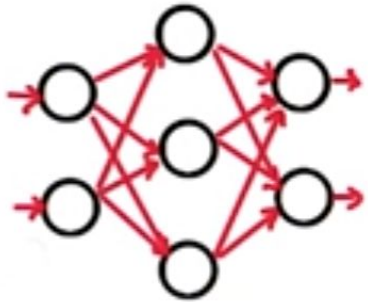
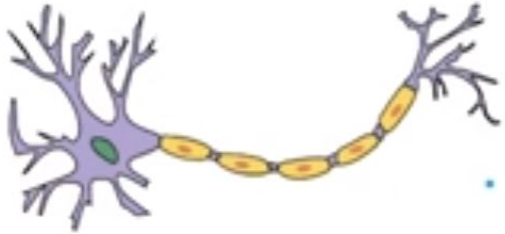
Redes Neurais Artificiais - 2



NEURONS?

HOW THE BRAIN WORKS?



NEUROMORPHIC ENGINEERING?

Revisão

Implementando o gradiente de descida

Sabemos como atualizar pesos: $\Delta w_{ij} = \eta * \delta_j * x_i$

Você aprendeu como implementar isso para uma única atualização, mas como traduzir esse código de modo que ele calcule muitas atualizações de peso de modo que a rede aprenda?

Média Quadrática dos Erros

Iremos fazer uma pequena mudança no modo como calculamos o erro aqui. Em vez de usarmos o SQE, usaremos a **média** dos erros quadrados (MEQ). Agora que estamos usando muitos dados, somar todos os pesos e todos os passos pode criar passos muito grandes, que fazem o gradiente descendente divergir. Para compensar isso, será necessário usar uma taxa de aprendizagem muito pequena. Em vez disso, nós dividiremos pelo número de dados observados, m para tirar a média. Desse modo, não importa quantos dados usemos, as taxas de aprendizado estarão tipicamente no intervalo entre 0,01 e 0,001. Então, podemos usar a MEQ (abaixo) para calcular o gradiente e o resultado da mesma forma que antes, porém, em um valor médio, em vez de uma somatória.

$$E = \frac{1}{2m} \sum_{\mu} (y^{\mu} - \hat{y}^{\mu})^2$$

Algoritmo Geral do Gradiente Descendente

Algoritmo geral para atualizar os pesos com gradiente descendente:

- Defina o passo como zero: $\Delta w_i = 0$
- Para cada observação nos dados de treinamento:
  - Passe adiante na rede, calculando o output $\hat{y} = f(\sum_i w_i x_i)$
  - Calcule o erro para aquela unidade de output, $\delta = (y - \hat{y}) * f'(\sum_i w_i x_i)$
  - Atualize o passo $\Delta w_i = \Delta w_i + \delta x_i$
- Atualize os pesos $w_i = w_i + \eta \Delta w_i / m$ onde η é a taxa de aprendizado e m é o número de observações. Aqui estamos tirando a média dos passos para ajudar a reduzir quaisquer variações nos dados de treinamento
- Repita por e rodadas.

Algoritmo Geral do Gradiente Descendente

Continuação… Algoritmo Geral:

É possível também atualizar os pesos para cada observação ao invés de tirar a média dos passos depois de passar por todas as observações.

Lembre-se que estamos usando a função de ativação sigmóide,

$$f(h) = 1/(1 + e^{-h})$$

E que o gradiente de uma sigmóide é

$$f'(h) = f(h)(1 - f(h)),$$

onde h é o input da unidade de output,

$$h = \sum_i w_i x_i$$

Algoritmo Geral do Gradiente Descendente

Implementando com Numpy

Em grande parte, isso é bem simples de fazer com Numpy. Primeiro, é necessário inicializar os pesos. Queremos que eles sejam tão pequenos quanto o input para uma função sigmóide é na região linear próxima ao 0 e não espremido nas pontas alta e baixa. Também é importante inicializar-los aleatoriamente de modo que todos comecem de lugares diferentes e divirjam, quebrando simetrias. Portanto, inicializamos os pesos de uma distribuição normal com centro em 0. Um bom valor para essa escala é $1/\sqrt{n}$ onde n é o número de unidades de input. Isso mantém o input da sigmóide baixo para números de unidades de input maiores.

```
weights = np.random.normal(scale=1/n_features**.5, size=n_features)
```

Algoritmo Geral do Gradiente Descendente

Cont... Implementando com Numpy

O Numpy tem uma função que calcula o produto escalar de dois vetores, o que convenientemente calcula h para nós. O produto escalar multiplica dois vetores elemento por elemento, o primeiro elemento do vetor 1 é multiplicado pelo primeiro elemento do vetor 2 e assim por diante. Então, cada produto é somado.

```
# input to the output layer
output_in = np.dot(weights, inputs)
```

Por fim, podemos atualizar Δw_i e w_i incrementando-os com `weights += ...` o que é um jeito rápido de escrever `weights = weights + ...`.

Algoritmo Geral do Gradiente Descendente

Por fim, podemos atualizar Δw_i e w_i incrementando-os com `weights += ...` o que é um jeito rápido de escrever `weights = weights + ...`.

Dica de eficiência!

É possível poupar alguns cálculos já que estamos usando uma sigmóide. Na função sigmóide, $f'(h) = f(h)(1-f(h))$.

Isso significa que uma vez calculado $f(h)$, a ativação da unidade de output, você pode usá-la para calcular o gradiente para o gradiente de erro.

Algoritmo Geral do Gradiente Descendente

Tarefa 3 - Exercício de programação

O objetivo é treinar a rede até que ela alcance um mínimo na média de erros quadrados (MEQ) dos dados de treinamento. Você deverá implementar:

- O output da rede: `output`.
- O erro do output: `error`.
- O termo do erro: `error_term`.
- Atualizar o passo: `del_w +=`.
- Atualizar os pesos: `weights +=`.

OBS: Para realizar o exercício, utilize os códigos disponíveis na tarefa 2. O arquivo a ser alterado é o gradient.py.