Backpropagation
Processus d'Apprentissage d'un Réseau de Neurones
rivo.link@gmail.com

1 Backpropagation - Neurone

Ceci est mon approche de la backpropagation, un processus d'apprentissage d'un réseau de neurones multicouches.

Soit un réseau de neurones à une couche cachée noté NN(i,j,k) où i est le nombre des entrées, j le nombre de neurones dans la couche cachée et k le nombre de neurones de sorties.

On notera $(x_i)_i$ les entrées présentées au neurone j de la couche cachée, $(w_{ij})_i$ les poids associés et b_j le biais pour ce neurone. Le neurone j aura donc la pré-activation z_j selon la formule:

$$z_j = \sum_i x_i w_{ij} + b_j \tag{1}$$

L'activation du neurone j de la couche cachée sera la sigmoïde de sa pré-activation, il aura donc comme sortie:

$$f(z_j) = \frac{1}{1 + e^{-z_j}} = f(\sum_i x_i w_{ij} + b_j) \tag{2}$$

Comme les sorties des j neurones de la couche cachée sont les entrée des k neurones de la couche de sortie, en notant $(x_j)_j$ ces entrées, $(w_{jk})_j$ les poids et b_k le biais, on aura pour le neurone k de la couche de sortie, la pré-activation z_k:

$$x_j = f(z_j)$$

$$z_k = \sum_j x_j w_{jk} + b_k \tag{3}$$

L'activation du neurone k, de la couche de sortie, sera la sigmoïde de sa pré-activation, il aura donc comme sortie:

$$f(z_k) = \frac{1}{1 + e^{-z_k}} = f(\sum_j x_j w_{jk} + b_k) \tag{4}$$

Finalement, en notant $\hat{y}_k$ la sortie du neurone k de la couche de sortie, alors $(\hat{y}_k)_k$ seront les sorties du réseau de neurones NN(i,j,k), et on aura les k-équations:

$$\hat{y}_k = f(z_k)$$

$$\hat{y}_k = f(\sum_j f(\sum_i x_i w_{ij} + b_j) w_{jk} + b_k) \tag{5}$$

2 Backpropagation - Apprentissage

On note $(y_k)_k$ les sorties attendues, associée aux entrées $(x_i)_i$ pour le réseau de neurones NN(i,j,k). On définit l'erreur de prédiction $E(\hat{y})$ par:

$$E(\hat{y}) = \sum_k \frac{1}{2}(y_k - \hat{y}_k)^2 \tag{6}$$

2.1 Apprentissage - Couche de sortie

La mise à jour du j-ème poids w_{jk} du neurone k de la couche de sortie se fera selon la formule:

$$w_{jk} := w_{jk} - \alpha \frac{\partial E(\hat{y})}{\partial w_{jk}} \tag{7}$$

$$\frac{\partial E(\hat{y})}{\partial w_{jk}} = \frac{\partial}{\partial w_{jk}} \sum_k \frac{1}{2}(y_k - \hat{y}_k)^2 \tag{8}$$

Comme, seul $\hat{y}_k$ dépend de w_{jk}, et les elements de la somme dont les indices se diffèrent de k ont une dérivée partielle nulle alors:

$$\begin{aligned}
\frac{\partial E(\hat{y})}{\partial w_{jk}} &= -\sum_k (y_k - \hat{y}_k)\frac{\partial \hat{y}_k}{\partial w_{jk}} \\
&= -(y_k - \hat{y}_k)\frac{\partial \hat{y}_k}{\partial w_{jk}} \\
&= -(y_k - \hat{y}_k)\frac{\partial f(z_k)}{\partial w_{jk}} \\
&= -(y_k - \hat{y}_k)\frac{\partial f(z_k)}{\partial z_k}\frac{\partial z_k}{\partial w_{jk}} \\
\frac{\partial E(\hat{y})}{\partial w_{jk}} &= -(y_k - \hat{y}_k)f'(z_k)\frac{\partial}{\partial w_{jk}}(\sum_j x_j w_{jk} + b_k)
\end{aligned} \tag{9}$$

Finalement, comme "la fonction f est la fonction sigmoid", il en résulte l'équation de variation de l'erreur $E(\hat{y})$ par rapport au poid w_{jk}, et par rapport au biais b_k:

$$\begin{aligned}
\frac{\partial E(\hat{y})}{\partial w_{jk}} &= -(y_k - \hat{y}_k)f(z_k)(1 - f(z_k))x_j \\
\frac{\partial E(\hat{y})}{\partial b_k} &= -(y_k - \hat{y}_k)f(z_k)(1 - f(z_k))
\end{aligned}$$

En d'autres termes, comme $\hat{y}_k = f(z_k)$, on a les nouvelles équations:

$$\begin{aligned}
\frac{\partial E(\hat{y})}{\partial w_{jk}} &= -\hat{y}_k(1 - \hat{y}_k)(y_k - \hat{y}_k)x_j \\
\frac{\partial E(\hat{y})}{\partial b_k} &= -\hat{y}_k(1 - \hat{y}_k)(y_k - \hat{y}_k)
\end{aligned}$$

2.2 Apprentissage - Couche cachée

La mise à jour du i-ème poids w_{ij} du neurone j de la couche cachée se fera selon la formule:

$$w_{ij} := w_{ij} - \alpha \frac{\partial E(\hat{y})}{\partial w_{ij}} \tag{10}$$

$$\frac{\partial E(\hat{y})}{\partial w_{ik}} = \frac{\partial}{\partial w_{ij}} \sum_k \frac{1}{2}(y_k - \hat{y}_k)^2 \tag{11}$$

Il faut noter qu'ici, la somme ne disparaît pas puisqu'elle ne depend pas de w_{ij}. Et comme seul $\hat{y}_k$ dépend de w_{ij}, alors on a:

$$\begin{aligned}
\frac{\partial E(\hat{y})}{\partial w_{ij}} &= -\sum_k (y_k - \hat{y}_k) \frac{\partial \hat{y}_k}{\partial w_{ij}} \\
&= -\sum_k (y_k - \hat{y}_k) \frac{\partial f(z_k)}{\partial w_{ij}} \\
&= -\sum_k (y_k - \hat{y}_k) \frac{\partial f(z_k)}{\partial z_k} \frac{\partial z_k}{\partial x_j} \frac{\partial x_j}{\partial z_j} \frac{\partial z_j}{\partial w_{ij}} \\
&= -\sum_k (y_k - \hat{y}_k) f'(z_k) (\frac{\partial}{\partial x_j} \sum_j x_j w_{jk} + b_k) \frac{\partial x_j}{\partial z_j} \frac{\partial z_j}{\partial w_{ij}} \\
&= -\sum_k (y_k - \hat{y}_k) f'(z_k) (w_{jk}) \frac{\partial x_j}{\partial z_j} \frac{\partial z_j}{\partial w_{ij}} \\
&= -\sum_k (y_k - \hat{y}_k) f'(z_k) (w_{jk}) \frac{\partial f(z_j)}{\partial z_j} \frac{\partial z_j}{\partial w_{ij}} \\
&= -\sum_k (y_k - \hat{y}_k) f'(z_k) (w_{jk}) f'(z_j) \frac{\partial z_j}{\partial w_{ij}} \\
\frac{\partial E(\hat{y})}{\partial w_{ij}} &= -\sum_k (y_k - \hat{y}_k) f'(z_k) (w_{jk}) f'(z_j) (\frac{\partial}{\partial w_{ij}} \sum_i x_i w_{ij} + b_j)
\end{aligned}$$

Finalement, il en résulte l'équation de variation de l'erreur $E(\hat{y})$ par rapport au poid w_{ij}. Et on en déduit sa variation par rapport au biais b_j, toujours en notant que f est la fonction sigmoid:

$$\begin{aligned}
\frac{\partial E(\hat{y})}{\partial w_{ij}} &= -\sum_k (y_k - \hat{y}_k) f(z_k)(1 - f(z_k)) w_{jk} f(z_j)(1 - f(z_j)) x_i \\
\frac{\partial E(\hat{y})}{\partial b_j} &= -\sum_k (y_k - \hat{y}_k) f(z_k)(1 - f(z_k)) w_{jk} f(z_j)(1 - f(z_j))
\end{aligned}$$

En d'autres termes, toujours en utilisant les équations $\hat{y}_k = f(z_k)$ et $x_j = f(z_j)$, on a les nouvelles equations:

$$\frac{\partial E(\hat{y})}{\partial w_{ij}} = -\sum_k (y_k - \hat{y}_k)\hat{y}_k(1-\hat{y}_k)w_{jk}x_j(1-x_j)x_i$$
$$\frac{\partial E(\hat{y})}{\partial b_j} = -\sum_k (y_k - \hat{y}_k)\hat{y}_k(1-\hat{y}_k)w_{jk}x_j(1-x_j)$$

3 Backpropagation - Implémentation

En informatique, la pratique est toujours plus explicite que la théorie, alors une implémentation en Java de la backpropagation sera donnée ci-après.

La methode train de la classe Network est la principale implémentation de l'algorithme de backpropagation:

```java
public void train(float[] inputs,int[] target){
   float[] yhat=setInput(inputs).getOutputs();
   float[] x=hidL.getOutputs();
   int[] y=target;

   Neuron[] hN=hidL.neurons;
   Neuron[] oN=outL.neurons;

   float gE_wij,gE_wjk;
   for(int j=0;j<hN.length;j++){
      for(int k=0;k<oN.length;k++){
         gE_wjk=alpha*(y[k]-yhat[k])*yhat[k]*(1-yhat[k]);
         oN[k].biais+=gE_wjk;
         oN[k].weights[j]+=gE_wjk*x[j];
      }

      for(int i=0;i<this.i;i++){
         gE_wij=0;
         float xi=hidL.neurons[0].inputs[i];
         for(int k=0;k<oN.length;k++){
            float wjk=outL.neurons[k].weights[j];
            gE_wij+=(y[k]-yhat[k])*yhat[k]*(1-yhat[k])*wjk*x[j]*(1-x[j]);
         }
         gE_wij*=alpha;
         hN[j].biais+=gE_wij;
         hN[j].weights[i]+=gE_wij*xi;
      }
   }
}
```

Ceci est la code source complete, il est aussi disponible sur: `https://github.com/RivoLink/AI_Doc/tree/master/backpropagation/java/src`

```java
// Neuron.java
import java.util.Random;

public class Neuron{

   static final Random r=new Random();

   public float biais;
   public final int size;
   public final float inputs[];
   public final float weights[];

   public Neuron(int size){
      this.size=size;
      this.inputs=new float[size];
      this.weights=new float[size];

      biais=r.nextFloat();
      for(int i=0;i<size;i++){
         weights[i]=r.nextFloat();
      }
   }

   public Neuron setInput(float inputs[]){
      for(int i=0;i<size;i++){
         this.inputs[i]=inputs[i];
      }
      return this;
   }

   public float getOutput(){
      float z=dot(inputs,weights)+biais;
      return (float)(1/(1+Math.exp(-z)));
   }

   public static float dot(float[] x,float[] w){
      if(x.length!=w.length)
         return 0;

      float dot=0;
      for(int i=0;i<x.length;i++){
         dot+=x[i]*w[i];
      }
      return dot;
   }
}
```

```java
// Layer.java
public class Layer{

   public final int nCount;
   public final Neuron[] neurons;

   public Layer(int iLen,int nCount){
      this.nCount=nCount;

      neurons=new Neuron[nCount];
      for(int i=0;i<nCount;i++){
         neurons[i]=new Neuron(iLen);
      }
   }

   public Layer setInputs(float[] inputs){
      for(int i=0;i<nCount;i++){
         neurons[i].setInput(inputs);
      }
      return this;
   }

   public float[] getOutputs(){
      float[] outputs=new float[nCount];
      for(int i=0;i<nCount;i++){
         outputs[i]=neurons[i].getOutput();
      }
      return outputs;
   }
}

// Network.java
public class Network{
   public final float alpha=0.1f;

   public final int i;
   public final Layer hidL;
   public final Layer outL;

   public Network(int i,int j,int k){
      this.i=i;
      this.hidL=new Layer(i,j);
      this.outL=new Layer(j,k);
   }

   public Network setInput(float... bits){
      hidL.setInputs(bits);
      return this;
   }
```

```
  public float[] getOutputs(){
    float[] x=hidL.getOutputs();
    return outL.setInputs(x).getOutputs();
  }

  public void train(float[] inputs,int[] target){
    float[] yhat=setInput(input).getOutputs();
    float[] x=hidL.getOutputs();
    int[] y=target;

    Neuron[] hN=hidL.neurons;
    Neuron[] oN=outL.neurons;

    float gE_wij,gE_wjk;
    for(int j=0;j<hN.length;j++){
      for(int k=0;k<oN.length;k++){
        gE_wjk=alpha*(y[k]-yhat[k])*yhat[k]*(1-yhat[k]);
        oN[k].biais+=gE_wjk;
        oN[k].weights[j]+=gE_wjk*x[j];
      }

      for(int i=0;i<this.i;i++){
        gE_wij=0;
        float xi=hidL.neurons[0].inputs[i];
        for(int k=0;k<oN.length;k++){
          float wjk=outL.neurons[k].weights[j];
          gE_wij+=(y[k]-yhat[k])*yhat[k]*(1-yhat[k])*wjk*x[j]*(1-x[j]);
        }
        gE_wij*=alpha;
        hN[j].biais+=gE_wij;
        hN[j].weights[i]+=gE_wij*xi;
      }
    }
  }
}
```

4 Backpropagation - Apprentissage Profond

En apprentissage profond, ou "Deep Learning", on ne se contente pas d'une ou deux couches cachées, il peut etre question de cent, cinq cent ou meme mille couches cachées, d'où vient l'appellation "profond".

4.1 Apprentissage Profond - Les entrées

En apprentissage profond, vu le nombre de couches cachées, la mise à jour des poids n'est plus chose aisée. Supposons qu'on aie j,k,l,...,q couches cachées, les q-entrées $(x_q)_q)$ de la couche de sortie seront données par les q-equations suivantes:

$$z_q = \sum_p f(... \sum_k f(\sum_j f(\sum_i x_i w_{ij} + b_j) w_{jk} + b_k) w_{kl} + b_l ...) w_{pq} + b_q$$

En effet,

$$\begin{aligned} z_j &= \sum_i x_i w_{ij} + b_j \\ z_k &= \sum_j f(z_j) w_{jk} + b_k \\ z_l &= \sum_k f(z_k) w_{kl} + b_l \\ ... &= ... \\ z_q &= \sum_p f(z_p) w_{pq} + b_q \end{aligned}$$

4.2 Apprentissage Profond - Poids

Ayant la variation de l'erreur de prediction $E(\hat{y})$ par rapport au poids $(w_{ij}) =)_i$ des j-neurones de la premiere couche cachée, toutes les autres variation s'en decoulent, alors:

Avec,

$$x_k = f(z_k), z_k = \sum_j f(z_j) w_{jk} + b_k$$

On a:

$$\begin{aligned} \frac{\partial E(\hat{y})}{\partial w_{ij}} &= -\sum_q (y_k - \hat{y}_q) f'(z_q) w_q f'(z_p) ... f'(z_k) w_k f'(z_j) w_j f'(z_i) x_i \\ \frac{\partial E(\hat{y})}{\partial b_i} &= -\sum_q (y_k - \hat{y}_q) f'(z_q) w_q f'(z_p) ... f'(z_k) w_k f'(z_j) w_j f'(z_i) \end{aligned}$$

En effet,

$$
\begin{aligned}
\frac{\partial E(\hat{y})}{\partial w_{ij}} &= -\sum_q (y_q - \hat{y}_q)\frac{\partial \hat{y}_q}{\partial w_{ij}} \\
&= -\sum_q (y_k - \hat{y}_q)\frac{\partial f(z_q)}{\partial w_{ij}} \\
\frac{\partial E(\hat{y})}{\partial w_{ij}} &= -\sum_q (y_q - \hat{y}_q)\frac{\partial f(z_q)}{\partial z_q}\frac{\partial z_q}{\partial x_p}\frac{\partial x_p}{\partial z_p}...\frac{\partial x_k}{\partial z_k}\frac{\partial z_k}{\partial x_j}\frac{\partial x_j}{\partial z_j}\frac{\partial z_j}{\partial w_{ij}}
\end{aligned}
$$