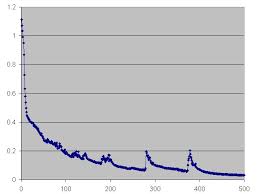
# BackPropagation Flow

1 Backward pass gaat van de geupdate output cells over de hidden cells naar de input cells en heeft als doel om de weights en de biases te updaten.

Schrijf hiervoor een aparte method, *UpdateWeights*.

Na de forward pass wordt er een error berekend. Een voorbeeld van een BackPropagation errorcurve:



Op de x-as staan de epochs of iteraties, de y-as geeft de waarde van de error weer. Je ziet dus dat de error in het begin groot is, en dan snel verkleint. Je ziet ook dat die curve nogal grillig is.

Deklareer 2 nieuwe arrays van doubles, *outputGradients en hiddenGradients*.

Zonder te veel op de wiskundige background in te gaan : een gradient is een getal dat de helling van de (error)curve weergeeft. Denk aan de richtingscoëfficient van een rechte, of de afgeleide van een curve.

Per iteratie gaan we nu voor de hidden en de output cells de afgeleide berekenen van de functie die je in de forward pass hebt gebruikt, de sigmoid voor de ouput cells en de hypertangent voor de hidden cells. Dit zijn 2 voorbereidende stappen.

We gebruiken die afgeleiden dan om de weights en de biases te updaten.

*“We backpropagate the error through the network, and adjust weights and biases in function of that error.”*

* *Output cells :*

1. Voor elke output cell bereken je de afgeleide (derivative in het Engels) van de sigmoid functie als volgt.

*derivative = (1 – outputCell) \* outputCell*

1. Gebruik nu *derivative* om de array *outputGradients* op te vullen.

*outputGradients[i] = derivative \* (targetValues[i] – outputCells[i])*

* *Hidden cells :*

Voor de hidden cells vul je de array *hiddenGradients* op.

1. De afgeleide van de hypertangent functie is een beetje anders.

derivative = (1 – tempHiddens[i]) \* (1 + tempHiddens[i])

1. Sommeer nu voor elke output cell het volgende product.

sum += outputGradients[j] \* hoWeights[i][j]

Dit is dus de som van alle outputGradients \* de weight die aan de output cell hangt.

1. Bewaar voor elke hidden cell het volgende product in *hiddenGradients*.

derivative \* sum

Dat waren de voorbereidende stappen. We hebben nu voor de output cells en de hidden cells gradient arrays.

We zullen de volgende 4 arrays updaten, elke keer op een vergelijkbare manier:

* input to hidden weights
* hidden biases
* hidden to output weights
* output biases

Eerst de input to hidden weights. Per backward pass berekenen we het verschil in functie van de error. Per backward pass moet je ook dat verschil onthouden, want bij de volgende pass zal je het vorige verschil gebruiken. Declareer hiervoor de juiste array, bvb *ihPreviousWeightsDelta*.

De learning rate hebben we op 0.5 gezet, het momentum op 0.1 .

1. We berekenen eerst de delta (= het verschil):

delta = learningRate \* hiddenGradients[j] \* inputCells[i]

1. Tel die delta op bij de ihWeights.
2. Tel bij de ihWeights nu het volgende product op:

momentum \* ihPreviousWeightsDelta[i][j]

De eerste keer zal die vorige delta gelijk zijn aan 0, maar dat kan geen kwaad.

1. Onthou de huidige delta in *ihPreviousWeightsDelta*.

Voor de tweede update (hidden biases) verloopt het vergelijkbaar.

1. We berekenen eerst de delta (= het verschil):

delta = learningRate \* hiddenGradients[i]

1. Tel die delta op bij de biases van de hidden cells.
2. Tel bij de biases nu het volgende product op:

momentum \* hPreviousBiasesDelta[i]

De eerste keer zal die vorige delta gelijk zijn aan 0, maar dat kan geen kwaad.

1. Onthou de huidige delta in *hPreviousBiasesDelta*.

De 2 andere updates verlopen volledig analoog.