

Table of contents

TF 2 and Keras (Sequential API)

TF 2 and Keras: Functional API

$x\}$ Dropout: Regularization through randomization - CP DD

6 | GPUs vs CPU

CPU
GPU

Tensorboard and Computational Graph

Batch Normalization

+ Section

1

+ Code + Text

Reconnect



▼

► TF 2 and Keras (Sequential API)

[] ↳ 13 cells hidden

► TF 2 and Keras: Functional API

[] ↳ 5 cells hidden

- ▶ Dropout: Regularization through randomization

[] ↳ 5 cells hidden

A set of small, light-gray navigation icons located at the bottom right of the page. From left to right, they include: an upward arrow, a downward arrow, a circular arrow, a speech bubble, a pencil, a square with rounded corners, a trash can, and three vertical dots.

► GPUs vs CPUs

→ 9 cells hidden

Table of contents

Q TF 2 and Keras (Sequential API)

TF 2 and Keras: Functional API

{x} Dropout: Regularization through randomization

GPUs vs CPUs

CPU

GPU

Tensorboard and Computational Graph

Batch Normalization

+ Section

٤٨٦

+ Code + Tex

Reconnect ▾

1

► TF 2 and Keras (Sequential API)

[] ↳ 13 cells hidden

► TF 2 and Keras: Functional API

[] ↳ 5 cells hidden

- ▶ Dropout: Regularization through randomization

[1] \hookrightarrow 5 cells hidden

A set of small, light-gray navigation icons located at the bottom right of the slide. From left to right, they include: a double arrow pointing up and down; a circular arrow; a double-headed horizontal arrow; a magnifying glass; a pencil; a square with rounded corners; a vertical bar; and three vertical dots.

► GPUs vs CPUs

→ 9 cells hidden

Chrome File Edit View History Bookmarks Profiles Tab Window Help

∞ TF2_Keras_Intro.ipynb - Colab ✓ srivastava14.pdf ✓ K Dropout layer ✓ Layer weight regularizers ✓ 1502.03167.pdf ✓ K BatchNormalization layer + colab.research.google.com/drive/1tyv-ViqseMy1rTwGc6aLbnXk-5MC13g#scrollTo=5Sq1TG4J4MDB Reconnect Update

Table of contents

TF 2 and Keras (Sequential API)

TF 2 and Keras: Functional API

{x} Dropout: Regularization through randomization

GPUs vs CPUs

CPU

GPU

Tensorboard and Computational Graph

Batch Normalization

+ Section

Reconnect

Update

Table of contents

+ Code + Text

Reconnect

Update

▶ TF 2 and Keras (Sequential API)

[] ↳ 13 cells hidden

▶ TF 2 and Keras: Functional API

[] ↳ 5 cells hidden

▶ Dropout: Regularization through randomization

[] ↳ 5 cells hidden

▶ GPUs vs CPUs

▶ 9 cells hidden

Up Down Undo Redo Comment Pen Copy Delete More

Page Number: 3 / 3

+ Code + Text

Reconnect ▾



▼

• TF 2 and Keras (Sequential API)

```
[ ] import numpy as np ✓  
import tensorflow as tf ✓  
from tensorflow import keras ✓  
from tensorflow.keras import layers ✓  
from tensorflow.keras.models import Sequential ✓  
from tensorflow.keras.layers import Dense, Activation, Dropout  
from tensorflow.keras.utils import to_categorical, plot_model  
from tensorflow.keras.datasets import mnist  
import matplotlib.pyplot as plt
```

```
[ ] print(tf.__version__)
```

2.8.2

```
[ ] # load mnist dataset  
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

∞ TF2_Keras_Intro.ipynb - Colab × srivastava14a.pdf × K Dropout layer × K Layer weight regularizers × 1502.03167.pdf × K BatchNormalization layer × + colab.research.google.com/drive/1tyv-ViqseMy1rTwGc6aLbnXk-5MC13g#scrollTo=sGo6QATHoTCL Reconnect ⚙️ Update

+ Code + Text

TF 2 and Keras (Sequential API)

{x}

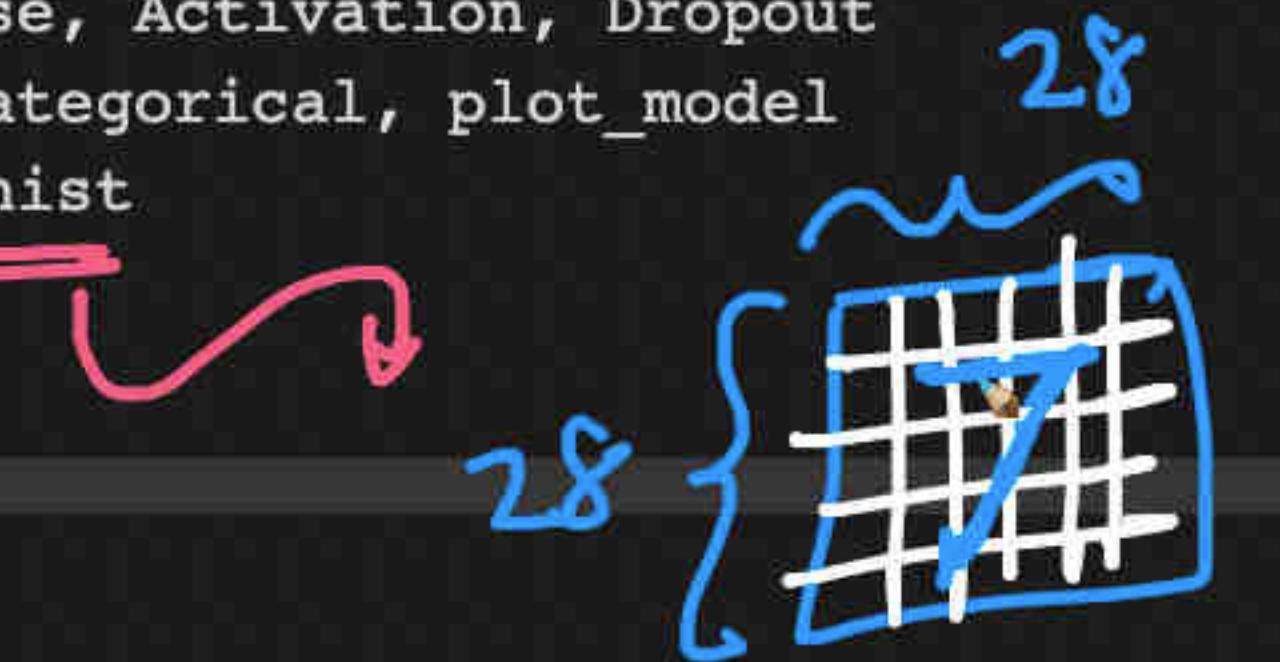
[] import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation, Dropout
from tensorflow.keras.utils import to_categorical, plot_model
from tensorflow.keras.datasets import mnist
import matplotlib.pyplot as plt

[] print(tf.__version__)

2.8.2

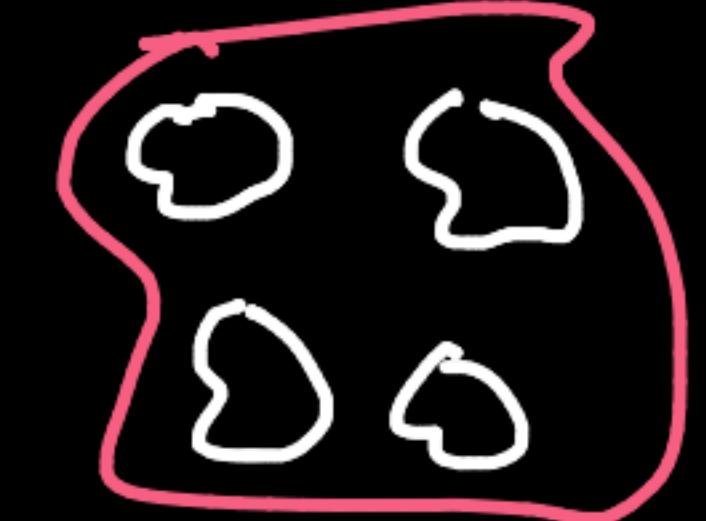
[] # load mnist dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()

Downloading data from https://storage.googleapis.com/tensorflow/tutorials/mnist/datasets/mnist.npz



Q

classical ML



1 vs rest → Logistic regrsn

Dim red (UMAP) + { Tree → ✓ DT, GBDT, RF ... }
each class → good amount of data
Softmax classifier ←

+ Code + Text

✓ RAM Disk

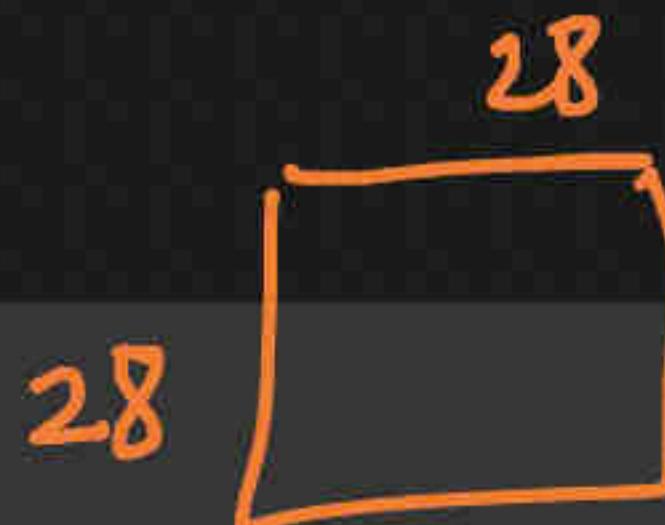
▼

```
[4] # load mnist dataset  
    (x_train, y_train), (x_test, y_test) = mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [=====] - 0s 0us/step
11501568/11490434 [=====] - 0s 0us/step
```

```
print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
```

D11 [(60000, 28, 28)
(60000,)
(10000, 28, 28)
(10000,)] D10



```
▶ idx=1234  
plt.imshow(x_train[idx], cmap='gray', interpolation='none'  
print(y_train[idx])
```

3

+ Code + Text

RAM Disk

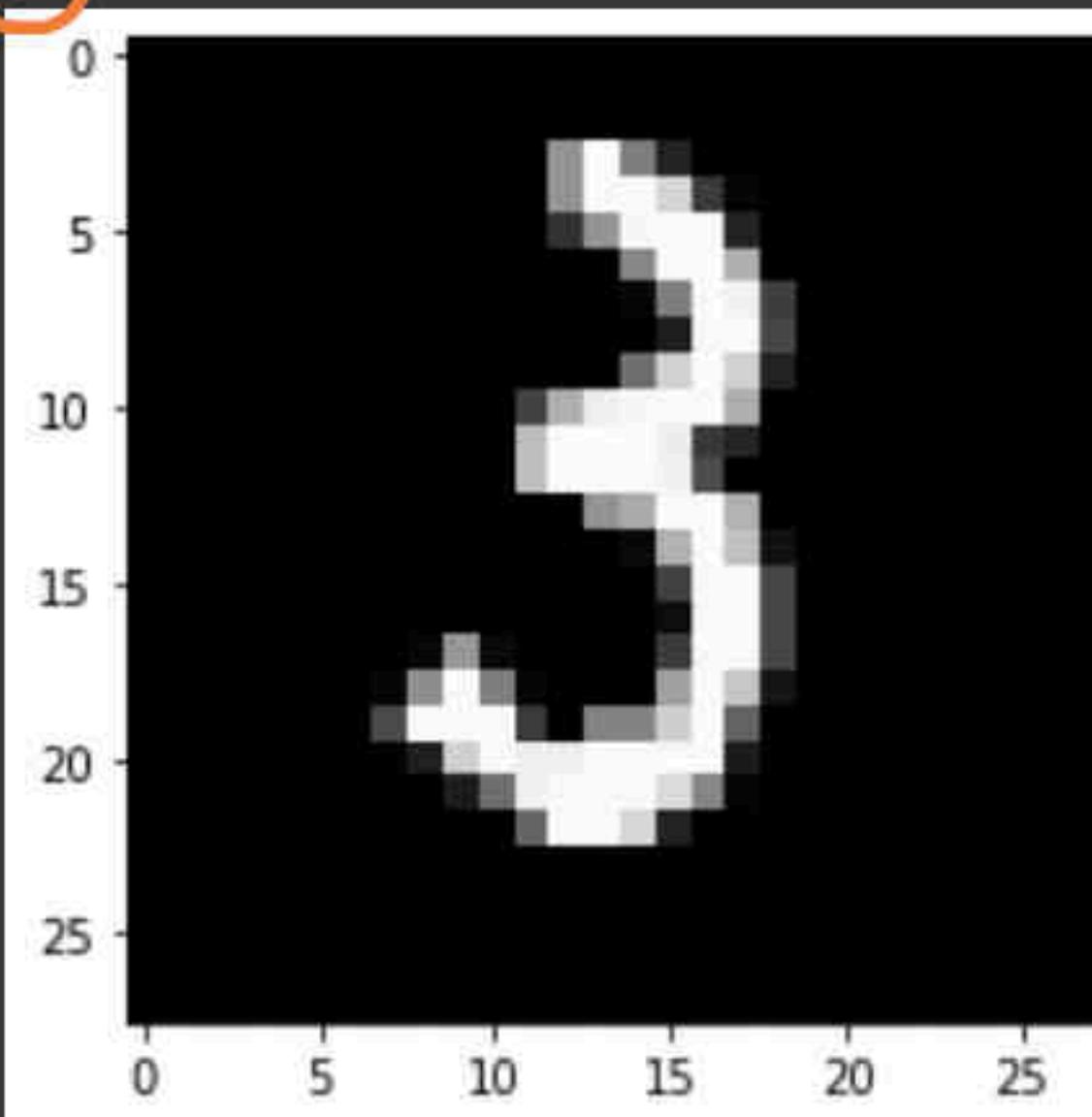


```
(60000, 28, 28)  
(60000, )  
(10000, 28, 28)  
(10000, )
```

```
[ ] idx=1234
    plt.imshow(x_train[idx], cmap='gray', interpolation='none')
    print(y_train[idx])
```

3

1





100 class classfn \rightarrow regression ?

$$\rightarrow \begin{bmatrix} c_1 & \dots & c_{100} \\ \underline{w}_1 & \dots & \underline{w}_{100} \\ 255 & \dots & 128 & \dots & 0 \end{bmatrix}$$

①

Ordinality

②

scale: number

Q

$$P_1 \dots P_{10}$$

$$H_p = -\sum p_i \log p_i$$

G

$$(-2) \cdot 0.5 \log 0.5 \\ = 0.3010$$

$$P_1 \quad P_2 \\ 0.5 \quad 0.5$$

$$(-10) \cdot 0.1 \cdot \log 0.1 = 1.0$$

$$P_1 \quad P_2 \quad \dots \quad P_{10} \\ 0.1 \quad 0.1 \quad \dots \quad 0.1$$

```
# number of class labels  
num_labels = len(np.unique(y_train))  
print(num_labels)
```

10

```
[ ] # convert y_i's to one-hot vectors
[ { y_train = to_categorical(y_train)
    y_test = to_categorical(y_test)
```

```
[ ] # Image: 2D matrix to 1D vector  
image_size = x_train.shape[1]  
input_size = image_size * image_size
```

```
# 2D to 1D + Normalize (0-255)
x_train = np.reshape(x_train, [-1, input_size])
x_train = x_train.astype('float32') / 255
x_test = np.reshape(x_test, [-1, input_size])
x_test = x_test.astype('float32') / 255
```

```
[ ] # Hyper-params of NN  
batch_size = 128 # typically 2^n  
hidden_units = 256
```

```
[ ] # convert y_i's to one-hot vectors  
y_train = to_categorical(y_train)  
y_test = to_categorical(y_test)
```

```
[ ] # Image: 2D matrix to 1D vector  
image_size = x_train.shape[1]  
input_size = image_size * image_size
```

```
# 2D to 1D + Normalize (0-255)
x_train = np.reshape(x_train, [-1, input_size])
x_train = x_train.astype('float32') / 255
x_test = np.reshape(x_test, [-1, input_size])
x_test = x_test.astype('float32') / 255
```

```
[ ] # Hyper-params of NN  
batch_size = 128 # typically 2^n  
hidden_units = 256
```

```
[ ] # Sequential API

# 3-layer MLP with ReLU
model = Sequential()
model.add(Dense(hidden_units, input_dim=input_size))
model.add(Activation('relu'))
```

28x28

768

6t | 2nd | ...

colab.research.google.com/drive/1tyv-ViqseMy1rTwGc6aLbnXk-5MC13g#scrollTo=lrBxXCgxpNaf

+ Code + Text

[] # convert y_i's to one-hot vectors
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)

{x}

[] # Image: 2D matrix to 1D vector
image_size = x_train.shape[1]
input_size = image_size * image_size

{ } # 2D to 1D + Normalize (0-255)
x_train = np.reshape(x_train, [-1, input_size])
x_train = x_train.astype('float32') / 255
x_test = np.reshape(x_test, [-1, input_size])
x_test = x_test.astype('float32') / 255

[] # Hyper-params of NN
batch_size = 128 # typically 2^n
hidden_units = 256

[] # Sequential API

3-layer MLP with ReLU
model = Sequential()
model.add(Dense(hidden_units, input_dim=input_size))
model.add(Activation('relu'))

RAM Disk

28x28 → 768dim

colab.research.google.com/drive/1tyv-ViqseMy1rTwGc6aLbnXk-5MC13g#scrollTo=lrBxXCgxpNaf

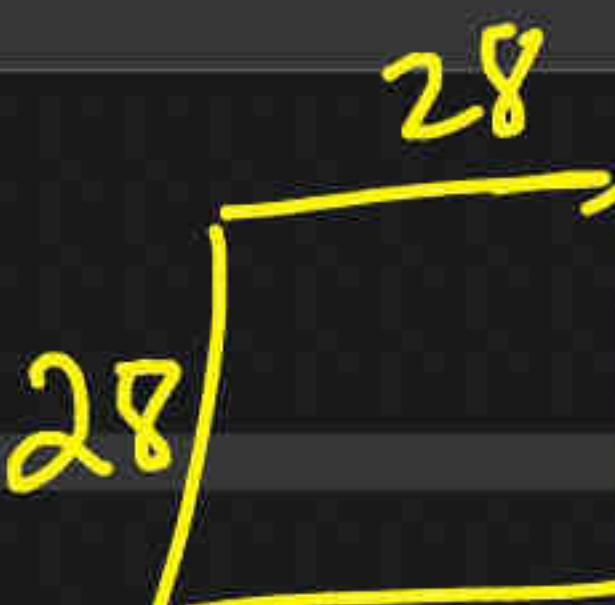
+ Code + Text

```
[ ] # convert y_i's to one-hot vectors
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)

[ ] # Image: 2D matrix to 1D vector
image_size = x_train.shape[1]
input_size = image_size * image_size
# 2D to 1D + Normalize (0-255)
x_train = np.reshape(x_train, [-1, input_size])
x_train = x_train.astype('float32') / 255
x_test = np.reshape(x_test, [-1, input_size])
x_test = x_test.astype('float32') / 255

[ ] # Hyper-params of NN
batch_size = 128 # typically 2^n
hidden_units = 256

[ ] # Sequential API
# 3-layer MLP with ReLU
model = Sequential()
model.add(Dense(hidden_units, input_dim=input_size))
model.add(Activation('relu'))
```



(black, 0 : min)
white
255 : max

{ Std → Mean-Centring }
Var-scaling

{ Images → Normalization }
Min-Max scaling

0 1

RAM Disk

Update

14 / 14

TF2_Keras_Intro.ipynb - Colab | srivastava14a.pdf | Dropout layer | Layer weight regularizers | 1502.03167.pdf | BatchNormalization layer | +

+ Code + Text

```
# IMAGE: 2D MATRIX TO 1D VECTOR
[ ] image_size = x_train.shape[1]
    input_size = image_size * image_size
```

```
# 2D to 1D + Normalize (0-255)
x_train = np.reshape(x_train, [-1, input_size])
x_train = x_train.astype('float32') / 255
x_test = np.reshape(x_test, [-1, input_size])
x_test = x_test.astype('float32') / 255
```

60K images

```
# Hyper-params of N
batch_size = 128 # ←
hidden_units = 256
```

```
[ ] # Sequential API

# 3-layer MLP with ReLU
model = Sequential()
model.add(Dense(hidden_units, input_dim=input_size))
model.add(Activation('relu'))
model.add(Dense(hidden_units))
model.add(Activation('relu'))
model.add(Dense(num_labels))
# this is the output for one-hot vector
model.add(Activation('softmax'))
```

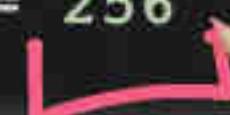
```
# IMAGE: 2D MATRIX TO 1D VECTOR
[ ] image_size = x_train.shape[1]
input_size = image_size * image_size

# 2D to 1D + Normalize (0-255)
x_train = np.reshape(x_train, [-1, input_size])
x_train = x_train.astype('float32') / 255
x_test = np.reshape(x_test, [-1, input_size])
x_test = x_test.astype('float32') / 255
```

```
# Hyper-params of NN  
batch_size = 128 # typically 2^  
hidden_units = 256
```

```
[ ] # Sequential API

# 3-layer MLP with ReLU
model = Sequential()
model.add(Dense(hidden_units, input_dim=input_size))
model.add(Activation('relu'))
model.add(Dense(hidden_units))
model.add(Activation('relu'))
model.add(Dense(num_labels))
# this is the output for one-hot vector
model.add(Activation('softmax'))
```



∞ TF2_Keras_Intro.ipynb - Colab | srivastava14a.pdf | K Dropout layer | K Layer weight regularizers | 1502.03167.pdf | K BatchNormalization layer | -

+ Code + Text

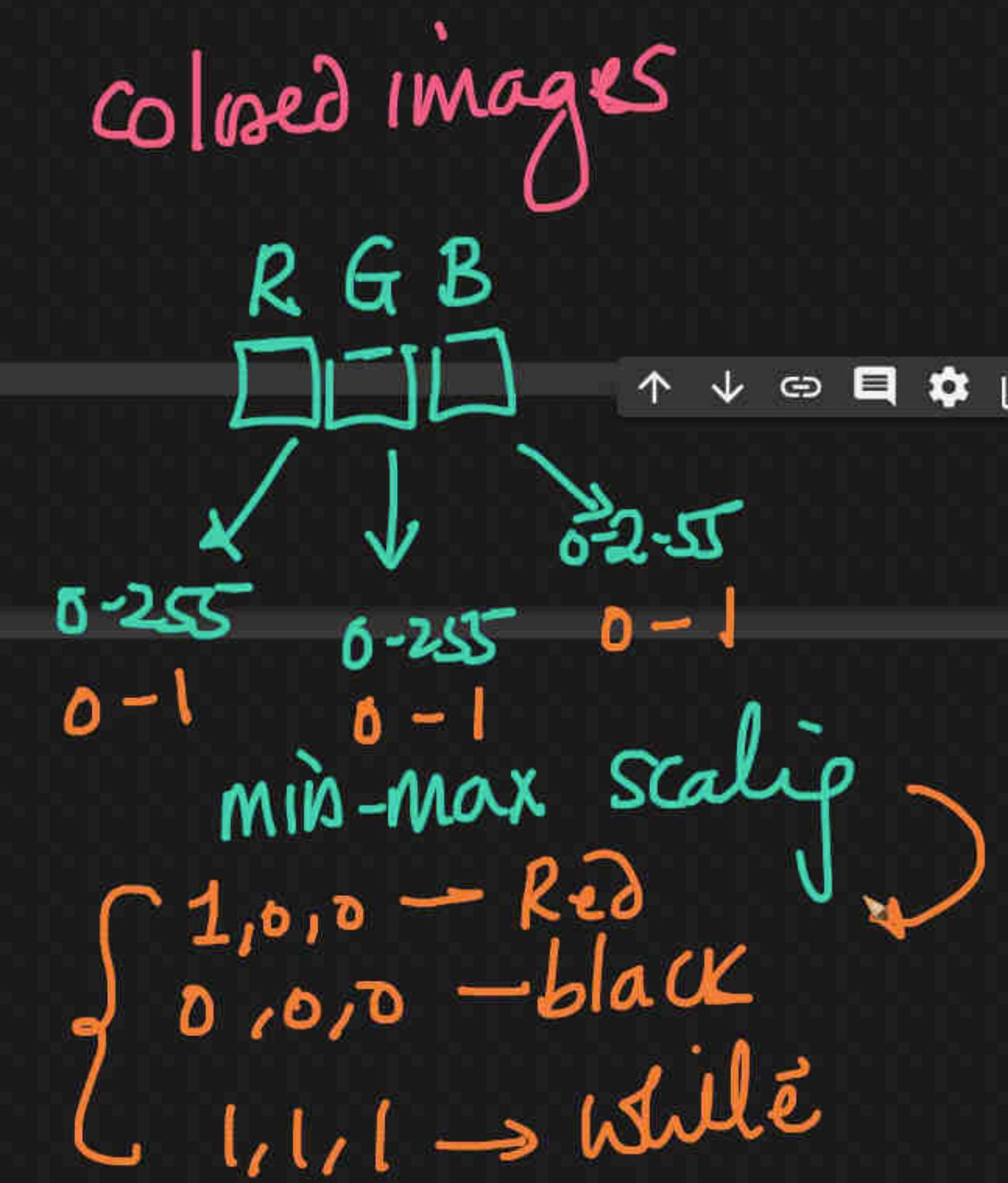
```
# Image: 2D matrix to 1D vector
[ ] image_size = x_train.shape[1]
    input_size = image_size * image_size

# 2D to 1D + Normalize (0-255)
x_train = np.reshape(x_train, [-1, input_size])
x_train = x_train.astype('float32') / 255
x_test = np.reshape(x_test, [-1, input_size])
x_test = x_test.astype('float32') / 255

# Hyper-params of NN
batch_size = 128 # typically 2^n
hidden_units = 256

[ ] # Sequential API

# 3-layer MLP with ReLU
model = Sequential()
model.add(Dense(hidden_units, input_dim=input_size))
model.add(Activation('relu'))
model.add(Dense(hidden_units))
model.add(Activation('relu'))
model.add(Dense(num_labels))
# this is the output for one-hot vector
model.add(Activation('softmax'))
```



colab.research.google.com/drive/1tyv-ViqseMy1rTwGc6aLbnXk-5MC13g#scrollTo=SN71uszYqbS9

+ Code + Text

```
# image: 2D matrix to 1D vector
[ ] image_size = x_train.shape[1]
input_size = image_size * image_size

# 2D to 1D + Normalize (0-255)
x_train = np.reshape(x_train, [-1, input_size])
x_train = x_train.astype('float32') / 255
x_test = np.reshape(x_test, [-1, input_size])
x_test = x_test.astype('float32') / 255

# Hyper-params of NN
batch_size = 128 # typically 2^n
hidden_units = 256
```

minibatch size
epoch → iterns ↗ batch-size

```
[ ] # Sequential API

# 3-layer MLP with ReLU
model = Sequential()
model.add(Dense(hidden_units, input_dim=input_size))
model.add(Activation('relu'))
model.add(Dense(hidden_units))
model.add(Activation('relu'))
model.add(Dense(num_labels))
# this is the output for one-hot vector
model.add(Activation('softmax'))
```

∞ TF2_Keras_Intro.ipynb - Colab | srivastava14a.pdf | K Dropout layer | K Layer weight regularizers | 1502.03167.pdf | K BatchNormalization layer | -

+ Code + Text

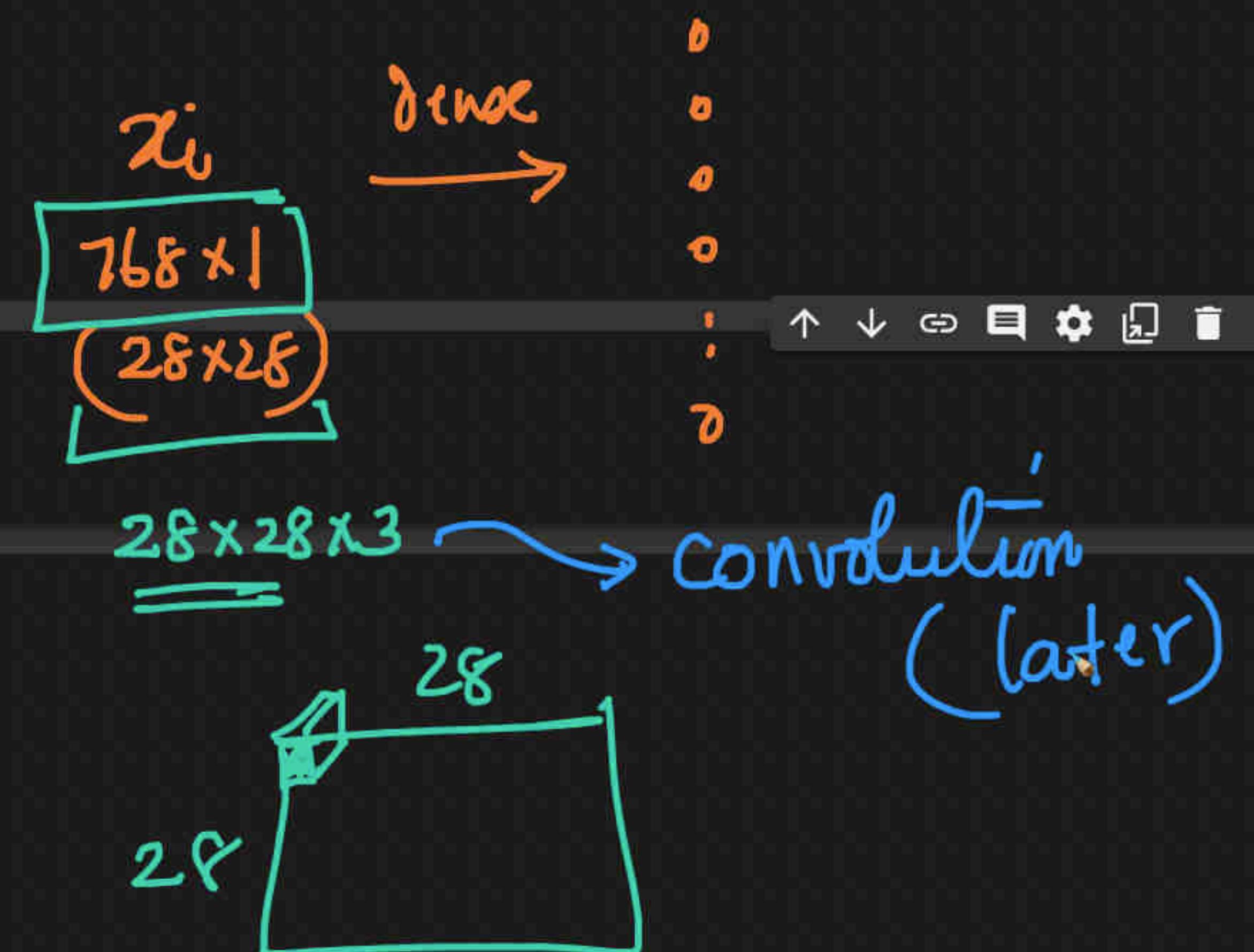
```
# IMAGE: 2D MATRIX TO 1D VECTOR
[ ] image_size = x_train.shape[1]
    input_size = image_size * image_size
```

```
# 2D to 1D + Normalize (0-255)
x_train = np.reshape(x_train, [-1, input_size])
x_train = x_train.astype('float32') / 255
x_test = np.reshape(x_test, [-1, input_size])
x_test = x_test.astype('float32') / 255
```

```
# Hyper-params of NN  
batch_size = 128 # typically 2^n  
hidden_units = 256
```

[] # Sequential API

```
# 3-layer MLP with ReLU
model = Sequential()
model.add(Dense(hidden_units, input_dim=input_size))
model.add(Activation('relu'))
model.add(Dense(hidden_units))
model.add(Activation('relu'))
model.add(Dense(num_labels))
# this is the output for one-hot vector
model.add(Activation('softmax'))
```



∞ TF2_Keras_Intro.ipynb - Colab × srivastava14a.pdf × K Dropout layer × K Layer weight regularizers × 1502.03167.pdf × K BatchNormalization layer × -

+ Code + Text

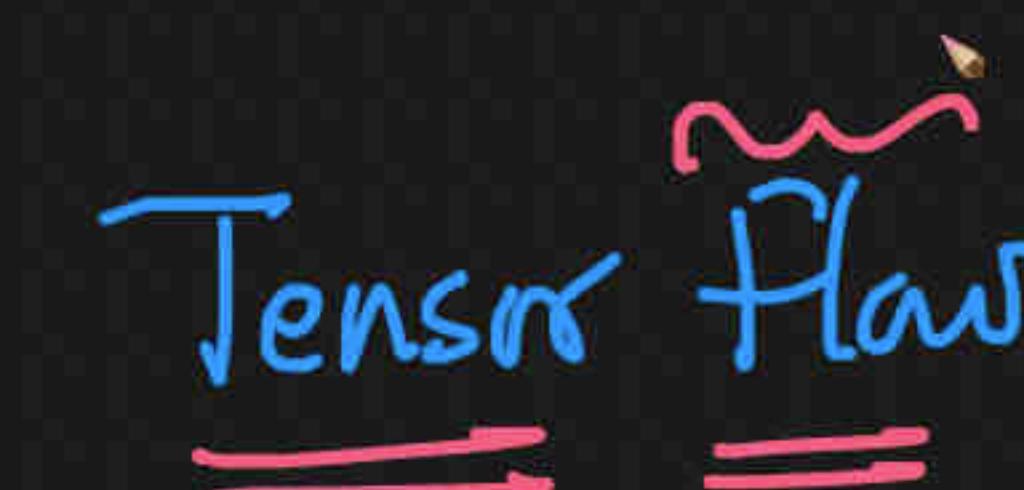
```
# IMAGE: 2D MATRIX TO 1D VECTOR
[ ] image_size = x_train.shape[1]
    input_size = image_size * image_size
```

```
# 2D to 1D + Normalize (0-255)
x_train = np.reshape(x_train, [-1, input_size])
x_train = x_train.astype('float32') / 255
x_test = np.reshape(x_test, [-1, input_size])
x_test = x_test.astype('float32') / 255
```

```
# Hyper-params of NN  
batch_size = 128 # typically 2^n  
hidden_units = 256
```

```
[ ] # Sequential API

# 3-layer MLP with ReLU
model = Sequential()
model.add(Dense(hidden_units, input_dim=input_size))
model.add(Activation('relu'))
model.add(Dense(hidden_units))
model.add(Activation('relu'))
model.add(Dense(num_labels))
# this is the output for one-hot vector
model.add(Activation('softmax'))
```



∞ TF2_Keras_Intro.ipynb - Colab × srivastava14a.pdf × K Dropout layer × K Layer weight regularizers × 1502.03167.pdf × K BatchNormalization layer × +

colab.research.google.com/drive/1tyv-ViqseMy1rTwGc6aLbnXk-5MC13g#scrollTo=gNv36JiGqjTK

+ Code + Text

activation (ACTIVATION) (None, 256) 0 ✓ RAM Disk

dense_1 (Dense) (None, 256) 65792

activation_1 (Activation) (None, 256) 0

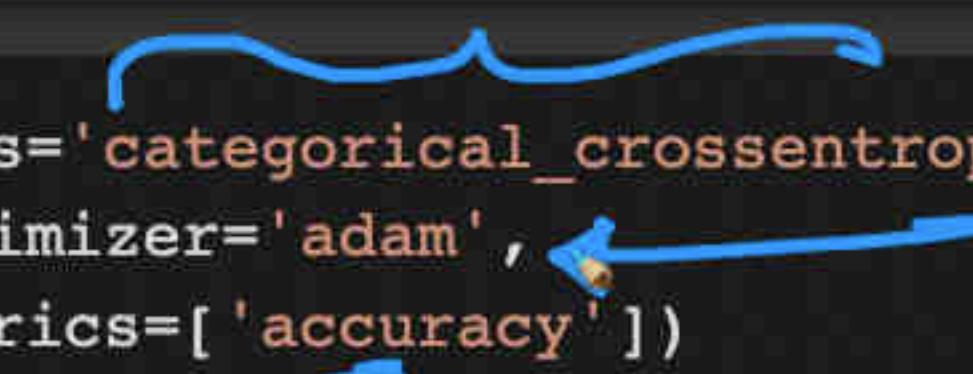
{x} dense_2 (Dense) (None, 10) 2570

activation_2 (Activation) (None, 10) 0

=====

Total params: 269,322
Trainable params: 269,322
Non-trainable params: 0

```
[ ] model.compile(loss='categorical_crossentropy',
                  optimizer='adam',
                  metrics=['accuracy'])
```



```
[ ] # train the network
%%time
model.fit(x_train, y_train, epochs=20, batch_size=batch_size)

#60000/128 ~ 469
```

EPOCH 1 / 20

22 / 24

TF2_Keras_Intro.ipynb - Colab srivastava14a.pdf Dropout layer Layer weight regularizers 1502.03167.pdf BatchNormalization layer + colab.research.google.com/drive/1tyv-ViqseMy1rTwGc6aLbnXk-5MC13g#scrollTo=gNv36JiGqjTK

+ Code + Text RAM Disk ✓

activation_2 (Activation) (None, 10) 0

Total params: 269,322
Trainable params: 269,322
Non-trainable params: 0

[] model.compile(loss='categorical_crossentropy',
optimizer='adam',
metrics=['accuracy'])

[] # train the network
%%time
model.fit(x_train, y_train, epochs=20, batch_size=batch_size)
#60000/128 ~ 469

Epoch 1/20
469/469 [=====] - 4s 3ms/step - loss: 0.2596 - accuracy: 0.9250
Epoch 2/20
469/469 [=====] - 1s 3ms/step - loss: 0.0949 - accuracy: 0.9712
Epoch 3/20
469/469 [=====] - 1s 3ms/step - loss: 0.0625 - accuracy: 0.9808
Epoch 4/20
469/469 [=====] - 2s 3ms/step - loss: 0.0433 - accuracy: 0.9864

∞ TF2_Keras_Intro.ipynb - Colab x srivastava14a.pdf x K Dropout layer x K Layer weight regularizers x 1502.03167.pdf x K BatchNormalization layer x + colab.research.google.com/drive/1tyv-ViqseMy1rTwGc6aLbnXk-5MC13g#scrollTo=gNv36JiGqjTK Reconnect + ⚙️ Update

+ Code + Text

```
%%time
[ ] model.fit(x_train, y_train, epochs=20, batch_size=batch_size)
#60000/128 ~ 469
```

epoch: 60K = 468...
128 ✓

```
Epoch 1/20
469/469 [=====] - 4s 3ms/step - loss: 0.2596 - accuracy: 0.9250
Epoch 2/20
469/469 [=====] - 1s 3ms/step - loss: 0.0949 - accuracy: 0.9712
Epoch 3/20
469/469 [=====] - 1s 3ms/step - loss: 0.0625 - accuracy: 0.9808
Epoch 4/20
469/469 [=====] - 2s 3ms/step - loss: 0.0433 - accuracy: 0.9864
Epoch 5/20
469/469 [=====] - 2s 5ms/step - loss: 0.0332 - accuracy: 0.9893
Epoch 6/20
469/469 [=====] - 2s 5ms/step - loss: 0.0237 - accuracy: 0.9923
Epoch 7/20
469/469 [=====] - 3s 6ms/step - loss: 0.0190 - accuracy: 0.9939
Epoch 8/20
469/469 [=====] - 3s 6ms/step - loss: 0.0185 - accuracy: 0.9938
Epoch 9/20
469/469 [=====] - 3s 6ms/step - loss: 0.0166 - accuracy: 0.9943
Epoch 10/20
469/469 [=====] - 2s 5ms/step - loss: 0.0140 - accuracy: 0.9956
Epoch 11/20
469/469 [=====] - 2s 4ms/step - loss: 0.0134 - accuracy: 0.9952
Epoch 12/20
```

decy in

24/26

∞ TF2_Keras_Intro.ipynb - Colab × srivastava14a.pdf × K Dropout layer × K Layer weight regularizers × 1502.03167.pdf × K BatchNormalization layer × + colab.research.google.com/drive/1tyv-ViqseMy1rTwGc6aLbnXk-5MC13g#scrollTo=gNv36JiGqjTK Reconnect Update

+ Code + Text

```
[ ] model.compile(loss='categorical_crossentropy',
                  optimizer='adam',
                  metrics=['accuracy'])

{x}
[ ] # train the network
%%time
model.fit(x_train, y_train, epochs=20, batch_size=batch_size)

#60000/128 ~ 469
```

```
Epoch 1/20
469/469 [=====] - 4s 3ms/step - loss: 0.2596 - accuracy: 0.9250
Epoch 2/20
469/469 [=====] - 1s 3ms/step - loss: 0.0949 - accuracy: 0.9712
Epoch 3/20
469/469 [=====] - 1s 3ms/step - loss: 0.0625 - accuracy: 0.9808
Epoch 4/20
469/469 [=====] - 2s 3ms/step - loss: 0.0433 - accuracy: 0.9864
Epoch 5/20
469/469 [=====] - 2s 5ms/step - loss: 0.0332 - accuracy: 0.9893
Epoch 6/20
469/469 [=====] - 2s 5ms/step - loss: 0.0237 - accuracy: 0.9923
Epoch 7/20
469/469 [=====] - 3s 6ms/step - loss: 0.0190 - accuracy: 0.9939
Epoch 8/20
469/469 [=====] - 3s 6ms/step - loss: 0.0185 - accuracy: 0.9938
Epoch 9/20
```



+ Code + Text

Epoch 11/20
[] 469/469 [=====] - 2s 4ms/step - loss: 0.0134 - accuracy: 0.9952

Epoch 12/20
469/469 [=====] - 1s 3ms/step - loss: 0.0103 - accuracy: 0.9969

Epoch 13/20
469/469 [=====] - 1s 3ms/step - loss: 0.0113 - accuracy: 0.9961

Epoch 14/20
469/469 [=====] - 1s 3ms/step - loss: 0.0100 - accuracy: 0.9965

Epoch 15/20
469/469 [=====] - 1s 3ms/step - loss: 0.0073 - accuracy: 0.9976

Epoch 16/20
469/469 [=====] - 1s 3ms/step - loss: 0.0082 - accuracy: 0.9973

Epoch 17/20
469/469 [=====] - 1s 3ms/step - loss: 0.0064 - accuracy: 0.9980

Epoch 18/20
469/469 [=====] - 1s 3ms/step - loss: 0.0071 - accuracy: 0.9976

Epoch 19/20
469/469 [=====] - 1s 3ms/step - loss: 0.0091 - accuracy: 0.9968

Epoch 20/20
469/469 [=====] - 1s 3ms/step - loss: 0.0076 - accuracy: 0.9973

CPU times: user 36.1 s, sys: 4.74 s, total: 40.8 s

Wall time: 38.6 s

<keras.callbacks.History at 0x7fec604b7190>

[] # test dataset performance
loss, acc = model.evaluate(x_test,
 y_test,
 batch_size=32)



+ Code + Text

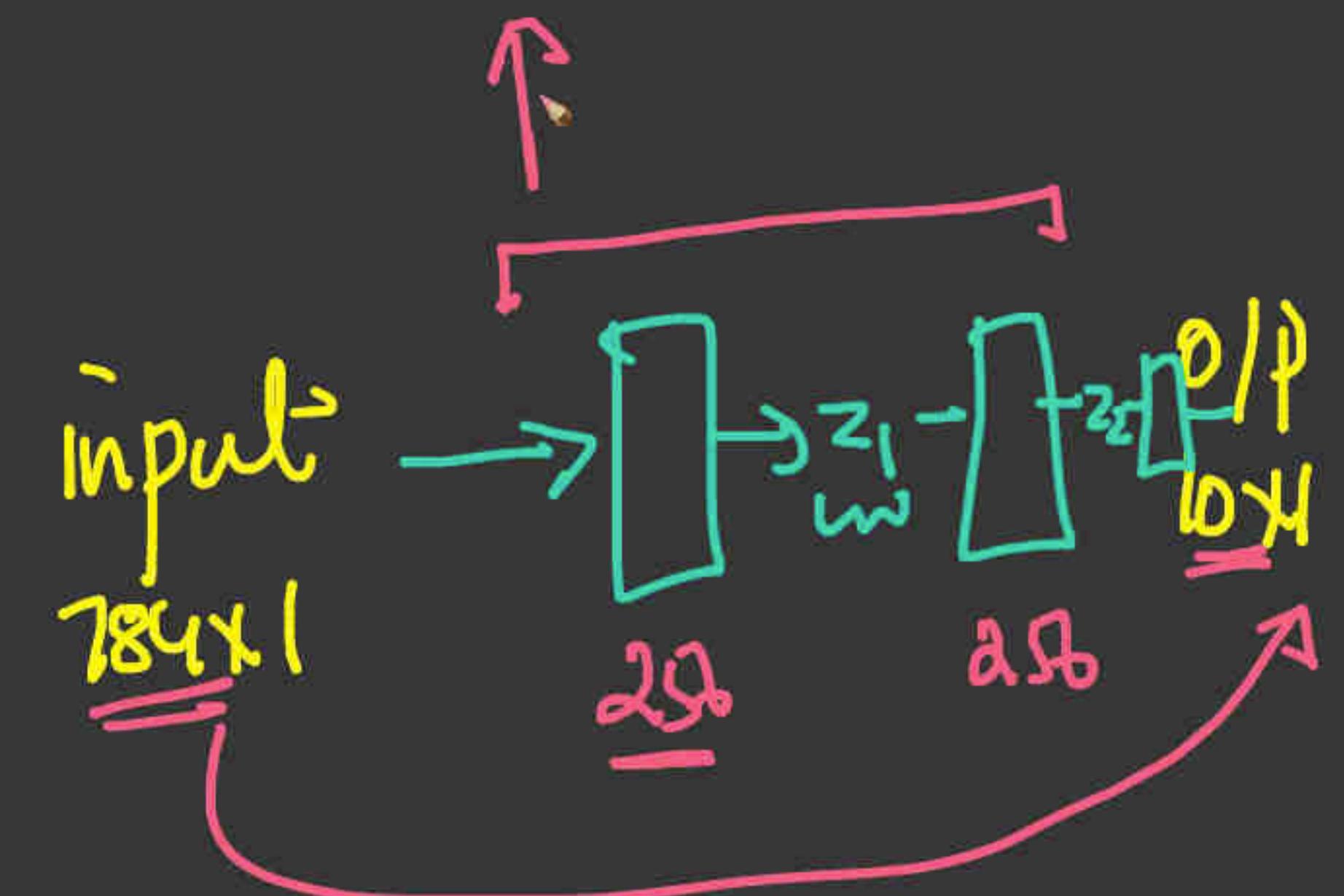
Model: "sequential"

Layer (type)	Output Shape	Param #
<hr/>		
dense (Dense)	(None, 256)	200960
activation (Activation)	(None, 256)	0
dense_1 (Dense)	(None, 256)	65792
activation_1 (Activation)	(None, 256)	0
dense_2 (Dense)	(None, 10)	2570
activation_2 (Activation)	(None, 10)	0
<hr/>		

Total params: 269,322

Trainable params: 269,322

Non-trainable params: 0



```
[ ] model.compile(loss='categorical_crossentropy',
                  optimizer='adam',
                  metrics=[ 'accuracy'])
```

∞ TF2_Keras_Intro.ipynb - Colab × srivastava14a.pdf × K Dropout layer × K Layer weight regularizers × 1502.03167.pdf × K BatchNormalization layer × +

colab.research.google.com/drive/1tyv-ViqseMy1rTwGc6aLbnXk-5MC13g#scrollTo=3t7v8jm9wDEb

+ Code + Text Reconnect

Epoch 16/20
469/469 [=====] - 1s 3ms/step - loss: 0.0082 - accuracy: 0.9973
Epoch 17/20
469/469 [=====] - 1s 3ms/step - loss: 0.0064 - accuracy: 0.9980
Epoch 18/20
469/469 [=====] - 1s 3ms/step - loss: 0.0071 - accuracy: 0.9976
Epoch 19/20
469/469 [=====] - 1s 3ms/step - loss: 0.0091 - accuracy: 0.9968
Epoch 20/20
469/469 [=====] - 1s 3ms/step - loss: 0.0076 - accuracy: 0.9973
CPU times: user 36.1 s, sys: 4.74 s, total: 40.8 s
Wall time: 38.6 s
<keras.callbacks.History at 0x7fec604b7190>

test dataset performance
loss, acc = model.evaluate(x_test,
y_test,
batch_size=batch_size,
verbose=0)
print("\nTest accuracy: %.1f%%" % (100.0 * acc))
print(loss)

Overfit or Underfit?

Test accuracy: 98.3%

0 09489700198173522

28 / 30

∞ TF2_Keras_Intro.ipynb - Colab × srivastava14a.pdf × K Dropout layer × K Layer weight regularizers × 1502.03167.pdf × K BatchNormalization layer × +

colab.research.google.com/drive/1tyv-ViqseMy1rTwGc6aLbnXk-5MC13g#scrollTo=3t7v8jm9wDEb

+ Code + Text Reconnect

Epoch 18/20
[] 469/469 [=====] - 1s 3ms/step - loss: 0.0071 - accuracy: 0.9976
Epoch 19/20
469/469 [=====] - 1s 3ms/step - loss: 0.0091 - accuracy: 0.9968
Epoch 20/20
469/469 [=====] - 1s 3ms/step - loss: 0.0076 - accuracy: 0.9973
CPU times: user 36.1 s, sys: 4.74 s, total: 40.8 s
Wall time: 38.6 s
<keras.callbacks.History at 0x7fec604b7190>

test dataset performance
loss, acc = model.evaluate(x_test,
 y_test,
 batch_size=batch_size,
 verbose=0)
print("\nTest accuracy: %.1f%%" % (100.0 * acc))
print(loss)

Overfit or Underfit?

Test accuracy: 98.3%
0.09489700198173523

Test

TF 2 and Keras: Functional API

+ Code + Text Reconnect

Epoch 17/20
[] 469/469 [=====] - 1s 3ms/step - loss: 0.0064 - accuracy: 0.9980

Epoch 18/20
469/469 [=====] - 1s 3ms/step - loss: 0.0071 - accuracy: 0.9976

Epoch 19/20
469/469 [=====] - 1s 3ms/step - loss: 0.0091 - accuracy: 0.9968

Epoch 20/20
469/469 [=====] - 1s 3ms/step - loss: 0.0076 - accuracy: 0.9973

CPU times: user 36.1 s, sys: 4.74 s, total: 40.8 s

Wall time: 38.6 s

<keras.callbacks.History at 0x7fec604b7190>

test dataset performance
loss, acc = model.evaluate(x_test,
 y_test,
 batch_size=batch_size,
 verbose=0)
print("\nTest accuracy: %.1f%%" % (100.0 * acc))
print(loss)

Overfit or Underfit?

Test accuracy: 98.3%
0.09489700198173523

Train loss

Test loss

ISX

min CE loss

+ Code + Text Reconnect

```
469/469 [=====] - 1s 3ms/step - loss: 0.0071 - accuracy: 0.9976
[ ] Epoch 19/20
469/469 [=====] - 1s 3ms/step - loss: 0.0091 - accuracy: 0.9968
Epoch 20/20
469/469 [=====] - 1s 3ms/step - loss: 0.0076 - accuracy: 0.9973
CPU times: user 36.1 s, sys: 4.74 s, total: 40.8 s
Wall time: 38.6 s
<keras.callbacks.History at 0x7fec604b7190>
```

```
# test dataset performance
loss, acc = model.evaluate(x_test,
                            y_test,
                            batch_size=batch_size,
                            verbose=0)
print("\nTest accuracy: %.1f%%" % (100.0 * acc))
print(loss)

# Overfit or Underfit?
```

Test accuracy: 98.3%
0.09439700198173523

TF 2 and Keras: Functional API

tensorflow.org/guide/keras/functional

TensorFlow

Install Learn API Resources Community Why TensorFlow

Search English GitHub Sign in

Overview Tutorials Guide Migrate to TF2 TF 1 ↗

Filter

Tensors Variables Automatic differentiation Graphs and functions Modules, layers, and models Training loops

Keras

The Sequential model

The Functional API

Training and evaluation with the built-in methods

Making new Layers and Models via subclassing

Save and load Keras models

Working with preprocessing layers

Customize what happens in Model.fit

Writing a training loop from scratch

Recurrent Neural Networks (RNN) with Keras

Masking and padding with Keras

Writing your own callbacks

Transfer learning and fine-tuning

Training Keras models with TensorFlow Cloud

Total params: 55,050
Trainable params: 55,050

You can also plot the model as a graph:

```
keras.utils.plot_model(model, "my_first_model.png")
```

input_1: InputLayer

dense: Dense

dense_1: Dense

dense_2: Dense

And, optionally, display the input and output shapes of each layer in the plotted graph:

```
keras.utils.plot_model(model, "my_first_model_with_shape_info.png", show_shapes=True)
```

On this page

Setup

Introduction

Training, evaluation, and inference

Save and serialize

Use the same graph of layers to define multiple models

All models are callable, just like layers

Manipulate complex graph topologies

Models with multiple inputs and outputs

A toy ResNet model

Shared layers

Extract and reuse nodes in the graph of layers

Extend the API using custom layers

When to use the functional API

Functional API strengths:

Functional API weakness:

Mix-and-match API styles

∞ TF2_Keras_Intro.ipynb - Colab | srivastava14a.pdf | K Dropout layer | K Layer weight regularizers | K 1502.03167.pdf | K BatchNormalization layer | The Functional API | TensorFlow | +

tensorflow.org/guide/keras/functional

TensorFlow

Install Learn API Resources Community Why TensorFlow Search English GitHub Sign in

Tensors
Variables
Automatic differentiation
Graphs and functions
Modules, layers, and models
Training loops

Keras
The Sequential model
The Functional API
Training and evaluation with the built-in methods
Making new Layers and Models via subclassing
Save and load Keras models
Working with preprocessing layers
Customize what happens in Model.fit
Writing a training loop from scratch
Recurrent Neural Networks (RNN) with Keras
Masking and padding with Keras
Writing your own callbacks
Transfer learning and fine-tuning
Training Keras models with TensorFlow Cloud

Filter

dense_2: Dense

And, optionally, display the input and output shapes of each layer in the plotted graph:

```
keras.utils.plot_model(model, "my_first_model_with_shape_info.png", show_shapes=True)
```

input_1: InputLayer
input: [(None, 784)]
output: [(None, 784)]

64
dense: Dense
input: (None, 784)
output: (None, 64)

64
dense_1: Dense
input: (None, 64)
output: (None, 64)

64
dense_2: Dense
input: (None, 64)
output: (None, 10)

On this page

Setup
Introduction
Training, evaluation, and inference
Save and serialize
Use the same graph of layers to define multiple models
All models are callable, just like layers
Manipulate complex graph topologies
Models with multiple inputs and outputs
A toy ResNet model
Shared layers
Extract and reuse nodes in the graph of layers
Extend the API using custom layers
When to use the functional API
Functional API strengths:
Functional API weakness:
Mix-and-match API styles

This figure and the code are almost identical. In the code version, the connection arrows are replaced by the call operation.

A "graph of layers" way to create models that closely

∞ TF2_Keras_Intro.ipynb - Colab | srivastava14a.pdf | K Dropout layer | K Layer weight regularizers | K 1502.03167.pdf | K BatchNormalization layer | The Functional API | TensorFlow | +

tensorflow.org/guide/keras/functional

TensorFlow

- Install
- Learn
- API
- Resources
- Community
- Why TensorFlow

Search

- English
- GitHub
- Sign in

Overview Tutorials Guide Migrate to TF2 TF 1 ↗

Filter

Tensors Variables Automatic differentiation Graphs and functions Modules, layers, and models Training loops

Keras

The Sequential model

The Functional API

Training and evaluation with the built-in methods

Making new Layers and Models via subclassing

Save and load Keras models

Working with preprocessing layers

Customize what happens in Model.fit

Writing a training loop from scratch

Recurrent Neural Networks (RNN) with Keras

Masking and padding with Keras

Writing your own callbacks

Transfer learning and fine-tuning

Training Keras models with TensorFlow Cloud

```
keras.utils.plot_model(model, "multi_input_and_output_model.png", show_shapes=True)
```

When compiling this model, you can assign different losses to each output. You can even assign different weights to each loss – to modulate their contribution to the total training loss.

```
model.compile(
    optimizer=keras.optimizers.RMSprop(1e-3),
    loss=[
        keras.losses.BinaryCrossentropy(from_logits=True),
        keras.losses.CategoricalCrossentropy(from_logits=True),
    ],
    loss_weights=[1.0, 0.2],
)
```

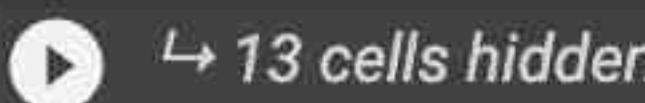
On this page

- Setup
- Introduction
- Training, evaluation, and inference
- Save and serialize
- Use the same graph of layers to define multiple models
- All models are callable, just like layers
- Manipulate complex graph topologies**
- Models with multiple inputs and outputs
- A toy ResNet model
- Shared layers
- Extract and reuse nodes in the graph of layers
- Extend the API using custom layers
- When to use the functional API
- Functional API strengths:
- Functional API weaknesses:
- Mix-and-match API styles

34 / 36

+ Code + Text

Reconnect ▾



↳ 13 cells hidden

A set of small, light-gray navigation icons typically found in software like Microsoft Word or Google Docs. From left to right, they include: a double-headed vertical arrow (for zoom), a double-headed horizontal arrow (for page navigation), a circular arrow (for refresh), a magnifying glass (for search), a double-headed arrow (for document navigation), a pencil (for edit), a double-headed arrow with a lock (for protection), a trash can (for delete), and three vertical dots (for more options).

▼ TF 2 and Keras: Functional API

```
[ ] print(x_train.shape)
```

(60000, 784)

```
[ ] inputs = keras.Input(shape=(784,))
inputs.shape
```

✓ TensorShape([None, 784])

```
[ ] # Functional API: much more intuitive  
# Refer: https://www.tensorflow.org/guide/keras/functional  
x1 = layers.Dense(hidden_units, activation="relu")(inputs)  
x2 = layers.Dense(hidden_units, activation="relu")(x1)  
outputs = layers.Dense(num_labels, activation="softmax")(x2)
```

```
model = keras.Model(inputs=inputs, outputs=outputs, name="simple_model")
model.summary()
```

+ Code + Text

Reconnect



```
[ ] inputs = keras.Input(shape=(784,))
inputs.shape
```

```
{x} TensorShape([None, 784])
```

```
[ ] # Functional API: much more intuitive
# Refer: https://www.tensorflow.org/guide/keras/functional
x1 = layers.Dense(hidden_units, activation="relu")(inputs)
x2 = layers.Dense(hidden_units, activation="relu")(x1)
outputs = layers.Dense(num_labels, activation="softmax")(x2)

model = keras.Model(inputs=inputs, outputs=outputs, name="simple_model")
model.summary()
```

Model: "simple_model"

Layer (type)	Output Shape	Param #
<hr/>		
input_1 (InputLayer)	[(None, 784)]	0
dense_3 (Dense)	(None, 256)	200960
dense_4 (Dense)	(None, 256)	65792

colab.research.google.com/drive/1tyv-ViqseMy1rTwGc6aLbnXk-5MC13g#scrollTo=sGo6QATHoTCL

+ Code + Text Reconnect  

```
[ ] inputs = keras.Input(shape=(784,))
inputs.shape
TensorShape([None, 784])
```

*shape
inputs [-, 784]*

```
[ ] # Functional API: much more intuitive
# Refer: https://www.tensorflow.org/guide/keras/functional
x1 = layers.Dense(hidden_units, activation="relu")(inputs)
x2 = layers.Dense(hidden_units, activation="relu")(x1)
outputs = layers.Dense(num_labels, activation="softmax")(x2)

model = keras.Model(inputs=inputs, outputs=outputs, name="simple_model")
model.summary()
```

*dense (256, Relu)
dense (256, Relu)
dense (10, Softmax)
op*

Model: "simple_model"

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 784)]	0
dense_3 (Dense)	(None, 256)	200960
dense_4 (Dense)	(None, 256)	65792

37 / 39

∞ TF2_Keras_Intro.ipynb - Colab × srivastava14a.pdf × K Dropout layer × K Layer weight regularizers × 1502.03167.pdf × K BatchNormalization layer × The Functional API | TensorFlow × + colab.research.google.com/drive/1tyv-ViqseMy1rTwGc6aLbnXk-5MC13g#scrollTo=sGo6QATHoTCL

+ Code + Text Reconnect

[] inputs = keras.Input(shape=(784,))
inputs.shape

{x} TensorShape([None, 784])

[] # Functional API: much more intuitive
Refer: <https://www.tensorflow.org/guide/keras/functional>
x1 = layers.Dense(hidden_units, activation="relu")(inputs)
x2 = layers.Dense(hidden_units, activation="relu")(x1)
outputs = layers.Dense(num_labels, activation="softmax")(x2)

{ model = keras.Model(inputs=inputs, outputs=outputs, name="simple_model")
model.summary()

Model: "simple_model"

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 784)]	0
dense_3 (Dense)	(None, 256)	200960
dense_4 (Dense)	(None, 256)	65792

38 / 40

∞ TF2_Keras_Intro.ipynb - Colab × srivastava14a.pdf × K Dropout layer × K Layer weight regularizers × 1502.03167.pdf × K BatchNormalization layer × The Functional API | TensorFlow × +

colab.research.google.com/drive/1tyv-ViqseMy1rTwGc6aLbnXk-5MC13g#scrollTo=RO_2IJAn0Xje

+ Code + Text Reconnect

Dropout: Regularization through randomization

▶ [] ↳ 5 cells hidden

▶ GPUs vs CPUs

[] ↳ 9 cells hidden

▶ Tensorboard and Computational Graph

[] ↳ 2 cells hidden

Batch Normalization

[] ↳ 4 cells hidden

Reconnect

Up Down Link Comment Edit Copy Delete More

File Edit View Insert Cell Kernel Help

40 / 42

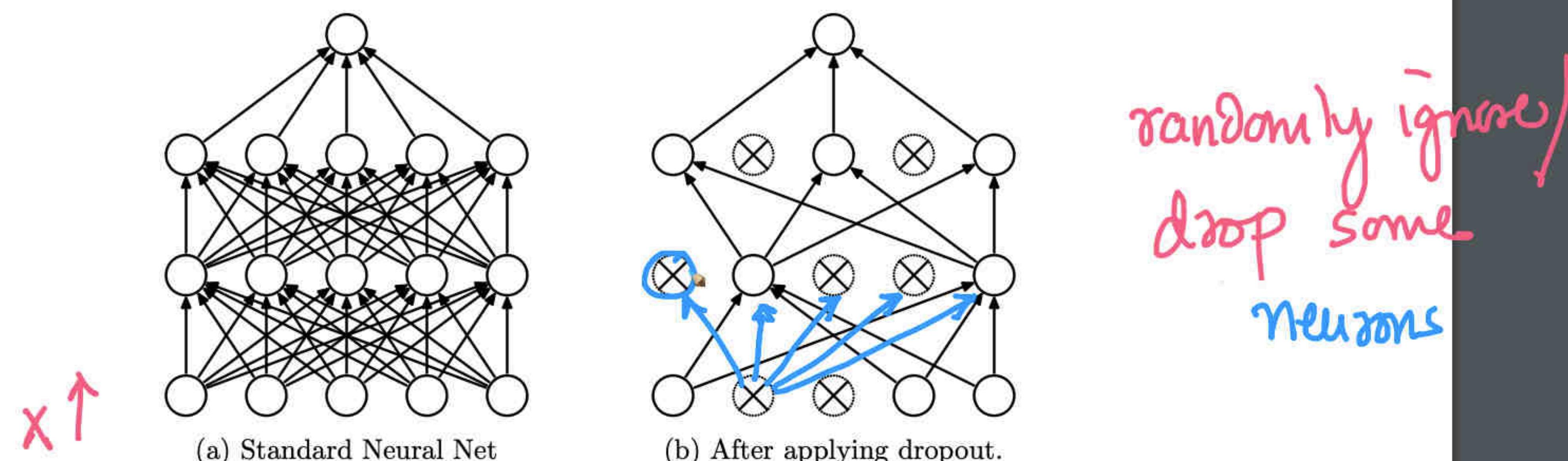


Figure 1: Dropout Neural Net Model. **Left:** A standard neural net with 2 hidden layers. **Right:** An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

its posterior probability given the training data. This can sometimes be approximated quite well for simple or small models (Xiong et al., 2011; Salakhutdinov and Mnih, 2008), but we would like to approach the performance of the Bayesian gold standard using considerably less computation. We propose to do this by approximating an equally weighted geometric mean of the predictions of the individual models, while sharing parameters.

srivastava14a.pdf

2 / 30 - 200% + ⌂ ⌂

Train

FP & BP

ilern ↑

(a) Standard Neural Net

Dropout layer

Layer weight regularizers

1502.03167.pdf

BatchNormalization layer

The Functional API | Tensor

cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf

Update

drop-out rate
= $0 \cdot 2$

randomly
 $\gamma = 0 - 1 \cdot 0$

$\gamma \leq 0 \cdot 2$

Figure 1: Dropout Neural Net Model. Left: A standard neural net with 2 hidden layers. Right: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

its posterior probability given the training data. This can sometimes be approximated quite well for simple or small models (Xiong et al., 2011; Salakhutdinov and Mnih, 2008), but we would like to approach the performance of the Bayesian gold standard using considerably less computation. We propose to do this by approximating an equally weighted geometric mean of the pred... share parameters.

srivastava14a.pdf

2 / 30 - 200% + ⌂ ⌂

srivastava14a.pdf

2000

$\text{rate} = 20\%$

$r = 0 - k_0$

$\{ \bar{d}_i \}_{i=1}^n$

(let) 20×500 learn

10,000

share parameters.

The diagram illustrates a neural network model. On the left, labeled '(a) Standard Neural Net', is a fully connected network with three layers: an input layer of 4 nodes, two hidden layers of 4 nodes each, and an output layer of 1 node. Every node in one layer is connected to every node in the next layer by a directed arrow pointing upwards. A blue arrow points from the input layer to the first hidden layer, and another blue arrow points from the second hidden layer to the output layer. Above this diagram, handwritten text reads 'rate = 20%' with two blue horizontal lines underneath. On the right, labeled '(b) After applying dropout.', is a similar network structure where some connections have been removed. A red circle highlights a node in the first hidden layer that has lost all its incoming connections, indicating it has been 'dropped'. Blue arrows point from the input layer to the first hidden layer and from the second hidden layer to the output layer, mirroring the connections in the standard network. Above this diagram, handwritten text reads ' $r = 0 - k_0$ ' with two blue horizontal lines underneath. To the right of the diagram, a blue curly brace groups several handwritten labels: ' \bar{d}_i ' followed by a sequence of numbers (1, 2, ..., n), and '(let) 20×500 learn' with two blue horizontal lines underneath. Below the diagram, handwritten text reads 'share parameters.' with two blue horizontal lines underneath.

Figure 1: Dropout Neural Net Model. **Left:** A standard neural net with 2 hidden layers. **Right:** An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

its posterior probability given the training data. This can sometimes be approximated quite well for simple or small models (Xiong et al., 2011; Salakhutdinov and Mnih, 2008), but we would like to approach the performance of the Bayesian gold standard using considerably less computation. We propose to do this by approximating an equally weighted geometric mean of the pred

share parameters.

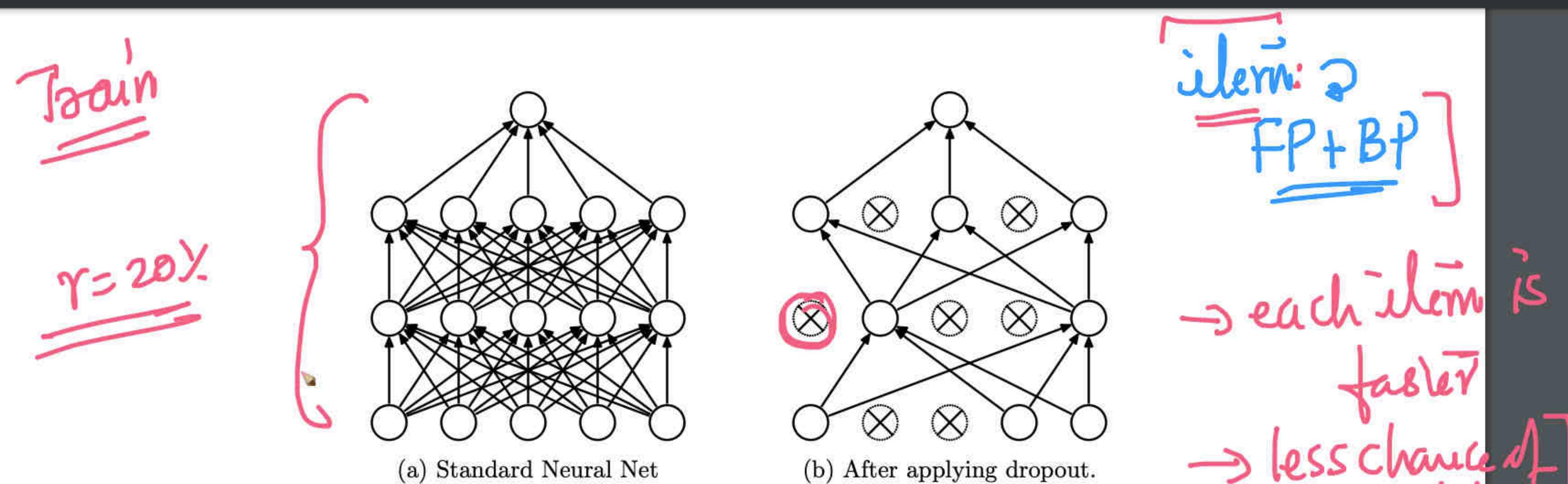


Figure 1: Dropout Neural Net Model. **Left:** A standard neural net with 2 hidden layers. **Right:** An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

its posterior probability given the training data. This can sometimes be approximated quite well for simple or small models (Xiong et al., 2011; Salakhutdinov and Mnih, 2008), but we would like to approach the performance of the Bayesian gold standard using considerably less computation. We propose to do this by approximating an equally weighted geometric mean of the predictions of the individual models. The resulting model will share parameters.

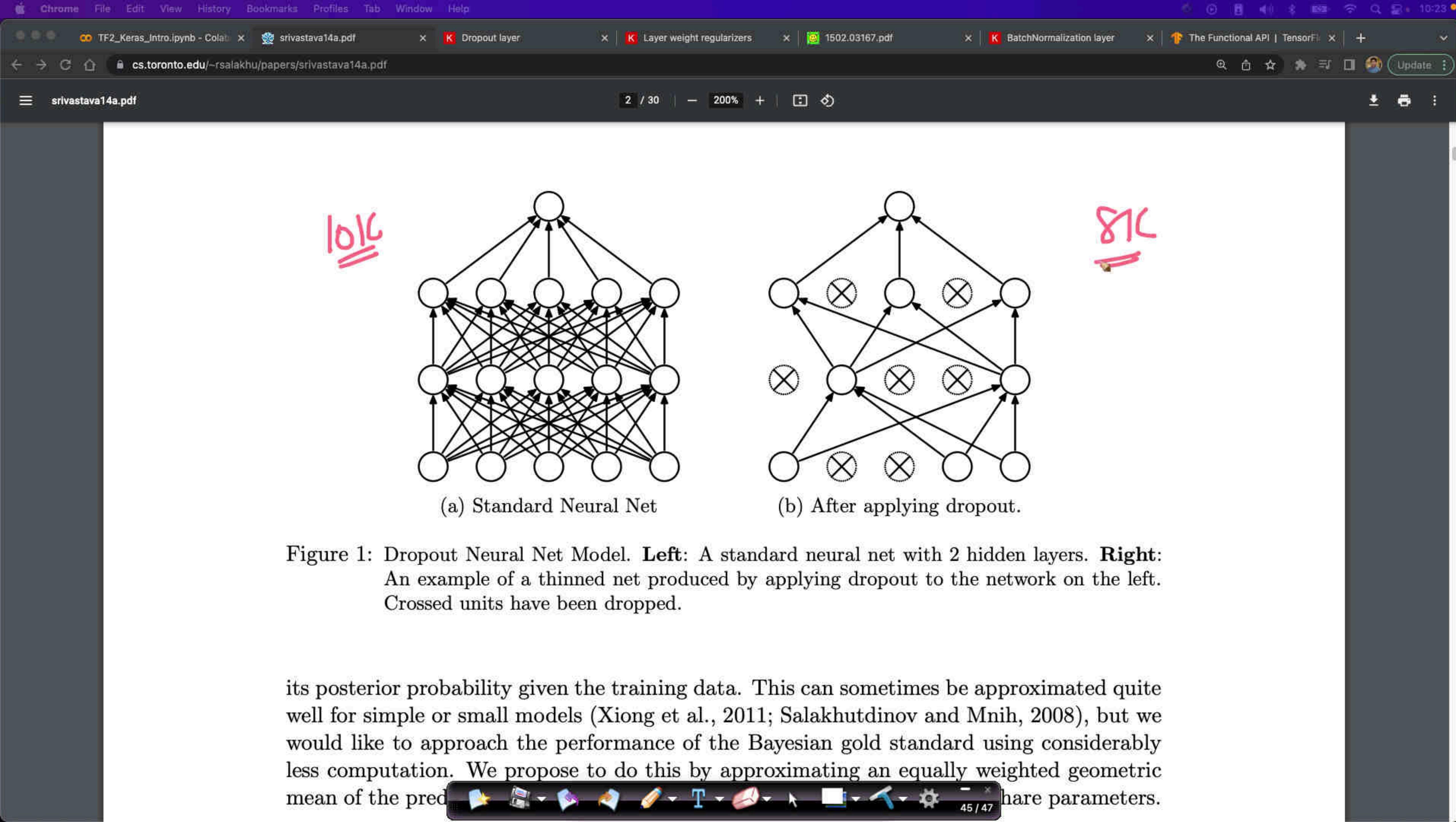


Figure 1: Dropout Neural Net Model. **Left:** A standard neural net with 2 hidden layers. **Right:** An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

its posterior probability given the training data. This can sometimes be approximated quite well for simple or small models (Xiong et al., 2011; Salakhutdinov and Mnih, 2008), but we would like to approach the performance of the Bayesian gold standard using considerably less computation. We propose to do this by approximating an equally weighted geometric mean of the predictions of all thinnings, share parameters.

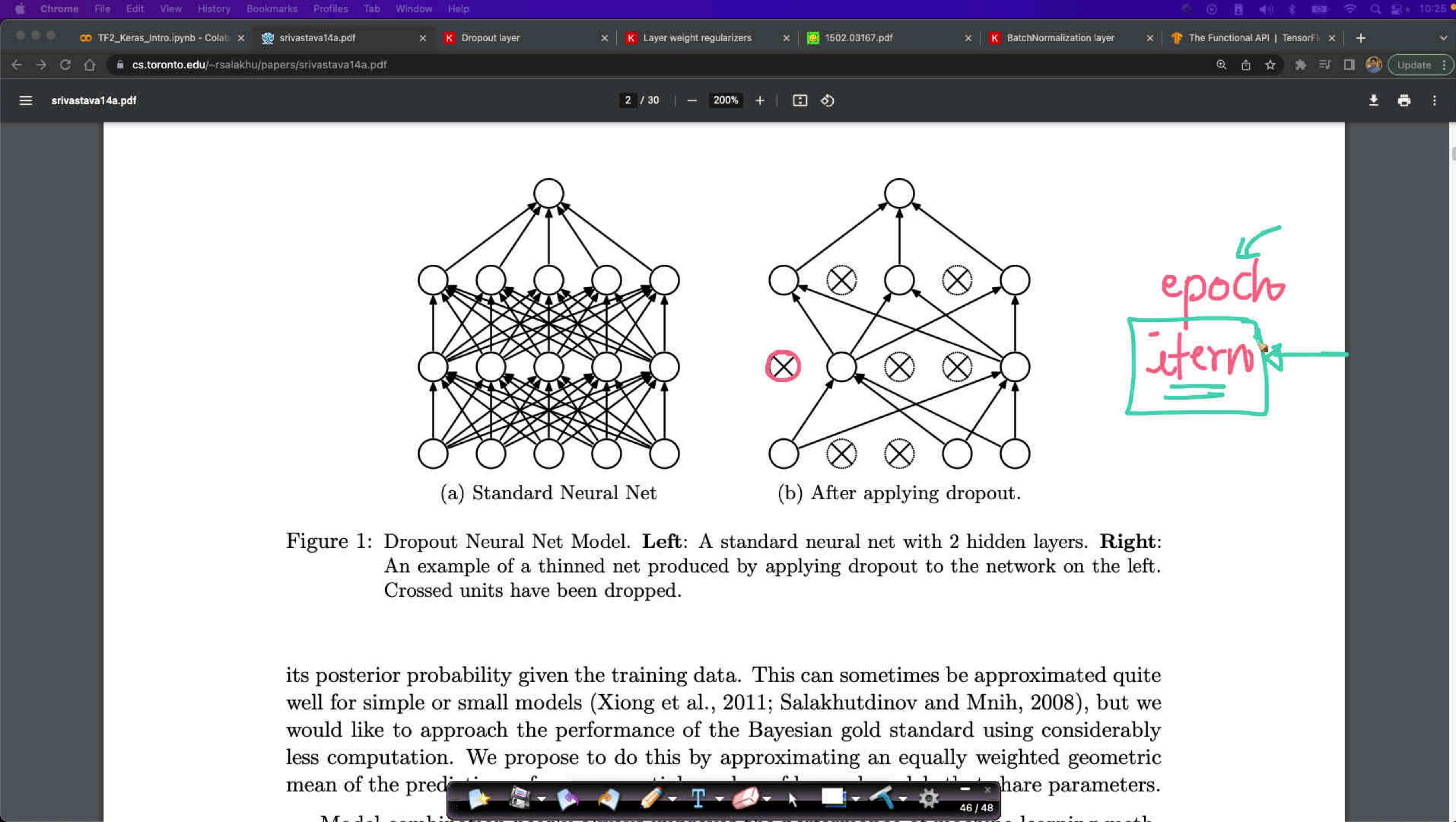


Figure 1: Dropout Neural Net Model. **Left:** A standard neural net with 2 hidden layers. **Right:** An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

its posterior probability given the training data. This can sometimes be approximated quite well for simple or small models (Xiong et al., 2011; Salakhutdinov and Mnih, 2008), but we would like to approach the performance of the Bayesian gold standard using considerably less computation. We propose to do this by approximating an equally weighted geometric mean of the predictions of a family of models that share parameters.

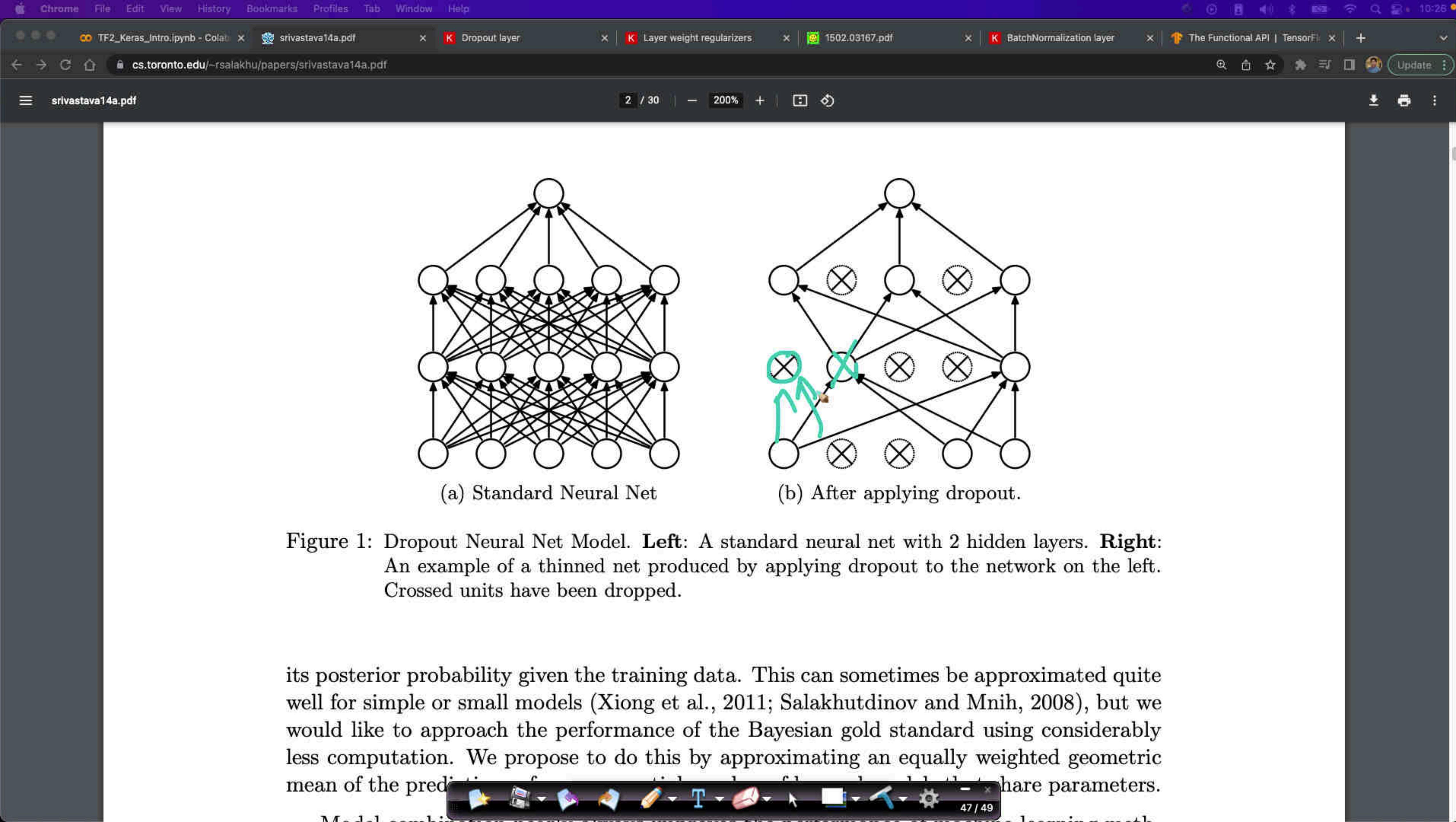


Figure 1: Dropout Neural Net Model. **Left:** A standard neural net with 2 hidden layers. **Right:** An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

its posterior probability given the training data. This can sometimes be approximated quite well for simple or small models (Xiong et al., 2011; Salakhutdinov and Mnih, 2008), but we would like to approach the performance of the Bayesian gold standard using considerably less computation. We propose to do this by approximating an equally weighted geometric mean of the predictions of a family of models that share parameters.

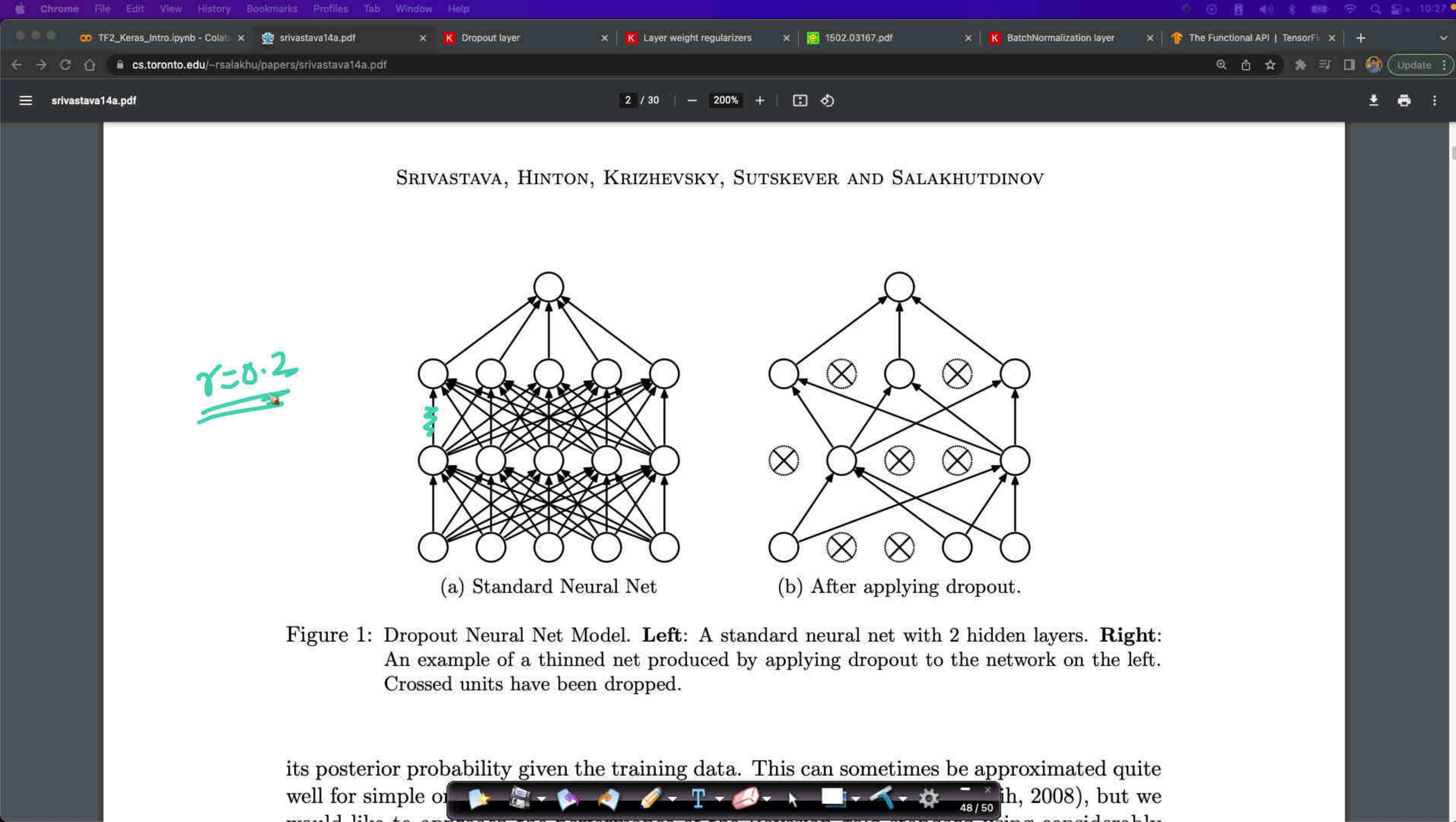


Figure 1: Dropout Neural Net Model. **Left:** A standard neural net with 2 hidden layers. **Right:** An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

its posterior probability given the training data. This can sometimes be approximated quite well for simple or linear models (Ghahramani and Ghahramani, 2008), but we would like to approximate the posterior distribution more accurately and efficiently.

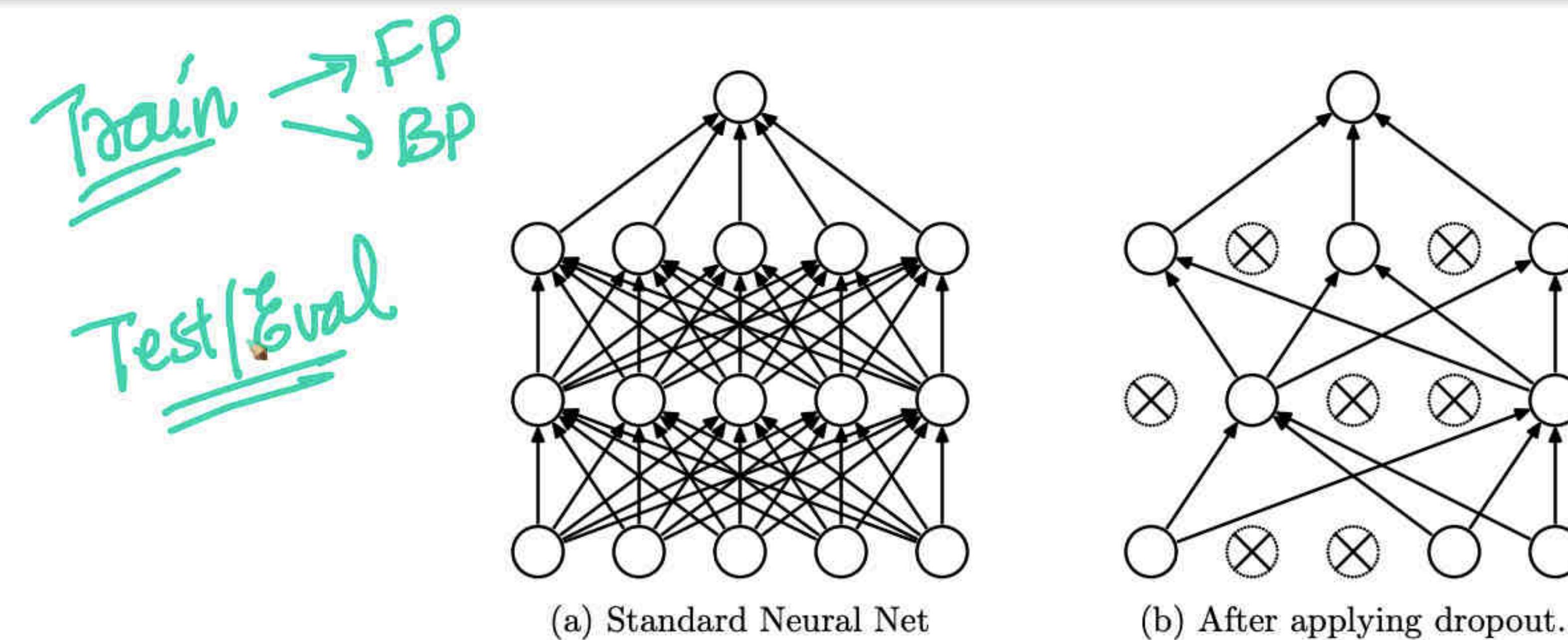


Figure 1: Dropout Neural Net Model. **Left:** A standard neural net with 2 hidden layers. **Right:** An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

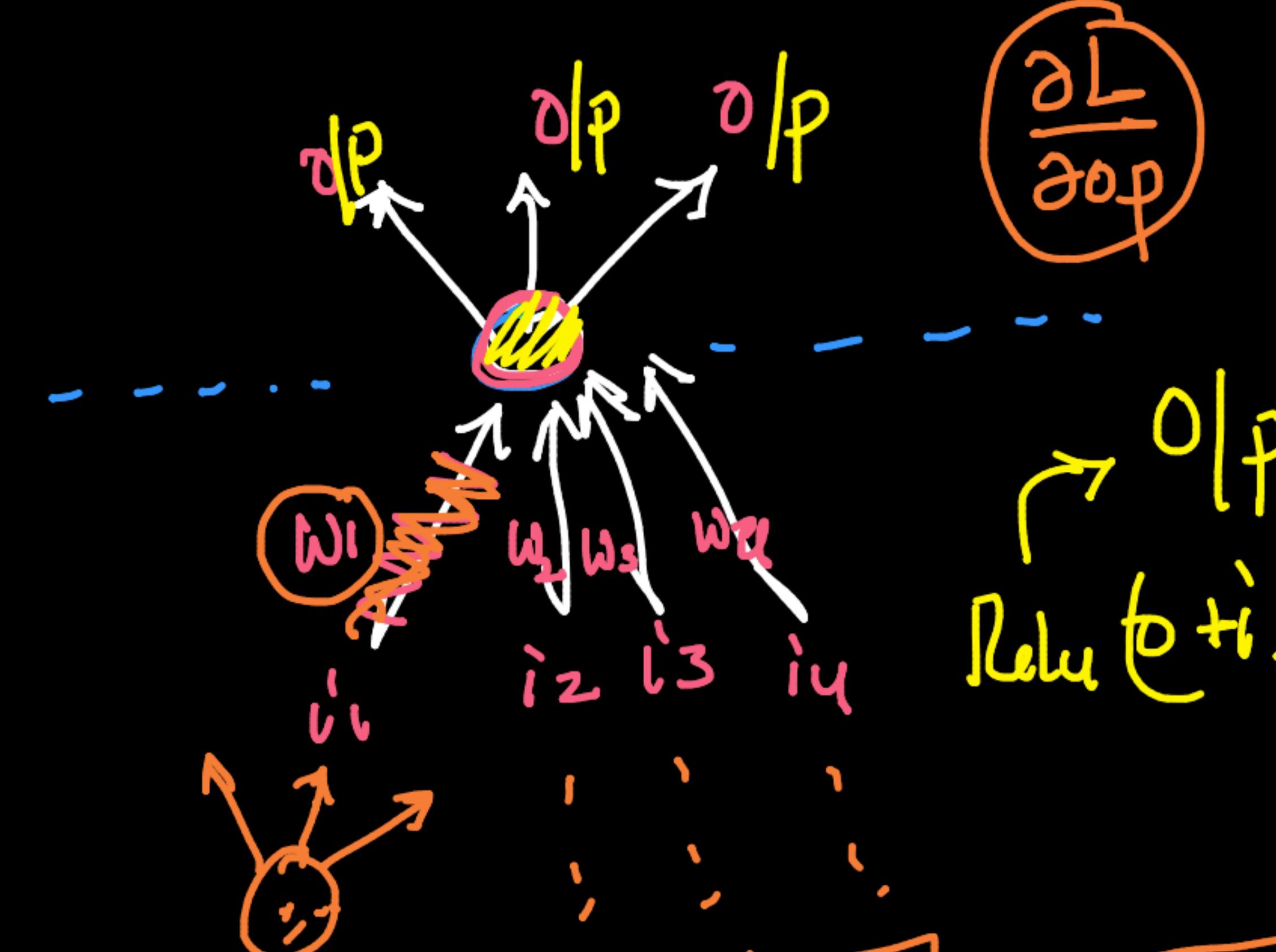
its posterior probability given the training data. This can sometimes be approximated quite well for simple or small models (Xiong et al., 2011; Salakhutdinov and Mnih, 2008), but we would like to approach the performance of the Bayesian gold standard using considerably less computation. We propose to do this by approximating an equally weighted geometric mean of the predictions of an exponential number of learned models that share parameters.



✓ { FP:
 =

✓ { BP:
 =

w_1 is not updated;

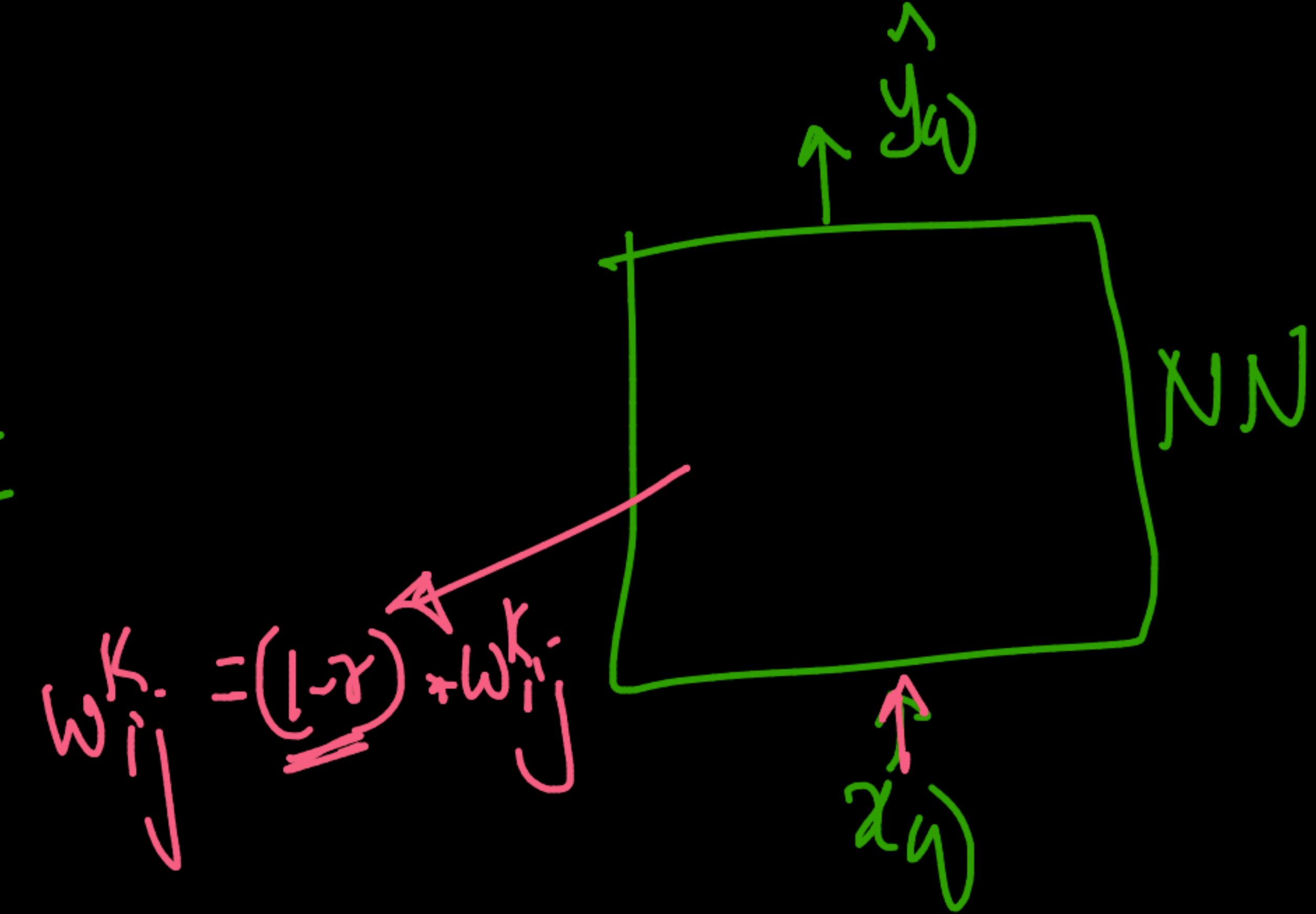


$$\text{ReLU}(i_1 w_1 + i_2 w_2 + i_3 w_3 + i_4 w_4)$$

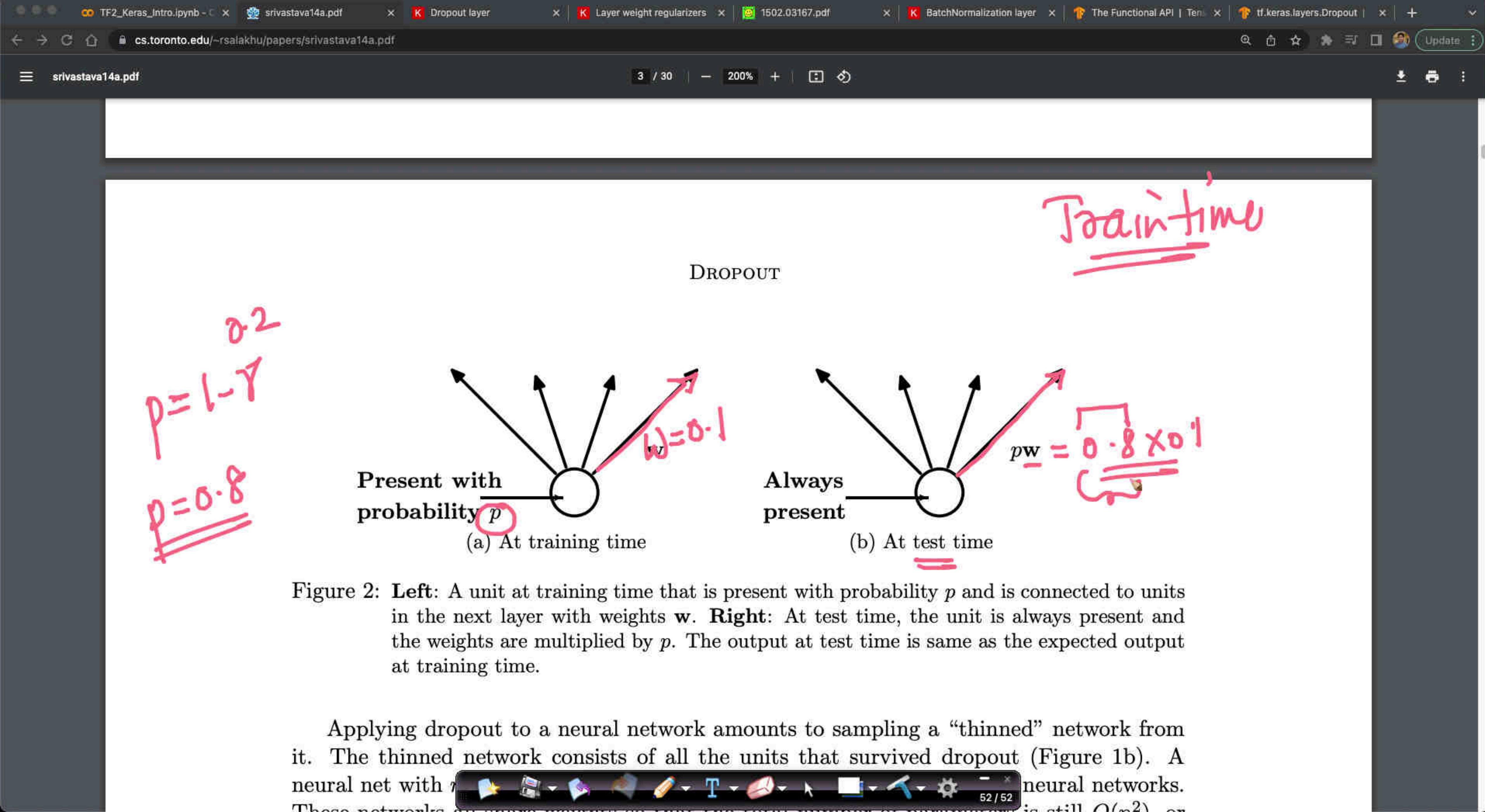
$\frac{\partial L}{\partial o_p}$... are not back propagated
on w_1 edge

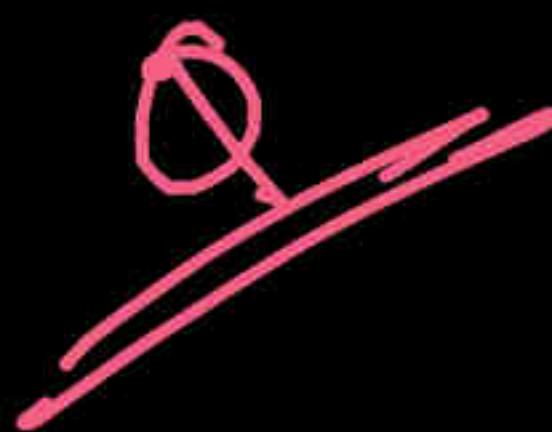
Dropout
~~edge~~ (edge-level)

Test:
 $\gamma = 0.2$

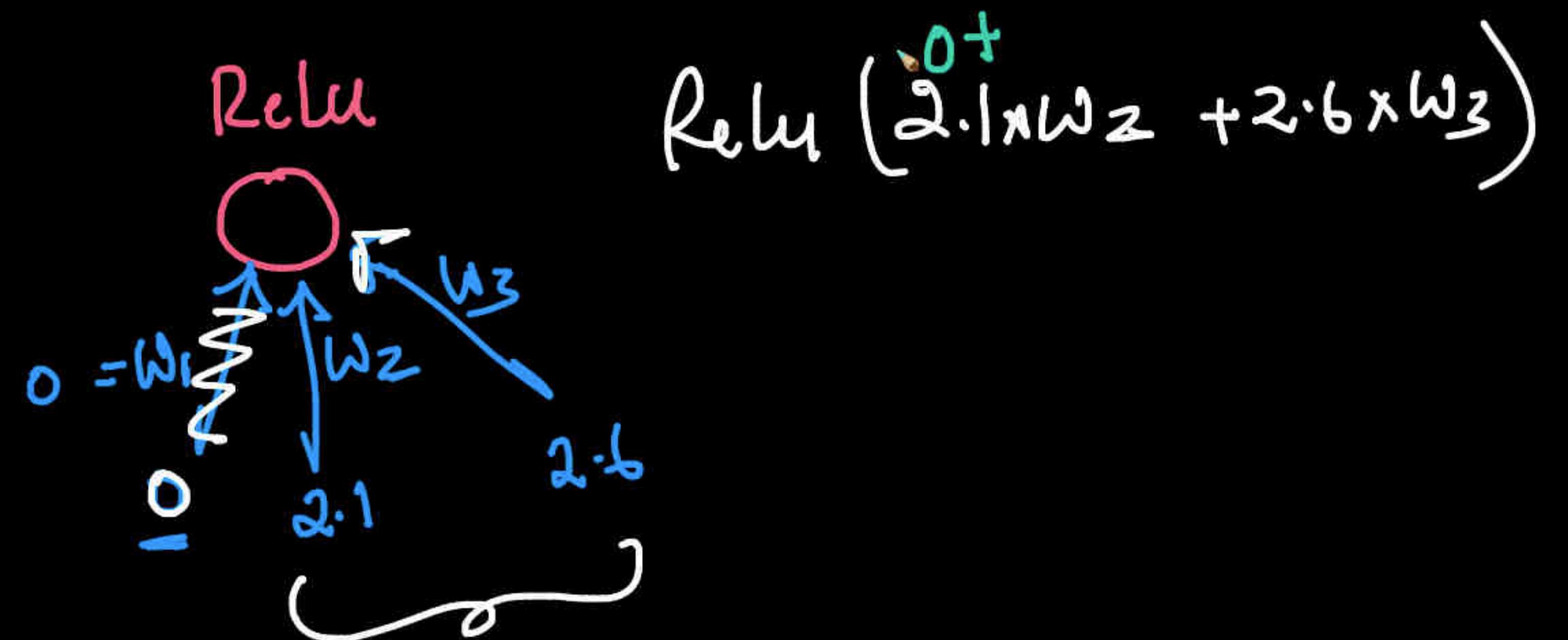


do NOT skip
any edge





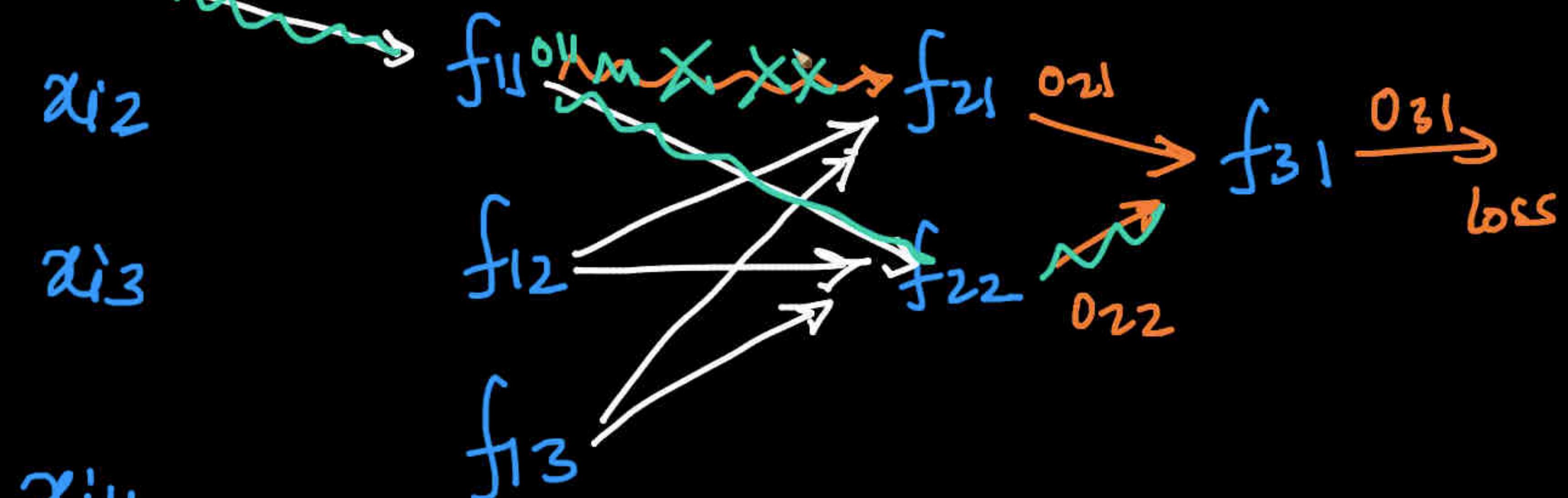
$$\gamma = 0.1 - 0.6$$





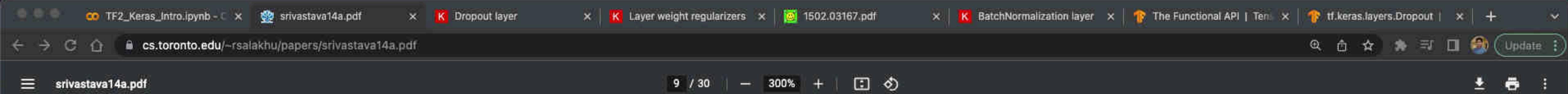
x_{i1} w_1
 x_{i2}
 x_{i3}
 x_{i4}

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial o_{31}} \cdot \frac{\partial o_{31}}{\partial o_{22}} \cdot \frac{\partial o_{22}}{\partial o_{11}} \cdot \frac{\partial o_{11}}{\partial w_1}$$





$$\gamma = 0.5$$



In order to test the robustness of dropout, classification experiments were done with networks of many different architectures keeping all hyperparameters, including p , fixed. Figure 4 shows the test error rates obtained for these different architectures as training progresses. The same architectures trained with and without dropout have drastically different test errors as seen as by the two separate clusters of trajectories. Dropout gives a huge improvement across all architectures, without using hyperparameters that were tuned specifically for each architecture.

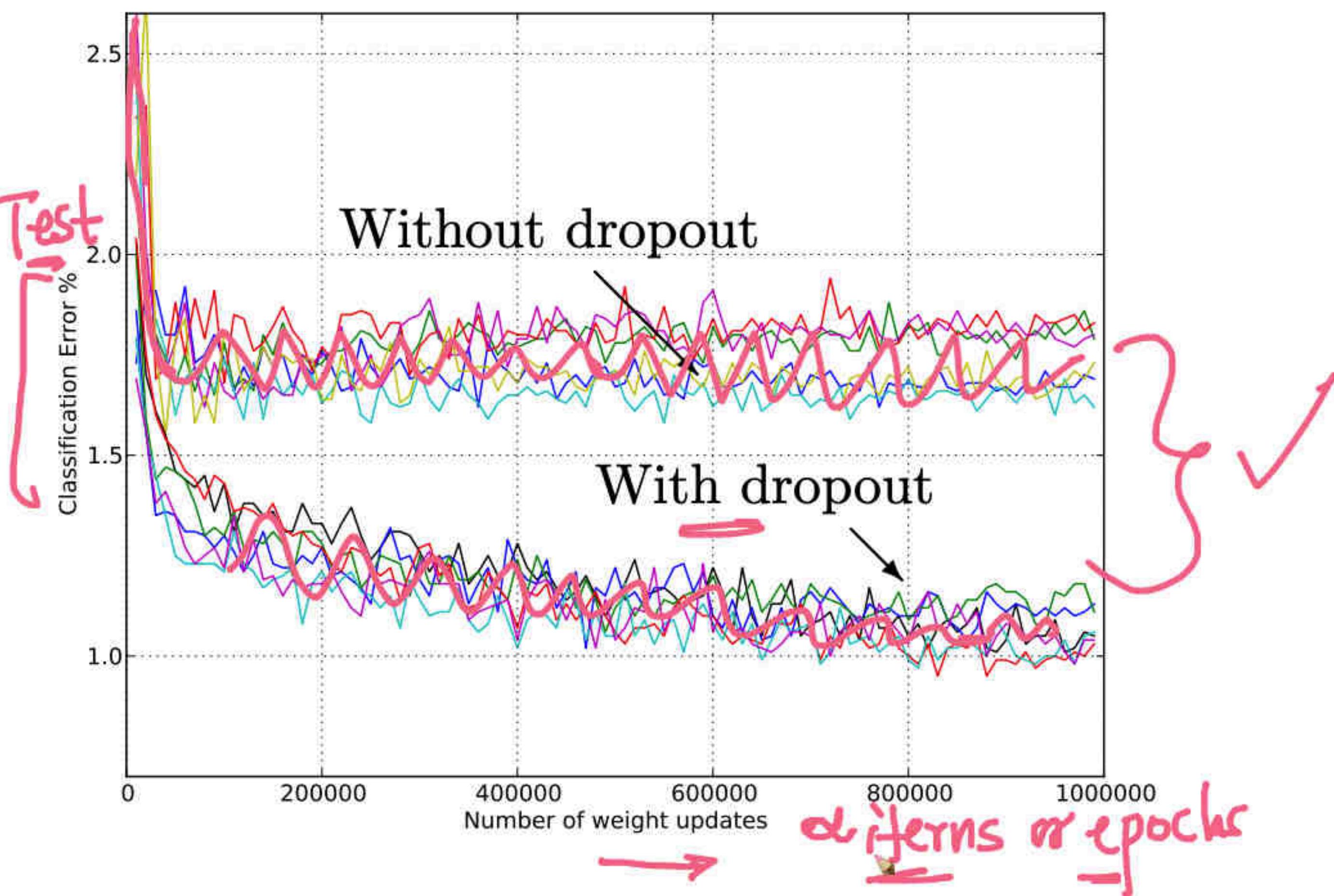


Figure 4: Test error for different architectures

1.2 STREET VIEW HOUSE NUMBER RECOGNITION WITH DROPOUT

∞ TF2_Keras_Intro.ipynb - C × srivastava14a.pdf × K Dropout layer × K Layer weight regularizers × 1502.03167.pdf × K BatchNormalization layer × The Functional API | Ten. × tf.keras.layers.Dropout × +

colab.research.google.com/drive/1tyv-ViqseMy1rTwGc6aLbnXk-5MC13g#scrollTo=VxWmm9wV0XHG

+ Code + Text Reconnect

Search {x}

▼ Dropout: Regularization through randomization

Model with Dropout

```
x = layers.Dense(hidden_units, activation="relu")(inputs)
x = Dropout(0.3)(x)
x = layers.Dense(hidden_units, activation="relu")(x)
x = Dropout(0.3)(x)
outputs = layers.Dense(num_labels, activation="softmax")(x)

model = keras.Model(inputs=inputs, outputs=outputs, name="simple_model")
model.summary()
```

Model: "simple_model"

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[None, 784]	0
dense_6 (Dense)	(None, 256)	200960
dropout (Dropout)	(None, 256)	0
dense_7 (Dense)	(None, 256)	65792

Reconnect

Update

Up Down Reload Settings Copy Paste Delete More

57 / 57

+ Code + Text

Dropout: Regularization through randomization



```
# Model with Dropout
x = layers.Dense(hidden_units, activation="relu")(inputs)
x = Dropout(0.3)(x)
x = layers.Dense(hidden_units, activation="relu")(x)
x = Dropout(0.3)(x)
outputs = layers.Dense(num_labels, activation="softmax")(x)

model = keras.Model(inputs=inputs, outputs=outputs, name="simple_model")
model.summary()
```

Model: "simple_model"

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[None, 784]	0
dense_6 (Dense)	(None, 256)	200960
dropout (Dropout)	(None, 256)	0
dense_7 (Dense)	(None, 256)	65792

+ Code + Text

Epoch 23/30
469/469 [=====] - 1s 3ms/step - loss: 0.0253 - accuracy: 0.9915
Epoch 24/30
469/469 [=====] - 2s 3ms/step - loss: 0.0272 - accuracy: 0.9913
Epoch 25/30
469/469 [=====] - 2s 3ms/step - loss: 0.0259 - accuracy: 0.9911
Epoch 26/30
469/469 [=====] - 2s 3ms/step - loss: 0.0240 - accuracy: 0.9917
Epoch 27/30
469/469 [=====] - 2s 4ms/step - loss: 0.0224 - accuracy: 0.9926
Epoch 28/30
469/469 [=====] - 2s 4ms/step - loss: 0.0224 - accuracy: 0.9927
Epoch 29/30
469/469 [=====] - 2s 4ms/step - loss: 0.0227 - accuracy: 0.9924
Epoch 30/30
469/469 [=====] - 2s 4ms/step - loss: 0.0219 - accuracy: 0.9924
CPU times: user 54 s, sys: 6.47 s, total: 1min 1s
Wall time: 1min 22s
<keras.callbacks.History at 0x7fec60036c50>

```
[ ] # test dataset performance
loss, acc = model.evaluate(x_test,
                           y_test,
                           batch_size=batch_size,
                           verbose=0)
print("\nTest accuracy: %.1f%%" % (100.0 * acc))
print(loss)
```

TF2_Keras_Intro.ipynb - C srivastava14a.pdf Dropout layer Layer weight regularizers 1502.03167.pdf BatchNormalization layer The Functional API | Ten. tf.keras.layers.Dropout +

colab.research.google.com/drive/1tyv-ViqseMy1rTwGc6aLbnXk-5MC13g#scrollTo=G6SixhLe2KvN

+ Code + Text Reconnect

469/469 [=====] - 2s 4ms/step - loss: 0.0224 - accuracy: 0.9926
Epoch 28/30
469/469 [=====] - 2s 4ms/step - loss: 0.0224 - accuracy: 0.9927
Epoch 29/30
469/469 [=====] - 2s 4ms/step - loss: 0.0227 - accuracy: 0.9924
Epoch 30/30
469/469 [=====] - 2s 4ms/step - loss: 0.0219 - accuracy: 0.9924
CPU times: user 54 s, sys: 6.47 s, total: 1min
Wall time: 1min 22s
<keras.callbacks.History at 0x7fec60036c50>

```
[ ] # test dataset performance
loss, acc = model.evaluate(x_test,
                            y_test,
                            batch_size=batch_size,
                            verbose=0)
print("\nTest accuracy: %.1f%%" % (100.0 * acc))
print(loss)

# Overfit or Underfit?
# Can we add more layers?
```

Test accuracy: 98.4%

0.07712692767381668

~~Q~~ ~~$\gamma = 0.2$~~

x_{i1} ~~w_{i1}^1~~ f_{i1}

 x_{i2} f_{i2} x_{i3} f_{i3} x_{iu} w_{ux3}^1

w_{ii}^1 : init
to randomly
value

itero:

 $w_{ii}^1 : 0.1 \text{ (let)}$

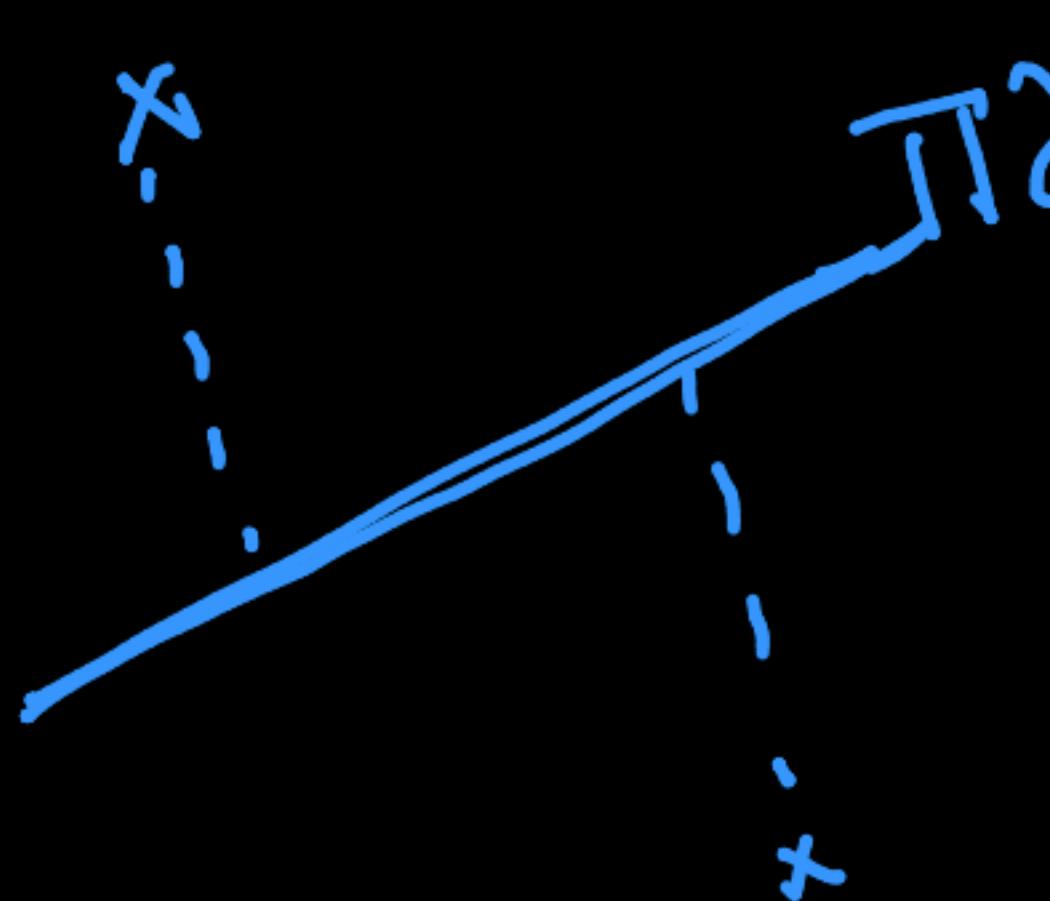
iter 1:

 $\left\{ \begin{array}{l} w_{ii}^1 = 0.1 \\ (\text{no update}) \end{array} \right.$

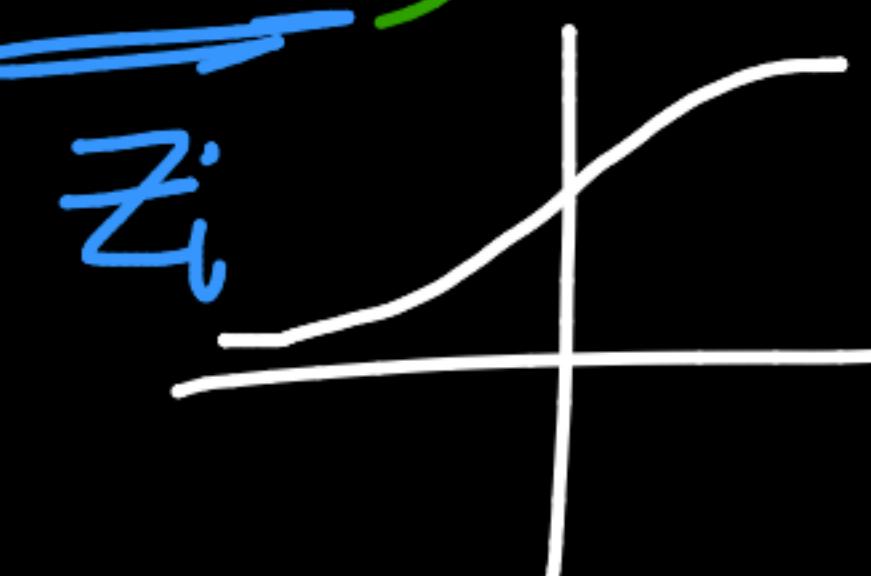
iter 2:

 $w_{ii}^1 = 0.1 - \eta \sum \frac{\partial L}{\partial w_i}$

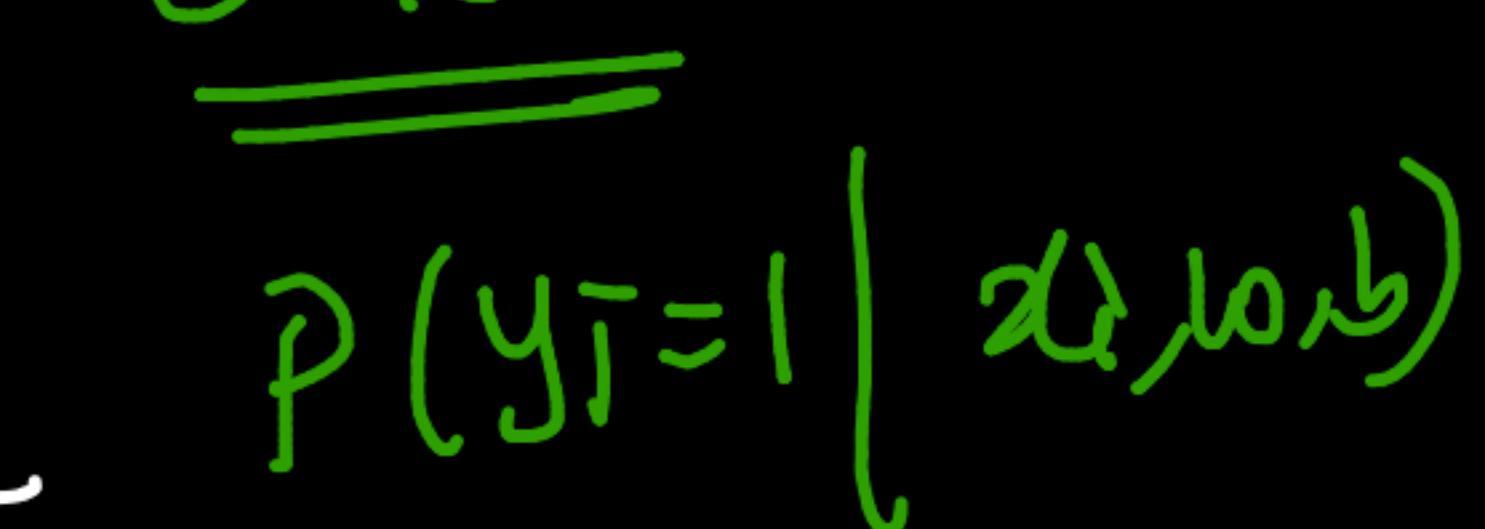
 Log-reg: - decision boundary: - linear

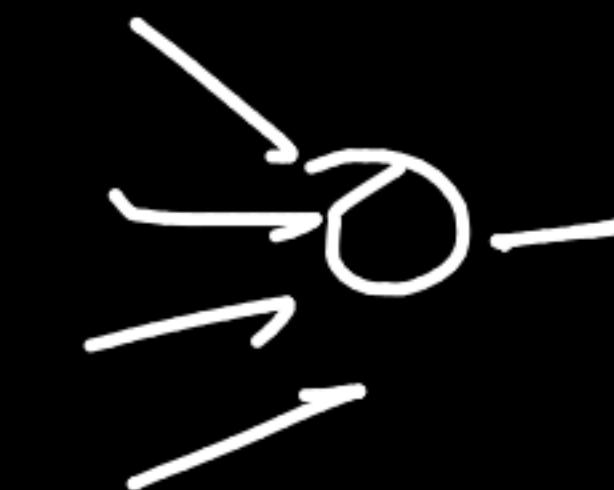


$$\text{if } (\underline{\omega}^T \underline{x}_i + b) \rightarrow z_i$$

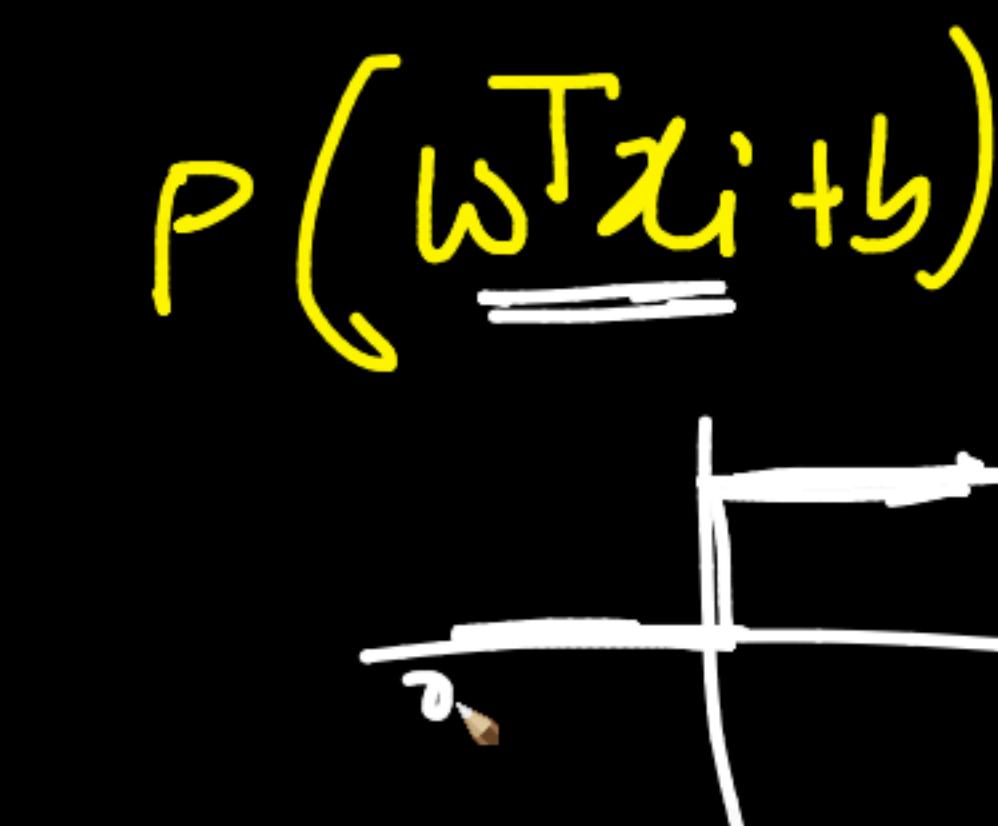


$$\Pr(y_i=1 | \underline{\omega}, \underline{x}_i, b)$$

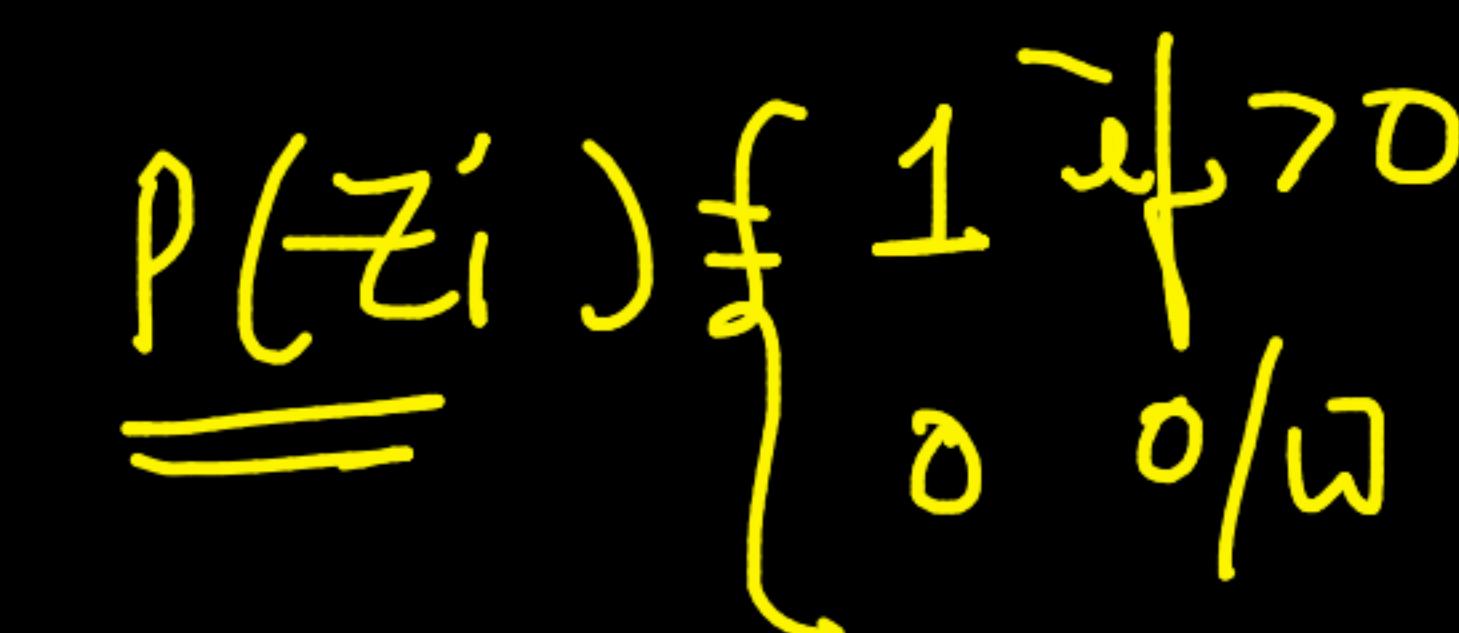


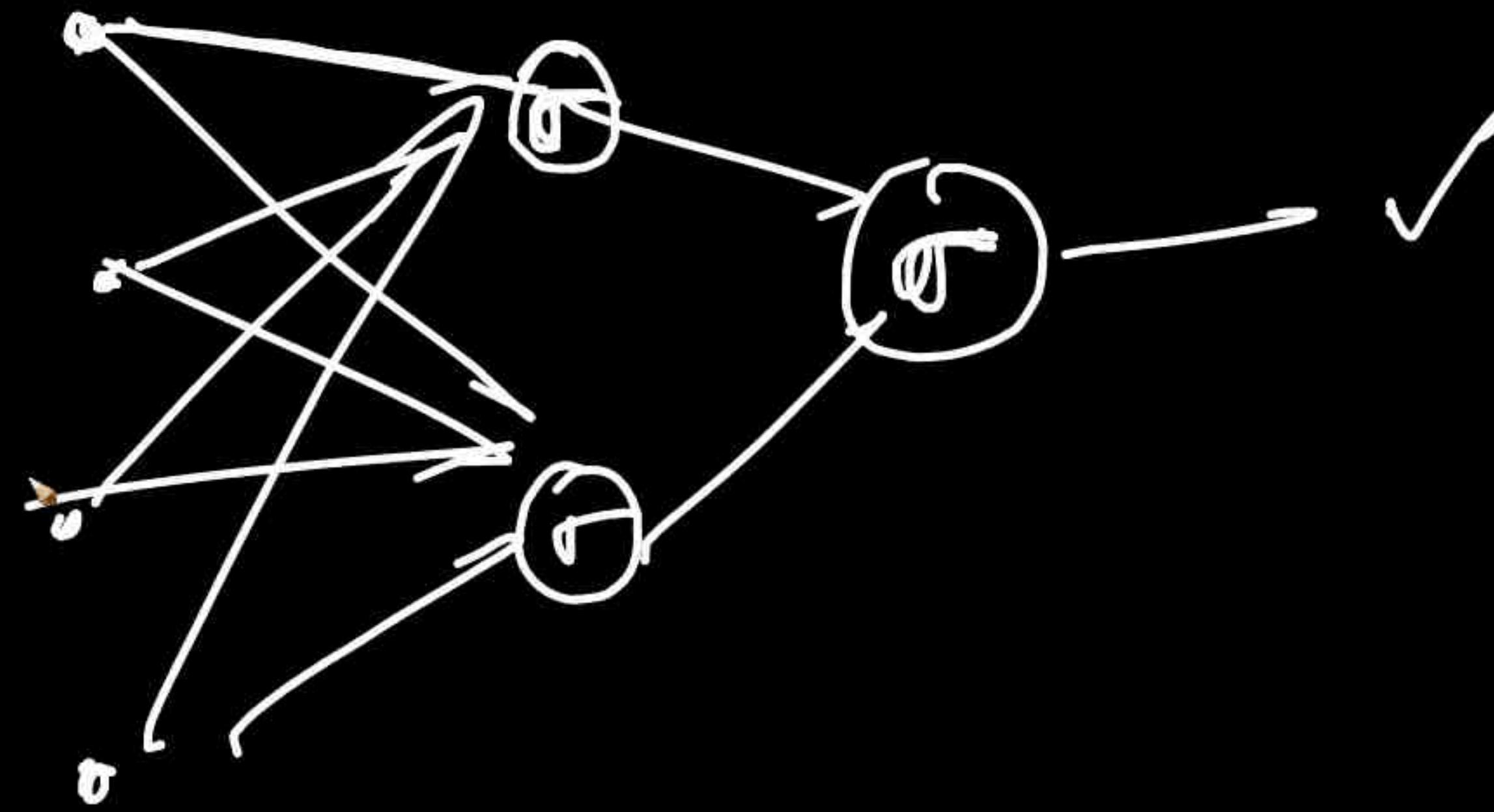


 Perception:-



$$\Pr(z_i > 0 | \underline{\omega}, \underline{x}_i, b)$$







$$\text{ReLU}(z) = \max(0, z)$$

3x + 5



$$\frac{\partial \text{ReLU}(3x+5)}{\partial x} \stackrel{\text{?}}{=} \frac{\partial \text{ReLU}(z)}{\partial z} \cdot \frac{\partial (3x+5)}{\partial x}$$

$\frac{\partial \text{ReLU}}{\partial z}$

$$= +1 \cdot 3$$

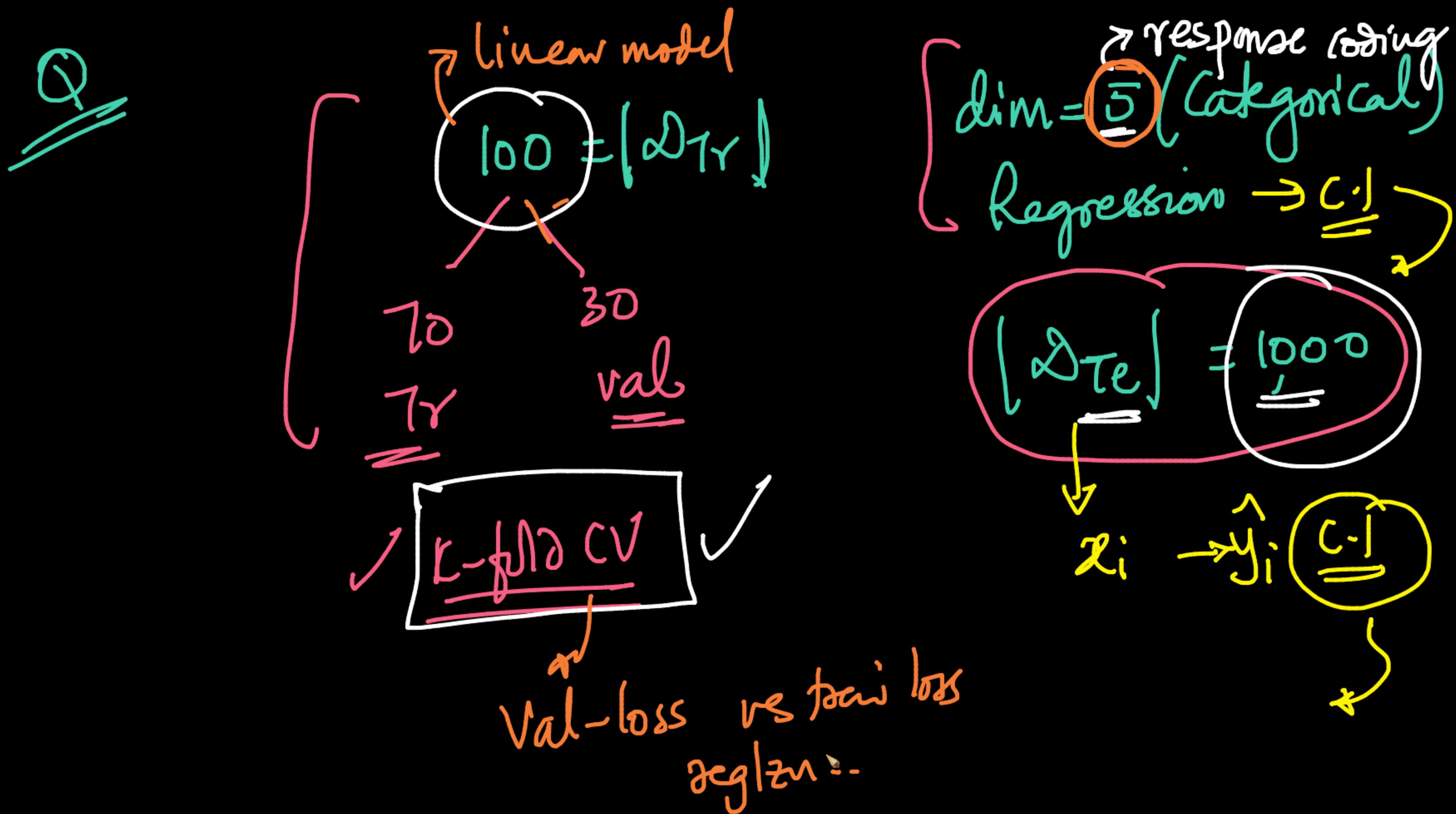
+3 ✓



OHE $\xrightarrow{c_1 \dots c_{100}}$ 100-class classfn \rightarrow regression

[Tree-based \rightarrow

{ Linear

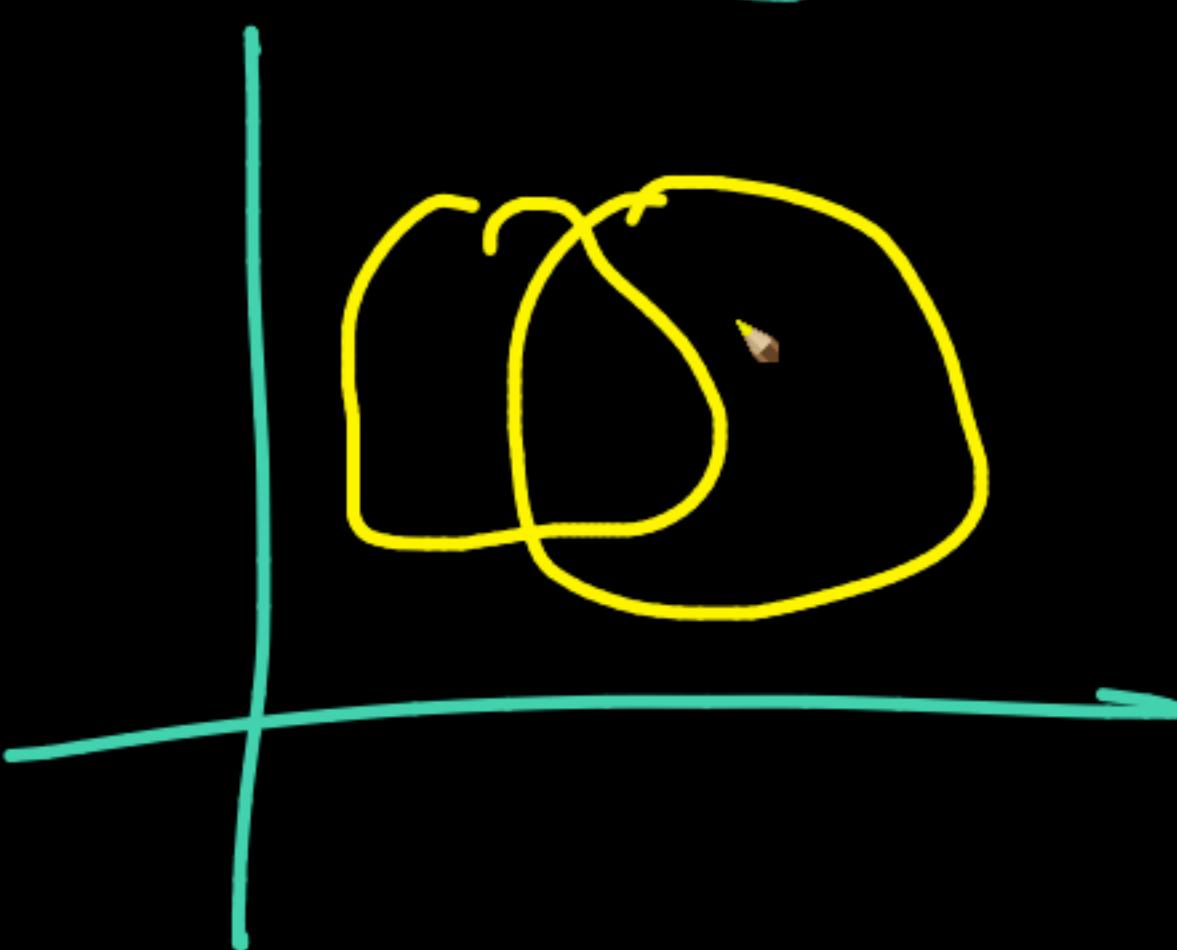




X^1 $n \times d$



2-classes
(50-50)

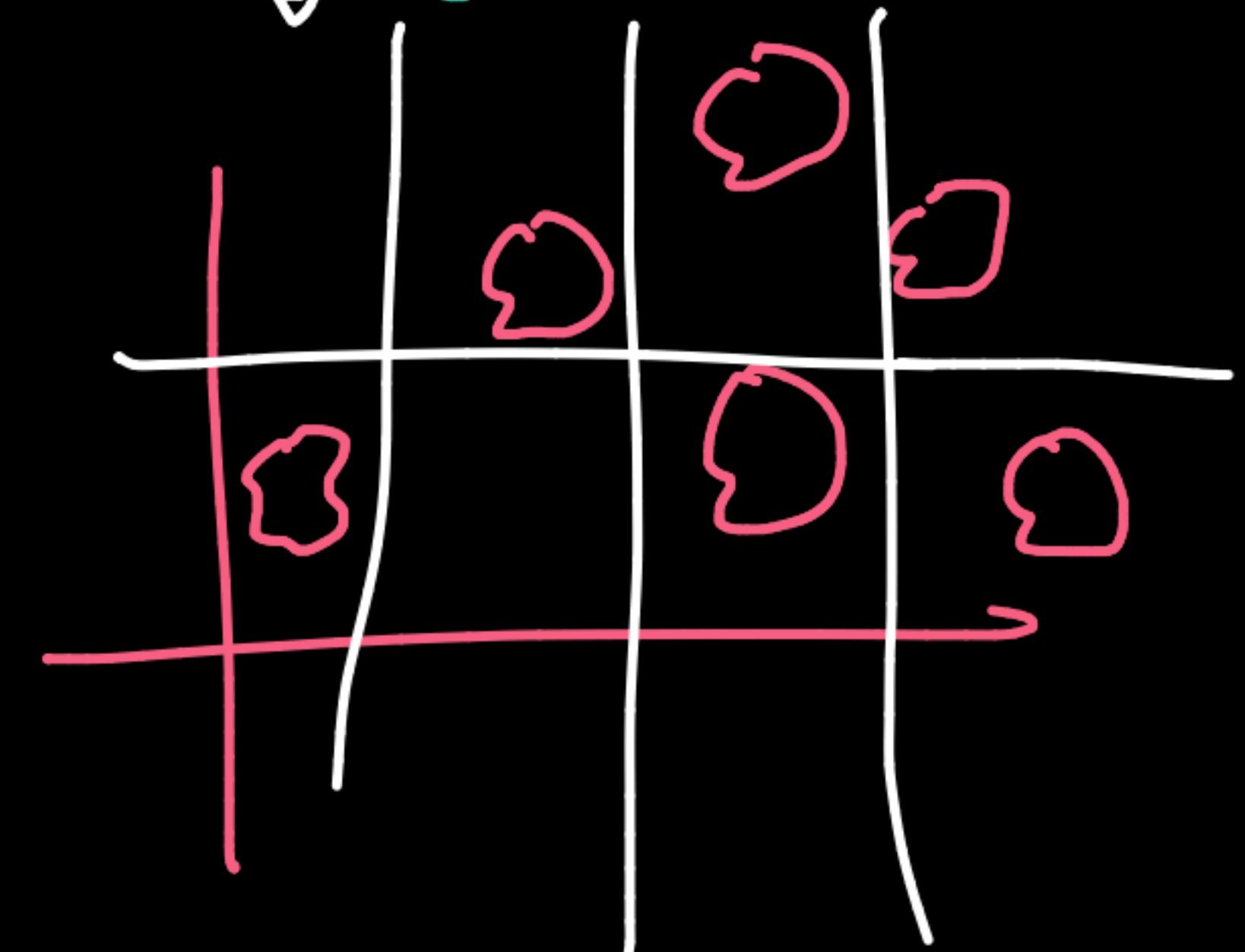


X^2 $n \times d$



20-classes

(5-5-5-5-5-5-5-5-5-5)



+ Code + Text

Reconnect



Test accuracy: 98.4%

0.07712692767381668

{x}

[]

▼ GPUs vs CPUs

► CPU

[] ↳ 4 cells hidden



► GPU

▶ ↳ 3 cells hidden

► Tensorboard and Computational Graph



68 / 68