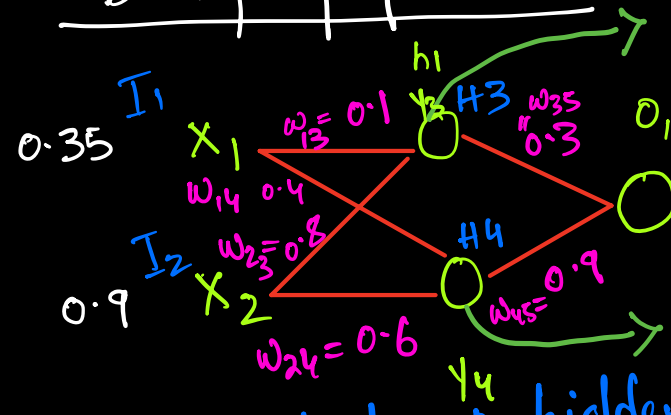
Today's Agenda

1) Backpropagation
2) Regularization
3) TensorboardL (Visualization of Val Traing Acc/Los)

Backpropagation



5 units = 2 input, 2 hidden neuron & 1 output

Assumptions

L.R = 1

0.V = 0.5

Forward Pass

$a_1 = (w_{13} * X_1) + (w_{23} * X_2) = (0.1 \times 0.35) + (0.8 \times 0.9)$

$= 0.0355 + 0.72$

$= 0.755$

$a_2 = (w_{14} * X_1) + (w_{24} * X_2) = (0.4 \times 0.35) + (0.6 \times 0.9)$

$= 0.68$

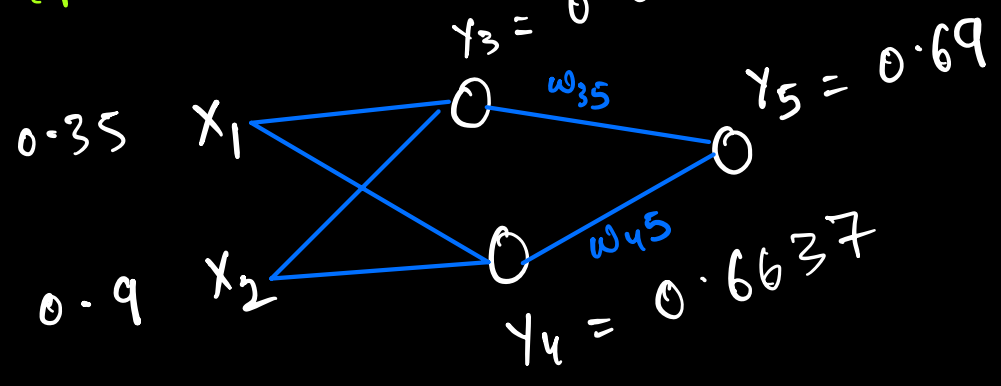
$Y_3 = \text{Sigmoid}(a_1) = 0.68$

$Y_4 = \text{Sigmoid}(a_2) = 0.6637$

$$a_3 = (w_{35} \times y_3) + (w_{45} \times y_4)$$
$$= (0.3 \times 0.68) + (0.9 \times 0.6637)$$
$$= 0.801$$
$$y_5 = \text{Sigmoid}(a_3) = 0.69$$

$$\text{Loss / Error} = y_{Act} - y_{pred} = 0.5 - 0.69$$
$$= -0.19$$

Weight Updation



target = actual = original = 0.5

Formula

Weight Change

$$\Delta w_{ij} = (\eta)(\delta_j)(o_i)$$

- η → LR
- o_i → output
- δ_j → Error Term

hidden unit output unit

$\eta = \text{L.R} = \underline{1}$

$\delta_j = o_j(1-o_j)(t_j - o_j)$ → output Neuron

$\delta_j = o_j(1-o_j)\sum_k w_{kj}$ → hidden Neuron

For output

$$\delta_5 = y(1-y)(y_{target} - y)$$

$$= 0.69(1-0.69)(0.5-0.69)$$

$$= -0.0406$$

For hidden

$$\delta_3 = y(1-y)\, w_{35} * \delta_5$$

$$= 0.68(1-0.68) * (0.3 \times -0.0406)$$

$$= -0.00265$$

$$\delta_4 = y(1-y)\, w_{45} \times \delta_5$$

$$= 0.6637(1-0.6637)\ 0.9 \times -0.0406)$$

$$= -0.0082$$

$\delta_5 = -0.0406$

$\delta_3 = -0.00265$

$\delta_4 = -0.0082$

Weight change

$\Delta w_{ij} = n\, \delta_j\, O_i$

- n → L.R
- δ_j → Error Term
- O_i → Output Y

Output Neuron (w.c)

$\Delta w_{45} = 1 \times \delta_5 \times (y_4)$ → Feed forward

$= 1 \times -0.0406 \times 0.6637 = -0.0269$

$w_{45}(new) = \Delta w_{45} + w_{old}$

$= -0.0269 + 0.9$

$= 0.8731$

$\left\{ w_{new} = w_{old} - \frac{\partial L}{\partial w_{old}} \right\}$

$w_{old} + \frac{\partial L}{\partial w_{old}}$

$\Delta_{35} = 1 \times \delta_5 \times O_3$

$= 1 \times -0.0406 \times 0.68$

$= -0.0276$

$Y_3 = O_3$

$Y_4 = O_4$

$Y_5 = O_5$

$w_{35}(new) = \Delta w_{35} + w_{35}\,old$

$= -0.0276 + 0.3$

$= .2724$

$w_{14} = \eta \delta_j O_i$

$= \eta \delta_4 X_1$

$= 1 \times -0.0082 \times 0.35$

$= -0.0028$

$w_{24} = \eta \delta_4 O_2$

$= 1 \times -0.0082 \times 0.9 = -.00738$

w_{14} new $= \Delta + w_{old} = -0.0028 + 0.4$
$= 0.397$

w_{24} new $= -.00738 + 0.6 = .5962$

$\Delta w_{13} = \eta \delta_3 o_1 = -.00265 \times 0.35$
$= -.0009$

$\Delta w_{23} = \eta \delta_3 o_2 = -.00265 \times 0.9$
$= -.0023$

w_{13} new $= \Delta w_{13} + w_{old} = -.0009 + 0.1$
$= 0.0991$

w_{23} new $= \Delta w_{23} + w_{old} = .0023 + 0.8$
$= 0.797$

$w_1 =$
$w_2 =$
$w_3 =$
$w_4 =$
$w_5 =$
$w_6 =$

→ 2nd Epoch *

1) Forward Propagation

1 Epoch = 10 iterati

Regularization

L1 Regularization (Lasso)

$$\text{Loss with L1} = \text{Loss} + \lambda \sum |w_i|$$

$\sum |w_i|$ → sum of absolute values of weights

λ → Regularization Term

λ Penalize the large weights

ML → 1) Feature Selection avoid overfitting

2) Sparse weights vectors

Ridge (L2 Regularization)

$$L_2 \text{ with Loss} = \text{Loss} + \lambda \sum (w_i^2)$$

L1 → weights zero

L2 → towards smaller value close to 0, but not zero

L2 → widely used in deep learning

```python
from tensorflow.keras import layers, models, regularizers

model = models.Sequential()
model.add(layers.Dense(64, kernel_regularizer=regularizers.l1(0.01), activation='rel
model.add(layers.Dense(64, kernel_regularizer=regularizers.l1(0.01), activation='rel
model.add(layers.Dense(num_classes, activation='softmax'))

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'
model.fit(x_train, y_train, epochs=10, batch_size=32)
```

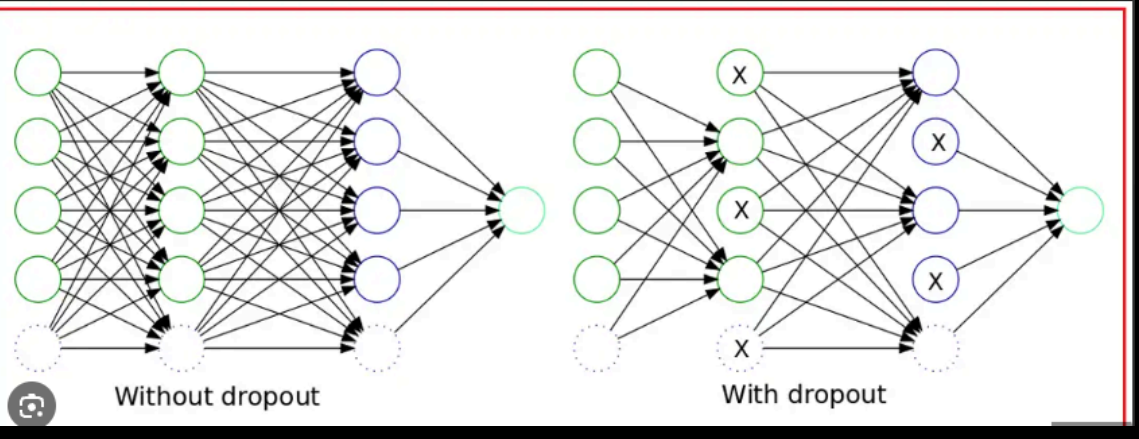
```python
from tensorflow.keras import layers, models, regularizers

model = models.Sequential()
model.add(layers.Dense(64, kernel_regularizer=regularizers.l2(0.01), activation='rel
model.add(layers.Dense(64, kernel_regularizer=regularizers.l2(0.01), activation='rel
model.add(layers.Dense(num_classes, activation='softmax'))

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'
model.fit(x_train, y_train, epochs=10, batch_size=32)
```

Dropout

Inactive / Deactive neurons in a layer

No Activation

Hide something from the network



0. (wx+b)

= 0

h1 → hidden layer

64 neurons

Dropout(0.1)

random selection of neurons = 6 neurons

in every iteration of epochs.

12 neurons = 20%

25% = 15 neurons

30% = 18 neurons

Hide & Show

by Generalized

Not more than 30%

Information loss from the network

```
# Define the model architecture with dropout regularization
model = Sequential([
    Dense(512, activation='relu', input_shape=(784,)),
    Dropout(0.2),  # Dropout with probability 0.2
    Dense(512, activation='relu'),
    Dropout(0.2),  # Dropout with probability 0.2
    Dense(10, activation='softmax')
])
```

512

General Thumb Rule

1) Don't use it in the start
2) Not more than 30%.
3) Use it when you are closing the network

(0.05)
(0.1)
(0.15)
(0.25) *

Penultimate layer
Ultimate layer (last layer)

Output X

input

Dense — penultimate — L
Dense — ultimate — L-1
Dense, softmax — Output layer — L-2
L-3

only on Training

1) Early Stopping

Stop the training at a certain point when there is no improvement.

Metrics → Acc / Loss

Epoch 1 . . . . . 0.21

Better

Epoch 15 . . . . . 0.04 ✓

0.0401

Epoch 25 . . . . . 0.04001

Epoch 35 . . . . . 0.03998

Epoch 99 . . . . . 0.3

35 - 99 epochs overfitting

Overfitting → noise

No Network change

Callbacks

Model Strategy

Training Strategy

python Copy code

```python
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.model_selection import train_test_split

# Generate some dummy data
X = np.random.rand(1000, 10)
y = np.random.randint(2, size=(1000,))

# Split data into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=

# Define a simple neural network model
model = Sequential()
model.add(Dense(64, activation='relu', input_dim=10))
model.add(Dense(1, activation='sigmoid'))

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Define early stopping criteria
early_stopping = EarlyStopping(monitor='val_loss', patience=5, verbose=1, restore_bes

# Train the model with early stopping
history = model.fit(X_train, y_train, epochs=100, batch_size=32, validation_data=(X_

# Evaluate the model
loss, accuracy = model.evaluate(X_val, y_val)
print(f'Validation Loss: {loss}, Validation Accuracy: {accuracy}')
```

```
ng criteria
yStopping(monitor='val_loss', patience=5, verbose=1, restore_best_weights=True)
```