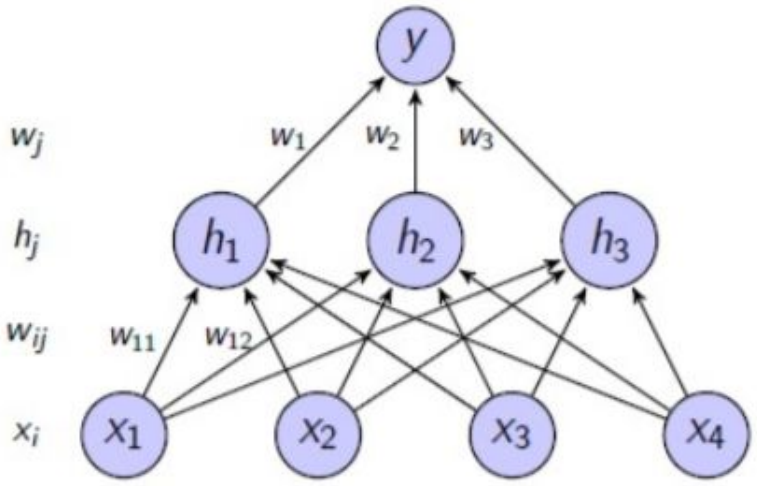
Aluno: Leonam Rezende Soares de Miranda
Matrícula: 2020681492

Artigo: Learning representations by back-propagating errors

Neste artigo é descrito um novo procedimento de aprendizado para redes neurais, o backpropagation, que se tornou um dos componentes principais para construção dos modelos atuais de deep learning.

O objetivo desse artigo era encontrar alguma maneira eficaz de conectar os elementos de uma rede neural arbitrária, de forma a torná-la eficaz para ser utilizada em alguma tarefa específica. Uma tarefa é definida por encontrar um vetor desejável de saída para cada combinação do vetor de entrada. Entretanto, aprender, treinar um modelo, se torna mais difícil quando são introduzidas camadas ocultas no modelo, cujos os estados não são definidos na tarefa.

Uma rede neural de suas camadas (uma sendo oculta) é apresentada na imagem a seguir:



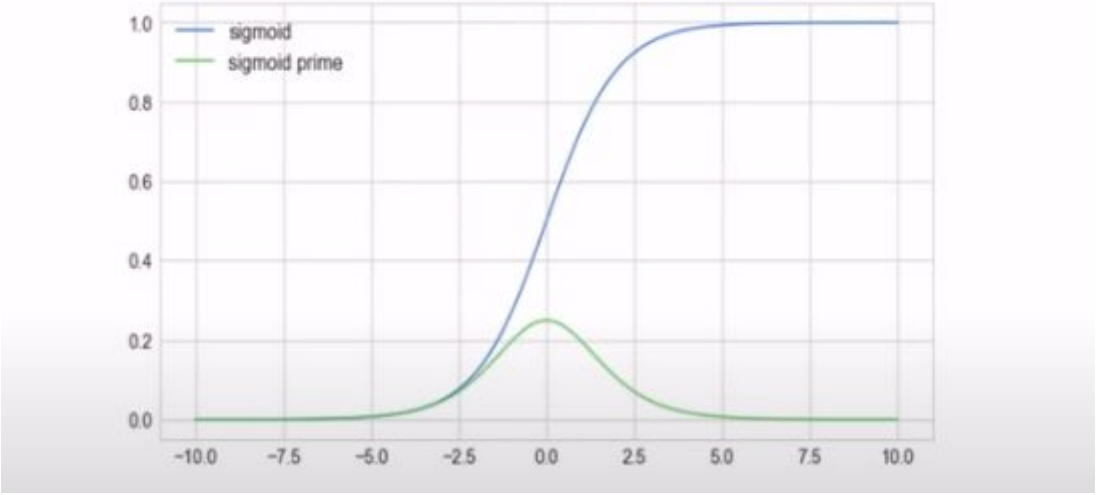
- $f(X) = \sigma(\sum w_j \times h_j) = \sigma(\sum w_j \times \sigma(\sum w_{ij} \times x_i))$

Sendo:

- **X** - matriz de entrada, sendo x_i , o componente i da matriz;
- **W** - matriz de pesos, sendo w_{ij} o peso que associa a componente de entrada *i* ao neurônio *j* da camada oculta e w_j o peso que associa o neurônio *j* da camada oculta a saída ;
- **σ** - função de ativação.

A função de ativação **σ** utilizada pelo artigo é a sigmoid:

$$\sigma(x) = \frac{1}{1+e^{-x}} \qquad \sigma'(x) = \sigma(x)(1-\sigma(x))$$



A função sigmoid é não linear e é facilmente derivável.
Podem ser adicionados vieses aos pesos, adicionando uma entrada extra a cada unidade (neurônio da rede). A saída de cada neurônio é uma função não linear de suas entradas.

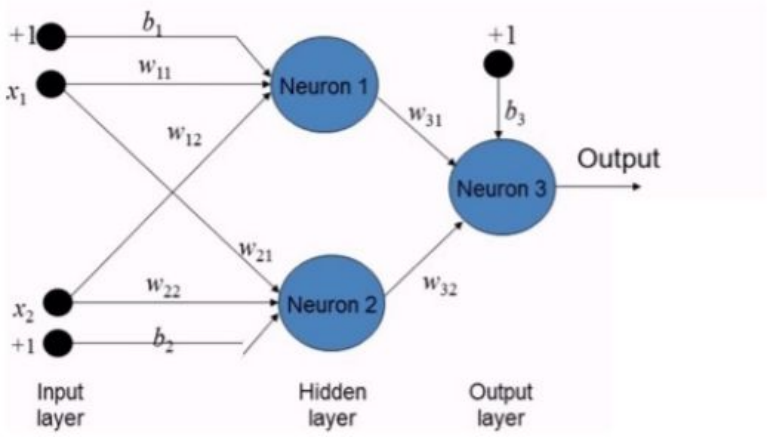
Assim, o objetivo é treinar a rede para encontrar o conjunto de pesos **W** que garanta que para cada vetor de entrada o vetor de saída tenha o mesmo valor (ou suficientemente perto) do vetor de saída desejável. O erro total é utilizado como métrica de performance para avaliar um conjunto de pesos; Assim o erro total é definido como:

$$L(W) = \frac{1}{2}\sum_{i}(\sigma(W^T X^{(i)}) - y^{(i)})^2$$

Sendo $y^{(i)}$ a saída desejada da amostra *i*.
Para minimizar a função de perda L(W), é necessário o uso de um método de otimização como o gradiente descendente. No gradiente descendente é necessário calcular as derivadas parciais da função de perda L(W) em função de cada peso da rede. A algoritmo de backpropagation é calculado seguindo os seguintes passos:

1. inicializa-se os pesos aleatoriamente
2. Para cada exemplo ($x^{(i)}$, $y^{(i)}$):
    a. Calcule o erro (forward pass)
    b. Calcule as derivadas parciais para cada neurônio
    c. Atualiza os pesos (backward pass)

Para exemplificar, considere o seguinte exemplo, onde são utilizados vieses e função de ativação sigmoid:



Após a inicialização:

- $w_{11} = -0.2,\ w_{12} = 0.1,\ w_{21} = -0.1,\ w_{22} = 0.3,\ w_{31} = 0.2,\ w_{32} = 0.3$
- $b_1 = 0.1,\ b_2 = 0.1,\ b_3 = 0.2$

Assumindo:

- Saída desejada *o* =0.9
- Taxa de aprendizado *r* = 0.25

Considerando a primeira iteração, temos:

Forward pass:

- $v_1 = 1 \times b_1 + x_1 \times w_{11} + x_2 \times w_{12} = 1 \times 0.1 + 0.1 \times (-0.2) + 0.9 \times 0.1 = 0.17$
- $y_1 = \phi(v1) = \phi(0.17) = \frac{1}{1+\exp(-0.17)} = 0.542$
- $v_2 = 1 \times b_2 + x_1 \times w_{21} + x_2 \times w_{22} = 1 \times 0.1 + 0.1 \times (-0.1) + 0.9 \times 0.3 = 0.36$
- $y_2 = \phi(v2) = \phi(0.36) = \frac{1}{1+\exp(-0.36)} = 0.589$
- $v_3 = 1 \times b_3 + y_1 \times w_{31} + y - 2 \times w_{32} = 1 \times 0.2 + 0.542 \times 0.2 + 0.589 \times 0.3 = 0.485$
- $y_3 = \phi(v_3) = \phi(0.485) = \frac{1}{1+\exp(-0.485)} = 0.619$

O erro encontrado: $(o - y_3) = 0.9 - 0.619 = 0.281$

Como está sendo utilizada a função de ativação sigmoid, as derivadas parciais de cada neurônio são dadas pelas seguintes fórmulas:

Local gradients:

- $\delta o = \phi'(1 * b_3 + y_1 \times w_{31} + y_2 \times w_{32}) \times (o - y_3)$
- $\delta h_1 = \phi'(1 * b_1 + x_1 \times w_{11} + x_2 \times w_{21}) \times (\delta o \times w_{31})$
- $\delta h_2 = \phi'(1 * b_2 + x_2 \times w_{12} + x_2 \times w_{22}) \times (\delta o \times w_{32})$

Backward pass:

- $\delta o = \phi'(v_3) \times (o - y_3) = \phi'(0.485) \times 0.281 =$
  $\phi(0.485) \times (1 - \phi(0.485)) \times 0.281 = 0.619 \times (1 - 0.619) \times 0.281 = 0.0663$
- $\delta h_1 = \phi'(v_1) \times (\delta o \times w_{31}) = \phi'(0.17) \times 0.0663 \times 0.2 =$
  $\phi(0.17) \times (1 - \phi(0.17)) \times 0.01362 = 0.542 \times (1 - 0.542) \times 0.01362 = 0.0033$
- $\delta h_2 = \phi'(v_2) \times (\delta o \times w_{32}) = \phi'(0.36) \times 0.0663 \times 0.3 =$
  $\phi(0.36) \times (1 - \phi(0.36)) \times 0.01989 = 0.589 \times (1 - 0.589) \times 0.01989 = 0.0049$

Após o Backward pass, atualiza-se os pesos para a próxima iteração *n*, de acordo com as seguintes equações:

- $w(n+1) = w(n) + r \times \delta \times x$
- $w_{31}(n+1) = 0.2090$
- $w_{32}(n+1) = 0.3098$
- $w_{11}(n+1) = -0.1999$
- $w_{21}(n+1) = -0.099$
- $w_{12}(n+1) = -0.1008$
- $w_{22}(n+1) = -0.3011$

Os vieses também são atualizados com a mesma fórmula:

- $b(n+1) = b(n) + r \times \delta \times x$
- $b_3 = 0.2166$
- $b_2 = 0.1012$
- $b_1 = 0.1008$

Após a primeira iteração:

- $v_1 = 0.17 \rightarrow v_1 = 0.1715$
- $y_1 = 0.542 \rightarrow y_1 = 0.5428$
- $v_2 = 0.36 \rightarrow v_2 = 0.3622$
- $y_2 = 0.589 \rightarrow y_2 = 0.5896$
- $v_3 = 0.4851 \rightarrow v_3 = 0.5127$
- $y_3 = 0.619 \rightarrow y_3 = 0.6254$
- $error = (o - y_3) = 0.9 - 0.619 = 0.281 \rightarrow error =$
  $0.9 - 0.6254 = 0.2746$

Após mais algumas iterações:

- After second pass: $error = 0.2683$
- After third pass: $error = 0.2623$
- After fourth pass: $error = 0.2565$
- After 100 passes: $error = 0.0693$
- After 200 passes: $error = 0.0319$
- After 500 passes: $error = 0.0038$
- Error is getting reduced after each pass.

Observa-se que a aplicação do backpropagation é eficiente para o treinamento de uma rede neural. Entretanto a desvantagem mais óbvia desse algoritmo é que não é garantido que será encontrado o mínimo global, apenas um mínimo local, porém, segundo o autor do artigo, experiências com muitas tarefas raramente ficaram presas em mínimos locais que são significamente piores que o mínimo global.

Os autores não pensaram nisso na época mas uma correção do algoritmo para evitar mínimos locais, seria altera a função de custo L(w) para a regressão logística, que é convexa, de acordo com a seguinte fórmula:

$$L(W) = -(\sigma(W^T X^{(i)}) + (1 - \sigma(W^T X^{(i)}))log(1 - u^{(i)}))$$