

Trabalho Prático 2: Deep Learning

José Silva (2020000000) *zesilva2309@dcc.ufmg.br*

Maria Costa (2020000001) *mariac453964@hotmail.com*

Nome Sobrenome (Número de Matrícula) *email@host*

1 Convolutional Neural Networks

O código fonte fornecido¹ implementa um *CNN* em C++. Ele consiste em um programa funcional, capaz de ler a base de dados do *MNist*, e o classificar usando a arquitetura especificada.

Sugere-se que, antes de iniciar suas alterações sobre o código, você certifique que você consegue compilar e executar-lo com os parâmetros colocados abaixo.

O modelo base para *Convolutional Neural Networks* que será utilizado é o proposto em [?] por LeCun, expressa pela seguinte arquitetura:

- Uma camada de entrada
- Duas camadas convolucionais
 - Ambas as camadas usam *pooling*, implementado usando um *step* de tamanho 2.
 - A primeira camada possui 13 *feature maps*, a segunda 5.
 - Ambas as camadas usam *kerneis* quadrados de lado 5.
- Uma camada escondida de 100 unidades completamente conectadas
- Uma camada de saída, com um neurônio para cada classe (10, no caso)

O formato de chamada do programa de exemplo (*main.cpp*) para classificar a base do *MNist* é genericamente:

```
./CNN [convolution layer count N] N*[Feature map counts]  
      N*[Kernel sizes] N*[Step sizes] [hidden layer count M]  
      M*[unit counts] [epochs]
```

Onde:

- *Feature map counts*, *Kernel sizes* e *Step sizes* são parâmetros específicos a cada camada convolucional, que especificam a arquitetura do *CNN* (consulte as referências para compreender seus significados específicos).
- *unit counts* é o número de unidades em cada camada escondida
- *epochs* é o número de vezes que o modelo passará completamente pelos dados de entrada (refinando seus pesos a cada iteração).

¹Agradece-se a Ishtiaq Khan, e originalmente a Mike Oneill pelo código original

Em particular, para treinar o modelo de LeCun sobre os dados de entrada 10 vezes (épocas) tem-se:

```
./CNN 2 13 5 5 5 2 2 1 100 10
```

Note que podemos testar um *MLP* usando a mesma implementação, com a chamada:

```
./CNN 0 1 100 10
```

1.1 Questões teóricas

1. Explique o que são e para que servem cada um dos seguintes conceitos, no contexto de *CNNs*: (1) Feature Map, (2) Kernel, (3) Convolution, (4) Pooling, (5) O step-size usado na implementação
2. *CNNs* tipicamente possuem 4 tipos de camadas: Uma camada de entrada, uma ou mais camadas de convolução, uma ou mais “camadas escondidas”, e uma camada de saída. Explique a função que cada qual possui, e associe isto com seu padrão de conectividade.
3. O termo *kernel* em *CNNs* refere ao conjunto de pesos aplicados nas entradas dos *feature maps*. Este termo também aparece em outros modelos, como, por exemplo, o *SVM*. Neste caso particular, de que forma estes dois conceitos são similares? Qual é a significância do *kernel* na classificação de um *CNN*?

1.2 Experimento: Deslocamento

Nesse experimento, pede-se o seguinte:

1. Crie um novo *CNN* que aceite como entrada os dados do *MNist*, porém que tenha uma primeira camada convolucional com um único *Feature Map* de largura 25 (ou seja, com um kernel de largura 4).
2. Sem precisar treinar, aplique uma entrada qualquer do conjunto de treino na rede, e extraia o valor atingido pelos neurônios na camada convolucionária (definida no primeiro item). Plote este resultado em um mapa de calor e o insira aqui.
3. Altere a entrada de tal forma que todos os pixels são deslocados para a direita por 5 posições, e repita o procedimento anterior, inclusive o plot, porém agora na nova entrada.

Assumindo que você obteve êxito, você terá observado que as ativações na camada convolucional também se moveram. Com base nisso responda:

1. Por que eles se moveram desta forma? Ao desenvolver sua resposta, considere o fato do mesmo *kernel* ser aplicado no conjunto inteiro de pixels (ainda que de forma segmentada).
2. Qual a justificativa computacional para se usar um mesmo *kernel* em uma região extensa?
3. Porque isto é útil em um classificador de dados visuais? Considere em sua resposta o resultado do experimento.

4. Ao fixar um *kernel* único para uma região extensa, usa-se um conhecimento a priori dos dados classificados (visuais). Isto vai contra o agnosticismo quanto as entradas que *deep learning* professa. Há, no entanto, uma forma de se construir um conjunto de entradas sintéticas usando os dados já conhecidos que permitam que se treine uma nova rede, sem um *kernel*, de tal forma que ele fique com conjuntos de pesos quase identicos (os quais antes adviriam de um mesmo componente do *kernel*). Como poderia-se gerar estes dados sintéticos para se obter este resultado? Ao definir seu método, desconsidere a sua praticidade em termos de tempo de execução.

1.3 Experimento: Aprendendo a aprender

Um dos princípios de *deep learning* é o chamado *learning to learn*. Este princípio envolve várias ideias, uma das quais é a de que, aprendendo um conjunto de dados, outros conjuntos de dados, a princípio pouco relacionados, se tornam mais fáceis de se aprender. Exemplificando isto, imagina-se que, tendo aprendido o que é uma moto e o que é um ônibus, um bom algoritmo de aprendizado conseguiria aprender o que é um carro com mais facilidade, pois ele conseguiria relacionar algumas partes do que ele observou (rodas, janelas, portas, etc) com a imagem atual, e assim conseguir discernir-lo mais facilmente.

Nesta tarefa, pede-se o seguinte:

1. Divida seus dados de treino em dois conjuntos, um contendo os dígitos 2, 3 e 7, e outro com o restante.
2. Treine o modelo LeCun usando somente o conjunto de 7 dígitos.
3. Desative a propagação retrógrada nas duas camadas convolucionais e na camada de entrada (sobrando somente as camadas escondidas e a de saída). Note que será necessário para isto uma alteração no código.
4. Treine agora sobre o conjunto contendo somente os dígitos 2, 3 e 7, e plote o grafo acurácia no treino x época. Insira aqui o plot obtido, tal como a acurácia no teste (note que o teste não foi filtrado, e deve conter todos os dígitos)
5. Finalmente, crie uma nova rede aleatória e repita somente os passos 3 e 4. Note que agora você estará aprendendo a reconhecer os 3 dígitos em uma rede com camadas convolucionais aleatórias).

Ao gerar os gráficos e ao medir a acurácia no teste, os valores obtidos devem ser a média de pelo menos 8 execuções, e estar juntas ao seu desvio padrão.

Assumindo que se obteve êxito, seus resultados devem mostrar que a rede que teve o pré-treino obteve uma performance melhor do que a outra que não. Responda as seguintes perguntas:

1. Compare o experimento com o pré-treino com aquele que não o teve no quesito da velocidade de aprendizado das novas classes.
2. Assumindo um número de épocas infinitas, os modelos convergirão para o mesmo valor? Porque? Sugere-se a execução da segunda parte deste experimento por um número grande de épocas (mais de 30) para reforçar sua hipótese.
3. Você concorda que isto ilustra o princípio *learning to learn*? Se sim, justifique, caso contrário, justifique também, e, se conseguir, proponha uma outra metodologia que poderia testar esta propriedade neste modelo.

4. O que aconteceu com a acurácia das classes do primeiro conjunto (7 dígitos) após o segundo treino (3 dígitos)? Porque isto ocorre? Que procedimento deveria ser usado para treinar uma nova classe dentro deste modelo?

1.4 Experimento: *CNN* vs *MLP*

Neste experimento propõe-se comparar a acurácia do *CNN* com o *MLP* (lembre-se que o modelo fornecido pode parametrizado como um *MLP*).

Utilizando os parâmetros fornecidos na introdução dessa seção, mudando o número de épocas de 10 para pelo menos 30, classifique os dígitos da base. Plote a acurácia no treino x épocas e seu desvio padrão para ambos os casos, e reporte as acurácias médias e seu desvio padrão obtidas no conjunto de teste. Repita cada execução pelo menos 8 vezes.

A diferença encontrada é significativa? Para ajudar a responder esta questão, efetue o teste de McNemar, preenchendo a seguinte tabela:

	<i>MLP</i> Errou	<i>MLP</i> Acertou
<i>CNN</i> Errou	a	b
<i>CNN</i> Acertou	c	d

Lembre-se também que uma mudança, por exemplo, entre 99.0% e 99.5% de acurácia é bastante significativa, pois representa uma redução em 50% nos erros.

2 Stacked Denoising Auto-Encoders

Existem várias variantes de *auto-encoders*. Neste trabalho, adotamos o modelo que toma o *auto-encoder* como sendo um *MLP* onde as entradas e as saídas são as mesmas, porém sem *Weight Mirroring* (isto é, os pesos que entram na camada escondida não são necessariamente iguais aos que saem). Pode-se facilmente gerar um *auto-encoder* simples usando o código disponibilizado, a título de exemplo:

```
class Autoencoder{
public:
    CCNN * CNN;
    Autoencoder(int ioSize, int hiddenSize) : CNN(NULL) {
        size_t sqrtIoSize = (int)sqrt((double)ioSize);
        int normIoSize = sqrtIoSize*sqrtIoSize;
        CNN = new CCNN(0, 1, NULL, NULL, NULL, &hiddenSize,
                      (size_t)normIoSize, sqrtIoSize);
    }
    ~Autoencoder(void){
        delete CNN;
    }
    void Calculate(double * input, double * output){
        CNN->Calculate(input, output);
    }
    void BackPropagate(double * io, double alpha){
        CNN->BackPropagate(io, alpha);
    }
};
```

Note que esta implementação possui a peculiaridade de necessitar que `iosize` seja um quadrado de inteiros.

Seguido as orientações colocadas nas referencias sobre *SdA*, e com base no trecho de código acima, pede-se que se implemente um *Stacked Denoising Auto-Encoder*. Um esboço do procedimento para se construir o *SdA* se segue:

1. Implemente o *auto-encoder* (veja o exemplo acima)
2. Implemente o *denoising auto-encoder*, aplicando um filtro que torna alguma fração aleatória f de cada entrada em zeros.
3. Implemente o sistema de pré-treino não supervisionado do *SdA*:
 - (a) Cada *denoising auto-encoder* é treinado sequencialmente
 - (b) Concluindo o treino de um dos níveis, passe todas as saídas do nível anterior (no caso da primeira camada, seriam as entradas) pelo nível, e armazena-se os valores dos componentes da camada escondida para cada caso; Este será o input da próxima camada.
4. Observando que a camada escondida de cada *denoising auto-encoder* é a entrada da próxima, implemente o sistema de treino supervisionado do *SdA*, criando uma nova rede composta somente pelos neurônios das camadas do meio, mudando a fonte (mas mantendo os pesos) das conexões que antes iam da entrada do *denoising auto-encoder* para a camada escondida do *dA* anterior. Na implementação fornecida, o produto final será um MLP com um número de camadas escondidas iguais ao número de *dAs* utilizados.

Na sua implementação, deve ser possível especificar a altura da pilha de *auto-encoders* e o número de componentes da camada do meio em cada caso (aceita-se que se tenha peculiaridades semelhantes ao exemplo colocado, desde que estes são devidamente informados e verificados por condicionais no código).

2.1 Questões teóricas

1. Qual a função de se treinar uma rede onde a camada de saída é igual a camada de entrada? De que forma isto poderia ser útil?
2. Porque que, tipicamente, usa-se menos neurônios na camada escondida do que na camada de entrada ou saída? O que se arrisca quando usa-se neurônios insuficientes? E quando se usa neurônios em excesso?
3. Qual a diferença funcional entre usar um *auto-encoder* com poucos neurônios na camada escondida e usar uma pilha de *auto-encoders* que perfazem a mesma redução, porém de forma gradual?
4. O que objetiva-se obter com a introdução de barulho na entrada do *auto-encoder* (criando com isso o *denoising auto-encoder*)?
5. Qual a utilidade da etapa final do procedimento de se criar um *SdA*, onde se interconecta as camadas escondidas formando um *MLP* enorme?

2.2 Experimento: Valores latentes

Uma das premissas utilizadas em muitos modelos de *deep learning* é a idéia de que a complexidade dos dados classificados advém da combinação de um número relativamente pequeno de features escondidas (denominados *valores latentes*). *Auto-encoders* ilustram isto nitidamente, ao passo que, para ter êxito, eles dependem da possibilidade de se representar sua entrada por meio de uma camada escondida usualmente menor que o tamanho dos dados sendo representados.

Neste experimento, pede-se que você treine um auto-encoder por 10 épocas sobre as primeiras 1000 entradas da base de treino do *MNist*, e então, após isto, que você valide seus resultados nas primeiras 1000 entradas da base de teste do *MNist*, usando como métrica o erro quadrático médio entre a saída e a entrada.

O experimento deverá primeiro ser conduzido no *auto-encoder*, e depois no *denoising auto-encoder*, devendo-se, em ambos os casos, variar o número de neurônios na camada escondida para valores na faixa $[0, 6||inputs||]$. Como de costume, os valores obtidos devem ser a média de pelo menos 8 execuções, e vir juntas de seus valores de desvio padrão.

Com base nos resultados observados, responda:

1. Como você caracterizaria o efeito sobre a acurácia de se aumentar e reduzir o número de neurônios na camada escondida?
2. Os problemas colocados na sua resposta da 2ª questão teórica apareceram? Caso contrário, hipotetize porque.
3. Em uma das referências recomendadas constrói-se um *SdA* para reconhecer a mesma base de dados que se usa aqui. Nele, utiliza-se um *denoising auto-encoder* com 500 neurônios na camada escondida para o primeiro nível na pilha de *auto-encoders*. Esse tamanho da camada escondida é justificável? Fortaleça sua resposta referenciando o resultado do seu experimento.

2.3 Classifique

Classifique o mesmo conjunto de dados utilizados no *CNN*, parametrizando seu modelo tanto no número de camadas escondidas como no número de neurônios neles. Quais foram as suas intuições iniciais, e quais resultados foram obtidos com elas? Qual caminho/procedimento conduziu aos parâmetros finais utilizados?

Inclua a média e desvio padrão dos valores de acurácia atingidos no conjunto de teste no final do experimento após pelo menos 8 execuções.

Compare este resultado com o resultado do *CNN*. Qual dos dois foi melhor? Hipotetize sobre as causas da relação observada.

Note que, por serem estruturas completamente conectadas, a execução dos *auto-encoders* poderá demorar bastante (por exemplo, usando os 500 neurônios na primeira camada, conforme sugerido na referência, tem-se um total de $2 * 500 * 784 = 784000$ conexões somente na primeira camada do *SdA*!).

3 Tarefas opcionais

Os seguintes itens são tarefas opcionais, que **podem conferir até 5 pontos extras**². A conclusão parcial dos mesmos será considerada parcialmente, e seu valor será atribuído con-

²Colocou-se na aula de apresentação do trabalho que estas atividades somente contariam contra eventuais erros no enunciado. Esta posição foi revisada.

siderando o quão interessante é, o esforço dedicado, e o impacto em performance (tempo e acurácia). No relatório, coloque e explique os itens que você fez como subseções desta seção, junto com uma explicação sobre os mesmos (além, é claro, de colocar seus resultados nos locais apropriados da documentação, se sua mudança mudar o resultado de algum experimento/métrica).

1. Uso do *validation set*, o qual deve ser implementado como critério de parada em ambos os modelos, e ser apresentado em todo lugar onde a acurácia no treino é pedido (junto a ele, não em substituição ao mesmo).
2. Uso de *k-fold cross validation*; O resultado da avaliação deve aparecer nos locais onde pede-se o erro no teste (porém, novamente, não em substituição ao mesmo). Este item somente não foi obrigatório por ser um dos focos do TP1, e pela base comparativa para o *MNist* ter sido classicamente o *test-set*.
3. Pré-processamento das entradas, por exemplo, por detecção de bordas. As imagens utilizadas aqui são os dados brutos dos pixels. Não é necessário melhorar a performance do modelo utilizando estas técnicas (ainda que é incentivado); Reporte a acurácia com e sem o seu pré-processamento.
4. Reconhecimento de outros data-sets de dígitos e/ou caracteres, em adição ao *MNist*. Por exemplo, <http://www.ee.surrey.ac.uk/CVSSP/demos/chars74k/> possui um dataset com caracteres alfa-numéricos em *.bmp*. A base possui caracteres tipografados, manuscritos, e de placas.
5. Otimização significativa do código fonte em qualquer um dos modelos (considere significativo como sendo pelo menos 30% mais rápido usando -O8 no g++).
6. Alteração no código/arquitetura do *CNN* que obtenha uma performance média de pelo menos 98.3% de acurácia no teste, após 30 épocas. Reporte, além da média de 8 execuções, o valor do desvio padrão.
7. Uso de outras métricas além da acurácia na hora de analisar e comparar os algoritmos.
8. Uma análise sobre quais dígitos que os classificadores estão errando (deve-se minimamente usar uma matriz de confusão).
9. Se, além destas tarefas opcionais você pensar em alguma outra (sejam métricas, experimentos, otimizações, etc), coloque-as aqui, junto a um argumento convincente de porque sua proposta é relevante. Para se assegurar que sua proposta é válida, sugere-se que você a discuta com o monitor.

4 Questionário pós trabalho

Esta seção é opcional, e será lida somente após a avaliação do restante do trabalho (visando não enviesar a avaliação). O objetivo aqui é principalmente pedagógico: Objetiva-se entender os sucessos, dificuldades e opiniões sobre o trabalho para que eventuais trabalhos futuros possam ser mais eficazes.

Vale lembrar que não há nada a se lucrar por exagerar esforços: As respostas aqui não influenciarão o próximo (último) trabalho da disciplina, ao passo que o feedback obtido aqui será visto só no fim do semestre, na data de entrega.

1. Qual foi o grau de dificuldade do trabalho? Como ele se compara com os trabalhos de outras disciplinas?
2. Quanto tempo seu foi gasto entre pesquisar, debater com os colegas, implementar e testar o trabalho, e rodar os experimentos? Se possível, divida sua análise nestes e em outros critérios que achar relevantes.
3. Quão úteis foram as referências fornecidas? Houve alguma outra referência que você achou mais explicativa?
4. A estrutura do trabalho permitiu que todos do grupo fizessem o trabalho? Os conhecimentos de uma parte foram bem repartidas entre os integrantes? Como o trabalho poderia ser estruturado para melhor garantir isto?
5. Houve uma correspondência entre as dificuldades reais e aparentes de cada item?
6. Adotou-se um formato diferente para este trabalho em relação a documentação. Como você avalia este formato?
7. Qual fração dos conhecimentos você diria que vieram da resposta as questões teóricas, da execução dos experimentos, e da implementação do *SdA*? Quais destes conhecimentos você julga ainda restarão com você daqui a um ano?
8. Havendo mais alguma colocação, sinta-se a vontade de o fazer-lo.

Referências

- [1] Fonte da base de dados MNist, com uma relação da performance de diferentes classificadores.
<http://yann.lecun.com/exdb/mnist/>
- [2] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-Based Learning Applied to Document Recognition” Proceedings of the IEEE, vol. 86, no. 11, pp. 2278-2324, Nov. 1998.