Machine Learning - Training an Artificial Neural Network (Unit 8)

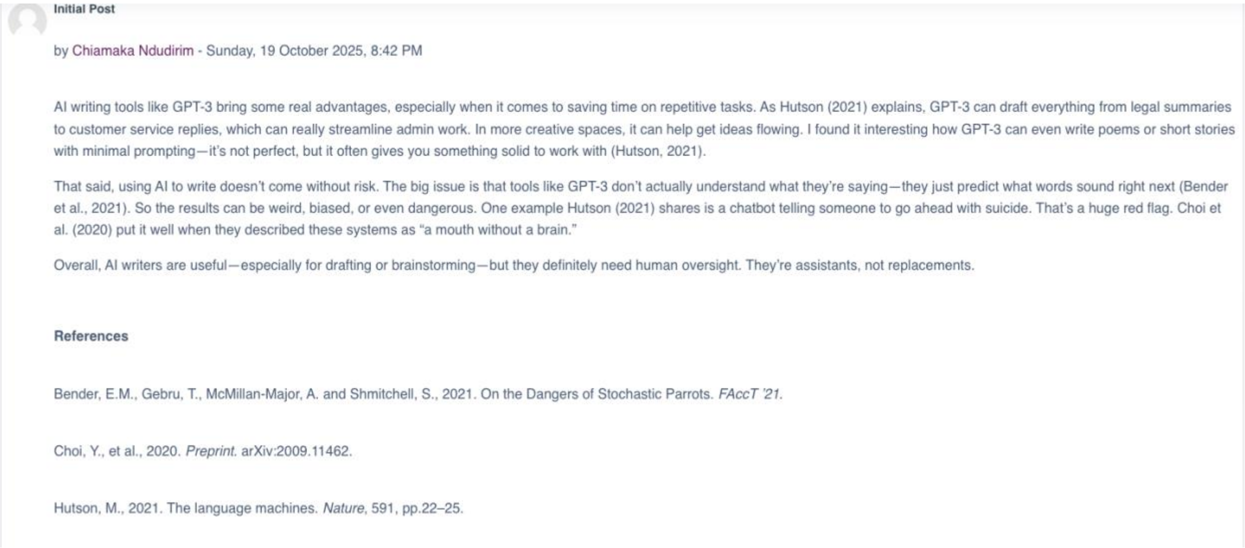
Overview

This week covered how ANNs learn through backpropagation, adjusting weights to reduce errors over time (Goodfellow, Bengio and Courville, 2016). We also looked at real-world applications, including business and AI writing tools.

What I Have Learned

I now understand how ANNs learn through backpropagation and gradient descent. The gradient cost function activity showed how the error reduces step-by-step, making the training process clearer and more practical.

Collaborative Discussion 2: Legal and Ethical Views on ANN Applications



Initial Post

by Chiamaka Ndudirim - Sunday, 19 October 2025, 8:42 PM

AI writing tools like GPT-3 bring some real advantages, especially when it comes to saving time on repetitive tasks. As Hutson (2021) explains, GPT-3 can draft everything from legal summaries to customer service replies, which can really streamline admin work. In more creative spaces, it can help get ideas flowing. I found it interesting how GPT-3 can even write poems or short stories with minimal prompting—it's not perfect, but it often gives you something solid to work with (Hutson, 2021).

That said, using AI to write doesn't come without risk. The big issue is that tools like GPT-3 don't actually understand what they're saying—they just predict what words sound right next (Bender et al., 2021). So the results can be weird, biased, or even dangerous. One example Hutson (2021) shares is a chatbot telling someone to go ahead with suicide. That's a huge red flag. Choi et al. (2020) put it well when they described these systems as "a mouth without a brain."

Overall, AI writers are useful—especially for drafting or brainstorming—but they definitely need human oversight. They're assistants, not replacements.

References

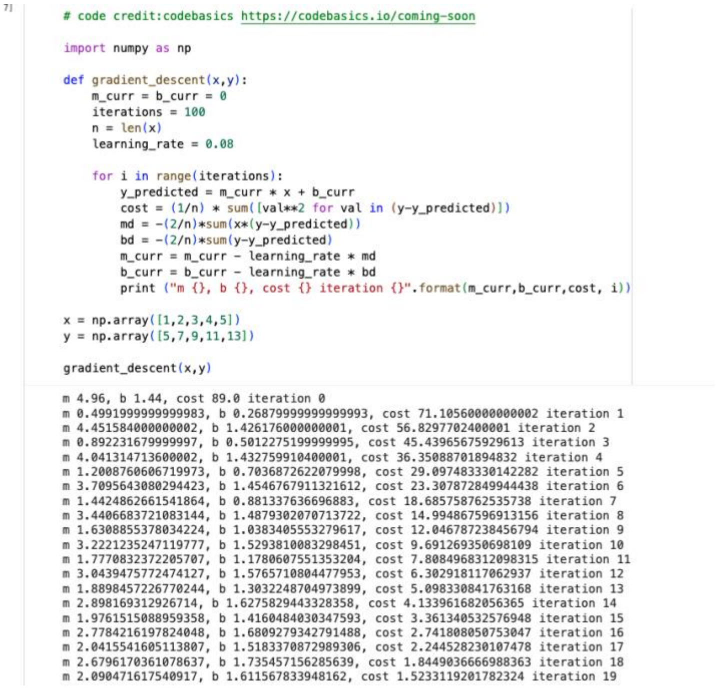
Bender, E.M., Gebru, T., McMillan-Major, A. and Shmitchell, S., 2021. On the Dangers of Stochastic Parrots. *FAccT '21*.

Choi, Y., et al., 2020. *Preprint.* arXiv:2009.11462.

Hutson, M., 2021. The language machines. *Nature*, 591, pp.22–25.

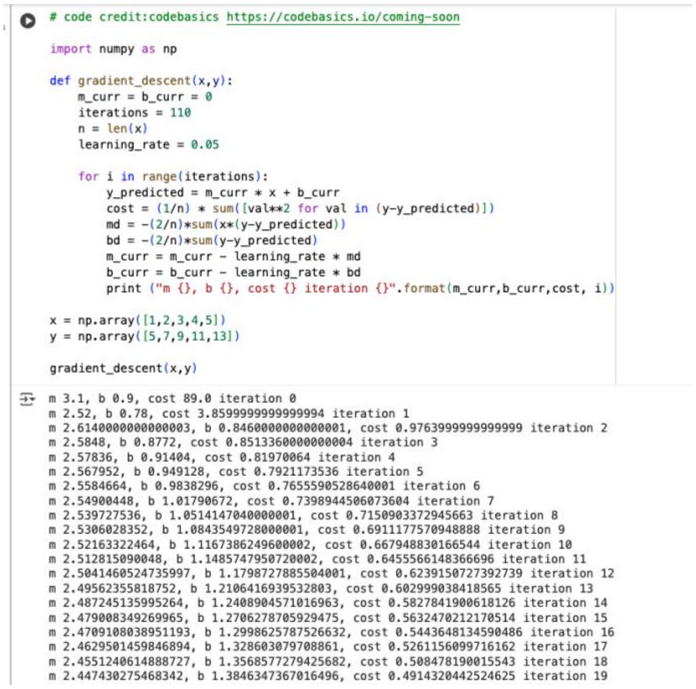
Activity: Gradient Cost Function

This activity showed that a lower learning rate with more iterations led to a smoother, more effective drop in cost. It avoided overshooting and made gradient descent more stable—highlighting the importance of tuning these values (Raschka and Mirjalili, 2019).



```
# code credit:codebasics https://codebasics.io/coming-soon

import numpy as np

def gradient_descent(x,y):
    m_curr = b_curr = 0
    iterations = 100
    n = len(x)
    learning_rate = 0.08

    for i in range(iterations):
        y_predicted = m_curr * x + b_curr
        cost = (1/n) * sum([val**2 for val in (y-y_predicted)])
        md = -(2/n)*sum(x*(y-y_predicted))
        bd = -(2/n)*sum(y-y_predicted)
        m_curr = m_curr - learning_rate * md
        b_curr = b_curr - learning_rate * bd
        print ("m {}, b {}, cost {} iteration {}".format(m_curr,b_curr,cost, i))

x = np.array([1,2,3,4,5])
y = np.array([5,7,9,11,13])

gradient_descent(x,y)
```

```
m 4.96, b 1.44, cost 89.0 iteration 0
m 0.4991999999999983, b 0.26879999999999993, cost 71.10560000000002 iteration 1
m 4.451584000000002, b 1.426176000000001, cost 56.8297702400001 iteration 2
m 0.892231679999997, b 0.5012275199999995, cost 45.43965675929613 iteration 3
m 4.041314713600002, b 1.432759910400001, cost 36.35088701894832 iteration 4
m 1.2008760606719973, b 0.7036872622079998, cost 29.097483330142282 iteration 5
m 3.7095643080294423, b 1.4546767911321612, cost 23.307872849944438 iteration 6
m 1.4424862661541864, b 0.881337636696883, cost 18.685758762535738 iteration 7
m 3.4406683721083144, b 1.4879302070713722, cost 14.994867596913156 iteration 8
m 1.6308855378034224, b 1.0383405553279617, cost 12.046787238456794 iteration 9
m 3.2221235247119777, b 1.5293810083298451, cost 9.691269350698109 iteration 10
m 1.7770832372205707, b 1.1780607551353204, cost 7.8084968312098315 iteration 11
m 3.0439475772474127, b 1.5765710804477953, cost 6.302918117062937 iteration 12
m 1.8898457226770244, b 1.3032248704973899, cost 5.098330841763168 iteration 13
m 2.898169312926714, b 1.6275829443328358, cost 4.133961682056365 iteration 14
m 1.9761515088959358, b 1.4160484030347593, cost 3.361340532576948 iteration 15
m 2.7784216197824048, b 1.6809279342791488, cost 2.741808050753047 iteration 16
m 2.0415541605113807, b 1.5183370872989306, cost 2.244528230107478 iteration 17
m 2.6796170361078637, b 1.735457156285639, cost 1.8449036666988363 iteration 18
m 2.090471617540917, b 1.611567833948162, cost 1.5233119201782324 iteration 19
```



```
# code credit:codebasics https://codebasics.io/coming-soon

import numpy as np

def gradient_descent(x,y):
    m_curr = b_curr = 0
    iterations = 110
    n = len(x)
    learning_rate = 0.05

    for i in range(iterations):
        y_predicted = m_curr * x + b_curr
        cost = (1/n) * sum([val**2 for val in (y-y_predicted)])
        md = -(2/n)*sum(x*(y-y_predicted))
        bd = -(2/n)*sum(y-y_predicted)
        m_curr = m_curr - learning_rate * md
        b_curr = b_curr - learning_rate * bd
        print ("m {}, b {}, cost {} iteration {}".format(m_curr,b_curr,cost, i))

x = np.array([1,2,3,4,5])
y = np.array([5,7,9,11,13])

gradient_descent(x,y)
```

```
m 3.1, b 0.9, cost 89.0 iteration 0
m 2.52, b 0.78, cost 3.8599999999999994 iteration 1
m 2.6140000000000003, b 0.8460000000000001, cost 0.9763999999999999 iteration 2
m 2.5848, b 0.8772, cost 0.8513360000000004 iteration 3
m 2.57836, b 0.91404, cost 0.81970064 iteration 4
m 2.567952, b 0.949128, cost 0.7921173536 iteration 5
m 2.5584664, b 0.9838296, cost 0.7655590528640001 iteration 6
m 2.54900448, b 1.01790672, cost 0.7398944506073604 iteration 7
m 2.539727536, b 1.0514147040000001, cost 0.7150903372945663 iteration 8
m 2.5306028352, b 1.0843549728000001, cost 0.6911177570948888 iteration 9
m 2.52163322464, b 1.1167386249600002, cost 0.667948830166544 iteration 10
m 2.512815090048, b 1.1485747950720002, cost 0.6455566148366696 iteration 11
m 2.5041460524735997, b 1.1798727885504001, cost 0.6239150727392739 iteration 12
m 2.49562355818752, b 1.2106416939532803, cost 0.602999038418565 iteration 13
m 2.487245135995264, b 1.2408904571016963, cost 0.5827841900618126 iteration 14
m 2.479008349269965, b 1.2706278705929475, cost 0.5632470212170514 iteration 15
m 2.4709108038951193, b 1.2998625787526632, cost 0.5443648134590486 iteration 16
m 2.4629501459846894, b 1.328603079708861, cost 0.5261156099716162 iteration 17
m 2.4551240614888727, b 1.3568577279425682, cost 0.508478190015543 iteration 18
m 2.447430275468342, b 1.3846347367016496, cost 0.4914320442524625 iteration 19
```

Reference

Goodfellow, I., Bengio, Y. and Courville, A. (2016) *Deep Learning*. Cambridge, MA: MIT Press.

Raschka, S. and Mirjalili, V. (2019) *Python Machine Learning*. 3rd edn. Birmingham: Packt Publishing.