Neural Network & deep learning

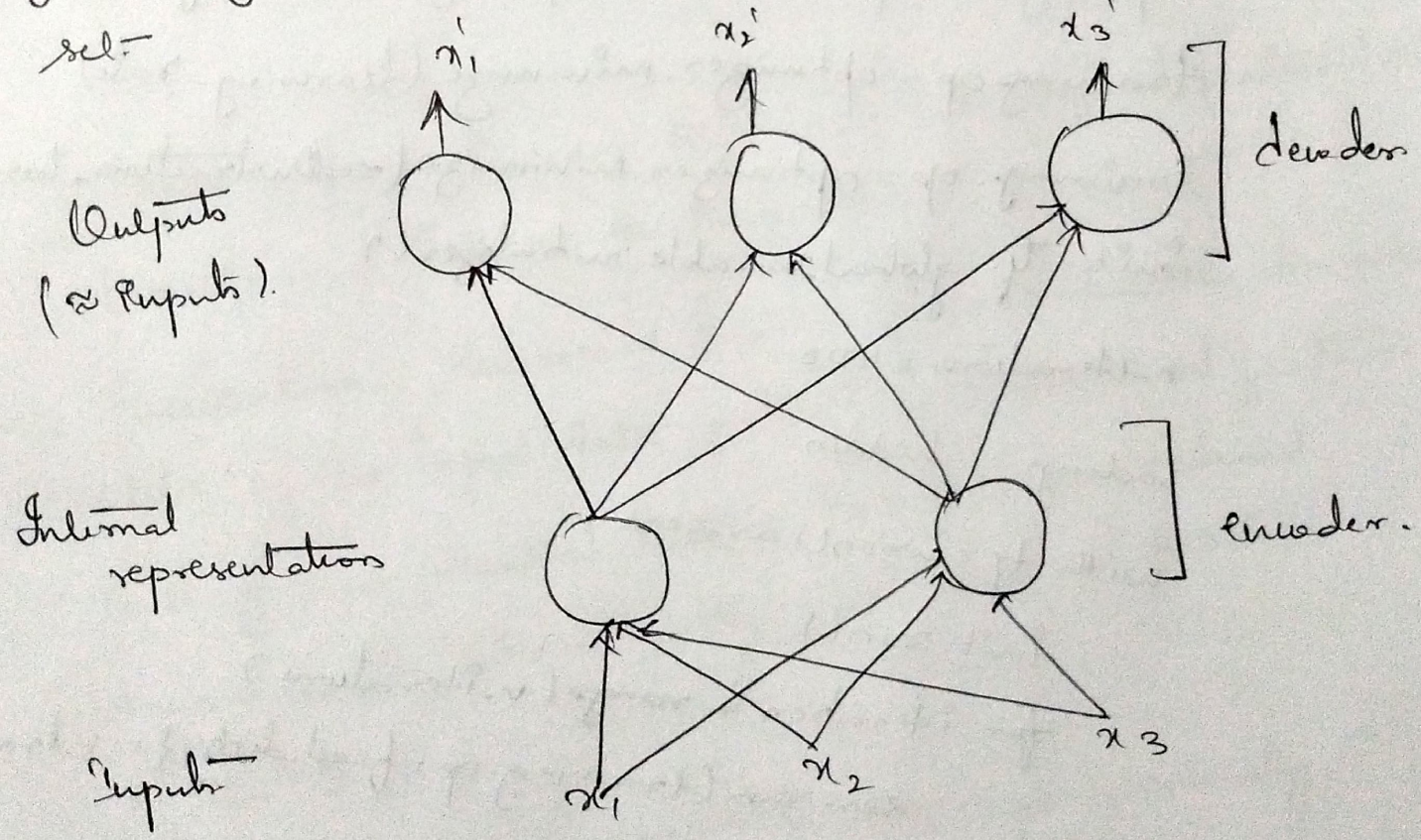
Assignment - 3

Pallavi K G
1JT20AI026.

1. what are autoencoders ? Explain the concept of efficient data representations.

Ans: Autoencoders are neural networks designed to learn efficient representations of input data, known as codings, without requiring labelled training data. these codings typically possess much lower dimensionality compared to the input data, making autoencoders effective tools for dimensionality reduction.

Autoencoders can function as generative modules capable of generating new data instances resembling those in the training set.

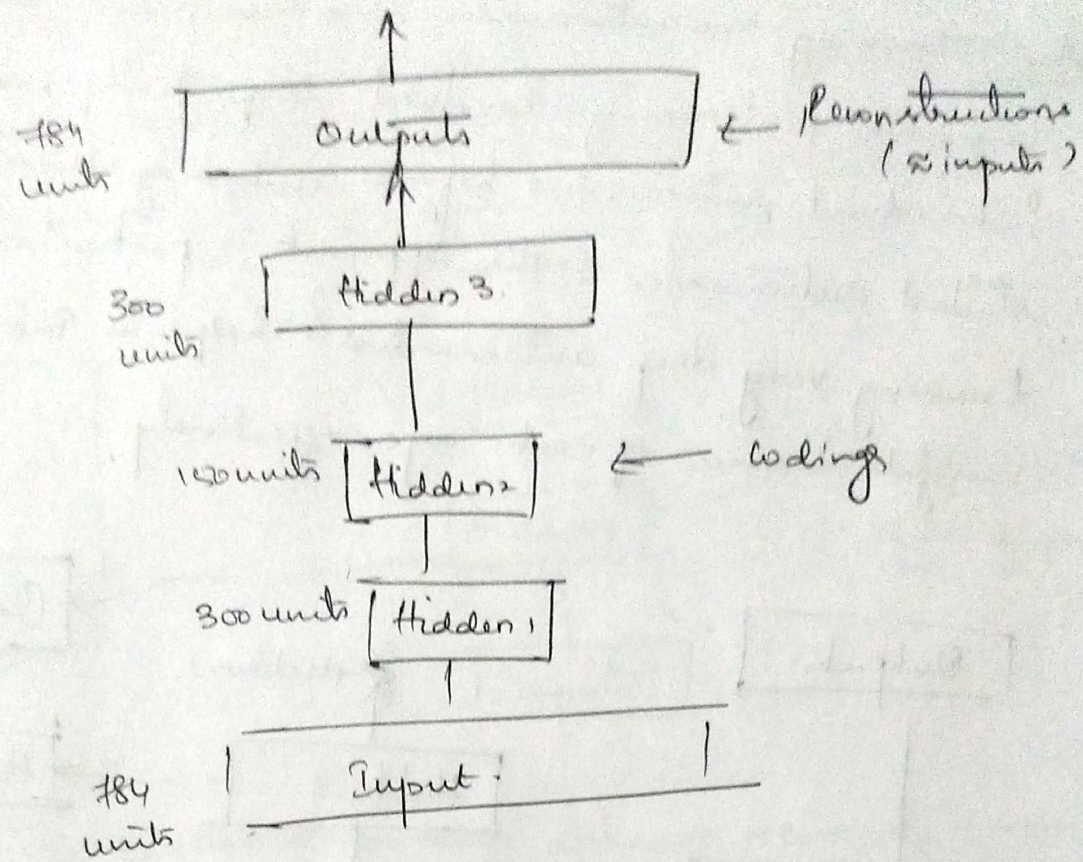


2. How to build a simple linear autoencoder to perform PCA on a 3D dataset, projecting it to 2D using python snippet.

Ans:

```
import tensorflow as tf.
from tensorflow.contrib.layers import fully_connected
n_inputs = 3
n_hidden = 2
n_outputs = n_inputs
x = tf.placeholder(tf.float32, shape = [none, n_inputs])
hidden = fully_connected (x, n_hidden, activation_fn = none)
Output = fully_connected (hidden, n_output, activation_fn = none)
reconstruction_loss = tf.reduce_mean = (tf.square_fn = none).
Optimizer = tf.train.AdamOptimizer (learning_rate)
training_op = optimizer.minimize (learning_rate)
training_op = optimizer.minimize (reconstruction_loss)
init = tf.global_variable_initializer()
n_iterations = 1000
codings = hidden
with tf.session() as sess:
    init.run()
    for iteration in range(n_iterations):
        sess.run (training_op, feed_dict = {x: x_train})
    codings_val = codings.eval (feed_dict = {x: x_test})
```

3. With neat a diagram explain stacked autoencoder and write python code to fix the weights of decoder layer.



→ Input layer: Recieves the input data, typical flattened or transformed into a suitable format.

→ Encoder layer: Successive hidden layers reduce the dimensionality of the input data extracting hierachical features

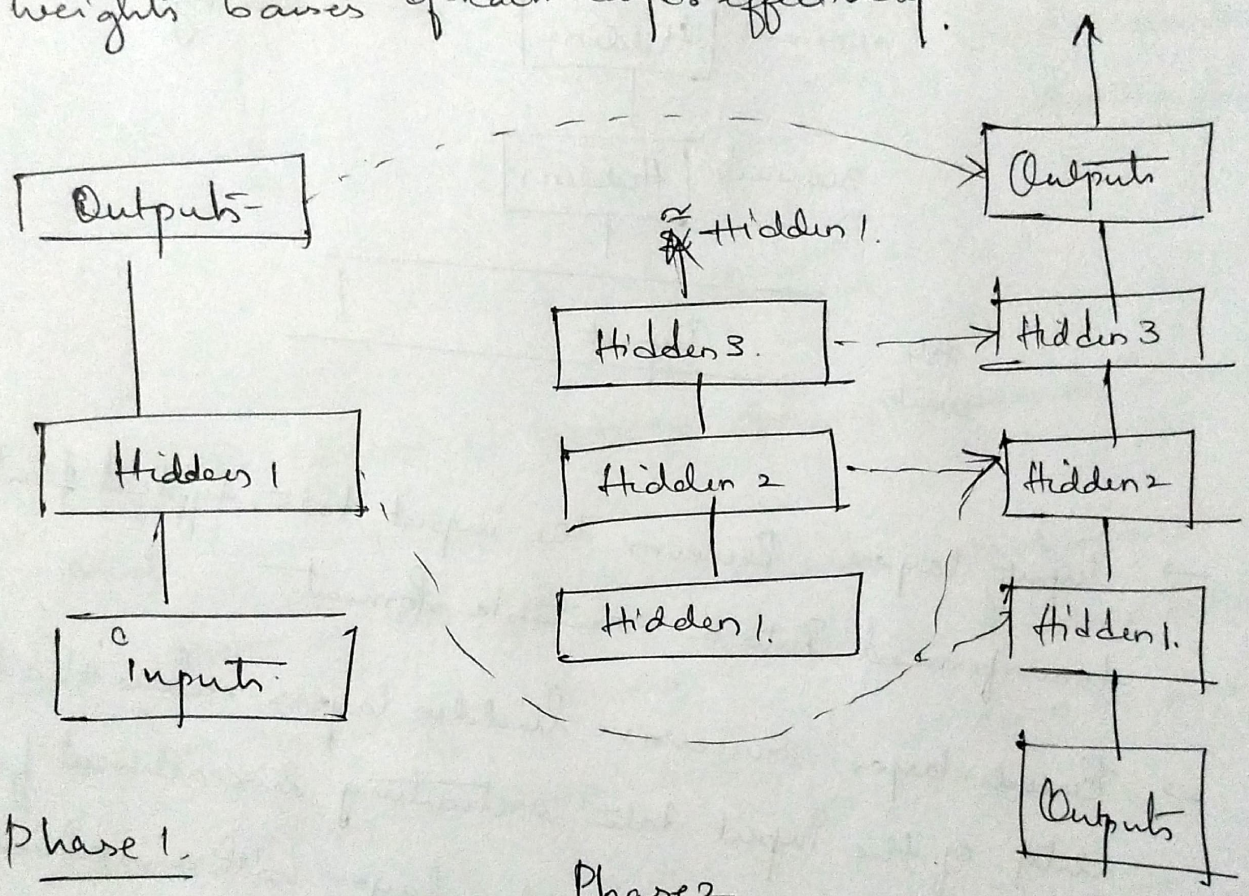
→ Coding layer: central hidden layer with a reduced dimensionality containing the most essential features of the i/p data.

→ Decoder layer: Successive hidden layers gradually reconstruction i/p data the coding layer.

→ Output layer: Reconstructs the i/p data at its original dimensionality.

4. With a neat diagram explain training one autoencoder at a time?

Ans: Training one autoencoder at a time, also known as layer-wise pretraining, involved sequentially training individual autoencoders before combining them into a stacked autoencoder. This approach is particularly useful to training very deep autoencoders & helps in initializing the weights & biases of each layer effectively.



Phase 1.

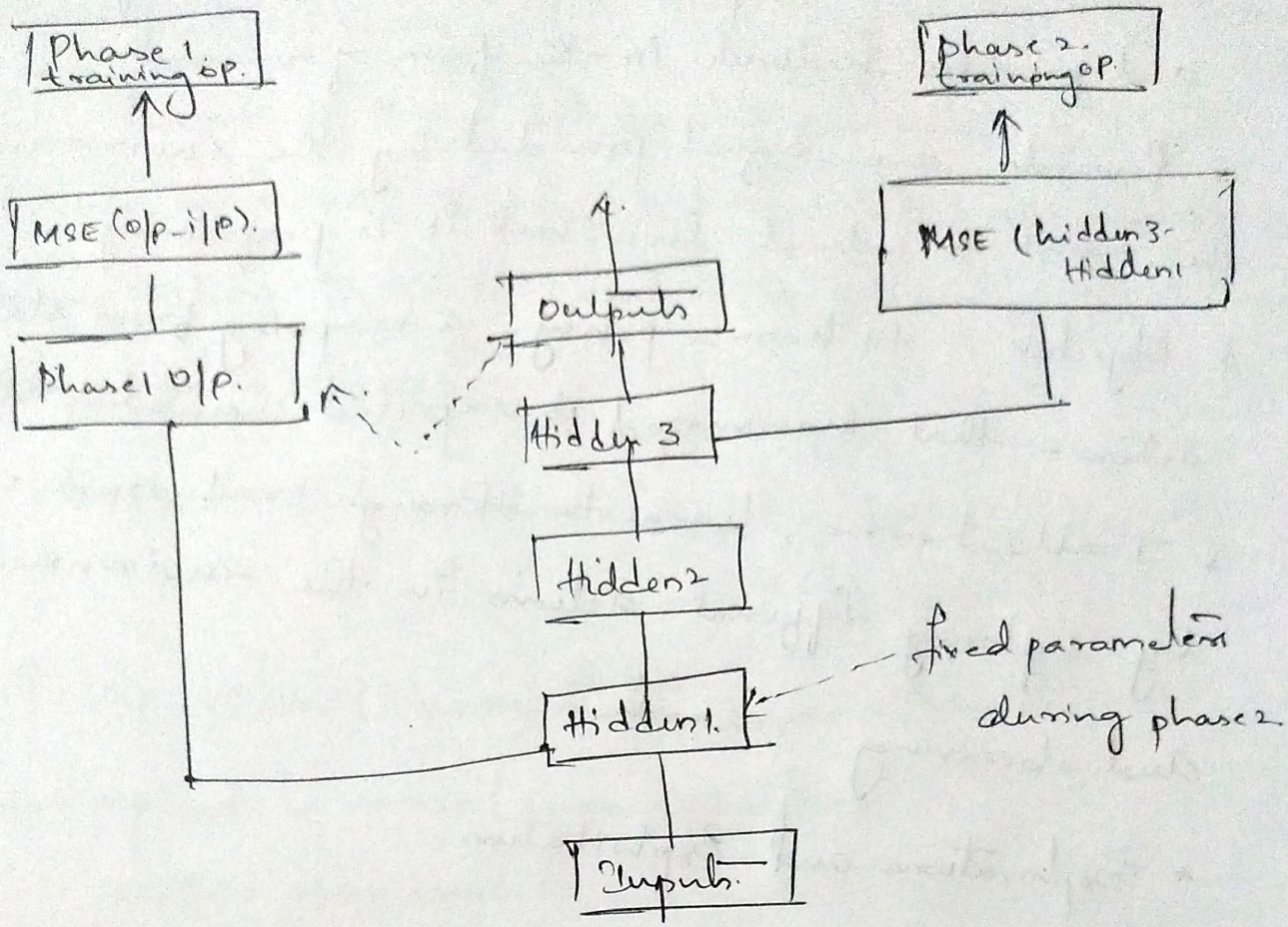
train the first encoder.

Phase 2

train the second auto encoder

Phase 3.

stack the auto encoder



5. Brief the concept of learning optimize rewards in reinforcement learning?

Ans: learning to optimize rewards in reinforcement learning involves training a s/w agent to make decisions & take actions within an environment to maximize its expected long-term rewards.

The agent learns through trail & error, adjusting its behaviours based on the rewards it receives from the environment.

* Agent and environment: agent interacts with the environment by making observations & taking actions.

the environment responds to the actions taken by the agent and provides feedback in the form of rewards.

* Rewards: are signal provided by the environment to the agent to indicate how well it is performing

* Objective : to learn a policy - a mapping from state to actions - that maximized its expected cumulative reward.

* Trail and error :- learns to through trail and error by exploring different actions in the environment and observing rewards.

* Explorations and Exploitation

* Applications

8. Explain markov Decision processes?

Ans: Markov decision process are mathematical framework used in the field of AI and decision theory to model decision-making problems in situations where outcomes are influenced by probabilistic transitions & involve sequential actions over time.

key component :

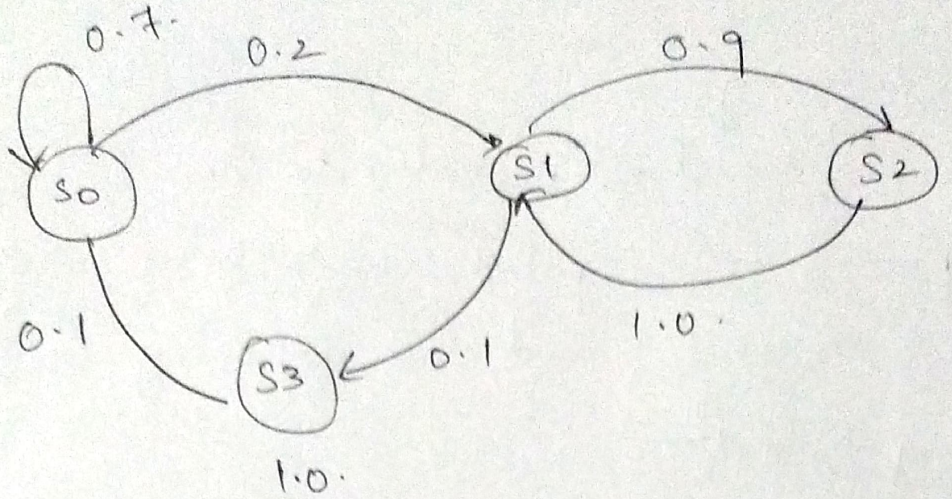
* States
* Actions
* Transition Probabilities
* Rewards
* Policy
* Value functions.
* Bellman equation

11. Explain Q learning

Ans: learns optimal action value function $Q^*(s,a)$ which represents the expected cumulative reward from taking action in state s and then following the optimal policy thereafter.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \cdot (r_{t+1} + \gamma \cdot \max_a Q(s_{t+1}, a) - Q(s_t, a_t))$$

```
Q = np.zeros((num_states, num_actions))
for episode in range(num_episodes):
    state = env.reset()
    done = False()
    while not done:
        action = epsilon_greedy_policy(Q, state,
                                       epsilon)
        next_state, reward, done = env.step(action)
        td_target = reward + discount_factor + np.max
                    (Q[next_state])]
        td_error = td_target - Q[state, action]
        Q[state, action] += learning_rate + td_error
        state = next_state.
```



9. Explain temporal difference learning

Ans: reinforcement learning algorithm that combines elements of dynamic programming and monte carlo methods

```
v = np.zeros(num_states)
for episode in range(num_episode):
    state = env.reset()
    done = False
    while not done:
        action = Policy(state)
        next_state, reward, done = env.step(action)
        td_target = reward + discount_factor *
                    v[next_state]
        td_error = td_target - v[state]
        v[state] += learning_rate * td_error
        state = next_state
```