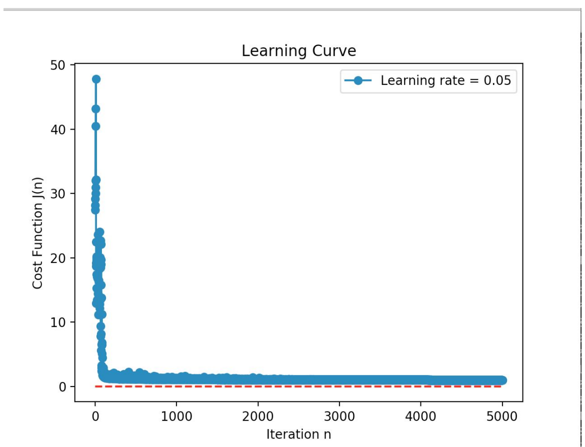
1.

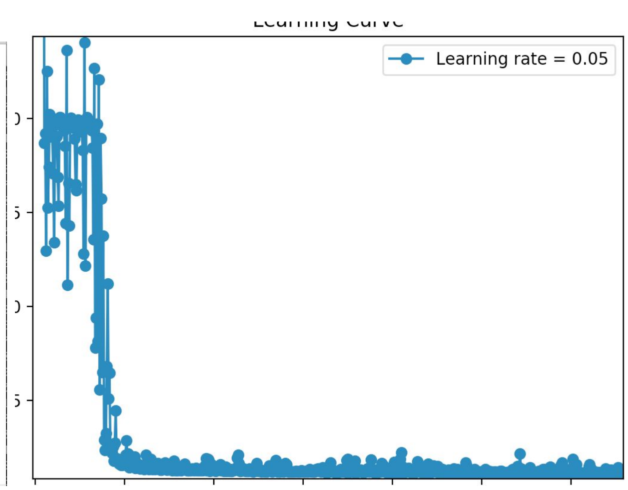
Online Learning	Batch Learning
Simplifies the algorithm and makes the data management easier.	Needs to keep a history of read and write processes to RAM and can take up too much space (store gradient)
Data arrived in a stream (continuous time)	Data collected over time
Difficult to generalize because each samlpe is evaluated	The average of data points is considered so this can lead to overgeneralization
Difficult to parallelize because data input is serial and can find a global optimum from computing gradient descent	Can ensure a local minimum and easier to parallelize
Is difficult to maintain a constant flow of data without disturbing already exisiting features	Provides a general framework that can be changed to other datasets as long as order is subject to IID samples
Updates weights every sample, single step residual error is often larger than the batch error	Updates weights every batch (True gradient, smaller residual error)

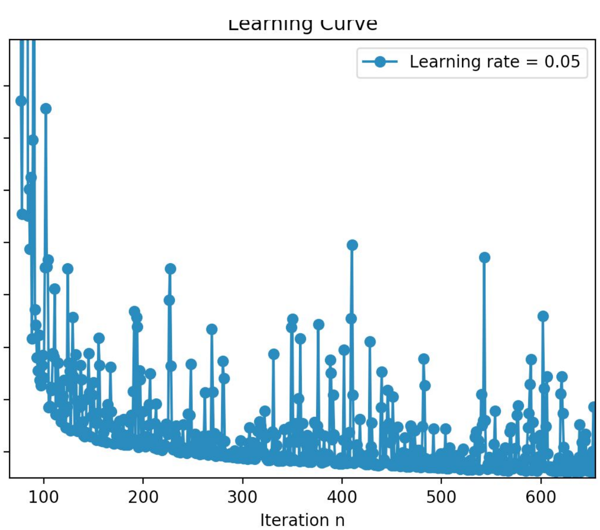
As a quick remark, the difference in the code between batch and online is that the data is fed in on a per-sample basis, which affects the period when the weights are updated.

Graphs:

Online train set(zoomed in pictures to show jaggedness from Online training):

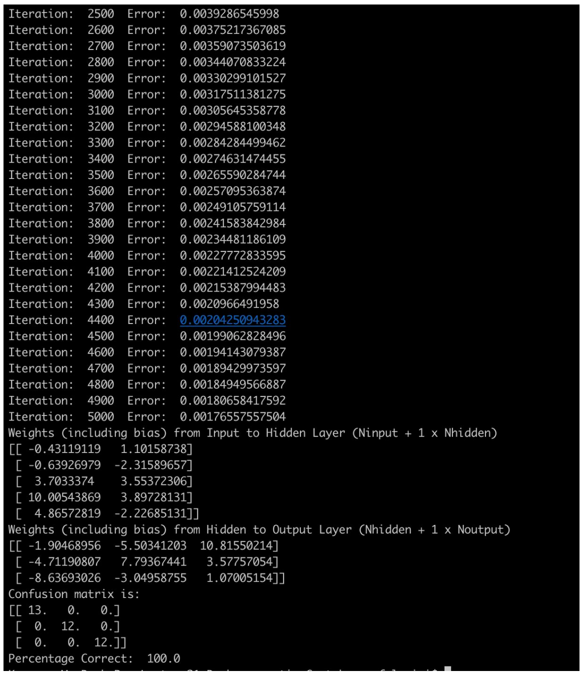




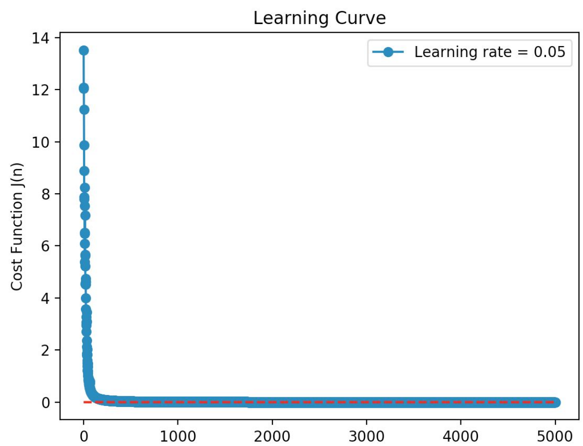


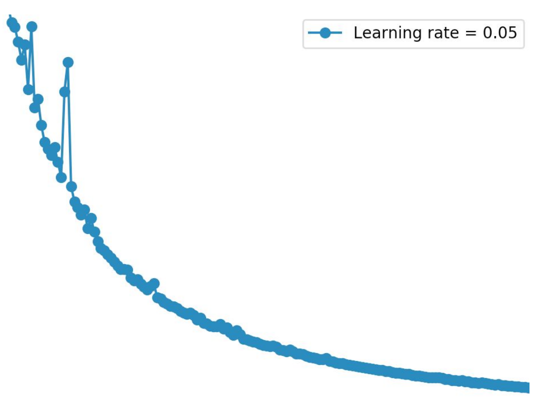
```
Iteration:  2500  Error:  1.04939803537
Iteration:  2600  Error:  1.047010395
Iteration:  2700  Error:  1.05611047029
Iteration:  2800  Error:  1.04450515978
Iteration:  2900  Error:  1.03806980562
Iteration:  3000  Error:  1.03478051764
Iteration:  3100  Error:  1.03379445703
Iteration:  3200  Error:  1.03465510187
Iteration:  3300  Error:  1.02860108081
Iteration:  3400  Error:  1.03124109664
Iteration:  3500  Error:  1.02574415825
Iteration:  3600  Error:  1.02574690118
Iteration:  3700  Error:  1.02191268229
Iteration:  3800  Error:  1.02105209383
Iteration:  3900  Error:  1.01938013392
Iteration:  4000  Error:  1.0185747752
Iteration:  4100  Error:  1.0171328057
Iteration:  4200  Error:  1.01632567951
Iteration:  4300  Error:  1.01535326973
Iteration:  4400  Error:  1.01452655829
Iteration:  4500  Error:  1.01367693589
Iteration:  4600  Error:  1.01291502173
Iteration:  4700  Error:  1.01235881153
Iteration:  4800  Error:  1.0116402866
Iteration:  4900  Error:  1.01107870552
Iteration:  5000  Error:  1.01058410361
Weights (including bias) from Input to Hidden Layer (Ninput + 1 x Nhidden)
[[  1.86585702  -3.66369571]
 [ -2.77964518  -0.10103151]
 [  4.0487157   17.11388026]
 [  3.13337107   5.7204361 ]
 [ -1.78313681   6.91574658]]
Weights (including bias) from Hidden to Output Layer (Nhidden + 1 x Noutput)
[[ -8.12915989   6.097603    -0.65844164]
 [ -5.70907565 -10.71959084   8.93223346]
 [ -1.97034982   3.83588004   8.66303542]]
Confusion matrix is:
[[ 11.   0.   0.]
 [  0.   8.   0.]
 [  0.   1.  17.]]
Percentage Correct:  97.2972972973
```

Online test set(zoomed in pictures to show jaggedness from Online training):

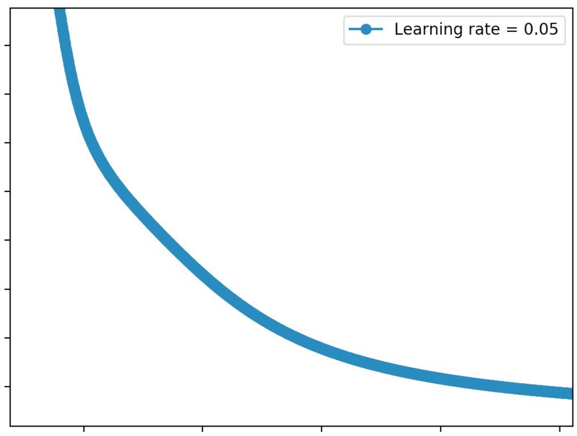


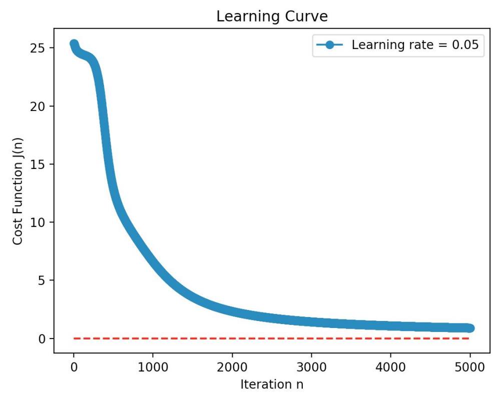
```
Iteration:  2500  Error:  0.0039286545998
Iteration:  2600  Error:  0.00375217367085
Iteration:  2700  Error:  0.00359073503619
Iteration:  2800  Error:  0.00344070833224
Iteration:  2900  Error:  0.00330299101527
Iteration:  3000  Error:  0.00317511381275
Iteration:  3100  Error:  0.00305645358778
Iteration:  3200  Error:  0.00294588100348
Iteration:  3300  Error:  0.00284284499462
Iteration:  3400  Error:  0.00274631474455
Iteration:  3500  Error:  0.00265590284744
Iteration:  3600  Error:  0.00257095363874
Iteration:  3700  Error:  0.00249105759114
Iteration:  3800  Error:  0.00241583842984
Iteration:  3900  Error:  0.00234481186109
Iteration:  4000  Error:  0.00227772833595
Iteration:  4100  Error:  0.00221412524209
Iteration:  4200  Error:  0.00215387994483
Iteration:  4300  Error:  0.0020966491958
Iteration:  4400  Error:  0.00204250943283
Iteration:  4500  Error:  0.00199062828496
Iteration:  4600  Error:  0.00194143079387
Iteration:  4700  Error:  0.00189429973597
Iteration:  4800  Error:  0.00184949566887
Iteration:  4900  Error:  0.00180658417592
Iteration:  5000  Error:  0.00176557557504
Weights (including bias) from Input to Hidden Layer (Ninput + 1 x Nhidden)
[[ -0.43119119   1.10158738]
 [ -0.63926979  -2.31589657]
 [  3.7033374    3.55372306]
 [ 10.00543869   3.89728131]
 [  4.86572819  -2.22685131]]
Weights (including bias) from Hidden to Output Layer (Nhidden + 1 x Noutput)
[[ -1.90468956  -5.50341203  10.81550214]
 [ -4.71190807   7.79367441   3.57757054]
 [ -8.63693026  -3.04958755   1.07005154]]
Confusion matrix is:
[[ 13.   0.   0.]
 [  0.  12.   0.]
 [  0.   0.  12.]]
Percentage Correct:  100.0
```





Batch train set(zoomed in to show smoothness):



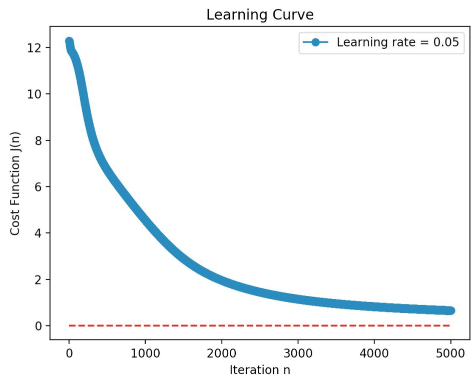


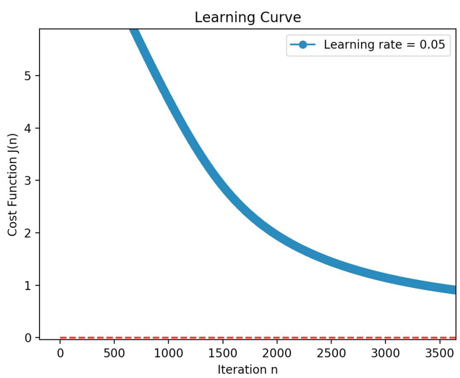

```
Iteration:  2500  Error:  1.75871479063
Iteration:  2600  Error:  1.68049106371
Iteration:  2700  Error:  1.61040830828
Iteration:  2800  Error:  1.54729586839
Iteration:  2900  Error:  1.49018602661
Iteration:  3000  Error:  1.43827355358
Iteration:  3100  Error:  1.39088428892
Iteration:  3200  Error:  1.34745053865
Iteration:  3300  Error:  1.30749165985
Iteration:  3400  Error:  1.27059862597
Iteration:  3500  Error:  1.23642167094
Iteration:  3600  Error:  1.2046603336
Iteration:  3700  Error:  1.17505538642
Iteration:  3800  Error:  1.14738225421
Iteration:  3900  Error:  1.12144561819
Iteration:  4000  Error:  1.09707496915
Iteration:  4100  Error:  1.07412092461
Iteration:  4200  Error:  1.05245216459
Iteration:  4300  Error:  1.0319528706
Iteration:  4400  Error:  1.01252057631
Iteration:  4500  Error:  0.994064356156
Iteration:  4600  Error:  0.976503293004
Iteration:  4700  Error:  0.959765176987
Iteration:  4800  Error:  0.943785396699
Iteration:  4900  Error:  0.928505991084
Iteration:  5000  Error:  0.91387483606
Weights (including bias) from Input to Hidden Layer (Ninput + 1 x Nhidden)
[[ 0.47104229 -0.47038907]
 [-0.59982782  2.51990378]
 [ 2.79315417 -2.63463579]
 [ 5.01430163 -2.2332706 ]
 [ 3.10029012  1.29554831]]
Weights (including bias) from Hidden to Output Layer (Nhidden + 1 x Noutput)
[[-2.02868285 -1.47819598  7.92066333]
 [ 6.75942834 -1.14910793 -1.87831086]
 [-1.60119023 -5.84813312 -0.89712784]]
Confusion matrix is:
[[ 11.   0.   0.]
 [  0.   7.   1.]
 [  0.   0.  18.]]
Percentage Correct:  97.2972972973
```

Batch test set

```
Iteration:  2500  Error:  1.45011983641
Iteration:  2600  Error:  1.37741463634
Iteration:  2700  Error:  1.31146937002
Iteration:  2800  Error:  1.25152563757
Iteration:  2900  Error:  1.19693528363
Iteration:  3000  Error:  1.14713284214
Iteration:  3100  Error:  1.10161673627
Iteration:  3200  Error:  1.05993710877
Iteration:  3300  Error:  1.02168822149
Iteration:  3400  Error:  0.986503696137
Iteration:  3500  Error:  0.954053312773
Iteration:  3600  Error:  0.924040518863
Iteration:  3700  Error:  0.896200160907
Iteration:  3800  Error:  0.870296208365
Iteration:  3900  Error:  0.846119402227
Iteration:  4000  Error:  0.823484848566
Iteration:  4100  Error:  0.802229614199
Iteration:  4200  Error:  0.782210387733
Iteration:  4300  Error:  0.76330125978
Iteration:  4400  Error:  0.745391660637
Iteration:  4500  Error:  0.728384477939
Iteration:  4600  Error:  0.712194363239
Iteration:  4700  Error:  0.696746226127
Iteration:  4800  Error:  0.681973907384
Iteration:  4900  Error:  0.667819018182
Iteration:  5000  Error:  0.65422993003
Weights (including bias) from Input to Hidden Layer (Ninput + 1 x Nhidden)
[[ 0.241202    0.33893679]
 [ 2.30198799  2.59434733]
 [-2.58042304 -1.88124831]
 [-5.07421693 -3.05445882]
 [-3.24132017  0.24775528]]
Weights (including bias) from Hidden to Output Layer (Nhidden + 1 x Noutput)
[[ 3.20228098  6.0117319  -2.75146842]
 [ 8.68268045 -0.19827484 -4.74358868]
 [-0.92727641 -3.90204576 -8.47167676]]
Confusion matrix is:
[[  9.   0.   0.]
 [  0.  15.   0.]
 [  0.   0.  13.]]
Percentage Correct:  100.0
```

:





Analytical questions:

1. $X = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ $d = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ $\eta = 0.1$ · use logistic function

error is calculated between expected outputs and the ones forward propagated. these errors are fed backwards from output layer to input layer, assigning blame for error and updating the weights as you go.

two sections
A) transfer derivative
B) Error Backpropagation.

Transfer Derivative
· given an output value, calculate its slope.
· for sigm(x) = $\frac{dsigm}{\partial x}$ = output * (1 - output) ⇒ t_d

Error BP-
at each neuron: $\boxed{\text{error} = (\text{expected} - \text{output}) * \text{t\_d}.}$ · "only @ output layer"

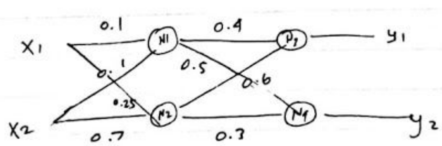
Hidden layer
the error in a hidden layer is a weighted sum of the errors in the output layer

$\boxed{\text{error\_hL} = (\text{weight\_k} * \text{error\_j}) * \text{td(output)}}$

error$_j$ - is the j^{th} neuron in the output layer, weight_k is the weight that connects the k^{th} neuron to the current neuron and output is output of current neuron.

Delta rule ?

$\boxed{\text{Update : weight} = \text{weight} + (\eta * \text{error} * \text{input}).}$



$N_1 = 0.2$ ~~N3 = 1~~ $N3 = (0.6 * N2) + (0.4\ N_1) = \begin{bmatrix} 0.65 \\ 0.385 \end{bmatrix}$ expected $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$

$N2 = 0.95$ ~~N4 = 0.8~~ $N4 = (0.5\ N1) + (0.3 N2)$

output layer

error for Neuron 3 = $(1-0.65) * 0.65 \cdot (1-0.65)$ $\boxed{= 0.2275}$

error for Neuron 4 = $(0-0.385) * 0.385 \cdot (1-0.385)$ $\boxed{= 0.236775}$

error for $N1_1$ $= (0.4 * 0.2275) * (0.2 \cdot (1-0.2))$ $= 0.0207025$

error for $N1_2$ $= (0.5 * 0.236775) * (0.2 \cdot (1-0.2))$ $+ 0.018942$

$\boxed{= 0.0396445}$

error for $N2_1$ $= (0.6 \cdot 0.2275) * (0.95 (1-0.95))$ $= 0.006484$

error for $N2_2$ $= (0.3 \cdot 0.236775) \cdot (0.95 (1-0.95))$ $+ 0.003374$ $\boxed{= 0.009858}$

2. $\tanh(x) = \frac{\sinh(z)}{\cosh(z)} = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$ $\quad \frac{\partial}{\partial z} \frac{\sinh(z)}{\cosh(z)} = \frac{\partial/\partial z \sinh \times \cosh - \partial/\partial z \cosh \cdot \sinh}{\cosh^2}$

$= \frac{\cosh^2 - \sinh^2}{\cosh^2} = 1 - \frac{\sinh^2}{\cosh^2} = \boxed{1-\tanh^2}$

so $\boxed{1 - f(z)^2}$

3. let $D_j S_i$ be a Jacobian matrix $R^n \rightarrow R^n$ look for $\frac{\partial s_i}{\partial a_j}$

redefine softmax as.

$$D_j S_i = \frac{\partial s_i}{\partial a_1} = \frac{\partial \frac{e^{a_i}}{\sum_{k=1}^{N} e^{a_k}}}{\partial a_j}$$

use quotient rule:

$g_i = e^{a_i}$

$h_i = \sum_{i=1}^{N} e^{a_i}$

for g_i

case A: if $i=j$ derivative is e^{a_j} else 0

h_i always e^{a_j}

case: $i = j$

$$\frac{\partial \dfrac{e^{a_i}}{\sum_{k=1}^{N} e^{a_k}}}{\partial a_j} = \frac{e^{a_i}\sum_{k=1}^{N} e^{a_k} - e^{a_j} e^{a_i}}{\sum_{k=1}^{N} {e^{a_k}}^{2}} = \frac{e^{a_i}\sum_{k=1}^{N} e^{a_k} - e^{a_j} e^{a_i}}{\left(\sum_{k=1}^{N} e^{a_k}\right)^2}$$

$$= \frac{e^{a_i}}{\sum_{k=1}^{N} e^{a_k}} \cdot \frac{\sum_{k=1}^{N} e^{a_k} - e^{a_j}}{\sum_{k=1}^{N} e^{a_k} - 0} = \boxed{S_i(1 - S_j)}$$

case $i \neq j$

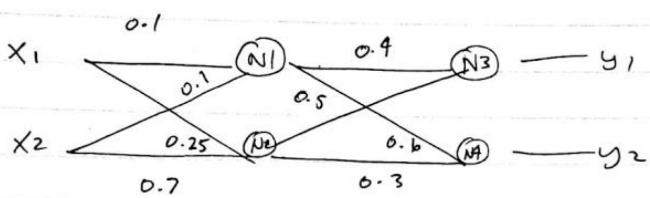
$$\frac{\partial \dfrac{e^{a_i}}{\sum_{k=1}^{N} e^{a_k}}}{\partial a_j} = \frac{\overset{0}{\cancel{e^{a_i}\sum_{k=1}^{N} e^{a_k}}} - e^{a_j} e^{a_i}}{\left(\sum_{k=1}^{N} e^{a_k}\right)^2}$$

$$= \frac{-e^{a_j} e^{a_i}}{\left(\sum_{k=1}^{N} e^{a_k}\right)^2} \Rightarrow \frac{-e^{a_j}}{\sum_{k=1}^{N} e^{a_k}} \frac{e^{a_i}}{\sum_{k=1}^{N} e^{a_k}} = \boxed{-S_j S_i}$$

$$\boxed{\text{so,} \quad D_j S_i = \begin{cases} S_i(1 - S_j) & i = j \\ -S_j S_i & i \neq j \end{cases}}$$

- Onlnie versus batch learning.
- batch can bring in bias and generalize too much.
- online uses better with functions of time. for sample based updates.
- online is faster, no need to store gradients and r/w operation
- batch avergages and can be stuck in local mihima.

$x = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \quad \begin{bmatrix} 0.2 \\ 0.95 \end{bmatrix}$



$w = w + (\eta * \text{error} * \text{input})$

$w_{11} = 0.1 \rightarrow 0.1 + (0.1 * 0.0396 * 1) = 0.10396$

$w_{12} = 0.25 \rightarrow 0.25 + (0.1 * 0.0099 * 1) = 0.25099$

$w_{21} = 0.1 \rightarrow 0.1 + (0.1 * 0.0396 * 1) = 0.10396$

$w_{22} = 0.7 \rightarrow 0.7 + (0.1 * 0.0099 * 1) = 0.70010$

$w_{N1N3} = 0.4 \rightarrow 0.6 + (0.1 * 0.2370 * 0.2) = 0.6047$

$w_{N1N4} = 0.5 \rightarrow 0.4 + (0.1 * 0.2375 * 0.2) = 0.5475$

$w_{N2N3} = 0.6 \rightarrow 0.6 + (0.1 * 0.95 * 0.2275) = 0.6216$

$w_{N2N4} = 0.3 \rightarrow 0.3 + (0.1 * 0.2370 * 0.95) = 0.6215$

tput?