Pontifícia Universidade Católica de Minas Gerais
Curso de Especialização – Inteligência Artificial e Aprendizado de Máquina
Disciplina: Redes Neurais e Deep Learning
Professor : Zenilton Kleber Gonçalves do Patrocínio Júnior

Questionário / Trabalho em Grupo N.01 (35 pontos)

Instruções:

- **Cada grupo deve produzir suas respostas separadamente**
- **As respostas podem ser manuscritas desde que sejam legíveis**
- **Deve ser entregue um único arquivo PDF com todas das respostas**

1) Conceitue e relacione: aprendizado de máquina, aprendizado de representação e *deep learning*. (20%)

Aprendizado de máquina é um processo de mapeamento estatístico de respostas, baseado em características brutas criadas a mão como qualquer coleta de dados. O aprendizado de representação, abstrai características desses dados antes de realizar o mapeamento para facilitá-lo e o Deep learning por sua vez é uma abordagem que utiliza redes neurais profundas para abstrair sub-características dessas características automaticamente, inferindo ainda mais informações sobre o dado antes de realizar o mapeamento. Exemplificando, para classificar veículos, o aprendizado de máquina pode olhar para características generalistas e dados brutos como peso e tentar relacioná-los, mas sabemos que existem pickups, motocicletas e carros muitas vezes com pesos e comprimentos parecidos dependendo da fabricante, já com a representação, poderíamos extrair a relação peso / comprimento e inferir quantia de eixos, facilitando mais a classificação de um veículo entre moto, carro e caminhão. Por fim o Deep learning infere ainda mais características abstratas como a razão peso ao longo dos eixos para classificar por exemplo um motor traseiro, dianteiro ou motor uniformemente distribuído, facilitando para o mapeamento. Em termos matemáticos, o aprendizado de máquina é uma função linear ou polinomial já o aprendizado de representação e deep learning são na verdade, multiplicações de matrizes encadeadas em que seus valores

são vários pesos e funções.

2)Conceitue função de perda e descreva seu uso na obtenção de um bom modelo. (20%)

A função de perda é um cálculo de proximidade das predições de um modelo e a sua resposta real. Dessa forma escolhendo essa função você muda qual métrica o seu modelo está buscando, por exemplo, escolhendo uma função de perda linear como a média, o seu modelo pode comportar grandes erro em poucas predições enquanto acerta na maioria, mas se você utiliza um função de erro quadrático, o seu modelo vai comportar mais erros menores em todas as predições do que um erro grande em poucas, pois esses erros grandes se tornam quadraticamente punitivos para ele. Portanto é de extrema importância entender que cada função se comporta diferente em cada problema e cabe a escolha correta da função que mais se adequa, para que o seu modelo se comporte como o esperado.

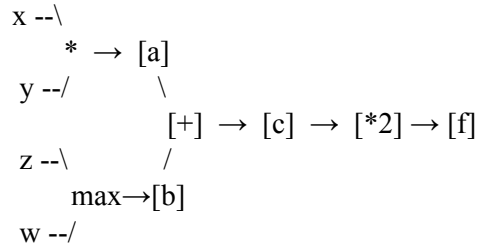
3) Forneça o pseudocódigo (algo em torno de 4~5 linhas) para o algoritmo "*SGD com minibatch*" e descreva sucintamente cada um de seus passos. (20%)

```
for epoch in range(num_epochs):
    np.random.shuffle(data)
    for i in range(0, len(data), batch_size):
        data_batch = data[i:i + batch_size] seleciona um segmento do conjunto de dados para formar um
                     minibatch, minibatchs são bons porquê permitem que o modelo atualize os pesos
                     mais frequentemente, mas com base em apenas uma parte dos dados, o que também
                     ajuda no overfitting
        weights_grad = evaluate_gradient(loss_fun, data_batch, weights) # é passada o batch que será
                       calculado e é realizado o cálculo a taxa instantânea(derivada) ∇L(w) entre a função
                       de perda com relação a cada peso. Geralmente é feito uma média do batch inteiro
                       para ter uma visão melhor do verdadeiro gradiente.
```

weights -= step_size * weights_grad # aqui é apenas atualizado os pesos com o produto do cálculo do gradiente com o tamanho do passo estabelecido, o sinal de menos existe porquê a derivada da a taxa de crescimento e queremos decrescer.

4) Construa o grafo de computação para a função $f(x, y, z, w) = 2 \times [(x \times y) + \max(z, w)]$. Em seguida, realize o *forward pass* considerando a entrada $(3, -4, 2, -1)$ e calcule as derivadas em relação a cada entrada usando *backpropagation*. (20%)

Grafo:



Forward pass e back propagation

a = 3 * (-4) = -12
b = max(2, -1) = 2
c = -12 + 2 = -10
f = 2 * (-10) = -20

Backpropagation:
Para calcular os gradientes em relação a cada entrada, comecamos com:
df/df = 1

Gradiente de f em relação a (c):
df/dc = 2

delta f = 1
delta c = delta f * df/dc = 1 * 2 = 2

Gradiente de c em relação a (a e b):
dc/da = 1
dc/db = 1
delta a = delta c * dc/da = 2 * 1 = 2
delta b = delta c * dc/db = 2 * 1 = 2

Gradiente de a em relação a (x e y):
da/dx = y = -4
da/dy = x = 3
delta x = delta a * da/dx = 2 * (-4) = -8
delta y = delta a * da/dy = 2 * 3 = 6

Gradiente de b em relação a (z e w):
db/dz = 1 (z e maior)
db/dw = 0 (w e menor)
delta z = delta b * db/dz = 2 * 1 = 2
delta w = delta b * db/dw = 2 * 0 = 0

Gradientes:
df/dx = -8
df/dy = 6
df/dz = 2
df/dw = 0

5) Descreva como deve se comportar um “bom” método de inicialização de pesos, abordando os problemas de quebra de simetria e saturação. Fale sobre alguma abordagem adequada para isso. (20%)

Ao iniciar os pesos precisamos primeiro garantir que os pesos não sejam zero, porque qualquer entrada x 0 é zero, então claramente isso não vai dar em nada. Também não podemos deixar que todos sejam iguais, pois cada neurônio oferecerá a mesma informação, então também é óbvio o porquê isso não vai dar muito certo. Além dos óbvios também vimos na matéria os problemas de gradiente como a saturação, onde os pesos podem ser insignificantes e muito próximos de zero ou até zero gerando uma divisão por 0 ou muito grandes explodindo os pesos de divergindo completamente. algumas das técnicas abordadas foram a inicialização de Xavier e HE.

A inicialização de Xavier observa o tamanho das camadas antes e depois de cada peso para que eles não variem de forma a causar os problemas citados acima e tenham de certa forma uma simetria, e por isso essa inicialização se comporta bem para funções de ativação centradas em zero.

A inicialização de HE é uma variação da de Xavier voltada para sua usabilidade na função ReLU, que não é simétrica. Com isso em mente ela observa apenas as camadas anteriores para limitar os pesos, assumindo que as posteriores muitas zeram por causa da natureza da ReLU.