Universidade Federal de Minas Gerais
Departamento de Ciência da Computação
Disciplina de Computação Natural

Trabalho Prático 3: Redes Neurais Artificiais

Data de entrega: 04/12/2017

O objetivo deste trabalho é colocar em prática conceitos relacionados a redes neurais. Iremos implementar uma rede capaz de resolver um problema de classificação. O foco do trabalho está na parte de parametrização do rede e não na implementação dos algoritmos em si, e por isso o uso de uma biblioteca que implemente redes neurais é primordial.

Diversas bibliotecas oferecem suporte para implementação de redes neurais. Recomendamos duas delas: Keras com TensorFlow[1] e PyTorch[2] .Caso você queria utilizar uma outra biblioteca, nos consulte. Ferramentas como o Weka ou Orange não devem ser utilizadas.

O problema de classificação a ser tratado pela rede neural é o de identificar em que parte da célula determinada proteína pode ser encontrada, e a base de dados está disponível no Moodle. A base tem 1429 proteínas, descritas por 8 atributos. A proteínas descritas podem ser encontradas no citoplasma (CYT), mitocôndria (MIT), em diferentes partes da membrana da própria proteína (ME1, ME2, ME3) ou fora da célula (EXC), em um total de 6 classes possíveis. Uma das características desse problema é o desbalanceamento das classes (enquanto a maior classe inclui 463 proteínas a menor inclui apenas 35), que faz com que a tarefa de classificação seja mais difícil. No geral, a taxa de acerto de classificadores nessa base de dados fica por volta dos 60%.

1. O primeiro passo após baixar a base de dados é verificar se serão necessárias transformações nos dados de entrada.
2. Escolha o tipo da rede neural que você irá utilizar (Perceptron, RBF, etc)
3. Escolhida a rede, deve-se pensar em sua arquitetura. É importante definir o que os nós de entrada e os nós de saída representarão, e quantos neurônios serão utilizados nessas duas camadas. Se a rede tiver uma camada escondida, temos também que escolher seu número de neurônios. A definição da função de ativação também é importante nessa fase.
4. Definida a arquitetura da rede, deve-se inicializar os pesos da rede e definir o valor inicial da taxa de aprendizagem e o número de épocas ou outro critério de parada.
5. A função de custo utilizada pelo algoritmo de aprendizado (e.g. back-propagation) também é um parâmetro importante da rede e afeta diretamente a velocidade de aprendizagem. Escolha com cuidado.
6. O treinamento pode ser feito utilizando um exemplo por vez (stocastic gradient descent) ou uma abordagem de mini-batches. Nessa abordagem, os pesos da rede são atualizados depois que uma quantidade *b* de exemplos é visto pela rede, onde *b* é o tamanho do mini-batch (*b* normalmente assume valores que variam de 2 a 100).
7. Os dados utilizados devem ser divididos em duas ou três partes, dependendo do critério de parada selecionado. Quando os dados são divididos em duas partes, a

[1] `https://keras.io/backend/`
[2] `http://pytorch.org/`

primeira (que normalmente corresponde a 70% do total) é utilizada para treinar a rede por um número máximo de épocas. Ao fim do treinamento, a segunda parte é utilizada para medir a capacidade de generalização da rede.
Já quando os dados são divididos em 3 partes, a primeira parte é novamente utilizada para treinar a rede. A segunda é utilizada para validar a rede, isto é, para medir seu erro. Nesse caso, o treinamento pode parar quando a rede atingir um erro mínimo nesse segundo conjunto de dados, chamado conjunto de validação. A terceira parte dos dados é utilizada apenas ao fim do treinamento, para medir a capacidade de generalização da rede.
Independente da abordagem utilizada, o ideal é que um procedimento de validação cruzada seja utilizado para garantir que os resultados de generalização não estão sendo obtidos ao acaso. Uma validação cruzada de 3 partições é suficiente para esse problema.

Guia de experimentação

1. O que acontece quando se aumenta o número de neurônios da camada escondida da rede? Isso afeta o número de épocas necessárias para convergência?
2. O que acontece quando se aumenta o número de camadas escondidas? O ganho no erro é grande o suficiente para justificar a adição de uma nova camada?
3. Qual o impacto da variação da taxa de aprendizagem na convergência da rede? O que acontece se esse parâmetro for ajustado automaticamente ao longo das diferentes épocas?
4. Compare o treinamento da rede com *gradient descent* estocástico com o mini-batch. A diferença em erro de treinamento versus tempo computacional indica que qual deles deve ser utilizado?
5. Qual a diferença do erro encontrado pela rede no conjunto de treinamento ou validação em relação ao erro encontrado no teste? Existe *overfitting*? Como ele pode ser evitado?
6. A base com que você trabalhou é bem desbalanceada. Você pode tentar contornar esse problema usando uma técnica de *oversampling*, ou seja, fazendo cópias dos exemplos das classes minoritárias para tornar a base mais balanceada. Por exemplo, a classe EXC tem 35 exemplos, e a classe MIT 244. De forma simples, você poderia fazer 6 cópias de cada exemplo da classe EXC, aumentando o número de exemplos dessa classe para 210, e utilizando todos eles no treinamento da rede. Fazendo um oversampling das classes minoritárias e retreinando a rede com os melhore parâmetros encontrados, o erro diminiu? Por quê?

<u>O que deve ser entregue:</u>

Um zip com:
1. Código.
2. Relatório: O relatório deve apresentar a arquitetura da rede utilizada, e gráficos que mostrem como diferentes valores dos parâmetros listados impactam na taxa de erro da rede. Além disso, você deve concluir o relatório listando os parâmetros que obtiveram os melhores resultados ao fim da análise experimental.