CE889 NEURAL NETWORKS AND DEEP LEARNING
FINAL ASSIGNMENT

University of Essex, School of Computer Science and Electronic Engineering

Goktug Can Simay, gs25411

PROJECT OVERVIEW

- **Objective:** Train a neural network to predict lander thruster and turning commands from its X and Y distances to the target.
- **Environment:** Randomly generated terrain and landing zones.
- Data collected from gameplay used for offline training.

https://github.com/simaygoktug/deep_neural_networks

DATA PROCESSING

1. Reading the raw CSV

```python
def read_csv_safely(path: Path) -> pd.DataFrame:
    df = pd.read_csv(path)
    try:
        numeric_like = pd.to_numeric(pd.Series(df.columns), errors="coerce").notna().all()
    except Exception:
        numeric_like = False
    if numeric_like:
        df = pd.read_csv(path, header=None)
    return df
```

2. Cleaning the data

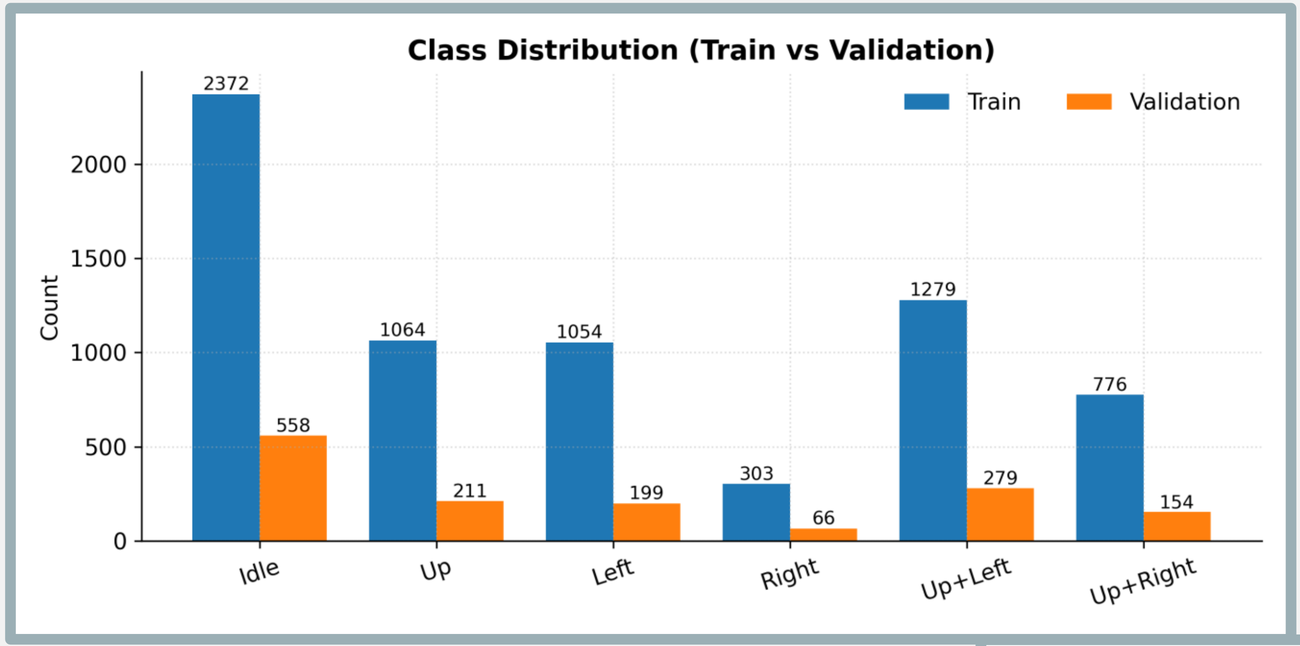
```python
def clean_dataframe(df: pd.DataFrame) -> pd.DataFrame:
    df = df.dropna(axis=1, how='all')
    df = df.replace([np.inf, -np.inf], np.nan)
    df = df.dropna(axis=0, how='all')
    df = df.ffill().bfill()
    df = df.dropna(axis=0, how='any')
    df = df.drop_duplicates()
    return df
```

3. Normalizing and splitting the data

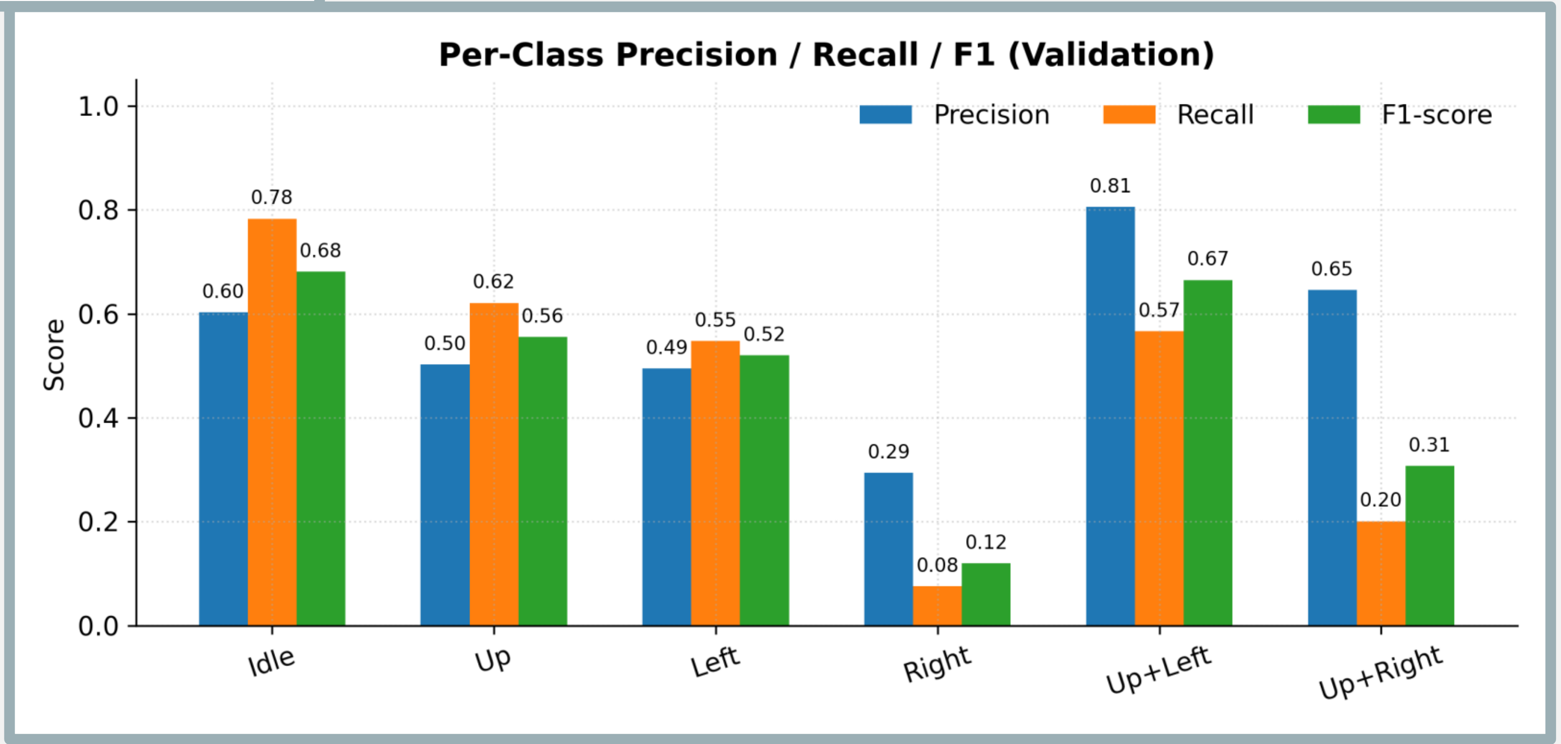
```python
def split_scale_three(df: pd.DataFrame, target_col: Optional[str], seed: int):
    numeric_cols = df.select_dtypes(include=[np.number]).columns.tolist()
    feature_cols = [c for c in numeric_cols if c != target_col] if target_col else numeric_cols
    if not feature_cols:
        raise ValueError("No numeric feature columns found to scale.")

    # 70 / 15 / 15 random split
    train_df, temp_df = train_test_split(df, test_size=0.30, random_state=seed, shuffle=True)
    val_df, test_df  = train_test_split(temp_df, test_size=0.50, random_state=seed, shuffle=True)

    scaler = MinMaxScaler()

    numeric_cols = df.select_dtypes(include=[np.number]).columns.tolist()
    scaler.fit(train_df[numeric_cols])

    def scale(part: pd.DataFrame) -> pd.DataFrame:
        out = part.copy()
        out[numeric_cols] = scaler.transform(part[numeric_cols])
        return out

    train_scaled = scale(train_df)
    val_scaled   = scale(val_df)
    test_scaled  = scale(test_df)
```

DATA ANALYSIS

1. Class Distribution



2. Metrics of Classes



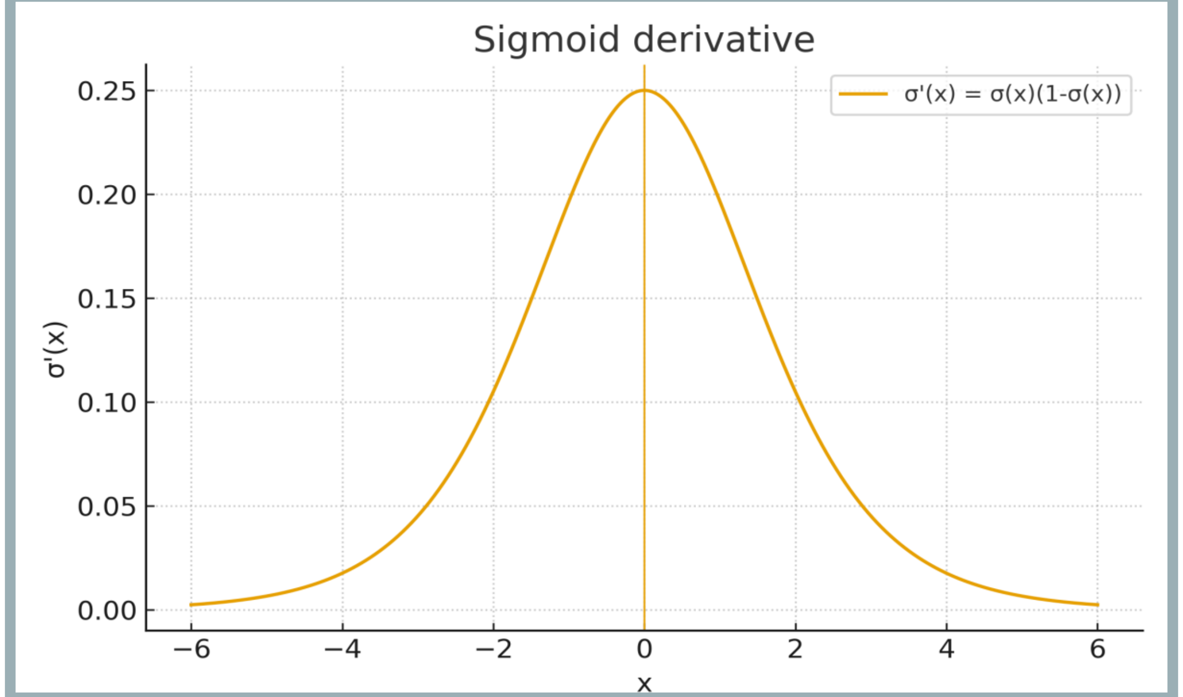
NETWORK DESIGN

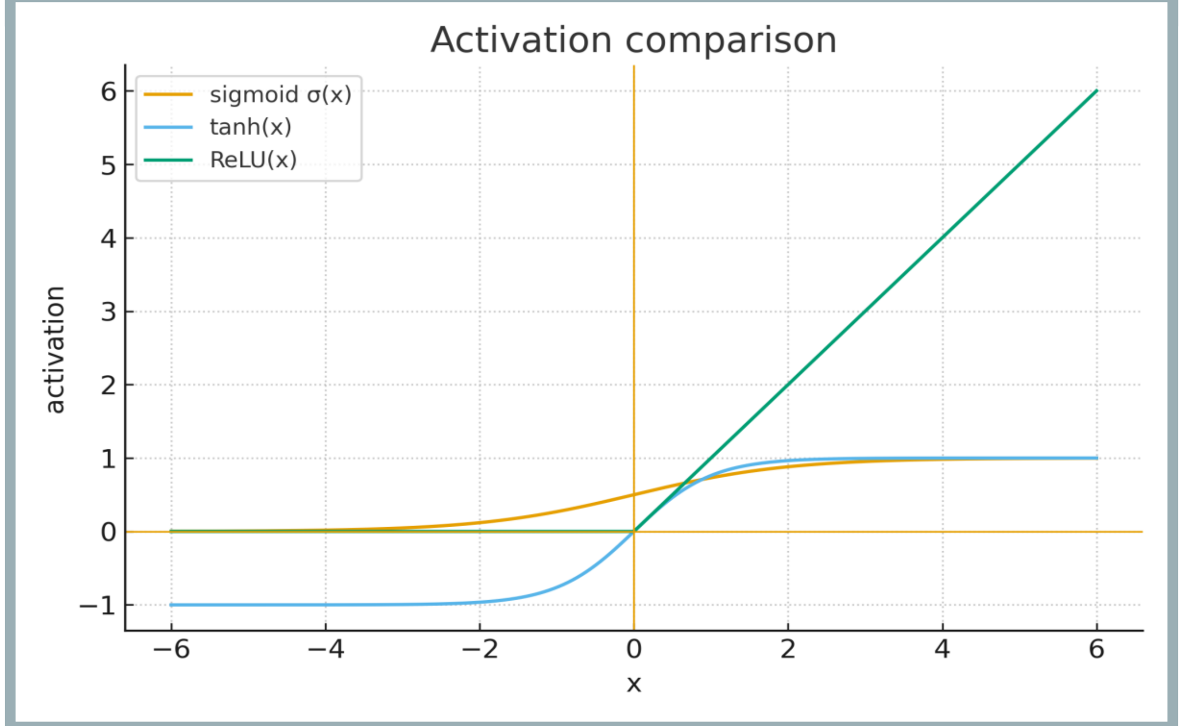
```python
class Neuron:
    def __init__(self, n_in: int):
        self.w = [random.uniform(-1.0,1.0) for _ in range(n_in)]
        self.b = random.uniform(-1.0,1.0)
        self.out=0.0; self.delta=0.0
        self.dw_prev=[0.0]*n_in; self.db_prev=0.0

    def fwd(self, x: List[float]) -> float:
        z = self.b
        for wi,xi in zip(self.w,x): z += wi*xi
        self.out = sigmoid(z); return self.out

class Layer:
    def __init__(self, n_in, n_neuron):
        self.neu = [Neuron(n_in) for _ in range(n_neuron)]
    def fwd(self, x: List[float]) -> List[float]:
        return [n.fwd(x) for n in self.neu]
```

```python
class MLP:
    def __init__(self, n_in, n_hidden, n_out, lr=0.05, momentum=0.9):
        self.h = Layer(n_in, n_hidden)
        self.o = Layer(n_hidden, n_out)
        self.lr=lr; self.mom=momentum

    def forward(self, x):
        h = self.h.fwd(x)
        y = self.o.fwd(h)
        return h, y

    def backprop(self, x, t):
        h, y = self.forward(x)
        for j, on in enumerate(self.o.neu):
            err = t[j] - y[j]
            on.delta = err * sd(on.out)
        for i, hn in enumerate(self.h.neu):
            s=0.0
            for on in self.o.neu: s += on.w[i]*on.delta
            hn.delta = s * sd(hn.out)
        # updating output
        for on in self.o.neu:
            for j, h_j in enumerate(h):
                dw = self.lr*on.delta*h_j + self.mom*on.dw_prev[j]
                on.w[j] += dw; on.dw_prev[j]=dw
            db = self.lr*on.delta + self.mom*on.db_prev
            on.b += db; on.db_prev=db
        # updating hidden
        for hn in self.h.neu:
            for j, xj in enumerate(x):
                dw = self.lr*hn.delta*xj + self.mom*hn.dw_prev[j]
                hn.w[j] += dw; hn.dw_prev[j]=dw
            db = self.lr*hn.delta + self.mom*hn.db_prev
            hn.b += db; hn.db_prev=db
        # mse for this sample (per-dim)
        return sum((tt-yy)**2 for tt,yy in zip(t,y))/max(1,len(t))
```

ACTIVATION FUNCTION SELECTION

```python
def sigmoid(x: float) -> float:
    if x < -60: return 0.0
    if x >  60: return 1.0
    return 1.0/(1.0+math.exp(-x))
```



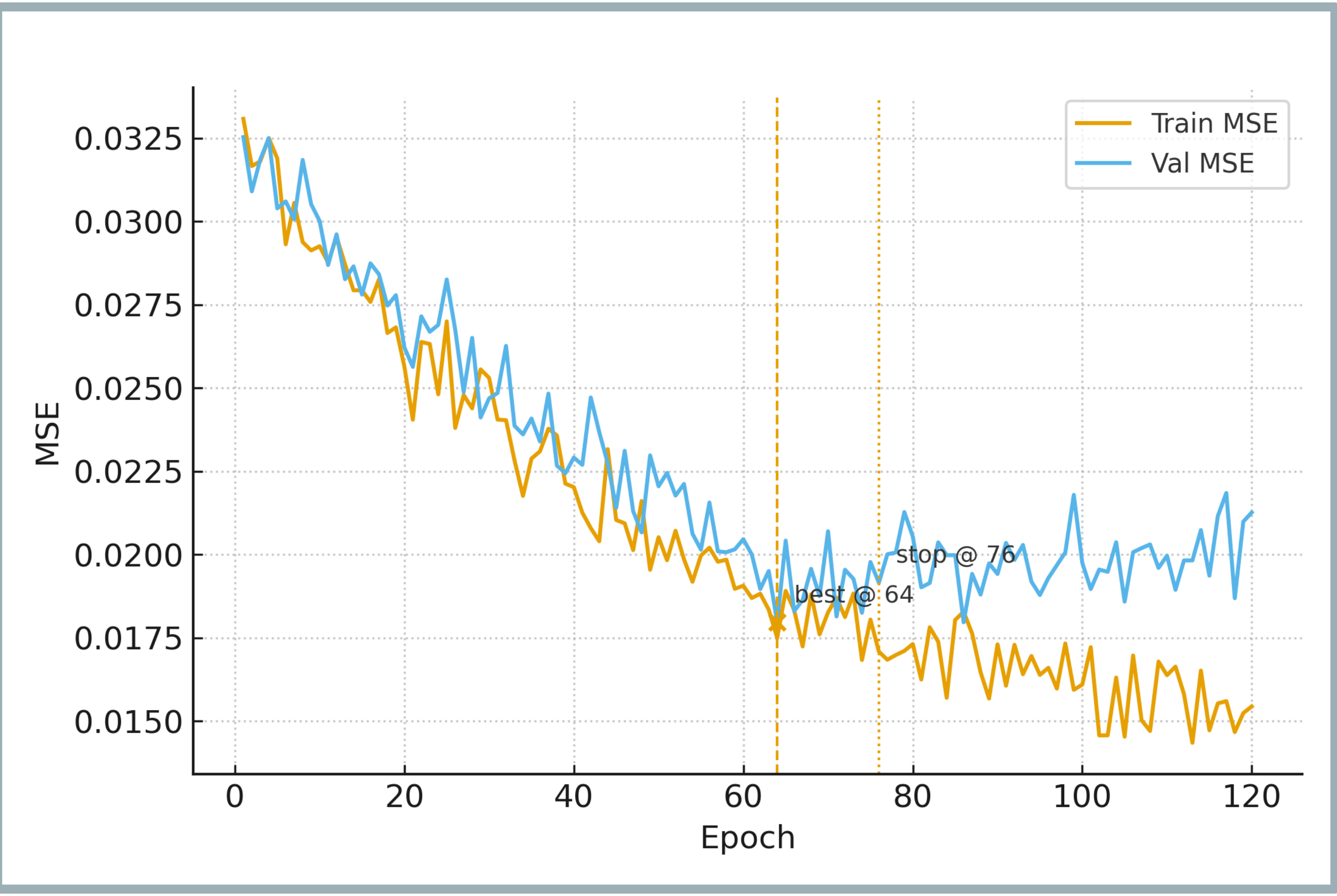


TRAINING PROCEDURE

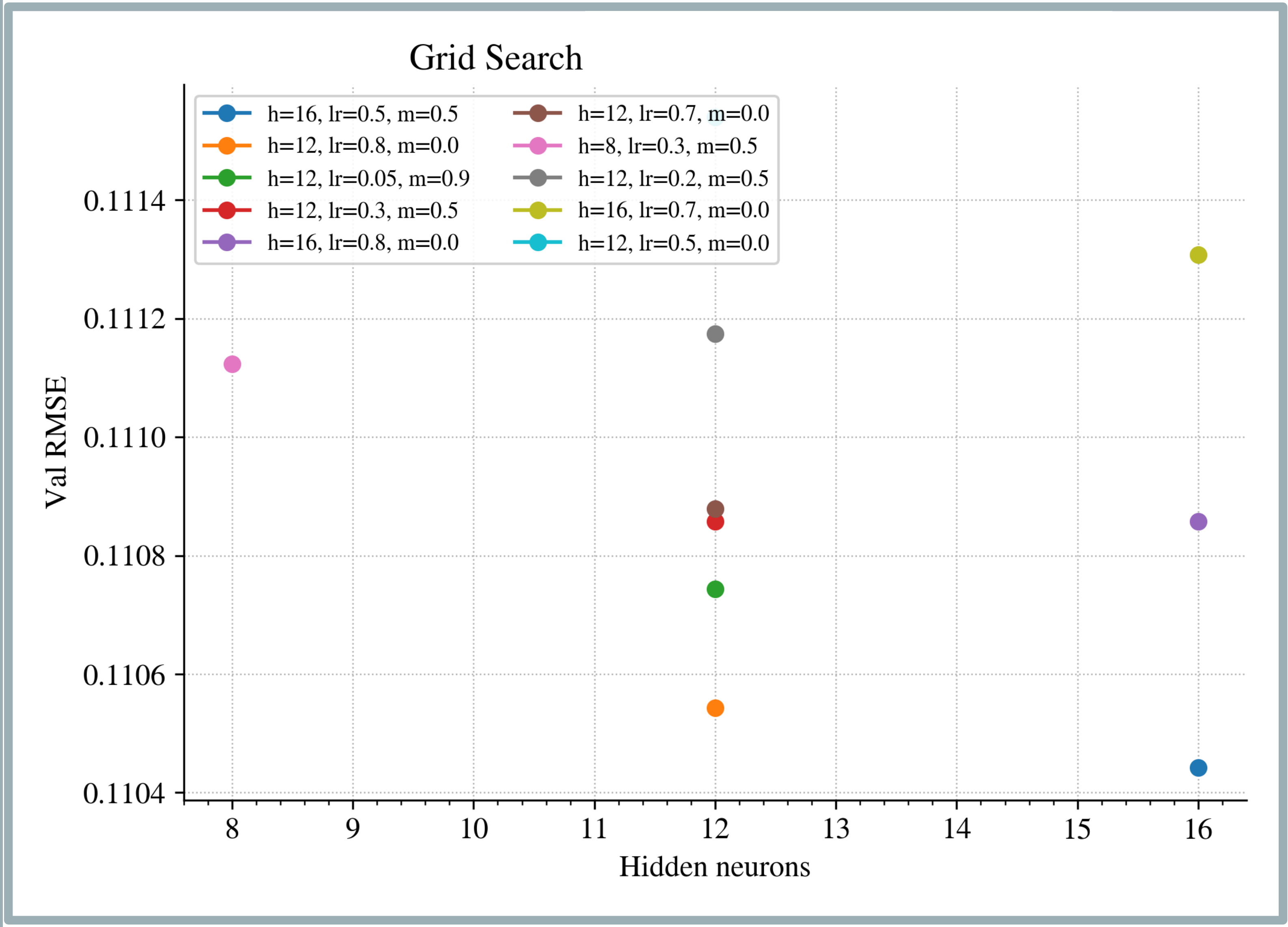
```
def train_loop(net: MLP, Xtr, Ytr, Xva=None, Yva=None, epochs=100, es: Optional[EarlyStop]=None):
    hist = {"epoch":[], "train_mse":[], "train_rmse":[], "val_mse":[], "val_rmse":[]}
    n=len(Xtr)
    for ep in range(1, epochs+1):
        idx=list(range(n)); random.shuffle(idx)
        tot=0.0
        for i in idx:
            tot += net.backprop(Xtr[i], Ytr[i])
        tr_mse = tot/max(1,n); tr_rmse = math.sqrt(tr_mse)
        va_mse = va_rmse = None
        if Xva and Yva:
            va_mse, va_rmse = evaluate(net, Xva, Yva, "val")
        hist["epoch"].append(ep); hist["train_mse"].append(tr_mse); hist["train_rmse"].append(tr_rmse)
        hist["val_mse"].append(va_mse if va_mse is not None else float("nan"))
        hist["val_rmse"].append(va_rmse if va_rmse is not None else float("nan"))
        if ep==1 or ep%max(1,epochs//10)==0 or ep==epochs:
            if va_mse is None:
                print(f"Epoch {ep}/{epochs} | train MSE={tr_mse:.6f} RMSE={tr_rmse:.6f}")
            else:
                print(f"Epoch {ep}/{epochs} | train {tr_mse:.6f}/{tr_rmse:.6f} | val {va_mse:.6f}/{va_rmse:.6f}")
        if es and va_mse is not None:
            if es.step(va_mse, net):
                print(f"Early stop @ {ep} (best={es.best:.6f})")
                es.restore(net); break
    return hist
```

EARLY STOPPING

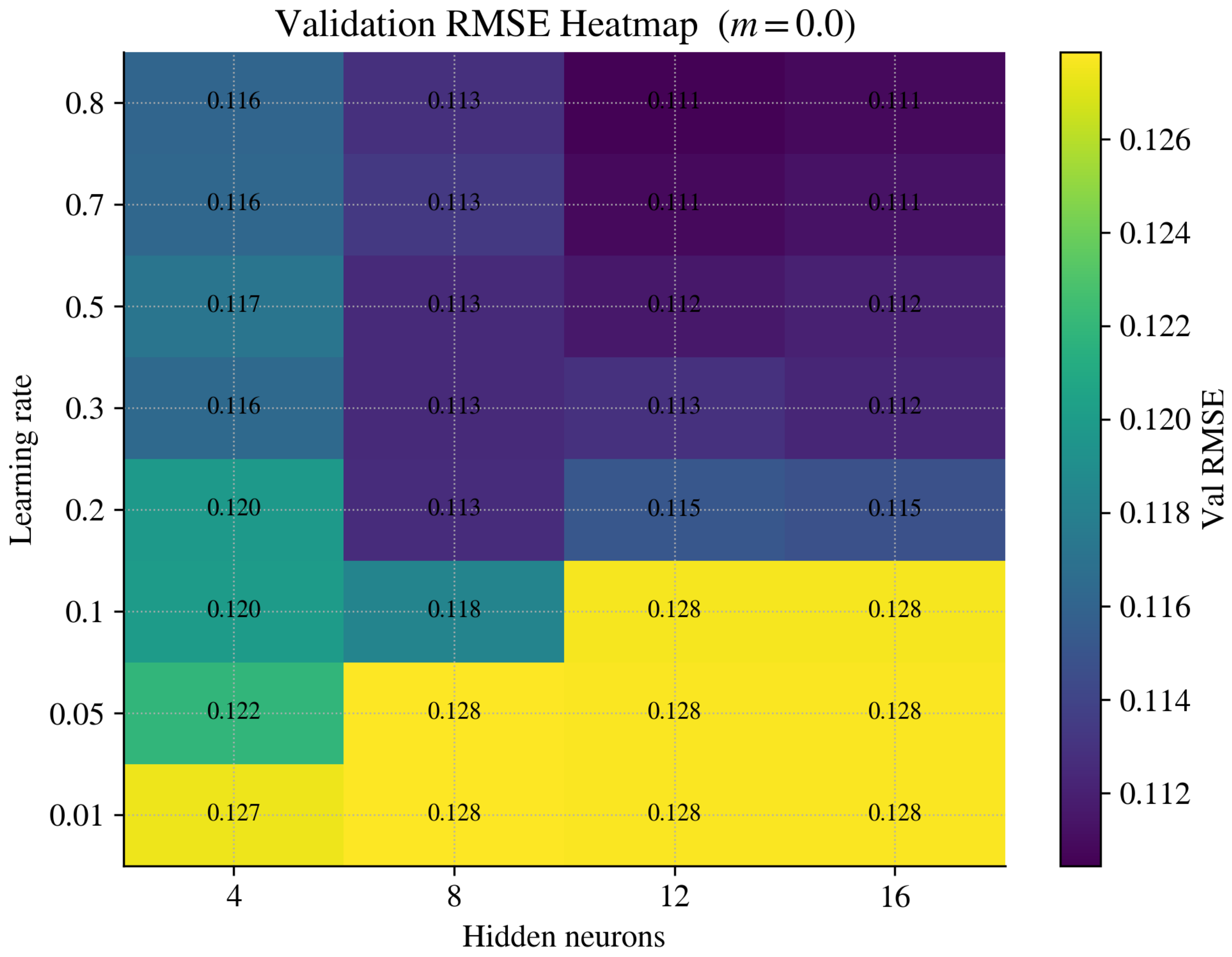
```
class EarlyStop:
    def __init__(self, patience=10, min_delta=1e-5):
        self.pat=patience; self.md=min_delta
        self.best=math.inf; self.buf=None; self.wait=0
    def step(self, cur, model: MLP):
        if cur + self.md < self.best:
            self.best = cur; self.wait=0; self.buf = snapshot(model)
            return False
        self.wait += 1
        return self.wait >= self.pat
    def restore(self, model: MLP):
        if self.buf: load_snapshot(model, self.buf)

def snapshot(model: MLP):
    return {
        "h_w":[n.w[:] for n in model.h.neu],
        "h_b":[n.b    for n in model.h.neu],
        "o_w":[n.w[:] for n in model.o.neu],
        "o_b":[n.b    for n in model.o.neu],
    }

def load_snapshot(model: MLP, s):
    for n,w in zip(model.h.neu, s["h_w"]): n.w=w[:]
    for n,b in zip(model.h.neu, s["h_b"]): n.b=b
    for n,w in zip(model.o.neu, s["o_w"]): n.w=w[:]
    for n,b in zip(model.o.neu, s["o_b"]): n.b=b
```



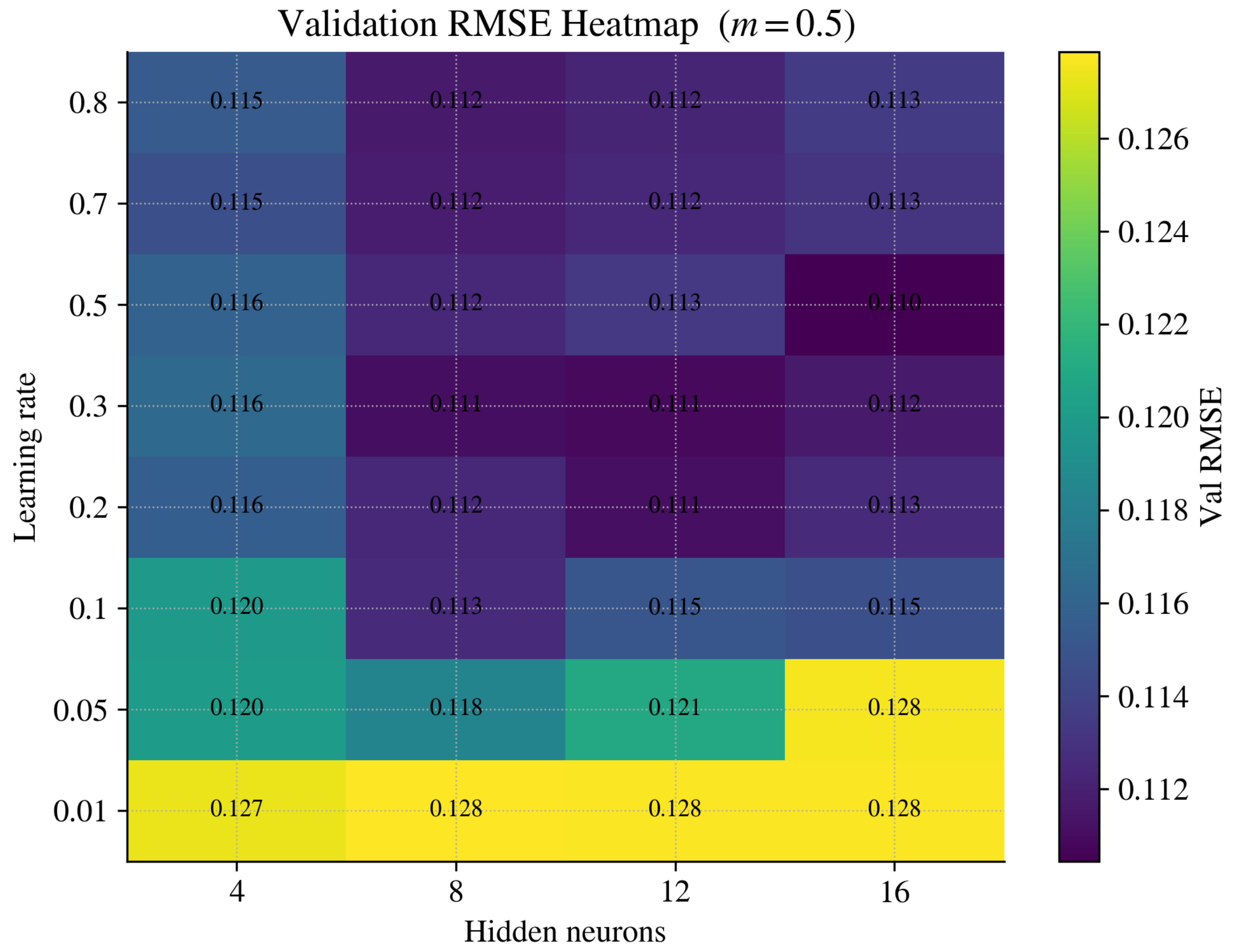
TRAINING PARAMETERS



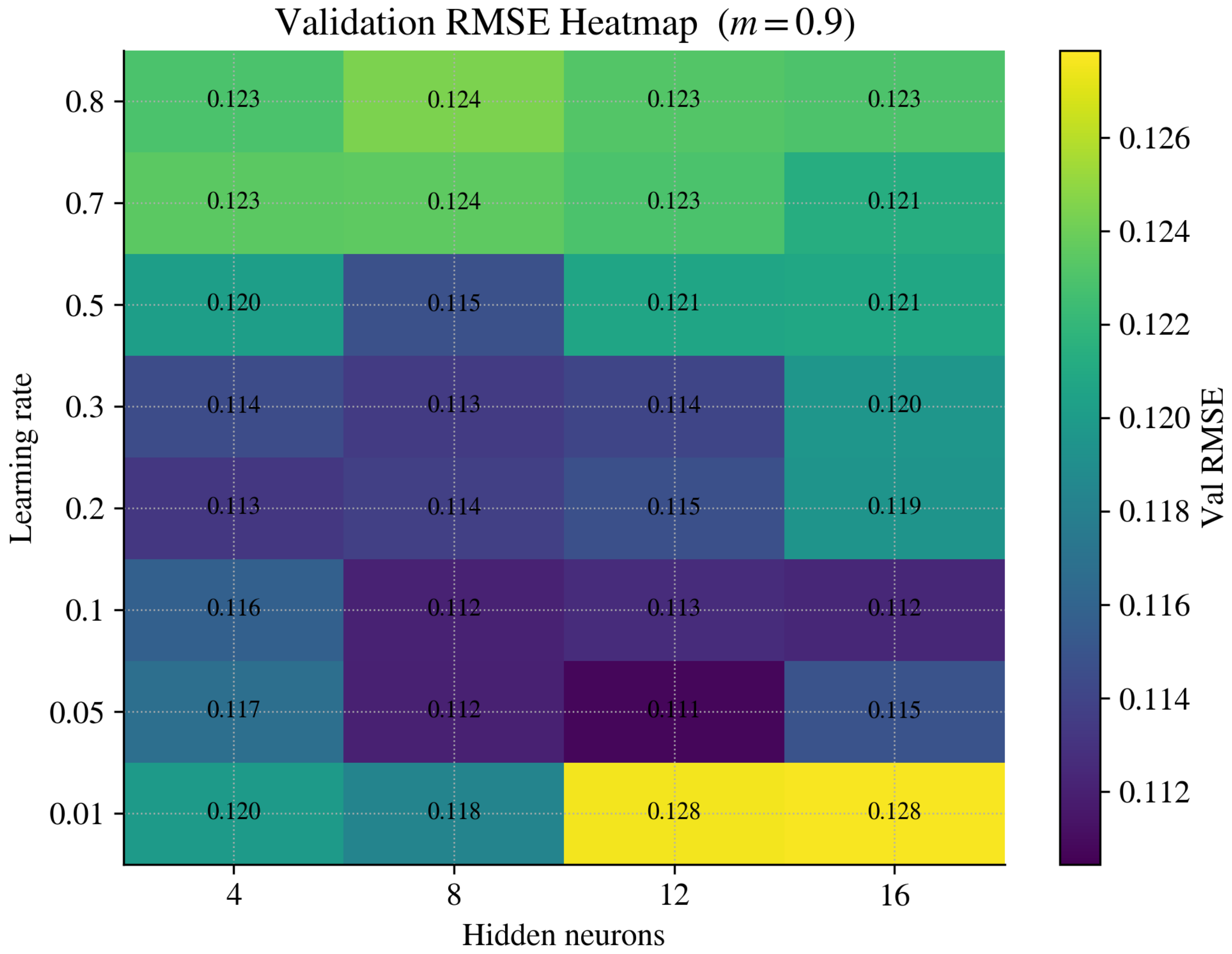
TRAINING PARAMETERS



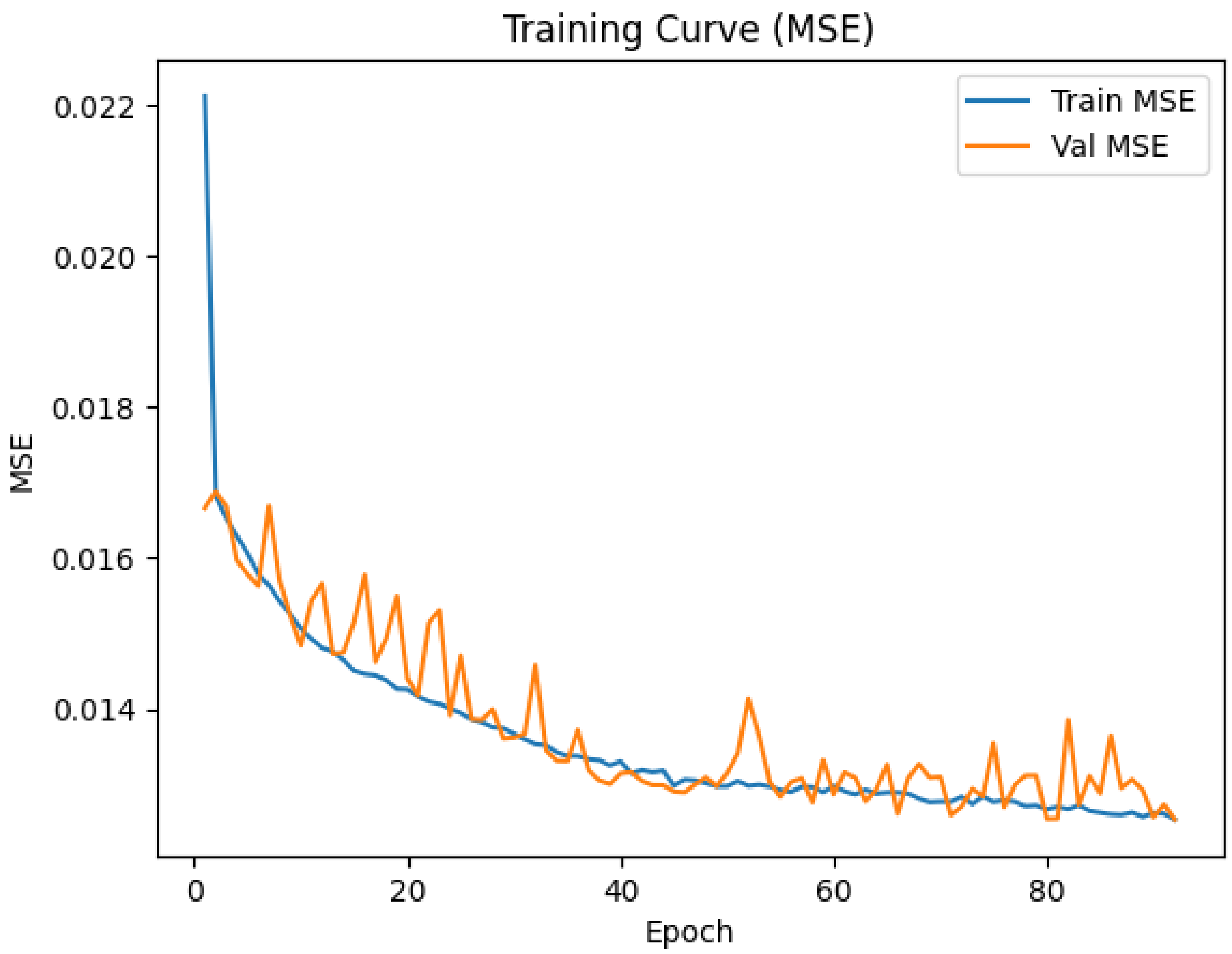
TRAINING PARAMETERS



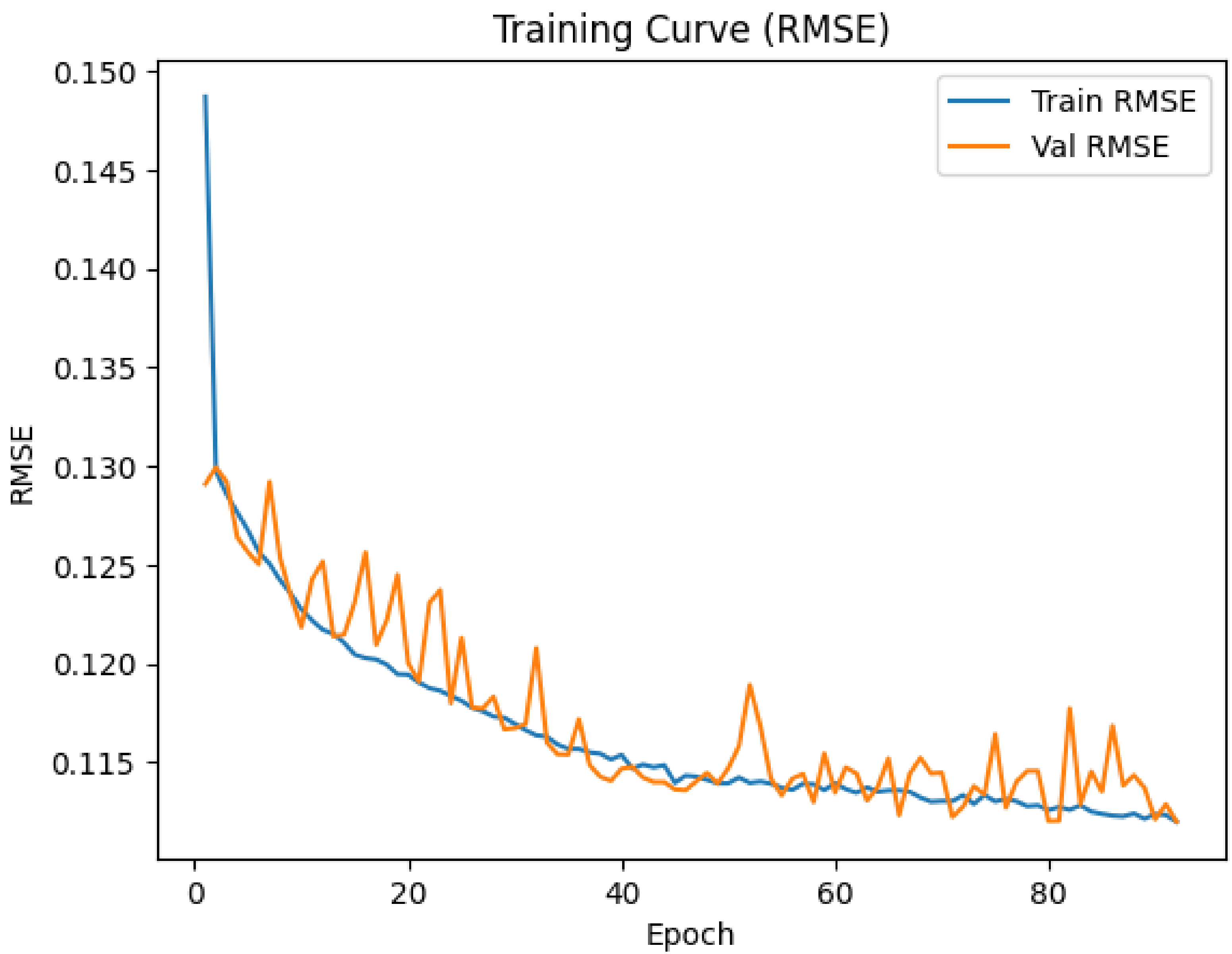
TRAINING PARAMETERS



PERFORMANCE METRICS



PERFORMANCE METRICS



REFERENCES

- Hagras, H. (2025). *CE889-7-AU: Neural Networks and Deep Learning* [Lecture notes], University of Essex.