[0]} \quad (3)$$

Depois dessa operação, é calculado $\sigma(x)$ para cada elemento x do vetor resultante da Equação (3).

Para calcular o gradiente usando "backpropagation", usamos uma notação para a rede que considera que o termo de "bias"da rede é representado pelos pesos de um neurônio de valor unitário [referenciar notas de aula]. Dessa forma, $\vec{b}$ é incorporado na matriz $\mathbf{A}$ e um neurônio unitário é acrescentado em cada camada (exceto na última). Uma nova matriz Θ é obtida para propagar na rede. Para converter a notação de matrizes de pesos $\mathbf{A}$ e vetores "bias" $\vec{b}$ para a matriz Θ, usamos:

$$\Theta^{[i]} \equiv \text{empilha}\left(\vec{b}^{[i]},\ \mathbf{A}^{[i]}\right)^T \quad (4)$$

onde a função "empilha"cria uma nova matriz em que a primeira linha é dada pelo primeiro argumento e as demais linhas são dadas pela matriz do segundo argumento. O "T"faz a transposta da matriz gerada. Na notação de Θ, h_Θ é a função equivalente à $\vec{F}$.

I.i Função de custo e minimização

A função de custo que descreve nosso problema é dada por[1,2]:

$$J(\Theta) = -\frac{1}{m}\left(\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} log(h_\Theta(\vec{x}^{(i)}))_k + (1-y_k^{(i)}) log(1-h_\Theta(\vec{x}^{(i)}))_k\right) \quad (5)$$

em que $\vec{y}^{(i)}$ é a saída (um vetor de K componentes), ou seja, a resposta esperada para o i-ésimo exemplo e $h_\Theta(\vec{x}^{(i)})$ é a saída obtida pela rede -também é um vetor de K componentes. Realizamos a soma indicada em (5) m vezes, que corresponde ao número de exemplos utilizados. Com a divisão $-\frac{1}{m}$, calculamos o custo médio de cada exemplo, e este será nosso parâmetro final.

Tendo isso em vista, para o projeto em questão, escolhemos o algoritmo regularizado, já que, ao utilizar este método, espera-se uma apredizagem mais rápida, além de ser uma forma de ajudar a combater o overfitting. Com isso, a equação final utilizada é dada por:

$$J(\Theta) = -\frac{1}{m}\left(\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} log(h_\Theta(\vec{x}^{(i)}))_k + (1-y_k^{(i)}) log(1-h_\Theta(\vec{x}^{(i)}))_k\right) + \frac{\lambda}{2m}\sum_{i=1}^{n}\theta_i^2 \quad (6)$$

II Backpropagation

Vimos que para calcular o gradiente da função de custo, Equação (5), podemos fazer [1]

$$\frac{\partial J(\Theta)}{\partial \Theta^{[l]}_{i,j}} = \alpha_j^{[l]}\delta_i^{[l+1]} \quad (7)$$

onde $\alpha_j^{[l]}$ é o valor do neurônio j da camada l e $\delta_i^{[l+1]}$ é o "erro"do neurônio j da camada $l+1$. Os erros podem ser obtidos calculados de forma recursiva do final pro início da rede (daí o nome "backpropagation"). Os erros da última camada, L, são dados por

$$\delta_i^{[L]} = a_i^{[L]} - y_i \quad (8)$$

e os erros das camadas anteriores são recursivamente calculados por

$$\delta_i^{[l]} = \left(\sum_{j=1}^{n_{l+1}} \Theta_{j,i}^{[l]}\ \delta_j^{[l+1]}\right) a_i^{[l]}(1-a_i^{[l]}) \quad (9)$$

onde n_l é o número de neurônios na camada l.

Essa operação deve ser feita para cada par de exemplo $(x^{(i)}, y^{(i)})$, e as derivadas parciais de cada exemplo devem ser somadas.

[1] Estamos considerando que o logarítmo de um vetor é um vetor com o logarítmo das componentes dos vetor de entrada.

[2] Podemos eliminar o somatório sobre K ao escrever a função de custo na forma vetorial:

$$J(\Theta) = -\frac{1}{m}\left(\sum_{i=1}^{m} \vec{y}^{(i)}\cdot log(h_\Theta(\vec{x}^{(i)})) + (\vec{1}-\vec{y}^{(i)})\cdot log(\vec{1}-h_\Theta(\vec{x}^{(i)}))\right)$$

II.i Cálculo do gradiente pelo método da secante

Outra forma de calcular o gradiente da função de custo seria fazer uma aproximação pela secante:

$$\frac{\partial J}{\partial \theta_i} = \frac{J(\theta + \epsilon_i) - J(\theta - \epsilon_i)}{2|\epsilon_i|} \tag{10}$$

Com este método, observamos a relação entre as derivadas, utilizando, apenas, cinco exemplos, e executando essa verificação, somente, na primeira iteração. Podemos observar abaixo tal comportamento.

```
8 new_theta, J_history = rede1.grad_desc(rede1.pesos_iniciais, alpha, max_iter, Lambda, epsilon, True)
```

```
1072 A razão entre as derivadas é: nan
1073 A razão entre as derivadas é: nan
1074 A razão entre as derivadas é: nan
1075 A razão entre as derivadas é: 1.0000000050522595
1076 A razão entre as derivadas é: 1.0000000004770953
1077 A razão entre as derivadas é: 0.9999999983020132
1078 A razão entre as derivadas é: 0.9999999999049421
```

Figura 2: as duas primeiras linhas de pixel têm pesos nulos ($2 \times 20 \times 25 \approx 1075$).

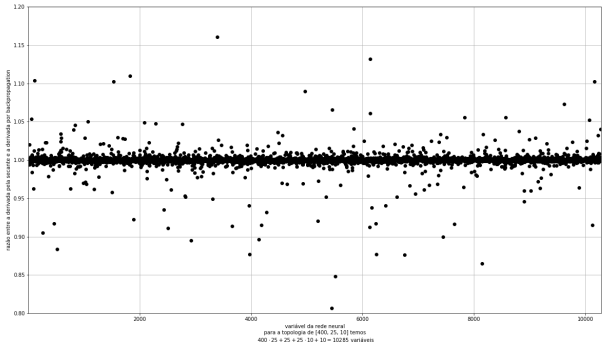


Figura 3: razão entre as derivadas parciais da função de custo calculadas pela secante e por backpropagation (imagem meramente ilustrativa).

Como é possível observar na figura anterior, os valores obtidos pelo método da secante e pelo gradiente descendente estão bem próximos. Com isso, temos a certeza de que o algoritmo está adequado. Além disso, vale ressaltar a que os pixels mais próximos à borda possuem valores de derivadas << 1, já que a escrita, normalmente, fica mais ao centro (por exemplo, o primeiro pixel, muito provavelmente, vai ser zero para todos os exemplos, então sua derivada sempr será zero). Com isso, uma medida que poderia ser tomada para a otimização do código é a retirada desses pixels obsoletos.

III Algoritmos de minimização

Para treinar a rede neural dividimos o conjunto total de exemplos em subconjuntos, atentando-se para a proporção de dados para cada dígito, que permaneceu equilibrada. Além disso, foram testados três algoritmos de Programação Não-Linear: Gradiente Descendente, Gradiente Conjudado, e BFGS.

O Gradiente Descendente obteve a pior performance, o que era esperado visto que é um método de convergência linear. Devido ao tempo necessário para treinar a rede com esse método, não foi possível alcançar taxas de acerto superiores a 65%. Na Figura (4), é possível observar a função de custo pelo número de iterações.

Os outros dois métodos citados convergiram mais rapidamente (além de serem métodos de ordem superior, foram escritos em linguagem de baixo nível e incorporados na biblioteca `scipy`, enquanto o Gradiente Descendente utilizado foi escrito em `python`). A Figura (10) ilustra a função de custo por iteração para o método BGFS. Esses dois métodos serão ilustrados nas próximas seções.

III.i Gradiente Descendente

A taxa de previsões corretas para a rede neural de topologia [400, 25, 10] foi de 63.4% sobre o conjunto de treino e de 65% sobre o conjunto de teste. Apesar de ter chamado atenção o fato do conjunto de teste ter performado melhor do que o conjunto de treino, não encontramos explicação além dos desvios estatísticos esperados pela aleatoriedade dos conjuntos. A proporção foi de 60% dos exemplos para treino -3000 imagens- e 40% para teste -2000 imagens.

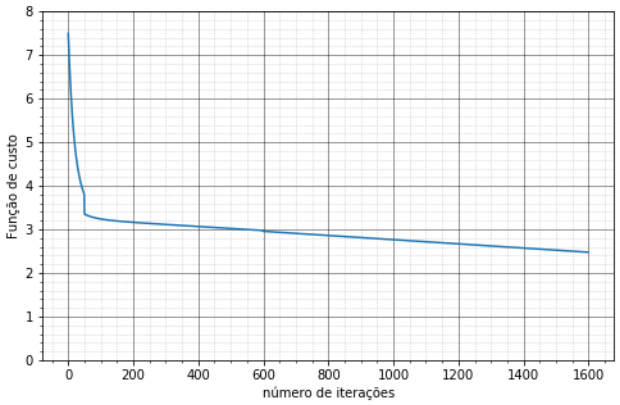


Figura 4: função de custo pelo número de iterações com o método Gradiente Descendente.

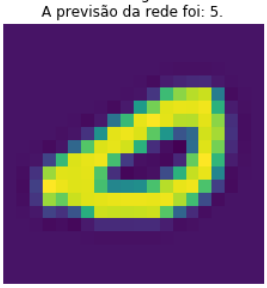
Na próxima página, estão alguns dígitos classificados de forma diferente do rótulo (valor "correto", entre aspas porque em alguns casos o rótulo é questionável mesmo para um humano) pela rede neural.

As imagens dos digitos classificados incorretamente foram selecionadas de forma aleatória. Nota-se que a rede neural confunde, com alguma frequência, os seguintes pares de algarismos: (0; 5), (1; 4), (2; 6).

A **taxa de previsões incorretas** por algarismo no conjunto de *testes* e o **percentual que eles representam no conjunto de** *treino* foram, respectivamente:

- 0: 19.19%, 10.07%
- 1: 4.90 %, 9.80 %
- 2: 51.06%, 9.47 %
- 3: 67.96%, 9.83 %
- 4: 33.68%, 10.03%
- 5: 65.09%, 10.07%
- 6: 5.41 %, 9.47 %
- 7: 32.00%, 10.43%
- 8: 40.82%, 10.33%
- 9: 30.43%, 10.50%

O rótulo do dígito é: 0.
A previsão da rede foi: 5.



O rótulo do dígito é: 5.
A previsão da rede foi: 0.

O rótulo do dígito é: 1.
A previsão da rede foi: 4.

O rótulo do dígito é: 6.
A previsão da rede foi: 2.

O rótulo do dígito é: 2.
A previsão da rede foi: 6.

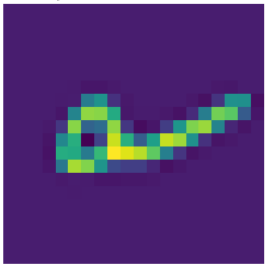
O rótulo do dígito é: 7.
A previsão da rede foi: 9.

O rótulo do dígito é: 3.
A previsão da rede foi: 8.

O rótulo do dígito é: 8.
A previsão da rede foi: 7.

O rótulo do dígito é: 4.
A previsão da rede foi: 1.

O rótulo do dígito é: 9.
A previsão da rede foi: 1.



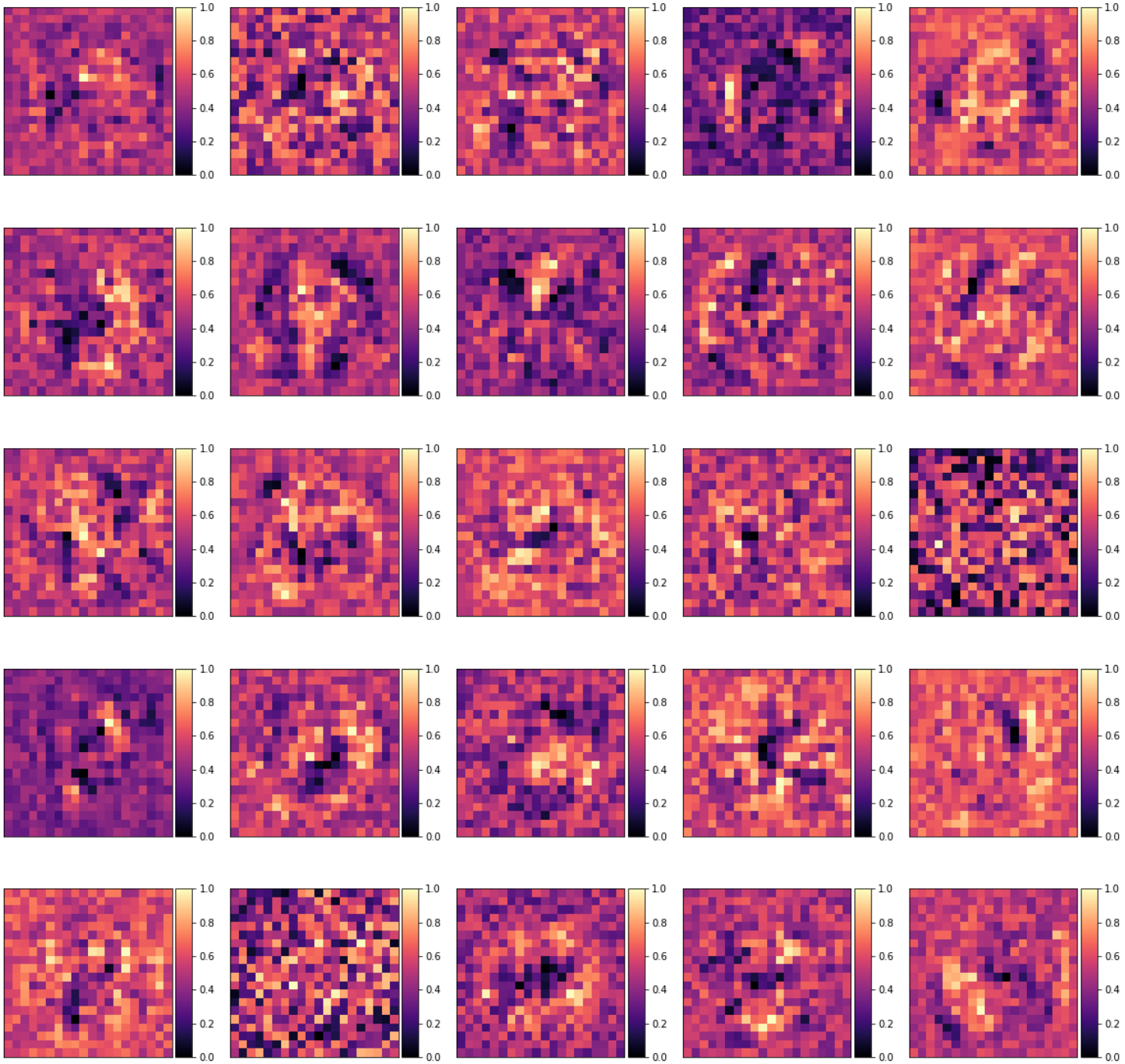


Figura 5: contribuição (após normalização-"scaling") de cada um dos 400 pixels em cada um dos 25 neurônios da camada escondida. Nesse ponto, a rede neural de topologia [400, 25, 10] acertava aprox. 99% dos dados do treino e aprox. 92% dos dados de teste. Ao continuar o treinamento, a performance da rede sobre os dados de teste começou a diminuir, indício de que treinar mais apenas causaria overfitting e não traria uma melhora real para a rede. A rede foi treinada usando a função de minimização `scipy.optimize.minimize` da linguagem `python`.

IV Conjuntos de treino, validação e teste

Anteriormente, o conjunto de exemplos foi dividido entre treino e teste. Nessa seção, os dados foram divididos entre três conjuntos: treino, validação e teste. O conjunto de validação é usado para avaliar a performance da rede neural ao variar hiperparâmetros, como por exemplo, a constante de regularização, λ. Após determinar qual o melhor hiperparâmetro, o conjunto de validação é incorporado ao conjunto de treino e a rede é otimizada novamente. Por fim, a rede otimizada tem a performance avaliada pelo conjunto de teste.

IV.i Curva de aprendizado

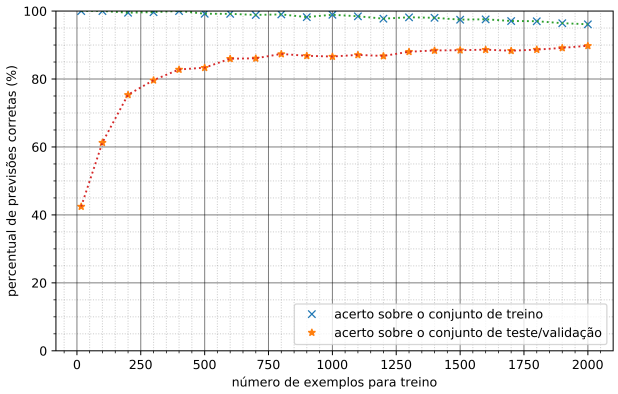


Figura 6: percentual de previsões corretas (para o conjunto de treino e para o conjunto de teste/validação) em função do número de elementos no conjunto de treinamento.

A curva de aprendizado apresentada na Figura (6) representa o percentual de acertos para os conjuntos de treino e teste em função do número de exemplos usados no treino, m. Todos os pontos do gráfico foram coletados com 50 iterações e $\lambda = 0.001$.

Para m pequeno, a rede neural apresenta traços claros de overfitting: praticamente 100% de acerto no conjunto de treino e cerca de 42% de acerto no conjunto de teste.

Com o aumento do número de exemplos, a taxa de acerto sobre o conjunto de treino cai, e a rede aprende a generalizar os exemplos de treino, o que contribui para a melhora da performance no conjunto de teste. Com a topologia da rede usada nesse trabalho, [400, 25, 10], vimos que é possível obter ≈92% de acerto sobre o conjunto de teste- Figura (5). Ao continuar o treinamento a partir desse ponto, entramos em uma região de overfitting e a taxa de previsões corretas sobre o conjunto de teste começa a diminuir.

Portanto, podemos dizer que quando temos poucos exemplos de treinamento, a variância da rede é alta, pois existem muitos parâmetros (pesos e bias) para uma quantidade relativamente pequena de exemplos para treino, o que gera o overfitting.

Ao aumentar a quantidade de exemplos de treino, percebemos que as curvas de acerto sobre os conjuntos de treino e de teste sugerem uma convergência entre aprox. 90% e 95%. Portanto, esse é um possível caso de underfitting, ou seja, alto viés. Isso significa que apenas aumentar a quantidade de exemplos para treinamento não vai melhorar a performance da rede neural para além de 95%. Será necessário aprimorar o modelo da rede neural para alcançar uma taxa de acertos superior. O aprimoramento pode ser feito, por exemplo, através do uso de mais neurônios na camada escondida, ou através do uso de mais camadas (ou os dois). Entretanto, como descrito na próxima seção, uma rede neural de topologia [400, 25, 10] não foi capaz de obter uma performance superior a 93% de acertos no conjunto de teste. Ou seja, para melhorar a performance da rede, além de aumentar os graus de liberdade (pesos), é preciso aumentar o número de exemplos.

IV.ii Uso do conjunto de validação para determinar a melhor constante de regularização

Treinamos a rede neural [400, 25, 10] para diferentes λ e para dois tamanhos de conjuntos de treino: 200 e 3000. A principal diferença foi que para o segundo caso, os casos $\lambda \in \{3, 10\}$ não apresentaram diferenças muito significativas dos demais, pois $m = 3000$ no denominador do termo da regularização na função de custo atenuou o efeito dessa penalização.

IV.ii.1 Conjunto de treino com duzentos exemplos

Para determinar a melhor constante de regularização (λ), treinamos a mesma rede neural de topologia [400, 25, 10] com o método do gradiente conjugado, `scipy.optimize.minimize(fun, x0, jac=True, method='CG', options = {'maxiter':20 })`, para $\lambda \in$ {0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, 3, 10}.

O conjunto de treino tinha 200 elementos, e os conjuntos de validação e teste tinham 1000 elementos cada. A taxa percentual de previsões corretas para o conjunto de treino foi acima de 98.5% em todos os casos, exceto para $\lambda = 3$ e $\lambda = 10$, que apresentaram, respectivamente, 14% e 11.5%. Os resultados estão representados nas Figuras (7) e (8).

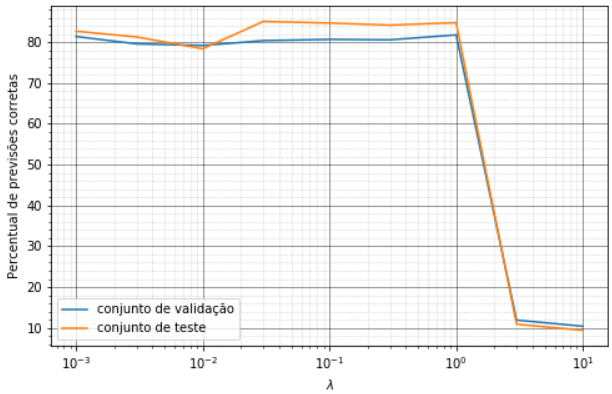


Figura 7: percentual de previsões corretas em função do parâmetro de regularização, $\lambda \in$ {0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, 3, 10}.

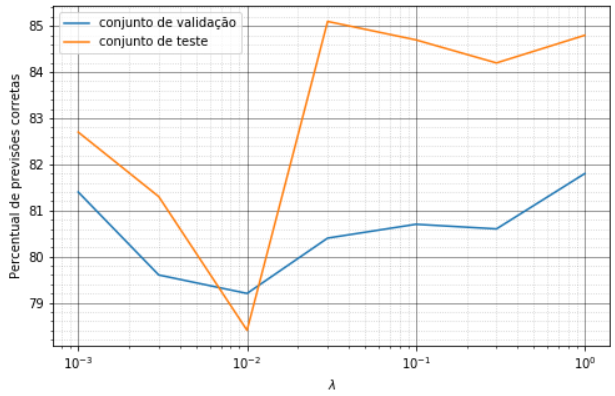


Figura 8: percentual de previsões corretas em função do parâmetro de regularização, $\lambda \in \{0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1\}$.

Nota-se que para $\lambda \in \{3, 10\}$ a regularização penaliza demais o algorítmo, e a taxa de acertos diminui aprox. 10%, que representa a mesma situação ao que obteríamos se sorteássemos ao acaso um dos 10 algarismos (assumindo conjuntos balanceados na distribuição dos algarismos e equiprobabilidade para ocorrência de cada caso).

Mesmo entre $\lambda = 0.001$ e $\lambda = 1$, não é tão assertivo afirmar uma relação entre a performance da rede neural e a escolha de λ, visto que a condição inicial de cada rede não foi a mesma (os pesos são inicializados de forma aleatória). Porém, tendo em vista os conceitos de viés e variância e observando a curva de aprendizado, que considera o percentual de acertos pelo valor de lambda (figuras 7 e 8), temos, como melhor escolha possível para λ, algo no intervalo $[0.001, 1]$, já que a variação entre o percentual de acertos dos conjuntos de treino e teste é pequena, além de estar em uma faixa aceitável de respostas corretas, quando comparado com demais resultados. Pela proximidade entre os percentuais, entendemos que não há overfitting, pois o modelo não está completamente adaptado ao conjunto de treinamento (acerto abaixo de 100%), e que também não há underfitting, uma vez que o número de acertos está relativamente alto, ou seja, a resposta esperada está próxima da obtida e, com isso, a função de custo não assume valores muito distintos para os conjuntos. Para estudar com maior confiabilidade essa região, seria necessário realizar esse mesmo procedimento várias vezes para obter uma média e um desvio padrão de cada ponto dos gráficos. Como o custo computacional seria muito alto e demandaria muito tempo, não foi possível trazer esses resultados para esse trabalho.

Entretanto, é possível dizer que, ao menos para essa topologia de rede neural, a regularização não é um parâmetro muito relevante, e pode ser mais vantajoso não regularizar a função de custo para economizar operações e tempo de execução.

A escolha do tamanho do conjunto de treino (200 elementos), que é relativamente pequeno em comparação com outras redes treinadas nesse trabalho, foi justificada porque a regularização visa evitar o overfitting. Se o conjunto de treino é suficiente grande, a chance de ocorrer overfitting é baixa, assim como a necessidade de regularizar a função de custo. Inclusive, isso é expresso no termo de regularização na equação de custo, $\frac{1}{2m}\sum_{i=0}^{n}\theta_i^2$; quanto maior o número de exemplos (m), menor o peso da regularização na função de custo. Por isso, escolhemos um conjunto de treino relativamente pequeno na expectativa de observar a contribuição de λ no treinamento da rede neural.

IV.ii.2 Conjunto de treino com três mil exemplos

O mesmo processo para determinar a melhor constante de regularização λ foi executado, porém dessa vez com três mil exemplos de treino, mil de validação e mil de teste, seguindo a proporção de 60% para 20% para 20%.

Novamente, treinamos a mesma rede neural de topologia [400, 25, 10] com o método do gradiente conjugado, `scipy.optimize.minimize(fun, x0, jac=True, method='CG', options = {'maxiter':10 })`, para $\lambda \in \{0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, 3, 10\}$.

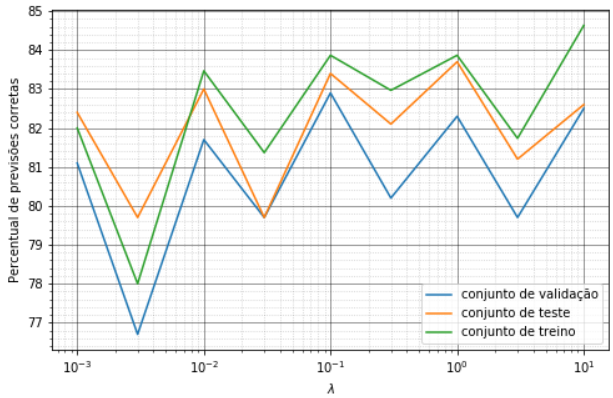


Figura 9: percentual de previsões corretas em função do parâmetro de regularização para os conjuntos de treino, validação e teste seguindo a proporção {60, 20, 20}%, de um total de cinco mil exemplos. $\lambda \in \{0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, 3, 10\}$.

Entretanto, a partir dos resultados obtidos, não é possível concluir que há um λ que otimiza melhor a rede neural, devido aos desvios estatísticos de natureza aleatória (como aleatoriedade e finitude dos conjuntos de exemplos, diferenças nas condições iniciais da minimização para cada λ, etc.)

V Estatística da evolução

Vamos ilustrar, através de uma rede neural de topologia [400, 25, 25, 10], a evolução do treinamento ao longo do tempo/número de iterações (minimização feita pelo método BFGS). Na Figura (10), podemos observar a evolução de cada algarismo, além da taxa de previsões corretas dos conjuntos de treino e de teste. Também é possível observar o momento em que a rede entra em regime de overfitting, quando a curva de teste atinge o ponto máximo. Não foi possível ver o decréscimo na taxa de acerto sobre o conjunto de teste pois não foram feitas iterações o suficiente. Esse efeito seria mais facilmente (e mais artificialmente) observado usando-se um conjunto de treino pequeno.

A rede neural foi treinada com 4000 exemplos e o conjunto de teste possuia 1000 exemplos, seguindo a proporção 80% para 20%.

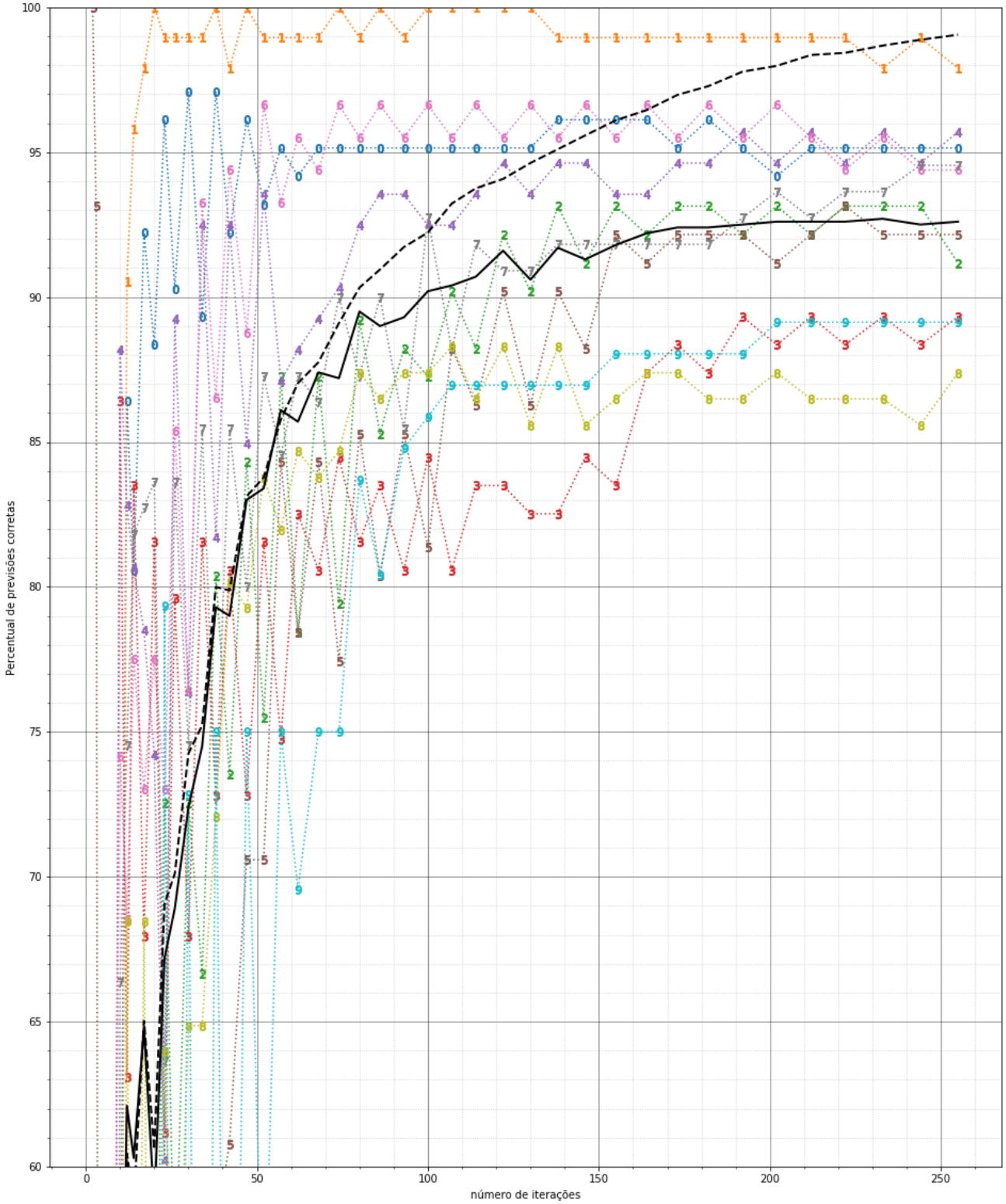


Figura 10: taxa de previsões corretas para cada algarismo (do conjunto de teste), para o conjunto de treino (linha preta tracejada) e para o conjunto de teste (linha preta contínua).

VI Conclusão

Começando pela teoria, o método de "backpropagation", para o cálculo de derivadas parciais dos pesos das redes neurais foi fundamental para a viabilidade desse tipo de algoritmo. Apesar de, numericamente, o método da secante apresentar resultados muito próximos, o fato dos valores dos neurônios da função de custo serem reaproveitados para calcular as derivadas economiza muito tempo de processamento.

A regularização, de fato, auxiliou o método a encontrar uma melhor solução, no entanto, não foi tão efetiva e aumentou, consideravelmente, o custo computacional.

Sobre a parte prática, vimos que antes de aplicar a rede neural a situações reais, o conjunto de teste pode ser incorporado ou não ao conjunto de treino com o intuito de ter uma solução mais generalizada. Entretanto, o conjunto de teste é uma medida de quando a rede neural começa a entrar em regime de overfitting (basta observar o gráfico tal). Ao avaliar a taxa de previsões corretas sobre o conjunto de teste a cada iteração da minimização, ela indica a partir de qual momento devemos parar de treinar por excesso de especialização no conjunto de treino. Se o conjunto de teste for reincorporado ao final do modelo, esse controle é perdido.

Para redes neurais, o monitoramento da função de custo sobre o conjunto de teste ao longo do treinamento é uma métrica boa para controlar o overfitting.

Por fim, observamos que a parte mais crucial para um bom desempenho de redes neurais é a quantidade de exemplos. A constante de regularização λ surtiu pouco ou nenhum efeito sobre a rede, assim como a introdução de mais uma camada escondida de 25 neurônios. É claro que ainda existe uma infinidade de configurações e topologias, mas o resultado mais claro desse trabalho veio da curva de aprendizado, que mostra como a performance da rede neural sobre o conjunto de teste melhora com o aumento do número de exemplos.

Referências

1. João Batista Florindo, Aulas de MS960, segundo semestre de 2020, UNICAMP.
2. T. Hastie, R. Tibshirani and J. H. Friedman, The Elements of Statistical Learning; Data Mining, Inference, and Prediction. Springer, 2009.
3. S. Shalev-Shwartz and S. Ben-David, Understanding Machine Learning: From Theory to Algorithms. Cambridge University Press, 2014.
4. C. Bishop, Pattern Recognition and Machine Learning. Springer, 2006.