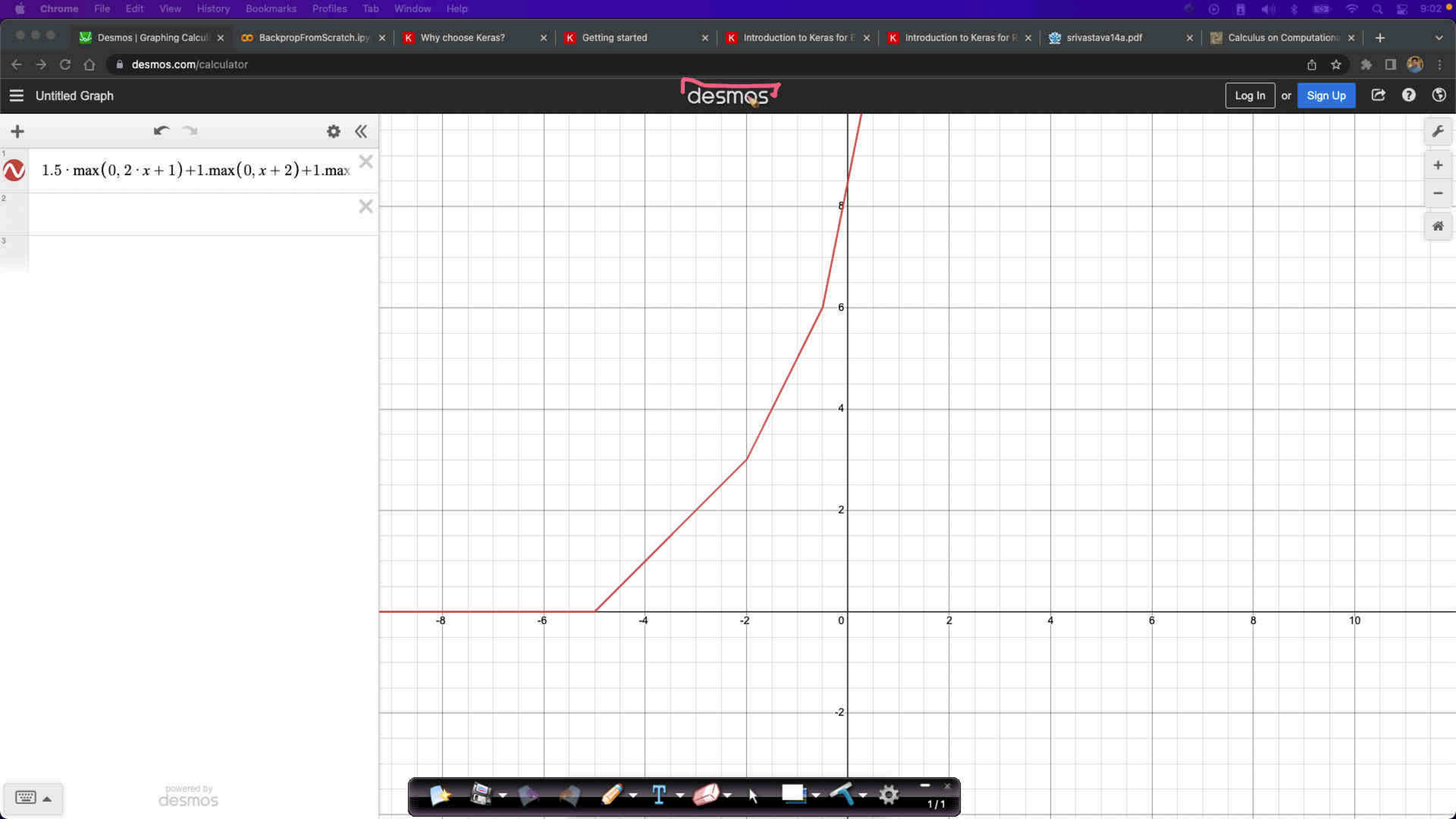
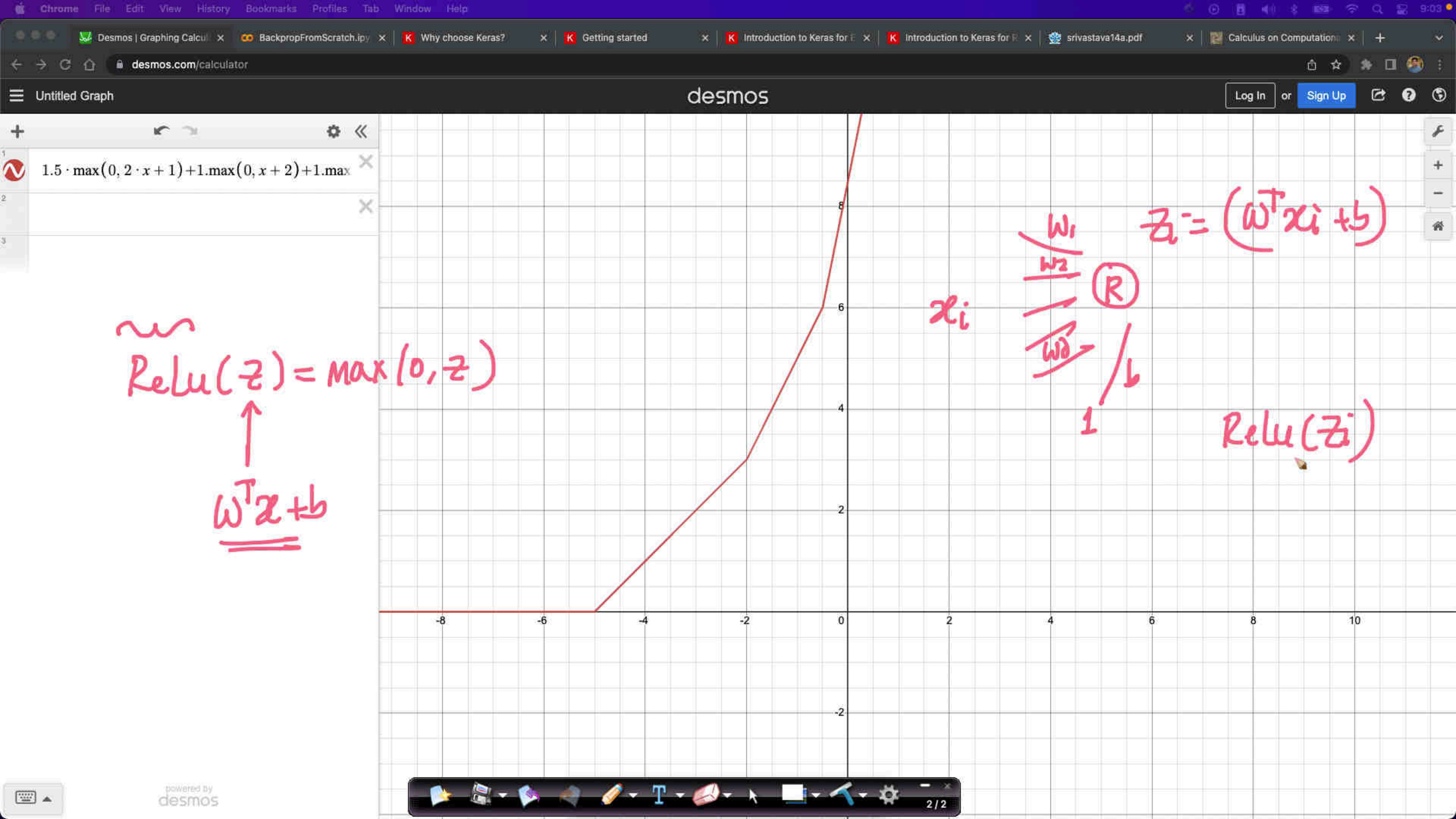

Chrome File Edit View History Bookmarks Profiles Tab Window Help
Desmos | Graphing Calcul
BackpropFromScratch.ipy
Why choose Keras?
Getting started
Introduction to Keras for F
Introduction to Keras for R
srivastava14a.pdf
Calculus on Computation
desmos.com/calculator
Untitled Graph
desmos
Log In
or
Sign Up
1
1.5 · max(0, 2 · x + 1)+1.max(0, x + 2)+1.max
2
3
8
6
4
2
-2
-8
-6
-4
-2
0
2
4
6
8
10
1/1
powered by
desmos


Chrome File Edit View History Bookmarks Profiles Tab Window Help
Desmos | Graphing Calcul
BackpropFromScratch.ipy
Why choose Keras?
Getting started
Introduction to Keras for
Introduction to Keras for
srivastava14a.pdf
Calculus on Computation
desmos.com/calculator
Untitled Graph
desmos
Log In
or
Sign Up
1.5 · max(0, 2 · x + 1)+1.max(0, x + 2)+1.max
Relu(z) = max(0, z)
W^T x + b
x_i
w1
w2
w3
b
1
R
z_i = (W^T x_i + b)
Relu(z_i)
powered by desmos
2 / 2

Chrome File Edit View History Bookmarks Profiles Tab Window Help

Desmos | Graphing Calcul | BackpropFromScratch.ipy | Why choose Keras? | Getting started | Introduction to Keras for F | Introduction to Keras for R | srivastava14a.pdf | Calculus on Computation

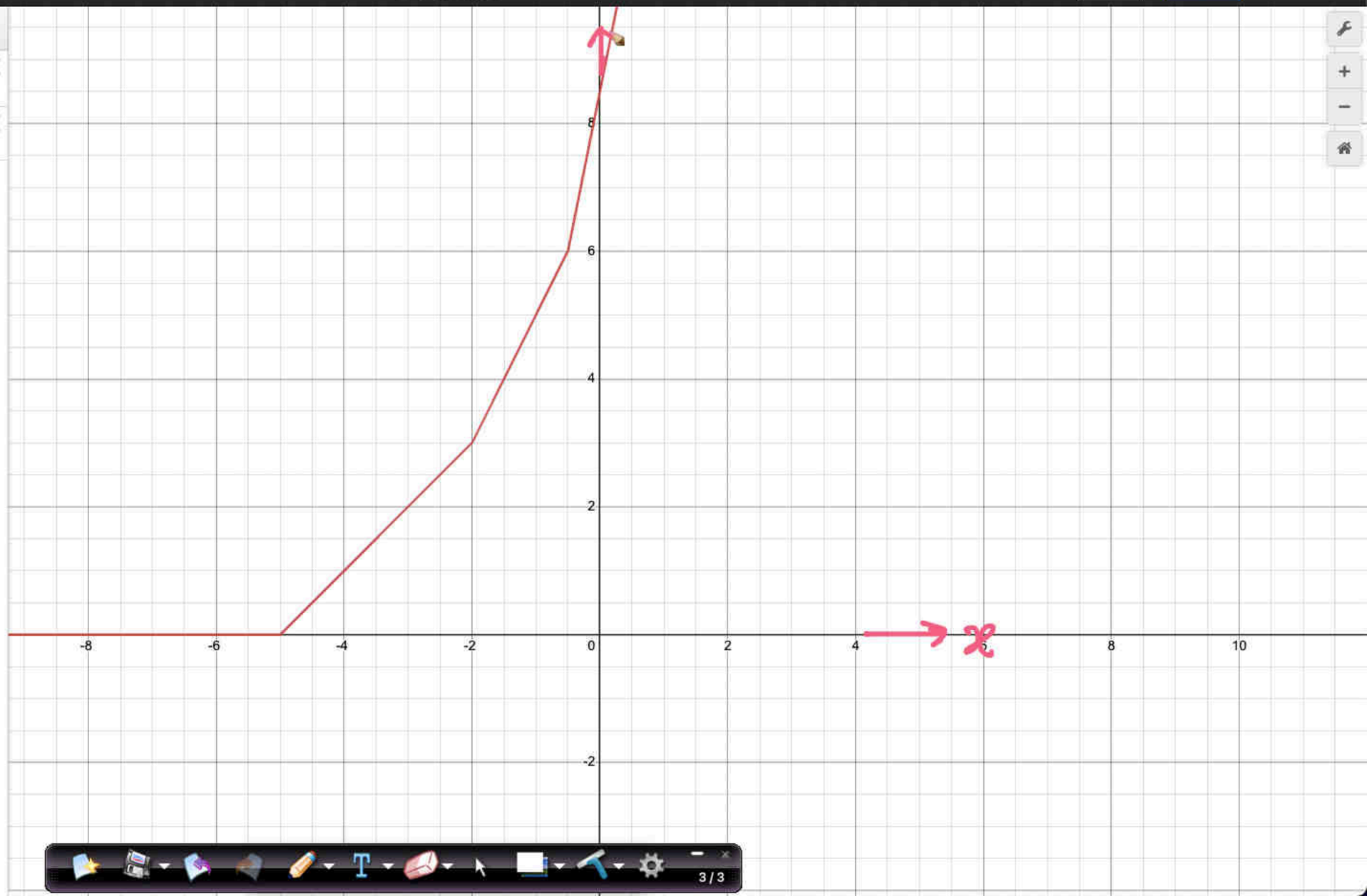
desmos.com/calculator

Untitled Graph

desmos

Log In or Sign Up

$1.5 \cdot \max(0, 2 \cdot x + 1) + 1.\max(0, x + 2) + 1.\max$



3 / 3

powered by desmos

Chrome File Edit View History Bookmarks Profiles Tab Window Help

Desmos | Graphing Calcul | BackpropFromScratch.ipy | Why choose Keras? | Getting started | Introduction to Keras for F | Introduction to Keras for R | srivastava14a.pdf | Calculus on Computation

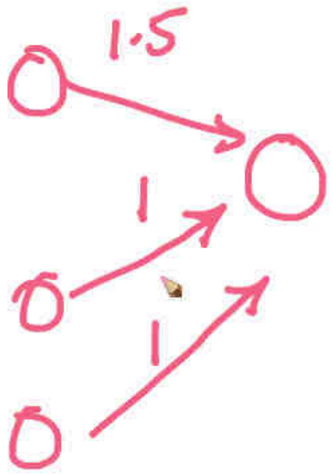
desmos.com/calculator

Untitled Graph

desmos

Log In or Sign Up

$1.5 \cdot \max(0, 2 \cdot x + 1) + 1.\max(0, x + 2) + 1.\max$



4 / 4

powered by desmos

Chrome File Edit View History Bookmarks Profiles Tab Window Help

Desmos | Graphing Calcul | BackpropFromScratch.ipy | Why choose Keras? | Getting started | Introduction to Keras for F | Introduction to Keras for R | srivastava14a.pdf | Calculus on Computation

desmos.com/calculator

Untitled Graph

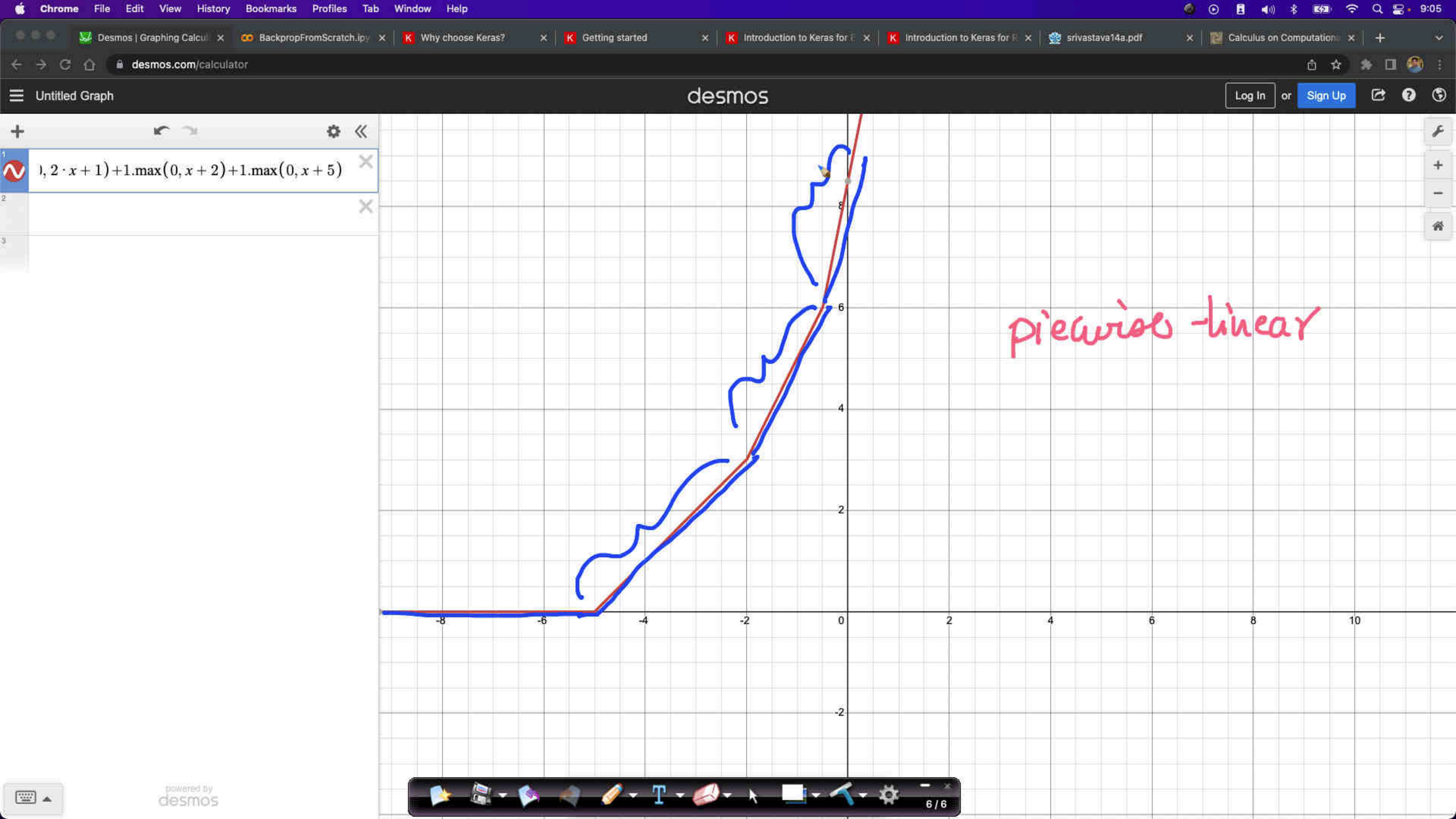
desmos

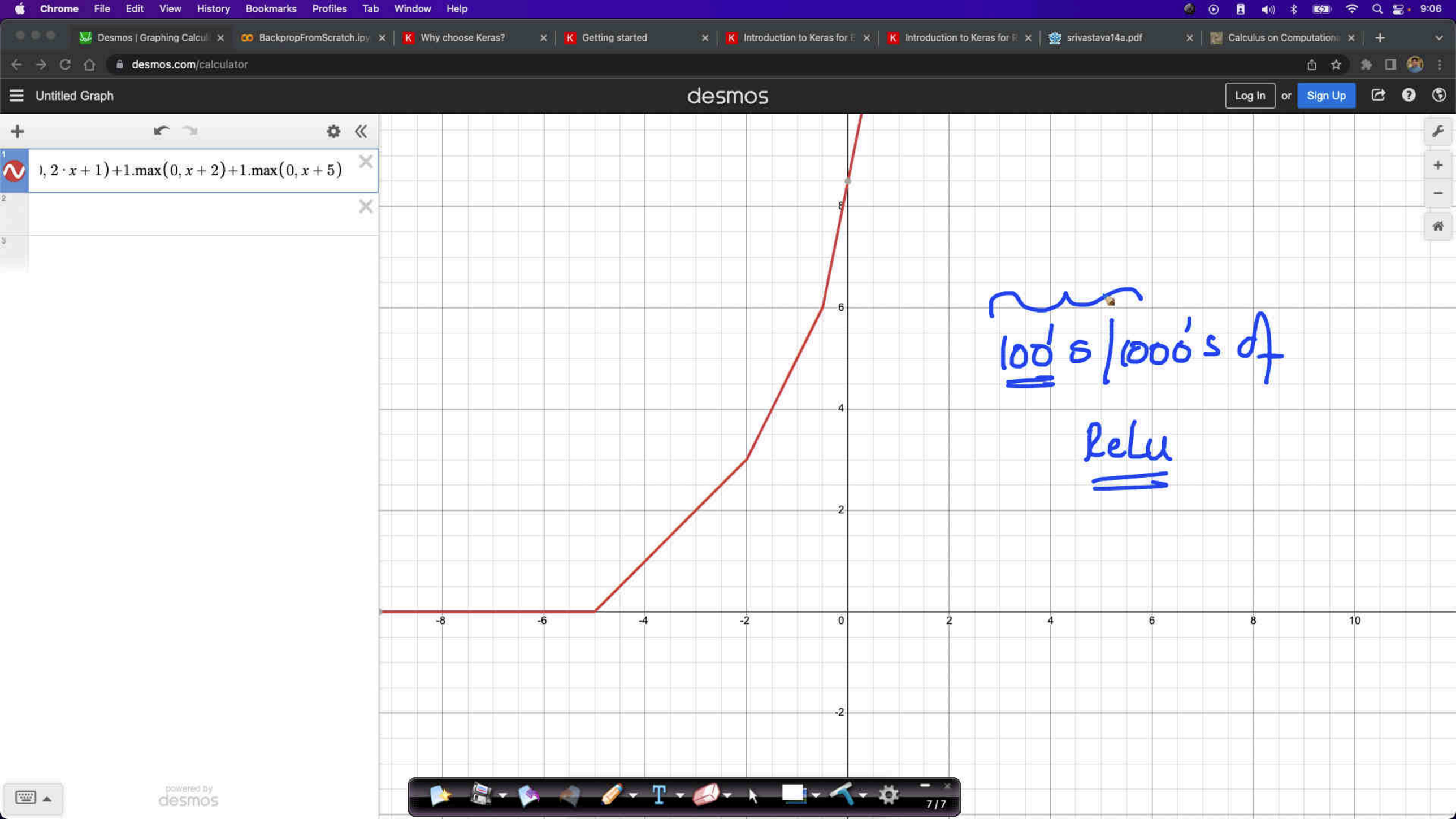
Log In or Sign Up

$1.5 \cdot \max(0, 2 \cdot x + 1) + 1.\max(0, x + 2) + 1.\max$

5 / 5

powered by desmos

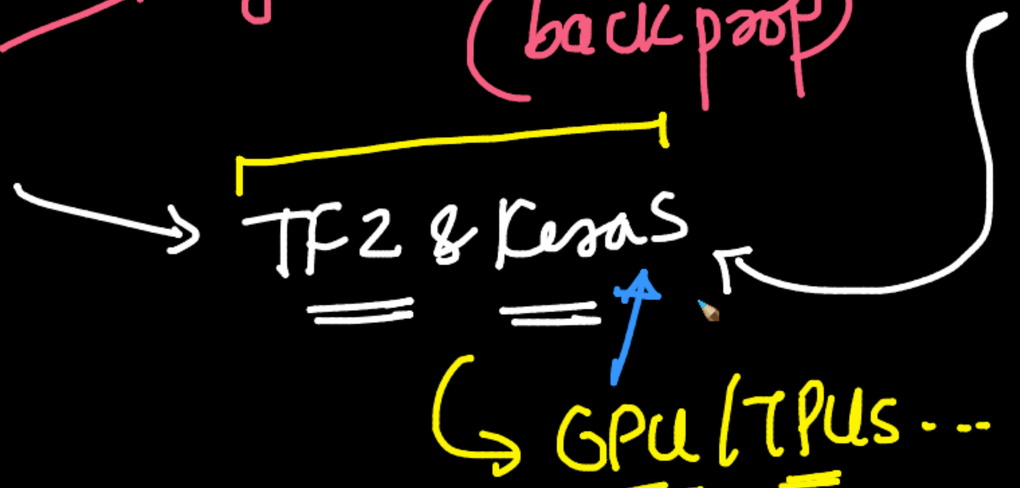

Chrome File Edit View History Bookmarks Profiles Tab Window Help
9:05
Desmos | Graphing Calcul
BackpropFromScratch.ipy
Why choose Keras?
Getting started
Introduction to Keras for
Introduction to Keras for
srivastava14a.pdf
Calculus on Computation
desmos.com/calculator
Untitled Graph
desmos
Log In
or
Sign Up
), 2 · x + 1) + 1.max(0, x + 2) + 1.max(0, x + 5)
piecewise - linear
6 / 6
powered by desmos


Chrome File Edit View History Bookmarks Profiles Tab Window Help
Desmos | Graphing Calcul
BackpropFromScratch.ipy
Why choose Keras?
Getting started
Introduction to Keras for
Introduction to Keras for
srivastava14a.pdf
Calculus on Computation
desmos.com/calculator
Untitled Graph
desmos
Log In
or
Sign Up
), 2 · x + 1) + 1.max(0, x + 2) + 1.max(0, x + 5)
100s/1000's of
Relu
powered by desmos
7/7

Chrome File Edit View History Bookmarks Profiles Tab Window Help 9:09

Agenda:

1. Multi-class classfn: (softmax)
2. CE. loss (generalization of log-loss)
3. Code → Python → 3-class classifier (backprop)



Chrome File Edit View History Bookmarks Profiles Tab Window Help 9:08

4 Dropout

Chrome File Edit View History Bookmarks Profiles Tab Window Help 9:20

Binary - Case

$y_i \in \{0, 1\}$

$$\begin{cases} w^T x_i + b = z_i \\ \text{Sigmoid}(z_i) = P(y_i = 1 \mid x_i, w, b) = p_i \end{cases}$$

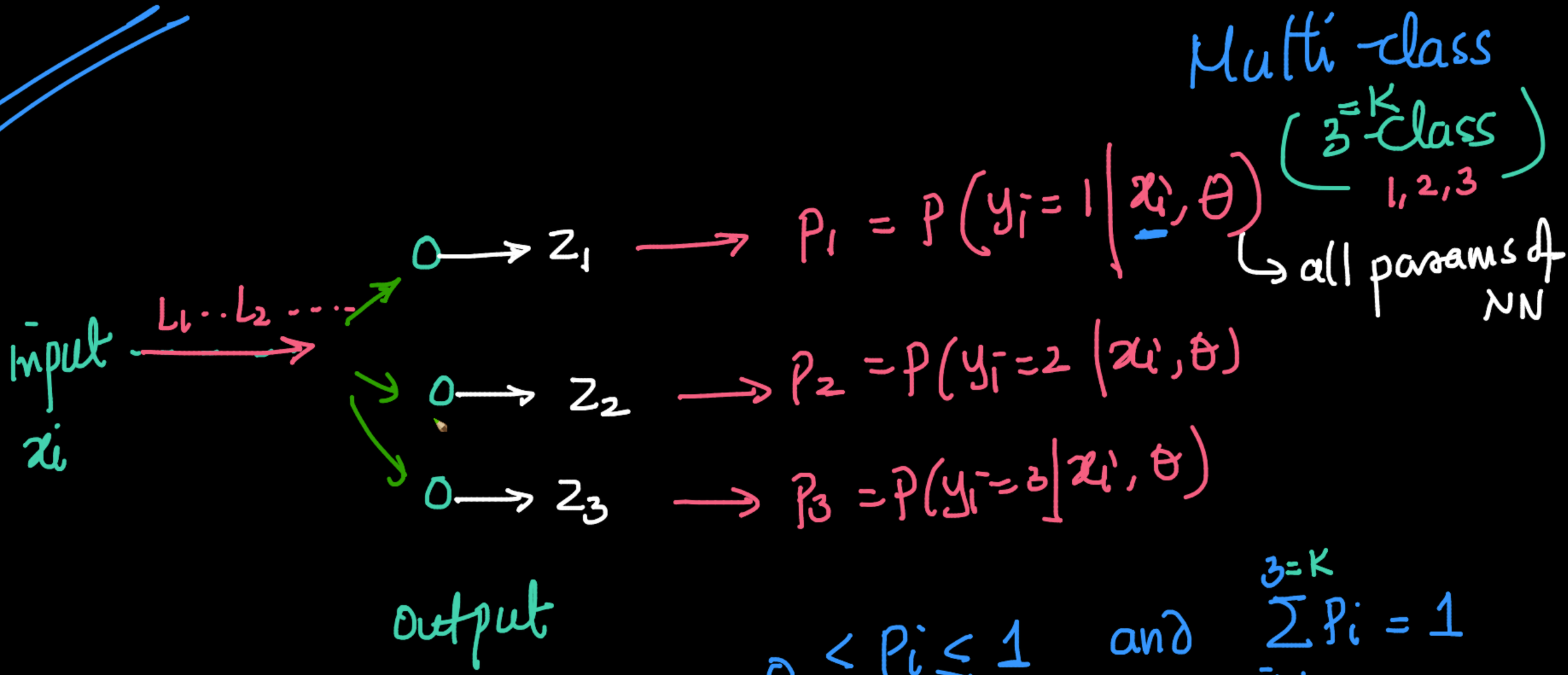
$$\hookrightarrow \frac{1}{1 + e^{-z_i}} = \frac{e^{z_i}}{1 + e^{z_i}}$$

$$P(y_i = 0 \mid x_i, w, b) = 1 - p_i$$

Log-loss: (Binary CE)

$$= -\left[y_i \log \hat{y}_i + (1-y_i) \log(1-\hat{y}_i) \right]$$

$\hat{y}_i \rightarrow p_i$


Multi-class
(3=K class)
1, 2, 3
input
x_i
$L_1 \cdot\cdot L_2 \cdots$
$o \rightarrow z_1 \rightarrow p_1 = P(y_i = 1 \mid x_i, \theta)$
all params of NN
$o \rightarrow z_2 \rightarrow p_2 = P(y_i = 2 \mid x_i, \theta)$
$o \rightarrow z_3 \rightarrow p_3 = P(y_i = 3 \mid x_i, \theta)$
output
$0 \leq p_i \leq 1$ and $\sum_{i=1}^{3=K} p_i = 1$
$\forall i : 1 \rightarrow K$

[Softmax:] →

$$p_1 = \frac{e^{z_1}}{\sum_{i=1}^{K} e^{z_i}}$$

$$p_2 = \frac{e^{z_2}}{\sum_{i=1}^{K} e^{z_i}}$$

$$p_3 = \frac{e^{z_3}}{\sum_{i=1}^{K} e^{z_i}}$$

$0 \leq p_i \leq 1$

if $K=3$;

$$p_1 + p_2 + p_3 = 1$$

softmax ≈ Sigmoid -like fn for multi-class Classfn..

Q

$$p_i = \frac{z_i}{\sum_{i=1}^{n} z_i} \qquad -\infty \leq z_i \leq \infty \}$$

$z_i : \quad 2 \qquad 6 \qquad 8$

$e^{z_i} : \quad e^{2} \qquad e^{6} \qquad 2^{8}$

$\hookrightarrow p_i : 0.662178$

Chrome File Edit View History Bookmarks Profiles Tab Window Help
Desmos | Graphing C | BackpropFromScratch | Why choose Keras? | Getting started | Introduction to Keras | Introduction to Keras | srivastava14a.pdf | Calculus on Computa | Softmax function Calc
keisan.casio.com/exec/system/15168444286206

Calculates the softmax function.

The softmax function is used in the activation function of the neural network.

vector	a
a_1	2
a_2	6
a_3	12

Execute | Clear | Store/Read | Print | 14digit

Softmax function

b	value
b_1	4.5285621964412E-5
b_2	0.0024725111823572
b_3	0.99748220319568

Softmax function $\sigma(\mathbf{z})_j$

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \text{ for } j = 1, \cdots, K.$$

Bookmarks
Not registered.
+ Bookmark

History
- Softmax function

Related Calculator
- Sigmoid function
- Derivative Sigmoid function
- Second Derivative Sigmoid function
- Sigmoid function (chart)
- Softsign function
- Derivative Softsign function
- Softsign function (chart)
- Softplus function
- Derivative Softplus function
- Softplus function (chart)
- Softmax function
- ReLU
- ReLU (chart)
- Leaky ReLU
- Leaky ReLU (chart)
- tanh(x) function

17 / 17

(Q) Why e^z are common

✓ $\left\{ \frac{\partial e^z}{\partial z} = e^z \right.$

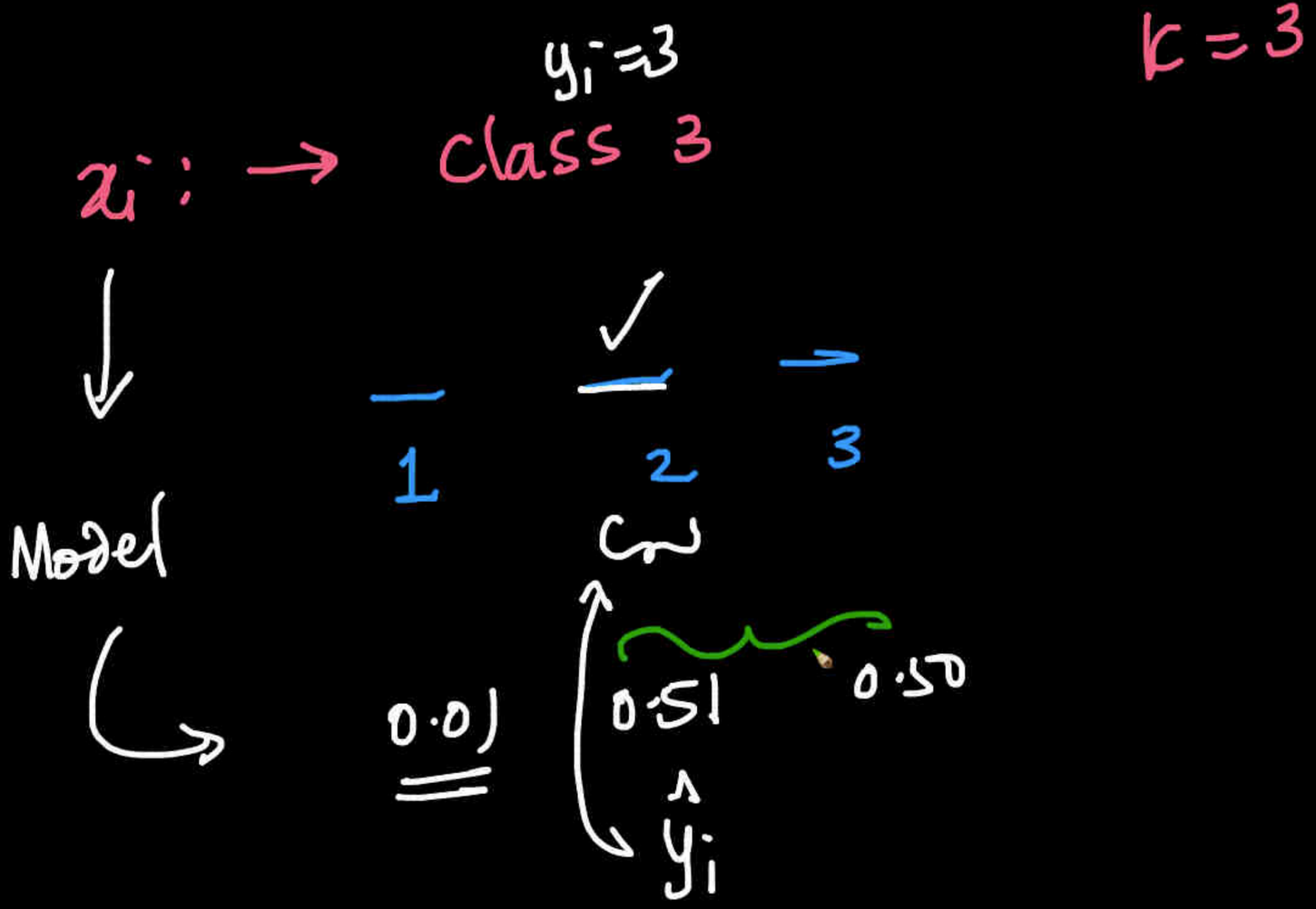
Chrome File Edit View History Bookmarks Profiles Tab Window Help 9:29

Q

Binary: $\bigcirc \longrightarrow z_i \longrightarrow P(y_i = 1) = p_1$

$p_0 = 1 - p_1$

k-ary $\bigcirc \longrightarrow 1, 2, 3, \ldots K$ (integer?) X


$y_i = 3$
x_i : → Class 3
$K = 3$
1
2
3
Model
0.01
0.51
0.50
$\hat{y}_i$

Chrome File Edit View History Bookmarks Profiles Tab Window Help 9:33

Log-loss in Binary Classfn.

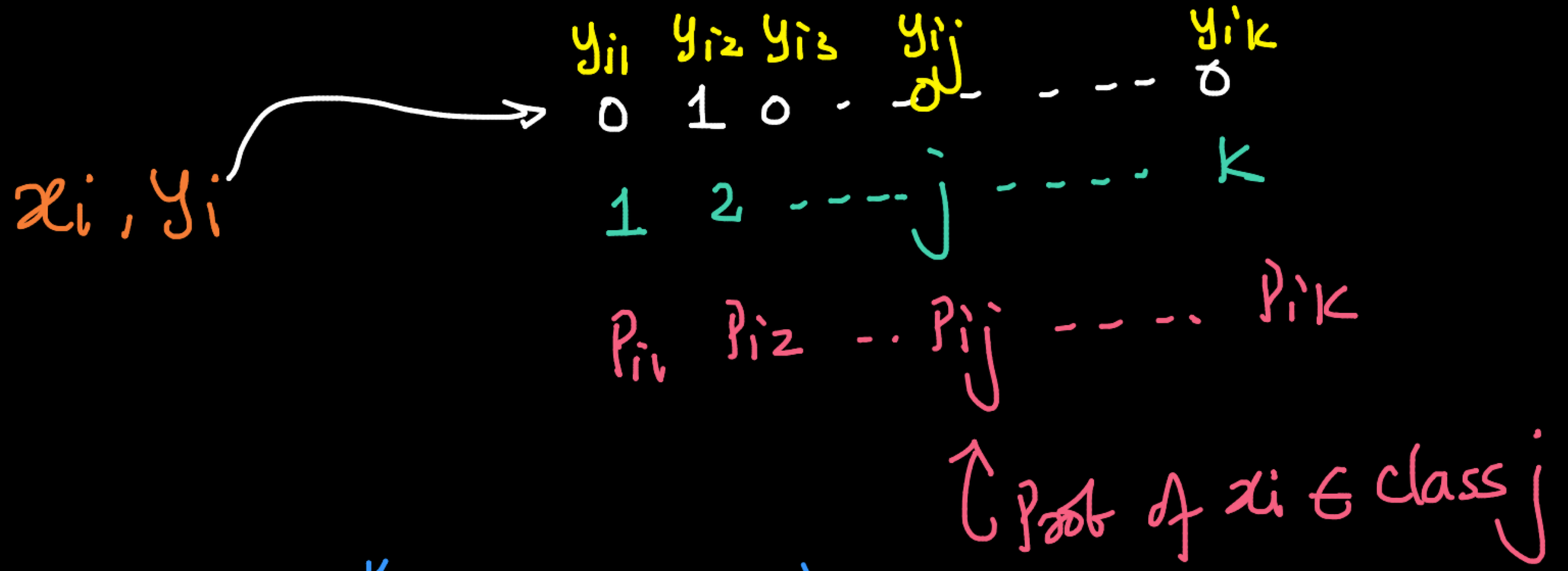
Multi-class Classfn

(Cross-Entropy)

P_{i1} P_{i2} P_{ij} ... P_{ik}

$x_i \rightarrow 1 \quad 2 \quad \cdots \quad j \quad \cdots \quad K$ classes

$\hookrightarrow P_{ij} \quad \forall j = 1 \rightarrow k$



$$CE_i = -\sum_{j=1}^{K} y_{ij} \log(P_{ij})$$

$$CE_i = -\sum_{j=1}^{K} y_{ij} \log P_{ij}$$

$y_i \in$ class 1

let $K = 2$

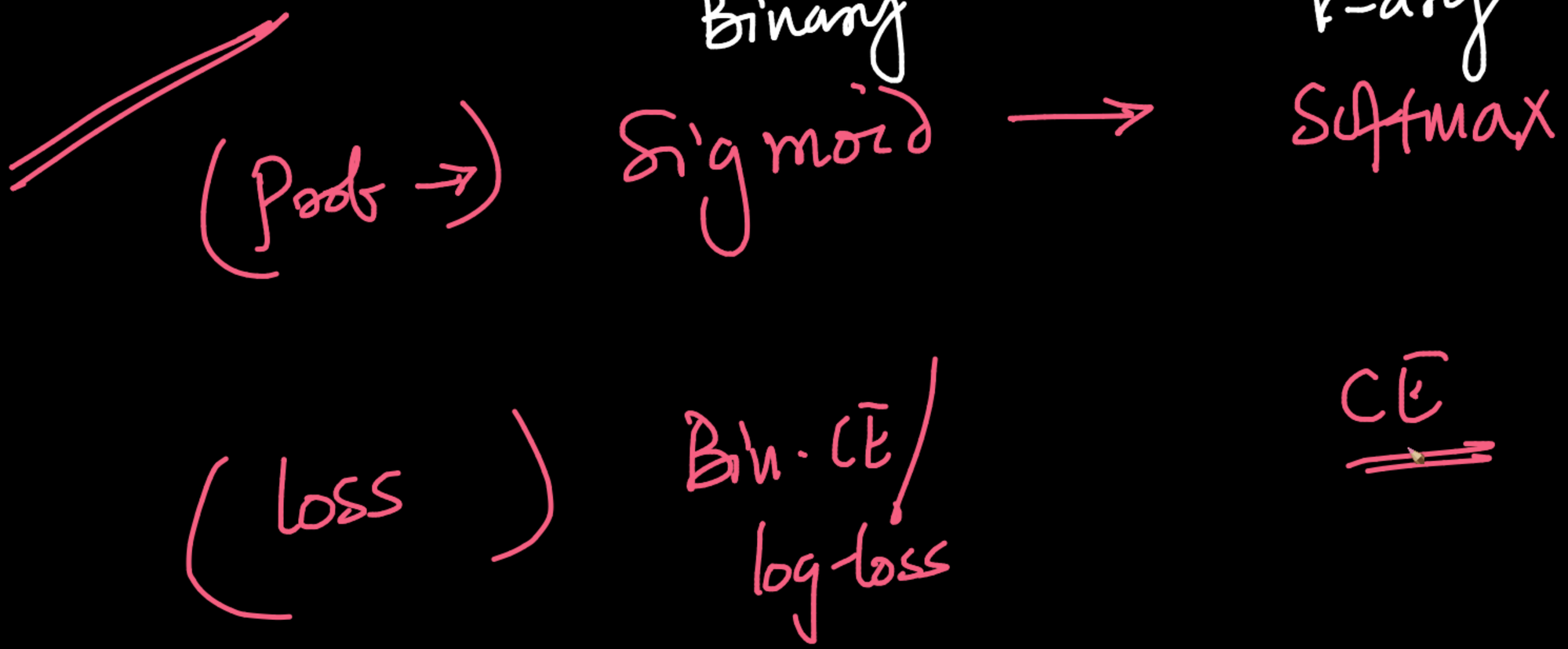
$$= -\left[y_{i1} \log P_{i1} + y_{i2} \log P_{i2} \right]$$

y_{i1}: 1

y_{i2}: $(1 - y_{i1})$

P_{i2}: $(1 - P_{i1})$

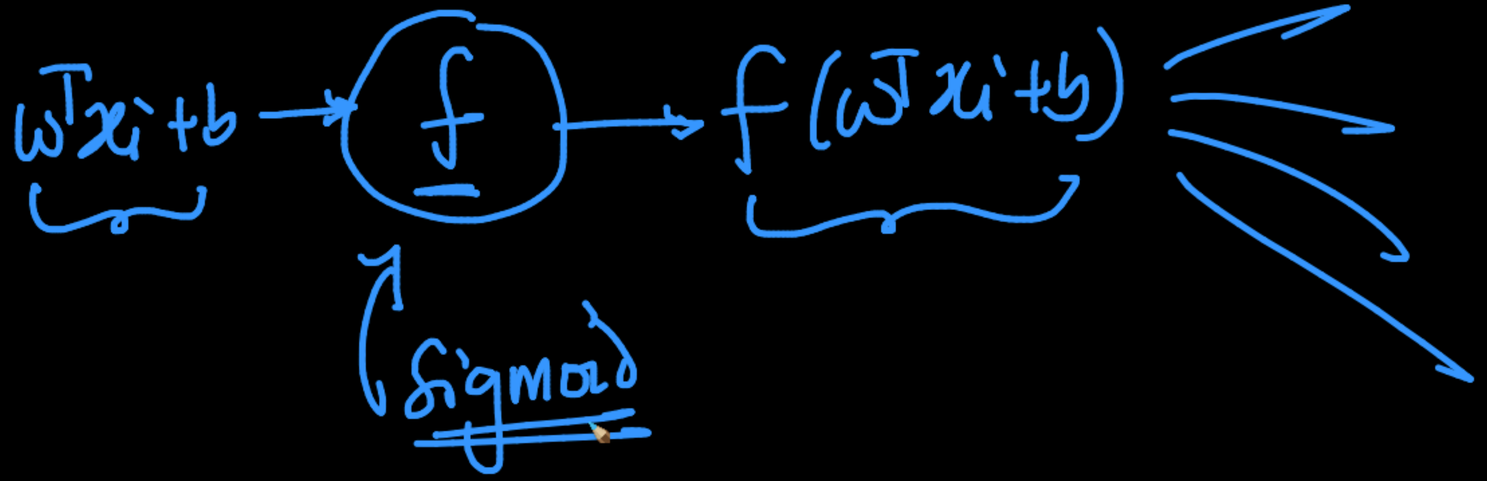
log-loss = Binary CE


Binary
k-ary
(Probs →)
Sigmoid
Softmax
(loss)
Bin. CE/
log loss
CE

Chrome File Edit View History Bookmarks Profiles Tab Window Help 9:47

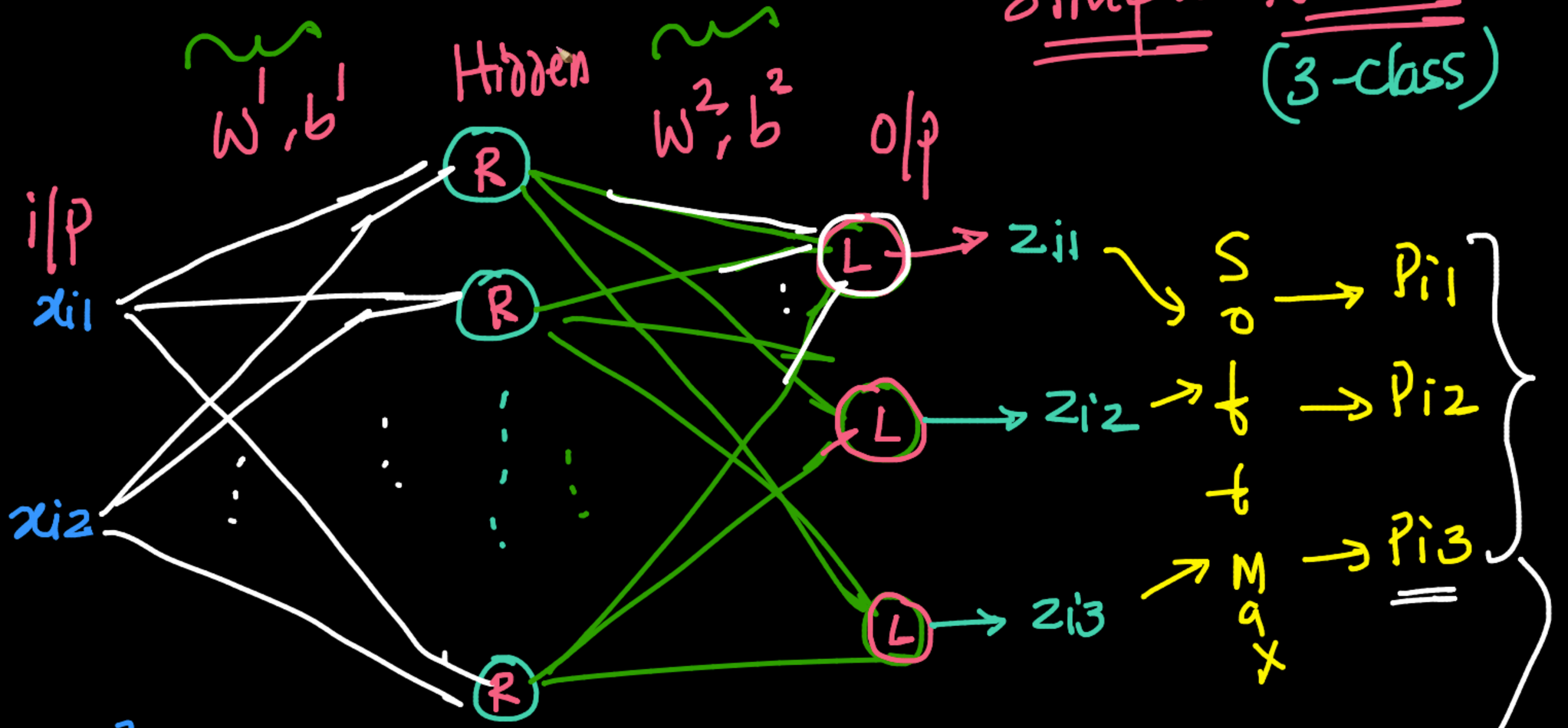
Q

z_1
z_2
z_3
→ softmax as actun-fn →
p_1
p_2
p_3



Chrome File Edit View History Bookmarks Profiles Tab Window Help 9:48

Simple-Network (3-class)



$x_q \in \mathbb{R}^2$

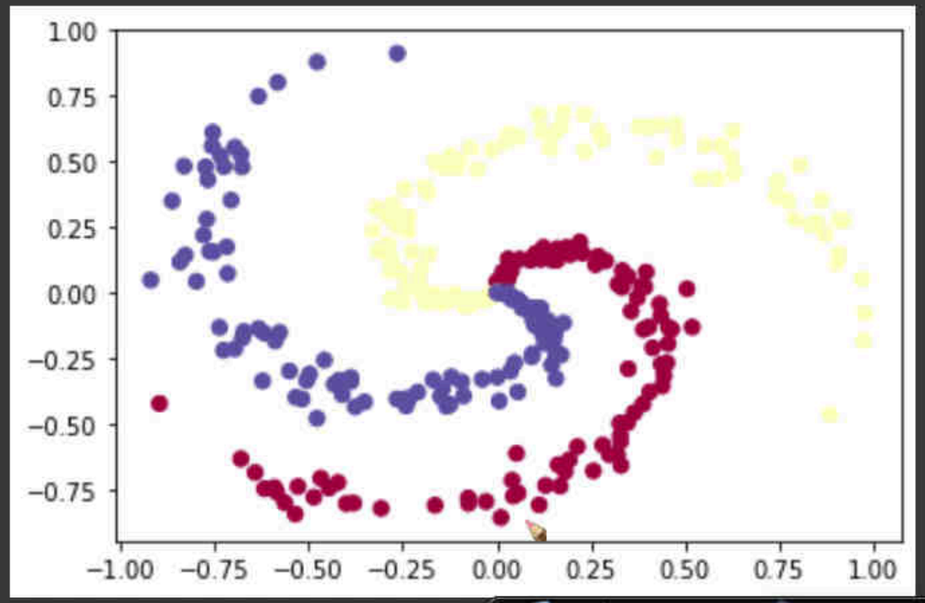
$x_i \in \mathbb{R}^2$

$L_{CE}(y_i^{OHE}, P_{ij})$

27 / 33

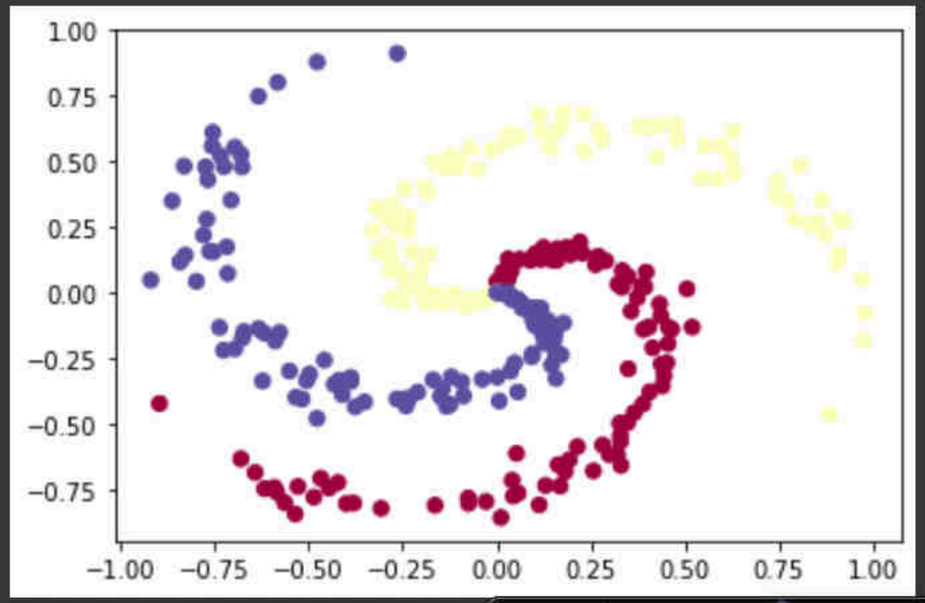
```
-rw-r--r-- 1 root root 12867 Sep  5 16:14 spiral.csv
```

```python
# Very simple case of a back-prop
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
df = pd.read_csv("spiral.csv")
plt.scatter(df["x1"], df["x2"], c=df["y"], s=40, cmap=plt.cm.Spectral)
plt.show()
X = df.iloc[:, :-1].to_numpy()
y = df.iloc[:, -1].to_numpy()
```



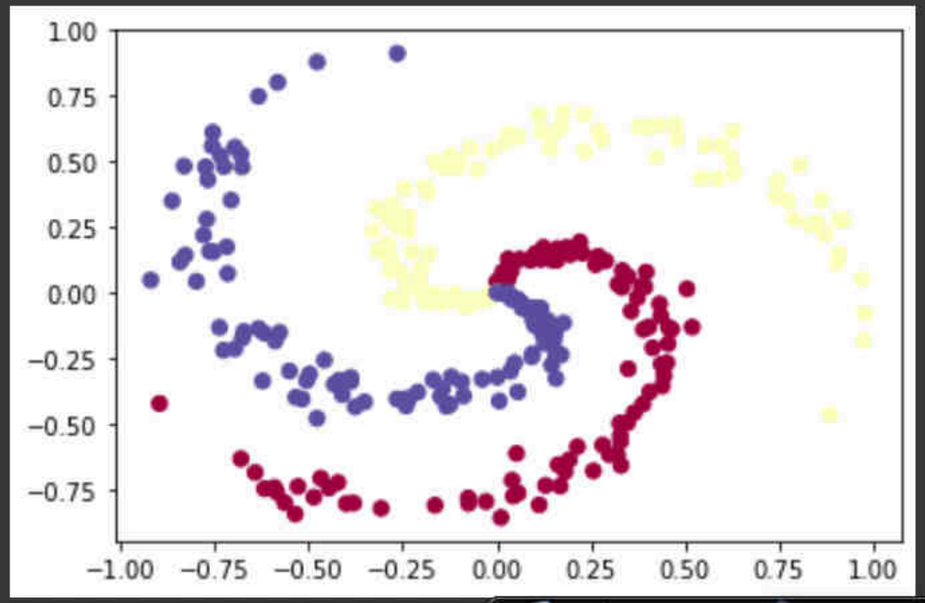
```
-rw-r--r-- 1 root root 12867 Sep  5 16:14 spiral.csv
```

```python
# Very simple case of a back-prop
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
df = pd.read_csv("spiral.csv")
plt.scatter(df["x1"], df["x2"], c=df["y"], s=40, cmap=plt.cm.Spectral)
plt.show()
X = df.iloc[:, :-1].to_numpy()
y = df.iloc[:, -1].to_numpy()
```



```
-rw-r--r-- 1 root root 12867 Sep  5 16:14 spiral.csv
```

```python
# Very simple case of a back-prop
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
df = pd.read_csv("spiral.csv")
plt.scatter(df["x1"], df["x2"], c=df["y"], s=40, cmap=plt.cm.Spectral)
plt.show()
X = df.iloc[:, :-1].to_numpy()
y = df.iloc[:, -1].to_numpy()
```



Chrome File Edit View History Bookmarks Profiles Tab Window Help 9:47

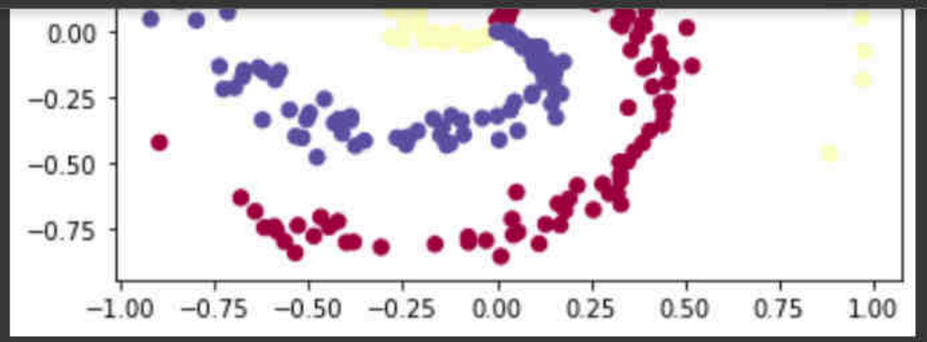
Desmos | Graphing C | BackpropFromScratch | Why choose Keras? | Getting started | Introduction to Keras | Introduction to Keras | srivastava14a.pdf | Calculus on Computa | Softmax function Calc

colab.research.google.com/drive/1vmwq6vCH4F42G9MoYzjcAWOx8rm97z4i#scrollTo=TFuRIHJrVLnk

+ Code + Text

RAM
Disk

[ ]



[ ]
```
df.head()
```

	x1	x2	y
0	0.000000	0.000000	0
1	-0.000650	0.010080	0
2	0.009809	0.017661	0
3	0.007487	0.029364	0
4	-0.000027	0.040404	0

[ ]
```
class NN:

    def __init__(self, n_features, n_hidden, n_classes):
```

32 / 32

Chrome File Edit View History Bookmarks Profiles Tab Window Help 9:47

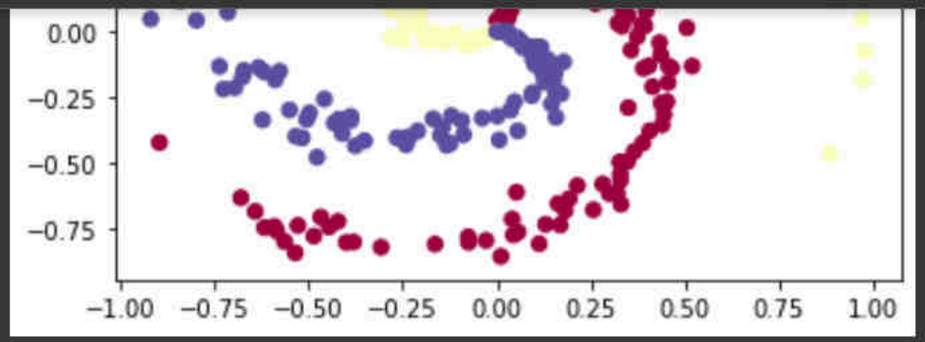
Desmos | Graphing C | BackpropFromScratch | Why choose Keras? | Getting started | Introduction to Keras | Introduction to Keras | srivastava14a.pdf | Calculus on Computa | Softmax function Cal

colab.research.google.com/drive/1vmwq6vCH4F42G9MoYzjcAWOx8rm97z4i#scrollTo=TFuRIHJrVLnk

+ Code + Text

RAM
Disk

[ ]



[ ] df.head()

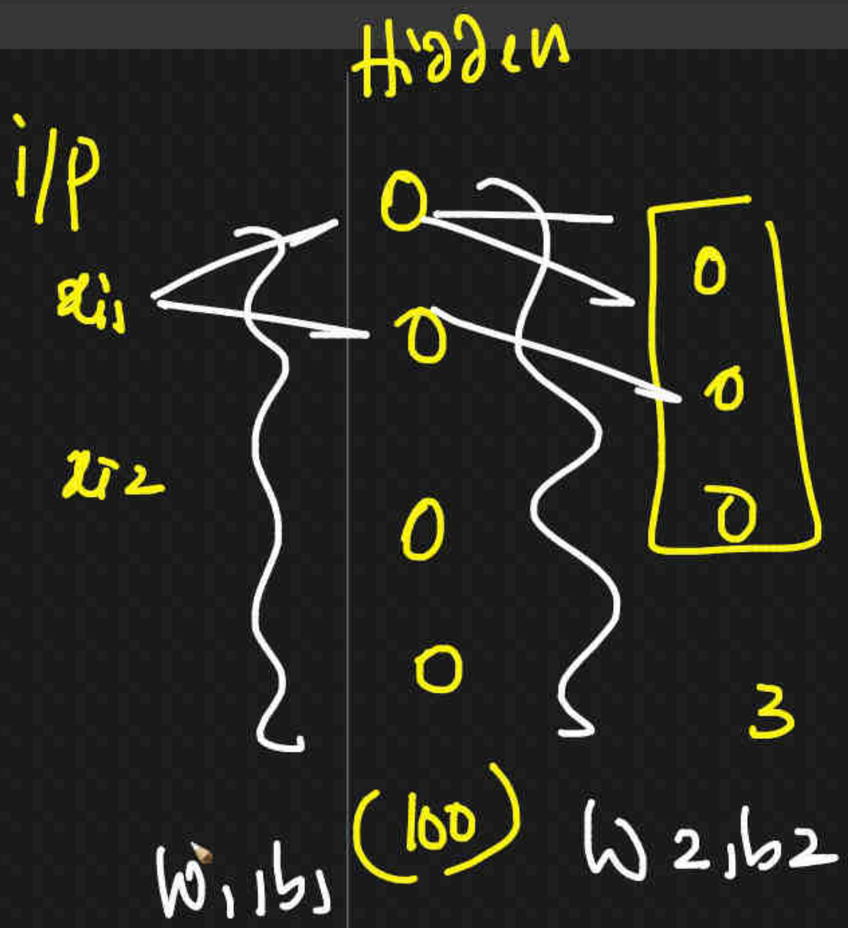
	x1	x2	y
0	0.000000	0.000000	0
1	-0.000650	0.010080	0
2	0.009809	0.017661	0
3	0.007487	0.029364	0
4	-0.000027	0.040404	0

[ ]

```
class NN:

    def __init__(self, n_features, n_hidden, n_classes):
        self.d = n_featu
```

32 / 32

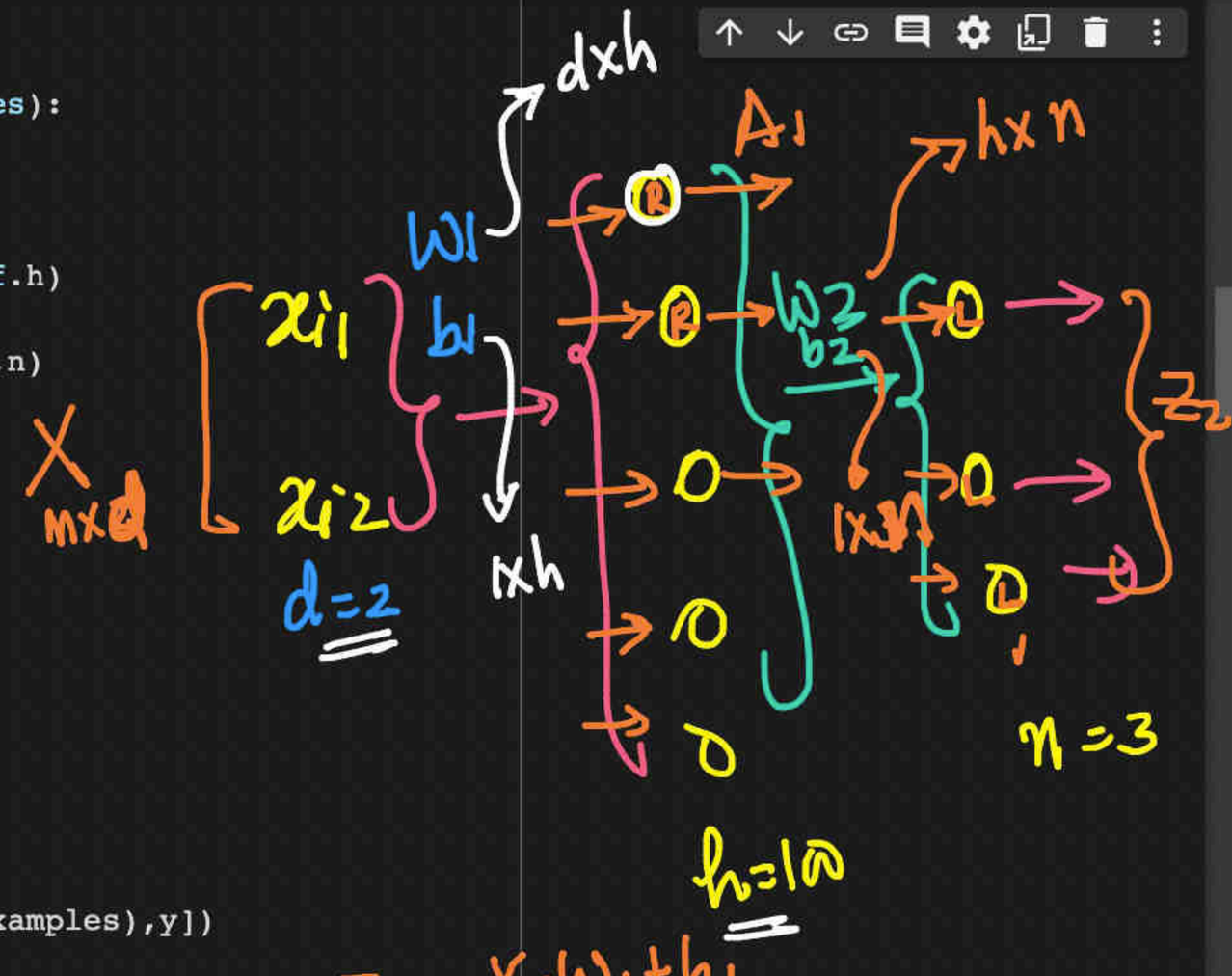
```
[ ]  class NN:

       def __init__(self, n_features, n_hidden, n_classes):
           self.d = n_features
           self.h = n_hidden
           self.n = n_classes
           self.W1 = 0.01 * np.random.randn(self.d, self.h)
           self.b1 = np.zeros((1,self.h))
           self.W2 = 0.01 * np.random.randn(self.h,self.n)
           self.b2 = np.zeros((1,self.n))

       def fwd_prop(self, X):
           Z1 = np.dot(X, self.W1) + self.b1
           A1 = np.maximum(0, Z1)
           Z2 = np.dot(A1, self.W2) + self.b2
           Z2 = np.exp(Z2)
           A2 = Z2 / np.sum(Z2, axis=1, keepdims=True)
           return A1, A2

       def cce_loss(self, y, probs):
           num_examples = y.shape[0]
           correct_logprobs = -np.log(probs[range(num_examples),y])
           loss = np.sum(correct_logprobs)/num_examples
           return loss
```



```
[ ] class NN:

        def __init__(self, n_features, n_hidden, n_classes):
            self.d = n_features
            self.h = n_hidden
            self.n = n_classes
            self.W1 = 0.01 * np.random.randn(self.d, self.h)
            self.b1 = np.zeros((1,self.h))
            self.W2 = 0.01 * np.random.randn(self.h,self.n)
            self.b2 = np.zeros((1,self.n))

        def fwd_prop(self, X):
            Z1 = np.dot(X, self.W1) + self.b1
            A1 = np.maximum(0, Z1)
            Z2 = np.dot(A1, self.W2) + self.b2
            Z2 = np.exp(Z2)
            A2 = Z2 / np.sum(Z2, axis=1, keepdims=True)
            return A1, A2

        def cce_loss(self, y, probs):
            num_examples = y.shape[0]
            correct_logprobs = -np.log(probs[range(num_examples),y])
            loss = np.sum(correct_logprobs)/num_examples
            return loss
```

```
        self.n = n_classes
        self.W1 = 0.01 * np.random.randn(self.d, self.h)
        self.b1 = np.zeros((1,self.h))
        self.W2 = 0.01 * np.random.randn(self.h,self.n)
        self.b2 = np.zeros((1,self.n))

    def fwd_prop(self, X):
        Z1 = np.dot(X, self.W1) + self.b1
        A1 = np.maximum(0, Z1)
        Z2 = np.dot(A1, self.W2) + self.b2
        Z2 = np.exp(Z2)
        A2 = Z2 / np.sum(Z2, axis=1, keepdims=True)
        return A1, A2

    def cce_loss(self, y, probs):
        num_examples = y.shape[0]
        correct_logprobs = -np.log(probs[range(num_examples),y])
        loss = np.sum(correct_logprobs)/num_examples
        return loss

    def back_prop(self, X, A1, A2, y):
        # compute the gradient on scores
        num_examples = y.shape[0]
        dZ2 = A2
        dZ2[range(num_examples),y] -= 1
        dZ2 /= num_examples
        # first backprop
```

```python
class NN:

    def __init__(self, n_features, n_hidden, n_classes):
        self.d = n_features
        self.h = n_hidden
        self.n = n_classes
        self.W1 = 0.01 * np.random.randn(self.d, self.h)
        self.b1 = np.zeros((1,self.h))
        self.W2 = 0.01 * np.random.randn(self.h,self.n)
        self.b2 = np.zeros((1,self.n))

    def fwd_prop(self, X):
        Z1 = np.dot(X, self.W1) + self.b1
        A1 = np.maximum(0, Z1)
        Z2 = np.dot(A1, self.W2) + self.b2
        Z2 = np.exp(Z2)
        A2 = Z2 / np.sum(Z2, axis=1, keepdims=True)
        return A1, A2

    def cce_loss(self, y, probs):
        num_examples = y.shape[0]
        correct_logprobs = -np.log(probs[range(num_examples),y])
        loss = np.sum(correct_logprobs)/num_examples
        return loss

    def back_prop(self, X, A1, A2, y):
```



```python
class NN:

    def __init__(self, n_features, n_hidden, n_classes):
        self.d = n_features
        self.h = n_hidden
        self.n = n_classes
        self.W1 = 0.01 * np.random.randn(self.d, self.h)
        self.b1 = np.zeros((1,self.h))
        self.W2 = 0.01 * np.random.randn(self.h,self.n)
        self.b2 = np.zeros((1,self.n))

    def fwd_prop(self, X):
        Z1 = np.dot(X, self.W1) + self.b1
        A1 = np.maximum(0, Z1)
        Z2 = np.dot(A1, self.W2) + self.b2
        Z2 = np.exp(Z2)
        A2 = Z2 / np.sum(Z2, axis=1, keepdims=True)
        return A1, A2

    def cce_loss(self, y, probs):
        num_examples = y.shape[0]
        correct_logprobs = -np.log(probs[range(num_examples),y])
        loss = np.sum(correct_logprobs)/num_examples
        return loss

    def back_prop(self, X, A1, A2, y):
```

```
def fwd_prop(self, X):
    Z1 = np.dot(X, self.W1) + self.b1
    A1 = np.maximum(0, Z1)
    Z2 = np.dot(A1, self.W2) + self.b2
    Z2 = np.exp(Z2)
    A2 = Z2 / np.sum(Z2, axis=1, keepdims=True)
    return A1, A2

def cce_loss(self, y, probs):
    num_examples = y.shape[0]
    correct_logprobs = -np.log(probs[range(num_examples),y])
    loss = np.sum(correct_logprobs)/num_examples
    return loss

def back_prop(self, X, A1, A2, y):
    # compute the gradient on scores
    num_examples = y.shape[0]
    dZ2 = A2
    dZ2[range(num_examples),y] -= 1
    dZ2 /= num_examples
    # first backprop into parameters W2 and b2
    dW2 = np.dot(A1.T, dZ2)
    db2 = np.sum(dZ2, axis=0, keepdims=True)
    # next backprop into hidden layer, A1
    dA1 = np.dot(dZ2, self.W2.T)
    # backprop the R
```

38 / 40

```
def fwd_prop(self, X):
    Z1 = np.dot(X, self.W1) + self.b1
    A1 = np.maximum(0, Z1)
    Z2 = np.dot(A1, self.W2) + self.b2
    Z2 = np.exp(Z2)
    A2 = Z2 / np.sum(Z2, axis=1, keepdims=True)
    return A1, A2

def cce_loss(self, y, probs):
    num_examples = y.shape[0]
    correct_logprobs = -np.log(probs[range(num_examples),y])
    loss = np.sum(correct_logprobs)/num_examples
    return loss

def back_prop(self, X, A1, A2, y):
    # compute the gradient on scores
    num_examples = y.shape[0]
    dZ2 = A2
    dZ2[range(num_examples),y] -= 1
    dZ2 /= num_examples
    # first backprop into parameters W2 and b2
    dW2 = np.dot(A1.T, dZ2)
    db2 = np.sum(dZ2, axis=0, keepdims=True)
    # next backprop into hidden layer, A1
    dA1 = np.dot(dZ2, self.W2.T)
    # backprop the R
```

$y_j = 1$

39 / 41

Chrome File Edit View History Bookmarks Profiles Tab Window Help 10:01

Desmos | Graphing C... | BackpropFromScratch | Why choose Keras? | Getting started | Introduction to Keras | Introduction to Keras | srivastava14a.pdf | Calculus on Computa... | Softmax function Calc...

colab.research.google.com/drive/1vmwq6vCH4F42G9MoYzjcAWOx8rm97z4i#scrollTo=uPrKn3MnjCeY

+ Code + Text

RAM
Disk

```python
def fwd_prop(self, X):
    Z1 = np.dot(X, self.W1) + self.b1
    A1 = np.maximum(0, Z1)
    Z2 = np.dot(A1, self.W2) + self.b2
    Z2 = np.exp(Z2)
    A2 = Z2 / np.sum(Z2, axis=1, keepdims=True)
    return A1, A2

def cce_loss(self, y, probs):
    num_examples = y.shape[0]
    correct_logprobs = -np.log(probs[range(num_examples),y])
    loss = np.sum(correct_logprobs)/num_examples
    return loss

def back_prop(self, X, A1, A2, y):
    # compute the gradient on scores
    num_examples = y.shape[0]
    dZ2 = A2
    dZ2[range(num_examples),y] -= 1
    dZ2 /= num_examples
    # first backprop into parameters W2 and b2
    dW2 = np.dot(A1.T, dZ2)
    db2 = np.sum(dZ2, axis=0, keepdims=True)
    # next backprop into hidden layer, A1
    dA1 = np.dot(dZ2, self.W2.T)
    # backprop the R[illegible]
```

$$CE = \frac{1}{m}\sum_{i=1}^{m} CE_i$$

$$CE_i = -\sum_{j=1}^{K} y_{ij} \log P_{ij}$$

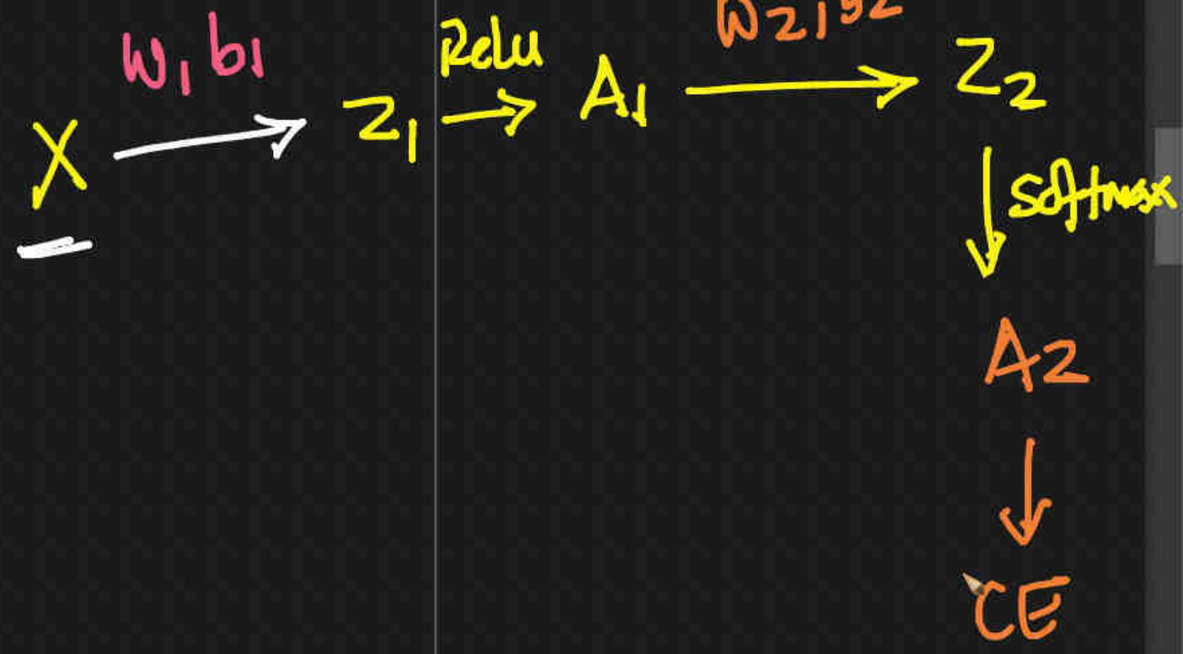
let $y_i \in$ class 2

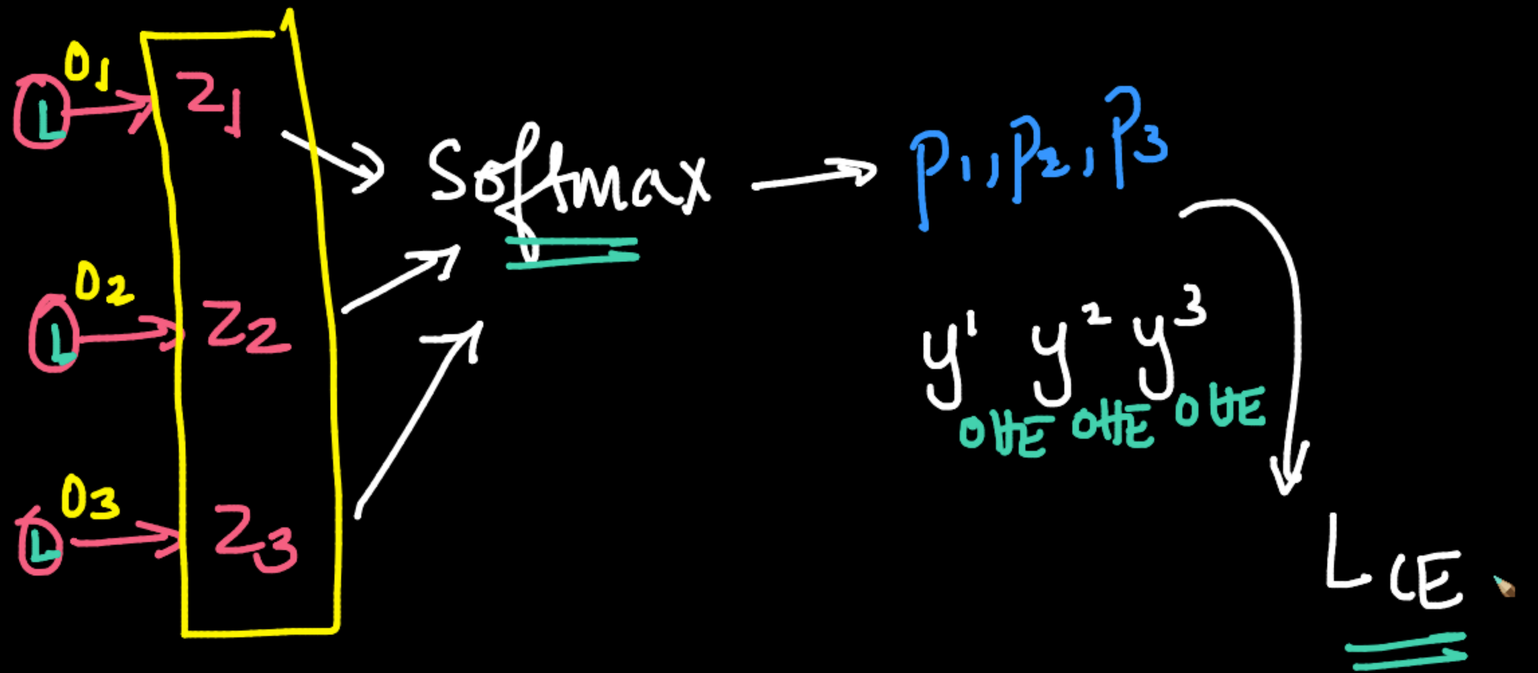
$$\begin{cases} y_{i2} = 1 \\ y_{ij \neq 2} = 0 \end{cases}$$

40 / 42

```
def fwd_prop(self, X):
    Z1 = np.dot(X, self.W1) + self.b1
    A1 = np.maximum(0, Z1)
    Z2 = np.dot(A1, self.W2) + self.b2
    Z2 = np.exp(Z2)
    A2 = Z2 / np.sum(Z2, axis=1, keepdims=True)
    return A1, A2

def cce_loss(self, y, probs):
    num_examples = y.shape[0]
    correct_logprobs = -np.log(probs[range(num_examples),y])
    loss = np.sum(correct_logprobs)/num_examples
    return loss

def back_prop(self, X, A1, A2, y):
    # compute the gradient on scores
    num_examples = y.shape[0]
    dZ2 = A2
    dZ2[range(num_examples),y] -= 1
    dZ2 /= num_examples
    # first backprop into parameters W2 and b2
    dW2 = np.dot(A1.T, dZ2)
    db2 = np.sum(dZ2, axis=0, keepdims=True)
    # next backprop into hidden layer, A1
    dA1 = np.dot(dZ2, self.W2.T)
    # backprop the R
```

```
        num_examples = y.shape[0]
        correct_logprobs = -np.log(probs[range(num_examples),y])
        loss = np.sum(correct_logprobs)/num_examples
        return loss

    def back_prop(self, X, A1, A2, y):
        # compute the gradient on scores
        num_examples = y.shape[0]
        dZ2 = A2
        dZ2[range(num_examples),y] -= 1
        dZ2 /= num_examples
        # first backprop into parameters W2 and b2
        dW2 = np.dot(A1.T, dZ2)
        db2 = np.sum(dZ2, axis=0, keepdims=True)
        # next backprop into hidden layer, A1
        dA1 = np.dot(dZ2, self.W2.T)
        # backprop the ReLU non-linearity
        dA1[A1 <= 0] = 0
        # finally into W,b
        dZ1 = dA1
        dW1 = np.dot(X.T, dZ1)
        db1 = np.sum(dZ1, axis=0, keepdims=True)
        return dW1, db1, dW2, db2

    def fit(self, X, lr, reg, max_iters):
        num_examples = X.shape[0]
```




o_1
o_2
o_3
z_1
z_2
z_3
Softmax
p_1, p_2, p_3
y^1 y^2 y^3
one hot
one hot
one hot
L_{CE}

Chrome File Edit View History Bookmarks Profiles Tab Window Help 10:05

$$\frac{\partial L_{CE}}{\partial z_1} \cdot \frac{\partial L_{CE}}{\partial z_2} = \frac{\partial L_{CE}}{\partial z_3}$$

Chrome File Edit View History Bookmarks Profiles Tab Window Help 10:12

3-class Classfn

$$\frac{\partial L_{CE}}{\partial z_j} \overset{\text{softmax}}{=} p_j - y_j \overset{\text{OHE}}{}$$

↳ Prob that $x \in$ Class j

$$\frac{\partial L_{CE}}{\partial z_1} = p_1 - 0$$

$$\frac{\partial L_{CE}}{\partial z_2}$$

(e.g)

$x \rightarrow$ Class 2

↳ Model → ~~p_1~~ p_2 ~~p_3~~

y_1^{OHE}	y_2^{OHE}	y_3^{OHE}
0	1	0
1	2	3

$$p_2 - 1 = \frac{\partial L_{CE}}{\partial z_2}$$

Chrome File Edit View History Bookmarks Profiles Tab Window Help
Desmos | Graphing | BackpropFromScr | Why choose Keras | Getting started | Introduction to Ke | Introduction to Ke | srivastava14a.pdf | Calculus on Comp | Softmax function | Derivative of the S
towardsdatascience.com/derivative-of-the-softmax-function-and-the-categorical-cross-entropy-loss-ffceefc081d1

$$\frac{\partial z_i}{\partial z_j} = \begin{cases} 1, & if\ i = j \\ 0, & otherwise \end{cases}$$

which can be concisely written using the indicator function $1\{\cdot\}$. The indicator function takes on a value of 1 if its argument is true, and 0 otherwise.

The second term on the right-hand side can be evaluated by applying the chain rule:

$$\frac{\partial}{\partial z_j} log\ s_i = 1\{i = j\} - \frac{1}{\sum_{l=1}^{n} e^{z_l}} \cdot \left(\frac{\partial}{\partial z_j} \sum_{l=1}^{n} e^{z_l} \right)$$

In the step above we used the derivative of the natural logarithm:

$$\frac{d}{dx} log(x) = \frac{1}{x}$$

Obtaining the partial derivative of the sum is trivial:

$$\frac{\partial}{\partial z_j} \sum_{l=1}^{n} e^{z_l} = \frac{\partial}{\partial z_j} [e^{z_1} + e^{z_2} + \cdots + e^{z_j} + \cdots + e^{z_n}] = \frac{\partial}{\partial z_j} [e^{z_j}] = e^{z_j}$$

Respond

Plugging the result into the form:

Thomas Kurbiel
212 Followers
Advanced Computer Vision & AI Research Engineer at APTIV Germany
Follow

More from Medium

Daniel Reiff in Towards Data Science
Generalizing Your Model: An Example With EfficientNetV2 and Cats & Dogs

iva vrtaric in Artificialis
Essential skill of Hypertuning with KerasTuner

Shenyan
A Machine Learning Template based on Pytorch Lightning

Sik-Ho Tsang
Review—There Are Many Consistent Explanations of Unlabeled Data: Why You Shoul...

Help Status Writers Blog Careers Privacy Terms About Knowable

Chrome File Edit View History Bookmarks Profiles Tab Window Help
Desmos | Graphing | BackpropFromScr | Why choose Keras | Getting started | Introduction to Ke | Introduction to Ke | srivastava14a.pdf | Calculus on Comp | Softmax function | Derivative of the S
towardsdatascience.com/derivative-of-the-softmax-function-and-the-categorical-cross-entropy-loss-ffceefc081d1

$$\frac{\partial \mathcal{L}}{\partial z_j} = \sum_{i=1}^{C} y_i \cdot s_j - y_j$$

Next, we pull s out of the sum, since it does not depend on index i:

$$\frac{\partial \mathcal{L}}{\partial z_j} = s_j \cdot \sum_{i=1}^{C} y_i - y_j = s_j - y_j$$

In the last step we used the fact, that the one-hot encoded vector $\boldsymbol{y}$ sums to 1. Remember that a one-hot encoded vector can be interpreted as a probability distribution with the probability mass centered around a single value. In the concise vector notation we get:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{z}} = \boldsymbol{s} - \boldsymbol{y}$$

More readings

Eli Bendersky's website
Raúl Gómez blog

363 6

47 / 47

Get started Sign In

Search

Thomas Kurbiel
212 Followers
Advanced Computer Vision & AI Research Engineer at APTIV Germany

Follow

More from Medium

Daniel Reiff in Towards Data Science
Generalizing Your Model: An Example With EfficientNetV2 and Cats & Dogs

iva vrtaric in Artificialis
Essential skill of Hypertuning with KerasTuner

Shenyan
A Machine Learning Template based on Pytorch Lightning

Sik-Ho Tsang
Review—There Are Many

```
def back_prop(self, X, A1, A2, y):
    # compute the gradient on scores
    num_examples = y.shape[0]
    dZ2 = A2
    dZ2[range(num_examples),y] -= 1
    dZ2 /= num_examples
    # first backprop into parameters W2 and b2
    dW2 = np.dot(A1.T, dZ2)
    db2 = np.sum(dZ2, axis=0, keepdims=True)
    # next backprop into hidden layer, A1
    dA1 = np.dot(dZ2, self.W2.T)
    # backprop the ReLU non-linearity
    dA1[A1 <= 0] = 0
    # finally into W,b
    dZ1 = dA1
    dW1 = np.dot(X.T, dZ1)
    db1 = np.sum(dZ1, axis=0, keepdims=True)
    return dW1, db1, dW2, db2

def fit(self, X, lr, reg, max_iters):
    num_examples = X.shape[0]
    for i in range(max_iters):
        #foward prop
        A1, A2 = self.fwd_prop(X)
        # calculate
```

$$\frac{\partial L}{\partial A_1} = \frac{\partial L}{\partial z_2} \cdot \frac{\partial z_2}{\partial A_1}$$

∂z_2 W_2

$\partial L / \partial W_2$, $\frac{\partial L}{\partial b_2}$

X ($m \times 2$) $\xrightarrow{W_1, b_1}$ z_1 $\xrightarrow{\text{ReLU}}$ A_1 $\xrightarrow{W_2, b_2}$ z_2 $\xrightarrow{\text{Softmax}}$ A_2 $\xrightarrow{}$ L_{CE}

$$\frac{\partial z_2}{\partial W_2} = A_1$$

$$A_1 \cdot W_2 + b_2 = z_2$$

$\frac{\partial L}{\partial z_2}$

$$\frac{\partial L}{\partial W_2} = \frac{\partial L}{\partial z_2} \cdot \frac{\partial z_2}{\partial W_2}$$

∂z_2 A_1

48 / 49

```
        db1 = np.sum(dZ1, axis=0, keepdims=True)
        return dW1, db1, dW2, db2

    def fit(self, X, lr, reg, max_iters):
        num_examples = X.shape[0]
        for i in range(max_iters):
            #foward prop
            A1, A2 = self.fwd_prop(X)
            # calculate loss
            data_loss = self.cce_loss(y, A2)
            reg_loss = 0.5*reg*np.sum(self.W1*self.W1) + 0.5*reg*np.sum(self.W2*self.W2)
            loss = data_loss + reg_loss

            if i % 1000 == 0:
                print("iteration %d: loss %f" % (i, loss))

            dW1, db1, dW2, db2  = self.back_prop(X, A1, A2, y)

            # add regularization gradient contribution
            dW2 += reg * self.W2
            dW1 += reg * self.W1

            # perform a parameter update
            self.W1 += -lr * dW1
            self.b1 += -lr * db1
            self.W2 += -lr * dW2
```

```
        dW1 = np.dot(X.T, dZ1)
        db1 = np.sum(dZ1, axis=0, keepdims=True)
        return dW1, db1, dW2, db2

    def fit(self, X, lr, reg, max_iters):
        num_examples = X.shape[0]
        for i in range(max_iters):
            #foward prop
            A1, A2 = self.fwd_prop(X)
            # calculate loss
            data_loss = self.cce_loss(y, A2)
            reg_loss = 0.5*reg*np.sum(self.W1*self.W1) + 0.5*reg*np.sum(self.W2*self.W2)
            loss = data_loss + reg_loss

            if i % 1000 == 0:
                print("iteration %d: loss %f" % (i, loss))

            dW1, db1, dW2, db2  = self.back_prop(X, A1, A2, y)

            # add regularization gradient contribution
            dW2 += reg * self.W2
            dW1 += reg * self.W1

            # perform a parameter update
            self.W1 += -lr * dW1
            self.b1 += -lr * db1
```

10,000

50 / 51

```
        dW1 = np.dot(X.T, dZ1)
        db1 = np.sum(dZ1, axis=0, keepdims=True)
        return dW1, db1, dW2, db2

    def fit(self, X, lr, reg, max_iters):
        num_examples = X.shape[0]
        for i in range(max_iters):
            #foward prop
            A1, A2 = self.fwd_prop(X)
            # calculate loss
            data_loss = self.cce_loss(y, A2)
            reg_loss = 0.5*reg*np.sum(self.W1*self.W1) + 0.5*reg*np.sum(self.W2*self.W2)
            loss = data_loss + reg_loss

            if i % 1000 == 0:
                print("iteration %d: loss %f" % (i, loss))

            dW1, db1, dW2, db2  = self.back_prop(X, A1, A2, y)

            # add regularization gradient contribution
            dW2 += reg * self.W2
            dW1 += reg * self.W1

            # perform a parameter update
            self.W1 += -lr * dW1
            self.b1 += -lr * db1
```

$L_2 \cdot reg$

$\frac{1}{2} w^T w$

$\frac{1}{2} \cdot b \cdot w$

51 / 52

```
        dW1 = np.dot(X.T, dZ1)
        db1 = np.sum(dZ1, axis=0, keepdims=True)
        return dW1, db1, dW2, db2

    def fit(self, X, lr, reg, max_iters):
        num_examples = X.shape[0]
        for i in range(max_iters):
            #foward prop
            A1, A2 = self.fwd_prop(X)
            # calculate loss
            data_loss = self.cce_loss(y, A2)
            reg_loss = 0.5*reg*np.sum(self.W1*self.W1) + 0.5*reg*np.sum(self.W2*self.W2)
            loss = data_loss + reg_loss

            if i % 1000 == 0:
                print("iteration %d: loss %f" % (i, loss))

            dW1, db1, dW2, db2  = self.back_prop(X, A1, A2, y)

            # add regularization gradient contribution
            dW2 += reg * self.W2
            dW1 += reg * self.W1

            # perform a parameter update
            self.W1 += -lr * dW1
            self.b1 += -lr * db1
```

```
        dW1 = np.dot(X.T, dZ1)
        db1 = np.sum(dZ1, axis=0, keepdims=True)
        return dW1, db1, dW2, db2

    def fit(self, X, lr, reg, max_iters):
        num_examples = X.shape[0]
        for i in range(max_iters):
            #foward prop
            A1, A2 = self.fwd_prop(X)
            # calculate loss
            data_loss = self.cce_loss(y, A2)
            reg_loss = 0.5*reg*np.sum(self.W1*self.W1) + 0.5*reg*np.sum(self.W2*self.W2)
            loss = data_loss + reg_loss

            if i % 1000 == 0:
                print("iteration %d: loss %f" % (i, loss))

            dW1, db1, dW2, db2  = self.back_prop(X, A1, A2, y)

            # add regularization gradient contribution
            dW2 += reg * self.W2
            dW1 += reg * self.W1

            # perform a parameter update
            self.W1 += -lr * dW1
            self.b1 += -lr * db1
```

```
        dW1 = np.dot(X.T, dZ1)
        db1 = np.sum(dZ1, axis=0, keepdims=True)
        return dW1, db1, dW2, db2

    def fit(self, X, lr, reg, max_iters):
        num_examples = X.shape[0]
        for i in range(max_iters):
            #foward prop
            A1, A2 = self.fwd_prop(X)
            # calculate loss
            data_loss = self.cce_loss(y, A2)
            reg_loss = 0.5*reg*np.sum(self.W1*self.W1) + 0.5*reg*np.sum(self.W2*self.W2)
            loss = data_loss + reg_loss

            if i % 1000 == 0:
                print("iteration %d: loss %f" % (i, loss))

            dW1, db1, dW2, db2  = self.back_prop(X, A1, A2, y)

            # add regularization gradient contribution
            dW2 += reg * self.W2
            dW1 += reg * self.W1

            # perform a parameter update
            self.W1 += -lr * dW1
            self.b1 += -lr * db1
```

```
        loss = data_loss + reg_loss

        if i % 1000 == 0:
            print("iteration %d: loss %f" % (i, loss))

        dW1, db1, dW2, db2  = self.back_prop(X, A1, A2, y)

        # add regularization gradient contribution
        dW2 += reg * self.W2
        dW1 += reg * self.W1

        # perform a parameter update
        self.W1 += -lr * dW1
        self.b1 += -lr * db1
        self.W2 += -lr * dW2
        self.b2 += -lr * db2

    def predict(self, X):
        A1 = np.maximum(0, np.dot(X, self.W1) + self.b1)
        Z2 = np.dot(A1, self.W2) + self.b2
        y_hat = np.argmax(Z2, axis=1)
        return y_hat

nn_model = NN(n_features=2, n_hidden=100, n_classes=3)
nn_model.fit(X, lr=1, reg=1e-3, max_iters=10000)
print('training accuracy: %.2f' % (np.mean(nn_model.predict(X) == y)))
```

$\partial L / \partial W_2$

loss + $\lambda \frac{1}{2} \cdot W_2^T W_2$

$+ \lambda \frac{1}{2} \cdot W_2^T W_2$

$\hookrightarrow \lambda \cdot W_2$

```
        # finally into W,b
        dZ1 = dA1
        dW1 = np.dot(X.T, dZ1)
        db1 = np.sum(dZ1, axis=0, keepdims=True)
        return dW1, db1, dW2, db2

    def fit(self, X, lr, reg, max_iters):
        num_examples = X.shape[0]
        for i in range(max_iters):
            #foward prop
            A1, A2 = self.fwd_prop(X)
            # calculate loss
            data_loss = self.cce_loss(y, A2)
            reg_loss = 0.5*reg*np.sum(self.W1*self.W1) + 0.5*reg*np.sum(self.W2*self.W2)
            loss = data_loss + reg_loss

            if i % 1000 == 0:
                print("iteration %d: loss %f" % (i, loss))

            dW1, db1, dW2, db2  = self.back_prop(X, A1, A2, y)

            # add regularization gradient contribution
            dW2 += reg * self.W2
            dW1 += reg * self.W1

            # perform a
```

Chrome File Edit View History Bookmarks Profiles Tab Window Help


```
            # add regularization gradient contribution
            dW2 += reg * self.W2
            dW1 += reg * self.W1

            # perform a parameter update
            self.W1 += -lr * dW1
            self.b1 += -lr * db1
            self.W2 += -lr * dW2
            self.b2 += -lr * db2

    def predict(self, X):
        A1 = np.maximum(0, np.dot(X, self.W1) + self.b1)
        Z2 = np.dot(A1, self.W2) + self.b2
        y_hat = np.argmax(Z2, axis=1)
        return y_hat

nn_model = NN(n_features=2, n_hidden=100, n_classes=3)
nn_model.fit(X, lr=1, reg=1e-3, max_iters=10000)
print('training accuracy: %.2f' % (np.mean(nn_model.predict(X) == y)))
```

```
iteration 0: loss 1.098614
iteration 1000: loss 0.305437
iteration 2000: loss 0.260879
iteration 3000: loss 0.255363
iteration 4000: loss 0.254227
```

Chrome File Edit View History Bookmarks Profiles Tab Window Help


```
            self.b2 += -lr * db2

    def predict(self, X):
        A1 = np.maximum(0, np.dot(X, self.W1) + self.b1)
        Z2 = np.dot(A1, self.W2) + self.b2
        y_hat = np.argmax(Z2, axis=1)
        return y_hat

nn_model = NN(n_features=2, n_hidden=100, n_classes=3)
nn_model.fit(X, lr=1, reg=1e-3, max_iters=10000)
print('training accuracy: %.2f' % (np.mean(nn_model.predict(X) == y)))
```

```
iteration 0: loss 1.098614
iteration 1000: loss 0.305437
iteration 2000: loss 0.260879
iteration 3000: loss 0.255363
iteration 4000: loss 0.254227
iteration 5000: loss 0.253764
iteration 6000: loss 0.264896
iteration 7000: loss 0.253992
iteration 8000: loss 0.253224
iteration 9000: loss 0.251573
training accuracy: 0.96
```

```
[ ] # create a 2D grid
    step = 0.02
    x_min, x_max = X[:, 0].m
```

Chrome File Edit View History Bookmarks Profiles Tab Window Help

Desmos | Graphing | BackpropFromScr | Why choose Kera | Getting started | Introduction to Ke | Introduction to Ke | srivastava14a.pdf | Calculus on Comp | Softmax function | Derivative of the S

colab.research.google.com/drive/1vmwq6vCH4F42G9MoYzjcAWOx8rm97z4i#scrollTo=uPrKn3MnjCeY

+ Code + Text

RAM
Disk

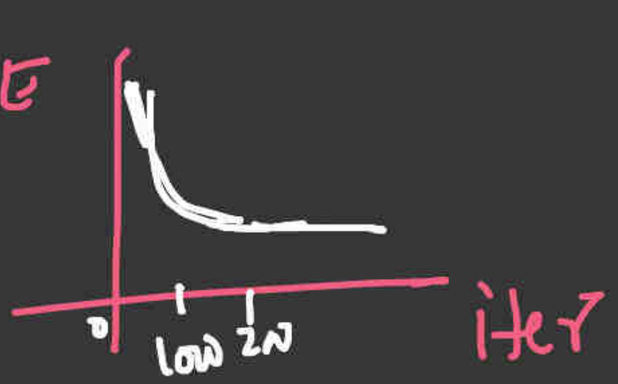
```
            self.b2 += -lr * db2

    def predict(self, X):
        A1 = np.maximum(0, np.dot(X, self.W1) + self.b1)
        Z2 = np.dot(A1, self.W2) + self.b2
        y_hat = np.argmax(Z2, axis=1)
        return y_hat

nn_model = NN(n_features=2, n_hidden=100, n_classes=3)
nn_model.fit(X, lr=1, reg=1e-3, max_iters=10000)
print('training accuracy: %.2f' % (np.mean(nn_model.predict(X) == y)))
```

```
iteration 0: loss 1.098614
iteration 1000: loss 0.305437
iteration 2000: loss 0.260879
iteration 3000: loss 0.255363
iteration 4000: loss 0.254227
iteration 5000: loss 0.253764
iteration 6000: loss 0.264896
iteration 7000: loss 0.253992
iteration 8000: loss 0.253224
iteration 9000: loss 0.251573
training accuracy: 0.96
```

```
[ ] # create a 2D grid
    step = 0.02
    x_min, x_max = X[:, 0].m
```

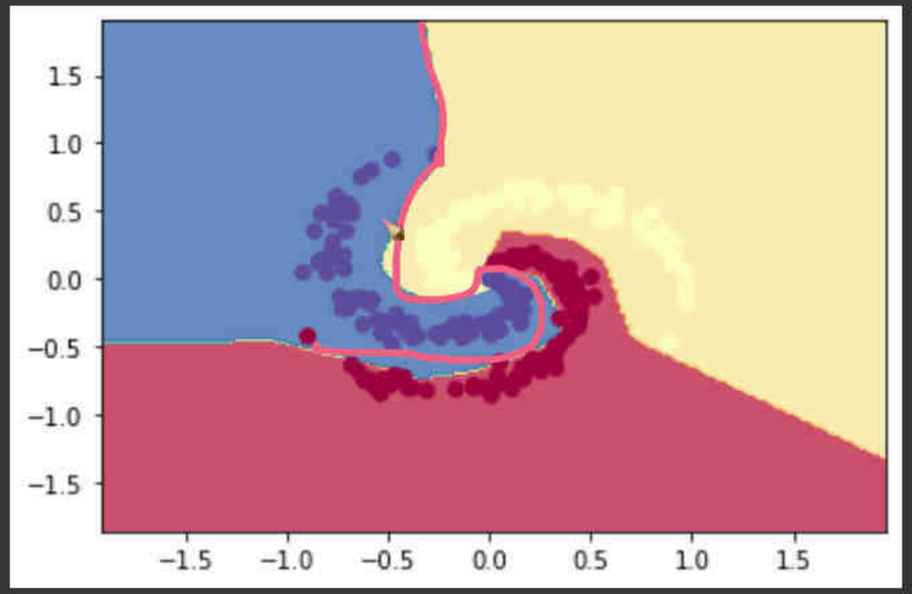
```
            self.b2 += -lr * db2

    def predict(self, X):
        A1 = np.maximum(0, np.dot(X, self.W1) + self.b1)
        Z2 = np.dot(A1, self.W2) + self.b2
        y_hat = np.argmax(Z2, axis=1)
        return y_hat

nn_model = NN(n_features=2, n_hidden=100, n_classes=3)
nn_model.fit(X, lr=1, reg=1e-3, max_iters=10000)
print('training accuracy: %.2f' % (np.mean(nn_model.predict(X) == y)))
```

```
iteration 0: loss 1.098614
iteration 1000: loss 0.305437
iteration 2000: loss 0.260879
iteration 3000: loss 0.255363
iteration 4000: loss 0.254227
iteration 5000: loss 0.253764
iteration 6000: loss 0.264896
iteration 7000: loss 0.253992
iteration 8000: loss 0.253224
iteration 9000: loss 0.251573
training accuracy: 0.96
```




```
[ ]  # create a 2D grid
     step = 0.02
     x_min, x_max = X[:, 0].m
```

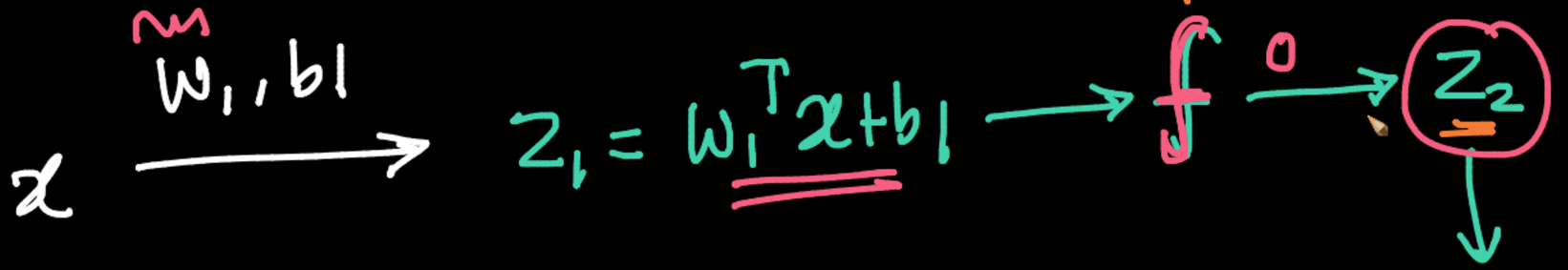
```
y_hat = nn_model.predict(np.c_[xx.ravel(), yy.ravel()])
y_hat = y_hat.reshape(xx.shape)

# plot
fig = plt.figure()
plt.contourf(xx, yy, y_hat, cmap=plt.cm.Spectral, alpha=0.8)
plt.scatter(X[:, 0], X[:, 1], c=y, s=40, cmap=plt.cm.Spectral)
plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())
plt.show()
```



Q

let $f(x) = x^2$

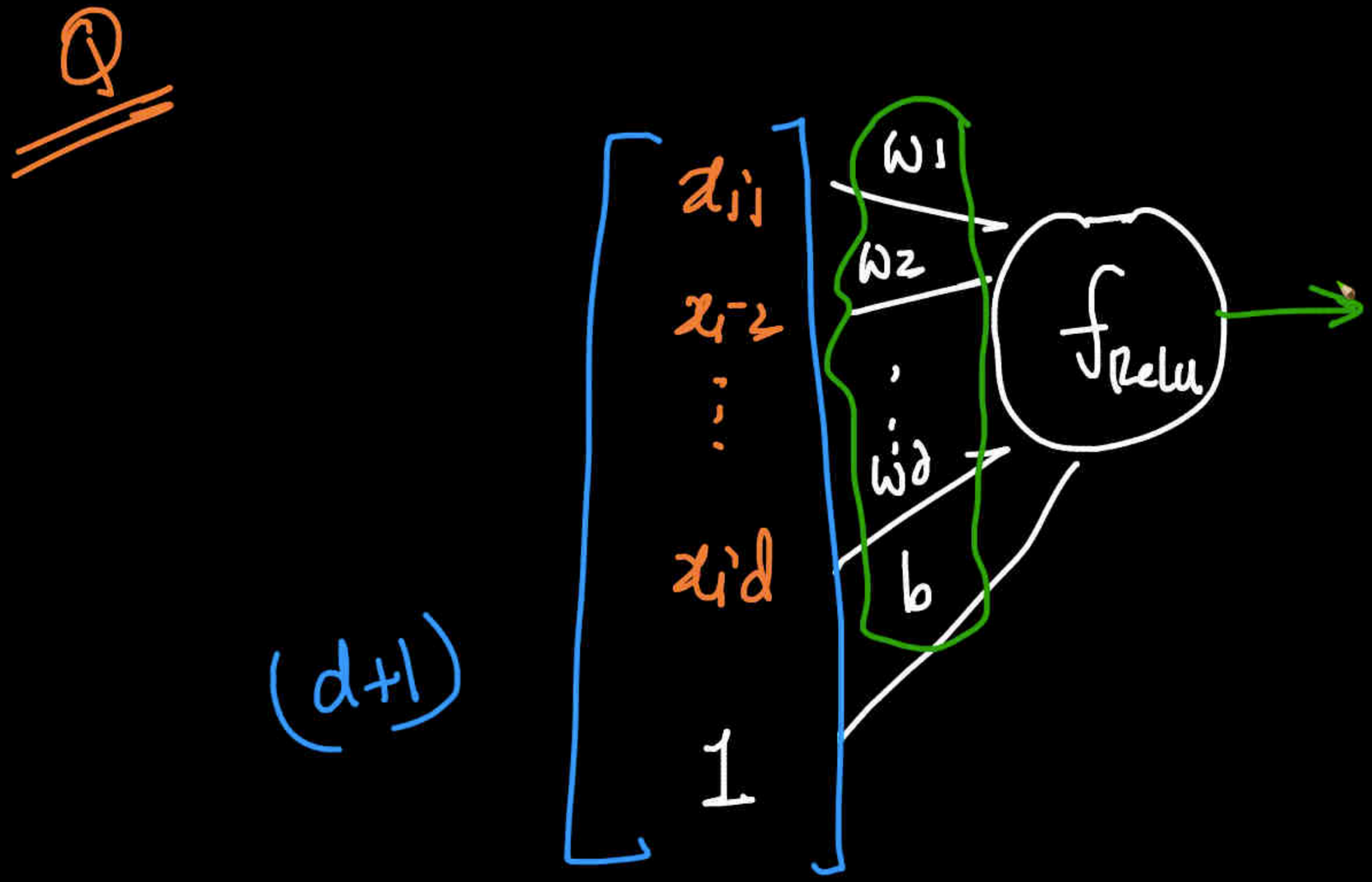
$z_1 = 2x + 3$

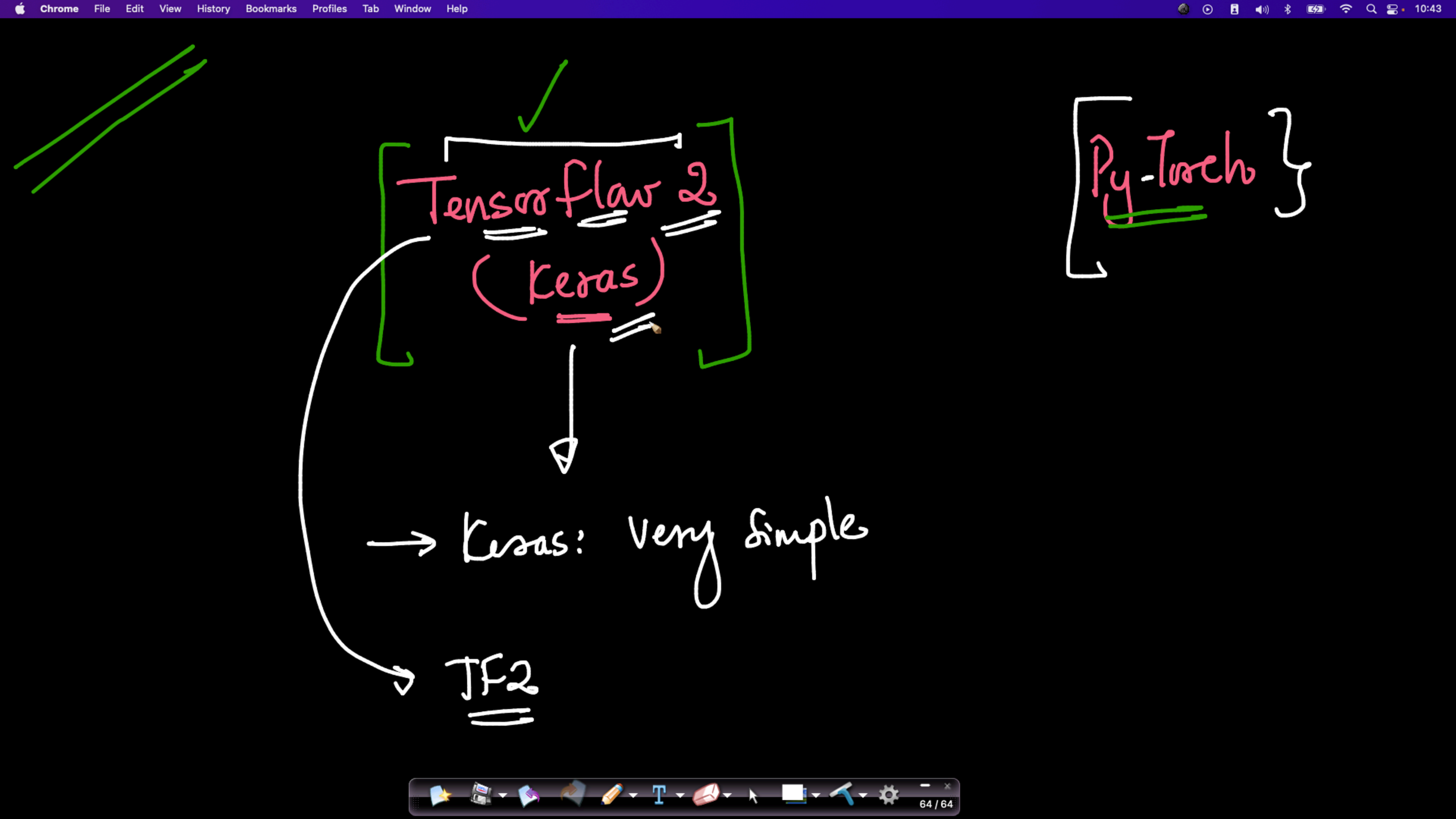


$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial z_2} \cdot \boxed{\frac{\partial z_2}{\partial w_1}}$$

Loss

$\hookrightarrow \frac{\partial (2x+3)^2}{\partial x} =$


Q
x_{i1}
x_{i2}
x_{id}
1
w_1
w_2
w_d
b
f_{Relu}
(d+1)


Tensorflow 2
(Keras)
PyTorch
→ Keras: Very simple
→ TF2

Chrome File Edit View History Bookmarks Profiles Tab Window Help 10:48

Desmos | Graphing | BackpropFromScr | Why choose Keras | Getting started | Introduction to Ke | Introduction to Ke | srivastava14a.pdf | Calculus on Comp | Softmax function | Derivative of the S

colab.research.google.com/drive/1vmwq6vCH4F42G9MoYzjcAWOx8rm97z4i#scrollTo=YMYElvUyWGrt

TF 2 and Keras: Sequential API

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers


print(tf.__version__)
```

```
2.8.2
```

```
[ ] #Keras Sequential API
    model = keras.Sequential()
    model.add(keras.Input(shape=(2,)))
    model.add(layers.Dense(100, activation="relu"))
    model.add(layers.Dense(3, activation="softmax"))
```

```
[ ] model.summary()
```

```
Model: "sequential_1"
```

Chrome File Edit View History Bookmarks Profiles Tab Window Help 10:48

Desmos | Graphing | BackpropFromScr | Why choose Keras | Getting started | Introduction to Ke | Introduction to Ke | srivastava14a.pdf | Calculus on Comp | Softmax function | Derivative of the S

colab.research.google.com/drive/1vmwq6vCH4F42G9MoYzjcAWOx8rm97z4i#scrollTo=YMYElvUyWGrt

+ Code + Text RAM Disk

TF 2 and Keras: Sequential API

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers


print(tf.__version__)
```

```
2.8.2
```

```
[ ] #Keras Sequential API
    model = keras.Sequential()
    model.add(keras.Input(shape=(2,)))
    model.add(layers.Dense(100, activation="relu"))
    model.add(layers.Dense(3, activation="softmax"))
```

```
[ ] model.summary()
```

```
Model: "sequential_1"
```

66 / 66

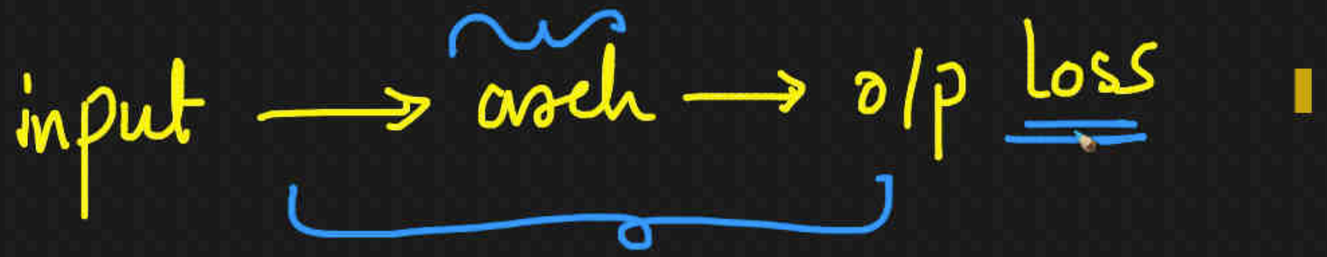
Chrome File Edit View History Bookmarks Profiles Tab Window Help 10:49
Desmos | Graphing | BackpropFromScr | Why choose Keras | Getting started | Introduction to Ke | Introduction to Ke | srivastava14a.pdf | Calculus on Comp | Softmax function | Derivative of the S
colab.research.google.com/drive/1vmwq6vCH4F42G9MoYzjcAWOx8rm97z4i#scrollTo=YMYElvUyWGrt

+ Code + Text

RAM
Disk

TF 2 and Keras: Sequential API

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers


print(tf.__version__)
```




```
2.8.2
```

```
[ ] #Keras Sequential API
    model = keras.Sequential()
    model.add(keras.Input(shape=(2,)))
    model.add(layers.Dense(100, activation="relu"))
    model.add(layers.Dense(3, activation="softmax"))
```

```
[ ] model.summary()
```

```
Model: "sequential_1"
```

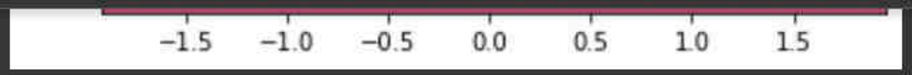
67 / 67

Chrome File Edit View History Bookmarks Profiles Tab Window Help 10:50

Desmos | Graphing | BackpropFromScr | Why choose Keras | Getting started | Introduction to Ke | Introduction to Ke | srivastava14a.pdf | Calculus on Comp | Softmax function | Derivative of the S

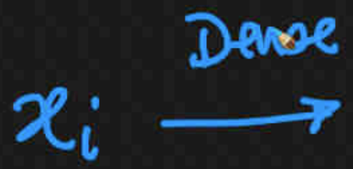
colab.research.google.com/drive/1vmwq6vCH4F42G9MoYzjcAWOx8rm97z4i#scrollTo=ZBvZDZaJWHHk

+ Code + Text RAM Disk



TF 2 and Keras: Sequential API

```
[ ] import tensorflow as tf
    from tensorflow import keras
    from tensorflow.keras import layers


    print(tf.__version__)
```




```
2.8.2
```

```
[ ] #Keras Sequential API
    model = keras.Sequential()
    model.add(keras.Input(shape=(2,)))
    model.add(layers.Dense(100, activation="relu"))
    model.add(layers.Dense(3, activation="softmax"))
```

```
[ ] model.summary()
```

Model: "sequential_1"

68 / 68

Chrome File Edit View History Bookmarks Profiles Tab Window Help 10:50

Desmos | Graphing | BackpropFromScr | Why choose Keras | Getting started | Introduction to Ke | Introduction to Ke | srivastava14a.pdf | Calculus on Comp | Softmax function | Derivative of the S

colab.research.google.com/drive/1vmwq6vCH4F42G9MoYzjcAWOx8rm97z4i#scrollTo=ZBvZDZaJWHHk

+ Code + Text

RAM
Disk

```
from tensorflow.keras import layers
```

[ ]

```
print(tf.__version__)
```

```
2.8.2
```

[ ]

```
#Keras Sequential API
model = keras.Sequential()
model.add(keras.Input(shape=(2,)))
model.add(layers.Dense(100, activation="relu"))
model.add(layers.Dense(3, activation="softmax"))
```

[ ]

```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_2 (Dense)             (None, 100)               300

 dense_3 (Dense)             (None, 3)                 303

=================================================================
Total params: 603
Trainable params: 603
```

Chrome File Edit View History Bookmarks Profiles Tab Window Help 10:53
Desmos | Graphing | BackpropFromScr | Why choose Keras | Getting started | Introduction to Ke | Introduction to Ke | srivastava14a.pdf | Calculus on Comp | Softmax function | Derivative of the S
colab.research.google.com/drive/1vmwq6vCH4F42G9MoYzjcAWOx8rm97z4i#scrollTo=ZBvZDZaJWHHk
+ Code + Text
RAM
Disk

```
from tensorflow.keras import layers
```

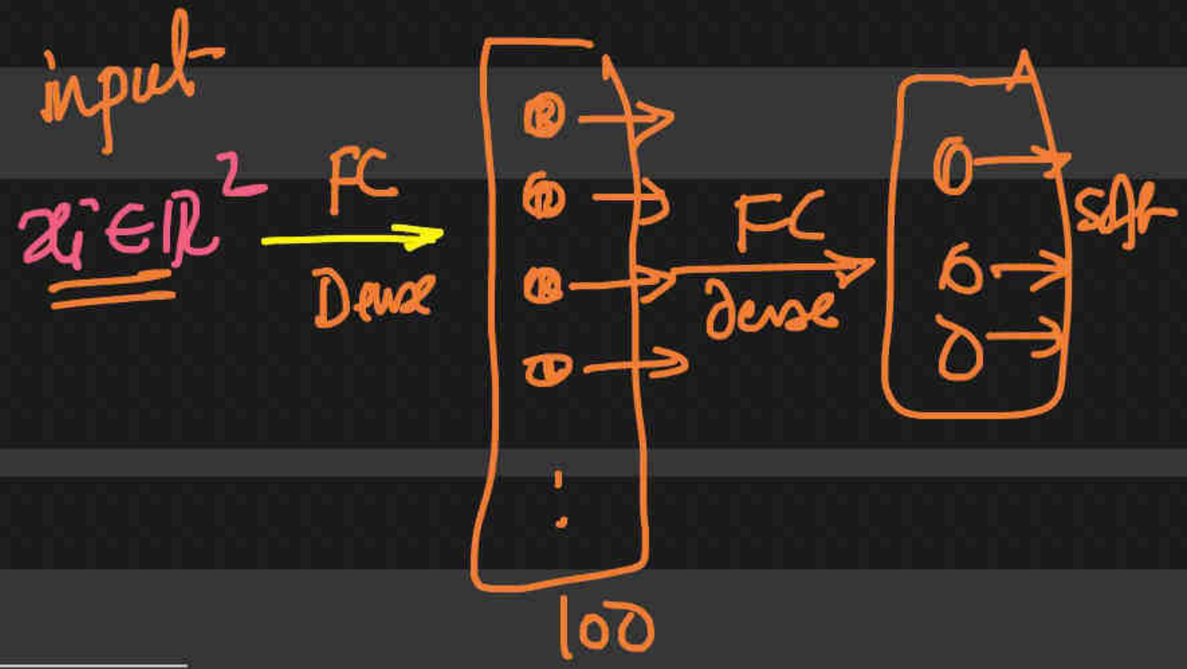
[ ]

```
print(tf.__version__)
```

```
2.8.2
```

[ ]

```
#Keras Sequential API
model = keras.Sequential()
model.add(keras.Input(shape=(2,)))
model.add(layers.Dense(100, activation="relu"))
model.add(layers.Dense(3, activation="softmax"))
```



[ ]

```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_2 (Dense)             (None, 100)               300

 dense_3 (Dense)             (None, 3)                 303

=================================================================
Total params: 603
Trainable params: 603
```

70 / 71

Chrome File Edit View History Bookmarks Profiles Tab Window Help 10:53

Desmos | Graphing | BackpropFromScr | Why choose Keras | Getting started | Introduction to Ke | Introduction to Ke | srivastava14a.pdf | Calculus on Comp | Softmax function | Derivative of the S

colab.research.google.com/drive/1vmwq6vCH4F42G9MoYzjcAWOx8rm97z4i#scrollTo=ZBvZDZaJWHHk

+ Code + Text RAM Disk

```
from tensorflow.keras import layers
```

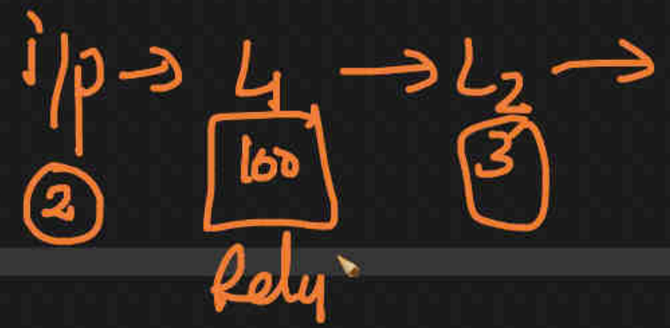
[ ]

```
print(tf.__version__)
```

```
2.8.2
```

[ ]

```
#Keras Sequential API
model = keras.Sequential()
model.add(keras.Input(shape=(2,)))
model.add(layers.Dense(100, activation="relu"))
model.add(layers.Dense(3, activation="softmax"))
```



[ ]

```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_2 (Dense)             (None, 100)               300

 dense_3 (Dense)             (None, 3)                 303

=================================================================
Total params: 603
Trainable params: 603
```

71 / 72

Chrome File Edit View History Bookmarks Profiles Tab Window Help 10:56

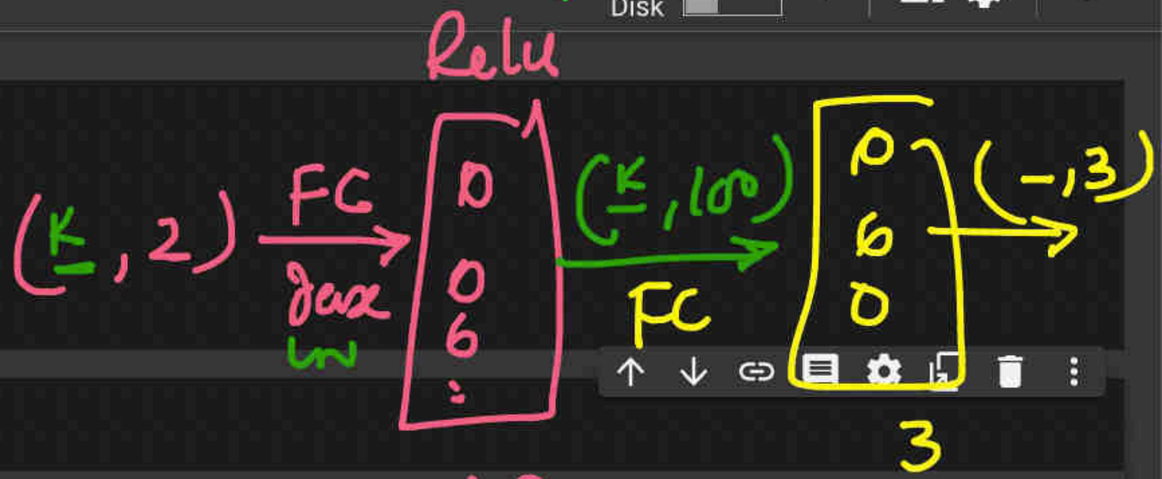
Desmos | Graphing | BackpropFromScr | Why choose Keras | Getting started | Introduction to Ke | Introduction to Ke | srivastava14a.pdf | Calculus on Comp | Softmax function | Derivative of the S

colab.research.google.com/drive/1vmwq6vCH4F42G9MoYzjcAWOx8rm97z4i#scrollTo=RI24jRUzgdGz

+ Code + Text

RAM
Disk

```
[ ] #Keras Sequential API
    model = keras.Sequential()
    model.add(keras.Input(shape=(2,)))
    model.add(layers.Dense(100, activation="relu"))
    model.add(layers.Dense(3, activation="softmax"))
```




```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_2 (Dense)             (None, 100)               300

 dense_3 (Dense)             (None, 3)                 303

=================================================================
Total params: 603
Trainable params: 603
Non-trainable params: 0
_________________________________________________________________
```

300 + 3

```
[ ] model.compile(optimizer=keras.optimizers.Adam(), # suggest in colab : Option+Esc in Mac
                  loss=keras.losses.CategoricalCrossentropy(),
```

72 / 73

Chrome File Edit View History Bookmarks Profiles Tab Window Help 10:56

Desmos | Graphing | BackpropFromScr | Why choose Keras | Getting started | Introduction to Ke | Introduction to Ke | srivastava14a.pdf | Calculus on Comp | Softmax function | Derivative of the S

colab.research.google.com/drive/1vmwq6vCH4F42G9MoYzjcAWOx8rm97z4i#scrollTo=RI24jRUzgdGz

+ Code + Text

RAM
Disk

```
[ ] #Keras Sequential API
    model = keras.Sequential()
    model.add(keras.Input(shape=(2,)))
    model.add(layers.Dense(100, activation="relu"))
    model.add(layers.Dense(3, activation="softmax"))
```

```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_2 (Dense)             (None, 100)               300

 dense_3 (Dense)             (None, 3)                 303

=================================================================
Total params: 603
Trainable params: 603
Non-trainable params: 0
_________________________________________________________________
```

```
[ ] model.compile(optimizer=keras.optimizers.Adam(), # suggest in colab : Option+Esc in Mac
                  loss=keras.losses.CategoricalCrossentropy(),
```

```
 dense_2 (Dense)             (None, 100)               300

 dense_3 (Dense)             (None, 3)                 303

=================================================================
Total params: 603
Trainable params: 603
Non-trainable params: 0
_________________________________________________________________
```

```
[ ] model.compile(optimizer=keras.optimizers.Adam(), # suggest in colab : Option+Esc in Mac
                  loss=keras.losses.CategoricalCrossentropy(),
                  metrics=[keras.metrics.BinaryAccuracy()])
```

```
[ ] print(X.shape)
    print(y.shape)
```

```
(300, 2)
(300,)
```

```
[ ] print(y)
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1
```

```
 dense_2 (Dense)              (None, 100)               300

 dense_3 (Dense)              (None, 3)                 303

=================================================================
Total params: 603
Trainable params: 603
Non-trainable params: 0
_________________________________________________________________
```

```
[ ] model.compile(optimizer=keras.optimizers.Adam(), # suggest in colab : Option+Esc in Mac
                  loss=keras.losses.CategoricalCrossentropy(),
                  metrics=[keras.metrics.BinaryAccuracy()])
```

```
[ ] print(X.shape)
    print(y.shape)
```

```
(300, 2)
(300,)
```

```
[ ] print(y)
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1
```

```
dense_2 (Dense)              (None, 100)               300

dense_3 (Dense)              (None, 3)                 303

=================================================================
Total params: 603
Trainable params: 603
Non-trainable params: 0
_________________________________________________________________
```

```
model.compile(optimizer=keras.optimizers.Adam(), # suggest in colab : Option+Esc in Mac
              loss=keras.losses.CategoricalCrossentropy(),
              metrics=[keras.metrics.BinaryAccuracy()])
```

```
print(X.shape)
print(y.shape)
```

```
(300, 2)
(300,)
```

```
print(y)
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0
```

```
dense_3 (Dense)              (None, 3)                 303

=================================================================
Total params: 603
Trainable params: 603
Non-trainable params: 0
_________________________________________________________________
```

```
[ ] model.compile(optimizer=keras.optimizers.Adam(), # suggest in colab : Option+Esc in Mac
                  loss=keras.losses.CategoricalCrossentropy(),
                  metrics=[keras.metrics.BinaryAccuracy()])
```

```
[ ] print(X.shape)
    print(y.shape)
```

```
(300, 2)
(300,)
```

```
[ ] print(y)
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

```
[ ] print(y)
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2]
```

```
history = model.fit(X, y, epochs=10, batch_size=32)
```

```
Epoch 1/10
/usr/local/lib/python3.7/dist-packages/tensorflow/python/data/ops/structured_function.py:265: UserWarning: Even thou
  "Even though the `tf.config.experimental_run_functions_eagerly` "
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-39-2de9726b75a6> in <module>
----> 1 history = model.fit(X, y, epochs=10, batch_size=32)

1 frames
/usr/local/lib/python3.7/dist-packages/keras/backend.py in categorical_crossentropy(target, output, from_logits,
axis)
   5081     target = tf.convert_to_tensor(target)
```

```
[ ] history = model.fit(X, y, epochs=10, batch_size=32)
```

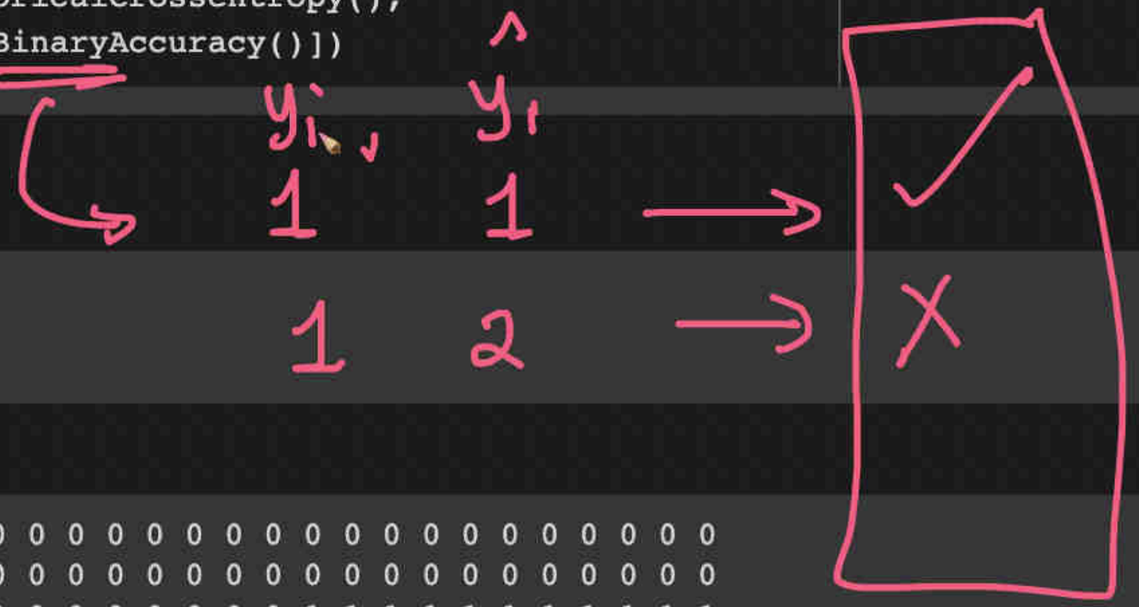
```
Epoch 1/10
/usr/local/lib/python3.7/dist-packages/tensorflow/python/data/ops/structured_function.py:265: UserWarning: Even thou
  "Even though the `tf.config.experimental_run_functions_eagerly` "
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-39-2de9726b75a6> in <module>
----> 1 history = model.fit(X, y, epochs=10, batch_size=32)

                                 1 frames
/usr/local/lib/python3.7/dist-packages/keras/backend.py in categorical_crossentropy(target, output, from_logits,
axis)
   5081     target = tf.convert_to_tensor(target)
   5082     output = tf.convert_to_tensor(output)
-> 5083     target.shape.assert_is_compatible_with(output.shape)
   5084
   5085     # Use logits whenever they are available. `softmax` and `sigmoid`

ValueError: Shapes (32, 1) and (32, 3) are incompatible
```

SEARCH STACK OVERFLOW

```
[ ] from tensorflow.keras.utils import to_categorical
```

```
Total params: 603
Trainable params: 603
Non-trainable params: 0
_________________________________________________________________
```

```
[ ] model.compile(optimizer=keras.optimizers.Adam(), # suggest in colab : Option+Esc in Mac
                  loss=keras.losses.CategoricalCrossentropy(),
                  metrics=[keras.metrics.BinaryAccuracy()])
```




```
[ ] print(X.shape)
    print(y.shape)
```

```
(300, 2)
(300,)
```

```
[ ] print(y)
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
```

80 / 81

Chrome File Edit View History Bookmarks Profiles Tab Window Help 11:02

m - pls = 300 (let)

GD → pass all m pls → Loss all m pls

$\frac{\partial L}{\partial w_j}$ using all pls

mini batch GD: batch-size = 32 out of 300 = M

$\frac{\partial L}{\partial w_j}$ using mini batch

Chrome File Edit View History Bookmarks Profiles Tab Window Help 11:03

SGD: use only 1 pt in each iterⁿ

Chrome File Edit View History Bookmarks Profiles Tab Window Help 11:06

m = 300 pts

mini-batch

epoch

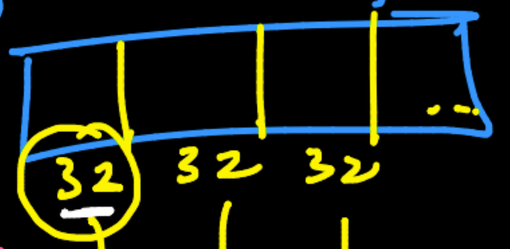
batch-size = 32

1 Shuffle random

2

3

m = 300 pts



iterⁿ ... → 10 iterns

84 / 84

```
(300, 2)
(300,)
```

```
[ ] print(y)
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2]
```

```
[ ] history = model.fit(X, y, epochs=10, batch_size=32)
```

```
Epoch 1/10
/usr/local/lib/python3.7/dist-packages/tensorflow/python/data/ops/structured_function.py:265: UserWarning: Even thou
  "Even though the `tf.config.experimental_run_functions_eagerly` "
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-39-2de9726b75a6> in <module>
----> 1 history = model.fit(X, y, epochs=10, batch_size=32)

1 frames
```

```
    5085   # Use logits whenever they are available. `softmax` and `sigmoid`
```

```
ValueError: Shapes (32, 1) and (32, 3) are incompatible
```

SEARCH STACK OVERFLOW

```
from tensorflow.keras.utils import to_categorical
y_OHE = to_categorical(y)
print(y_OHE)
```

```
[0. 0. 1.]
[0. 0. 1.]
[0. 0. 1.]
[0. 0. 1.]
[0. 0. 1.]
[0. 0. 1.]
[0. 0. 1.]
[0. 0. 1.]
[0. 0. 1.]
[0. 0. 1.]
[0. 0. 1.]
[0. 0. 1.]
[0. 0. 1.]
[0. 0. 1.]
[0. 0. 1.]
[0. 0. 1.]
```

Chrome File Edit View History Bookmarks Profiles Tab Window Help 11:07

Desmos | Graphing | BackpropFromScr | Why choose Keras | Getting started | Introduction to Ke | Introduction to Ke | srivastava14a.pdf | Calculus on Comp | Softmax function | Derivative of the S

colab.research.google.com/drive/1vmwq6vCH4F42G9MoYzjcAWOx8rm97z4i#scrollTo=RI24jRUzgdGz

+ Code + Text RAM Disk

```
[0. 0. 1.]
[0. 0. 1.]
[0. 0. 1.]
[0. 0. 1.]
[0. 0. 1.]
[0. 0. 1.]]
```

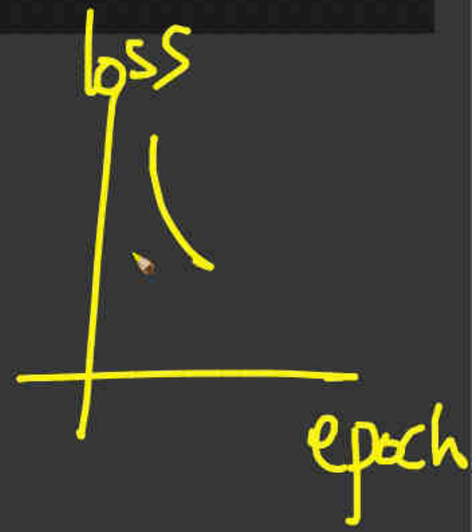
```
tf.config.run_functions_eagerly(True)
history = model.fit(X, y_OHE, epochs=10, batch_size=256)
```

```
Epoch 1/10
2/2 [==============================] - 0s 32ms/step - loss: 1.1073 - binary_accuracy: 0.6667
Epoch 2/10
2/2 [==============================] - 0s 29ms/step - loss: 1.0991 - binary_accuracy: 0.6667
Epoch 3/10
2/2 [==============================] - 0s 35ms/step - loss: 1.0913 - binary_accuracy: 0.6667
Epoch 4/10
2/2 [==============================] - 0s 26ms/step - loss: 1.0835 - binary_accuracy: 0.6667
Epoch 5/10
2/2 [==============================] - 0s 63ms/step - loss: 1.0758 - binary_accuracy: 0.6667
Epoch 6/10
2/2 [==============================] - 0s 26ms/step - loss: 1.0682 - binary_accuracy: 0.6667
Epoch 7/10
2/2 [==============================] - 0s 28ms/step - loss: 1.0606 - binary_accuracy: 0.6667
Epoch 8/10
2/2 [==============================] - 0s 27ms/step - loss: 1.0531 - binary_accuracy: 0.6667
Epoch 9/10
2/2 [==============================] - 0s 67ms/step - loss: 1.0458 - binary_accuracy: 0.6667
```

87 / 87

```
[ ] tf.config.run_functions_eagerly(True)
    history = model.fit(X, y_OHE, epochs=10, batch_size=256)
```

```
Epoch 1/10
2/2 [==============================] - 0s 32ms/step - loss: 1.1073 - binary_accuracy: 0.6667
Epoch 2/10
2/2 [==============================] - 0s 29ms/step - loss: 1.0991 - binary_accuracy: 0.6667
Epoch 3/10
2/2 [==============================] - 0s 35ms/step - loss: 1.0913 - binary_accuracy: 0.6667
Epoch 4/10
2/2 [==============================] - 0s 26ms/step - loss: 1.0835 - binary_accuracy: 0.6667
Epoch 5/10
2/2 [==============================] - 0s 63ms/step - loss: 1.0758 - binary_accuracy: 0.6667
Epoch 6/10
2/2 [==============================] - 0s 26ms/step - loss: 1.0682 - binary_accuracy: 0.6667
Epoch 7/10
2/2 [==============================] - 0s 28ms/step - loss: 1.0606 - binary_accuracy: 0.6667
Epoch 8/10
2/2 [==============================] - 0s 27ms/step - loss: 1.0531 - binary_accuracy: 0.6667
Epoch 9/10
2/2 [==============================] - 0s 67ms/step - loss: 1.0458 - binary_accuracy: 0.6667
Epoch 10/10
2/2 [==============================] - 0s 46ms/step - loss: 1.0385 - binary_accuracy: 0.6667
```




```
[ ] tf.config.run_functions_eagerly(True)
    history = model.fit(X, y_OHE, epochs=100, batch_size=256)
```

Chrome File Edit View History Bookmarks Profiles Tab Window Help 11:08

Desmos | Graphing | BackpropFromScr | Why choose Kera | Getting started | Introduction to Ke | Introduction to Ke | srivastava14a.pdf | Calculus on Comp | Softmax function | Derivative of the S

colab.research.google.com/drive/1vmwq6vCH4F42G9MoYzjcAWOx8rm97z4i#scrollTo=RI24jRUzgdGz

+ Code + Text RAM Disk

```
2/2 [==============================] - 0s 28ms/step - loss: 0.1136 - binary_accuracy: 0.9800
Epoch 998/1000
2/2 [==============================] - 0s 35ms/step - loss: 0.1134 - binary_accuracy: 0.9811
Epoch 999/1000
2/2 [==============================] - 0s 27ms/step - loss: 0.1132 - binary_accuracy: 0.9811
Epoch 1000/1000
2/2 [==============================] - 0s 24ms/step - loss: 0.1131 - binary_accuracy: 0.9811
```

TF & Keras: Functional API

[ ] ↳ 5 cells hidden

Computational Graph

[ ] ↳ 3 cells hidden

89 / 89

Chrome File Edit View History Bookmarks Profiles Tab Window Help

Simple NN (in Python)

[1] `!gdown 1dLOPwh01o3k8p_hK633ixhD1ehz6nNWk`

```
Downloading...
From: https://drive.google.com/uc?id=1dLOPwh01o3k8p_hK633ixhD1ehz6nNWk
To: /content/spiral.csv
100% 12.9k/12.9k [00:00<00:00, 17.1MB/s]
```

[2] `!ls -lrt`

```
total 20
drwxr-xr-x 1 root root  4096 Aug 31 13:47 sample_data
-rw-r--r-- 1 root root 12867 Sep  5 16:14 spiral.csv
```

```
# Very simple case of a back-prop
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
df = pd.read_csv("spiral.
plt.scatter(df["x1"], df["x2"], c=df["y"], s=40, cmap=plt.cm.Spectral)
```

90 / 90