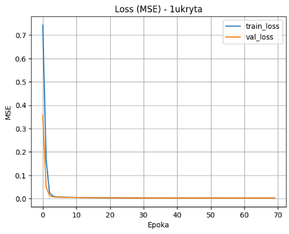
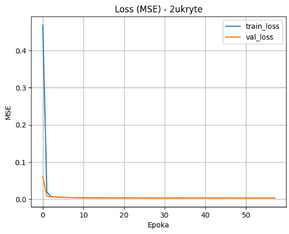
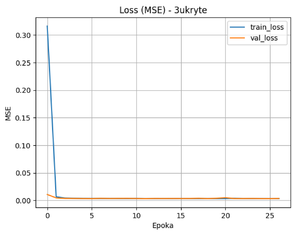
In [6]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers, callbacks
import matplotlib.pyplot as plt

np.random.seed(42)
tf.random.set_seed(42)

N = 5000
X = np.random.rand(N, 3) * np.array([50, 24, 10_000_000.0])

# Generowanie zależnych danych - y jako funkcja X + szum
y = np.zeros((N, 2))
y[:,0] = X[:,0]*5 + X[:,1]*2 + np.random.randn(N)*5       # Liczba wymaganych pracowników
y[:,1] = X[:,2]*0.001 + X[:,1]*50 + np.random.randn(N)*100 # Średnie wynagrodzenie

# Uporządkowanie do DataFrame (dla czytelności)
df = pd.DataFrame(np.hstack([X, y]), columns=[
    'n_projects', 'avg_duration_months', 'budget',
    'req_employees', 'avg_salary'
])

df.head()

# Podział na zbiór treningowy i testowy oraz skalowanie cech
X = df[['n_projects', 'avg_duration_months', 'budget']].values
y = df[['req_employees', 'avg_salary']].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

scaler_X = StandardScaler()
X_train_scaled = scaler_X.fit_transform(X_train)
X_test_scaled = scaler_X.transform(X_test)

# Standaryzacja targetów ułatwi trenowanie; zapiszemy skalery, aby odwrócić transformację
scaler_y = StandardScaler()
y_train_scaled = scaler_y.fit_transform(y_train)
y_test_scaled = scaler_y.transform(y_test)

# Funkcja pomocnicza do budowania modeli o różnych architekturach
def build_model(architecture='1ukryta', input_shape=(3,)):
    model = keras.Sequential()
    if architecture == '1ukryta':
        model.add(layers.Input(shape=input_shape))
        model.add(layers.Dense(64, activation='relu'))
    elif architecture == '2ukryte':
        model.add(layers.Input(shape=input_shape))
        model.add(layers.Dense(64, activation='relu'))
        model.add(layers.Dense(32, activation='relu'))
    elif architecture == '3ukryte':
        model.add(layers.Input(shape=input_shape))
        model.add(layers.Dense(128, activation='relu'))
        model.add(layers.Dense(64, activation='relu'))
        model.add(layers.Dense(32, activation='relu'))
    else:
        raise ValueError('Unknown architecture')
    model.add(layers.Dense(2))  # 2 wartości wyjściowe (regresja wielowymiarowa)
    model.compile(optimizer='adam', loss='mse', metrics=['mae'])
    return model

# Parametry treningu
EPOCHS = 200
BATCH_SIZE = 64
es = callbacks.EarlyStopping(monitor='val_loss', patience=15, restore_best_weights=True)

architectures = ['1ukryta', '2ukryte', '3ukryte']
histories = {}
models = {}
results = {}

for arch in architectures:
    print(f'\n--- Trening modelu:', arch, '---')
    model = build_model(arch, input_shape=(X_train_scaled.shape[1],))
    history = model.fit(
        X_train_scaled, y_train_scaled,
        validation_split=0.15,
        epochs=EPOCHS,
        batch_size=BATCH_SIZE,
        callbacks=[es],
        verbose=0
    )
    models[arch] = model
    histories[arch] = history
    y_pred_scaled = model.predict(X_test_scaled)
    y_pred = scaler_y.inverse_transform(y_pred_scaled)
    mse = mean_squared_error(y_test, y_pred)
    mae = mean_absolute_error(y_test, y_pred)
    results[arch] = {'mse': mse, 'mae': mae}
    print(f'Arch: {arch} -> MSE: {mse:.4f}, MAE: {mae:.4f}')

# Wykresy: historia MSE (loss) dla każdej architektury
for arch in architectures:
    h = histories[arch]
    plt.figure()
    plt.plot(h.history['loss'], label='train_loss')
    plt.plot(h.history['val_loss'], label='val_loss')
    plt.title(f'Loss (MSE) - {arch}')
    plt.xlabel('Epoka')
    plt.ylabel('MSE')
    plt.legend()
    plt.grid(True)
    plt.show()

# Wykres: rzeczywiste vs przewidywane (dla najlepszego modelu wg MSE)
best_arch = min(results, key=lambda k: results[k]['mse'])
print('\nNajlepsza architektura wg MSE:', best_arch, results[best_arch])

best_model = models[best_arch]
y_pred_best = scaler_y.inverse_transform(best_model.predict(X_test_scaled))

# Wykres dla obu wyjść w osobnych wykresach
plt.figure()
plt.scatter(y_test[:,0], y_pred_best[:,0], alpha=0.4)
plt.title('Rzeczywista vs Przewidywana - liczba wymaganych pracowników')
plt.xlabel('Rzeczywista liczba pracowników')
plt.ylabel('Przewidywana liczba pracowników')
plt.grid(True)
plt.show()

plt.figure()
plt.scatter(y_test[:,1], y_pred_best[:,1], alpha=0.4)
plt.title('Rzeczywiste vs Przewidywane - średnie wynagrodzenie')
plt.xlabel('Rzeczywiste średnie wynagrodzenie')
plt.ylabel('Przewidywane średnie wynagrodzenie')
plt.grid(True)
plt.show()

# Tabela wyników metryk dla modeli
results_df = pd.DataFrame(results).T
results_df
```

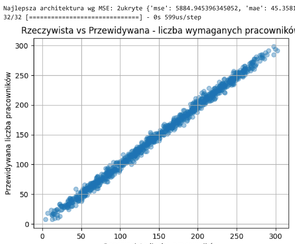
```
--- Trening modelu: 1ukryta ---
32/32 [==============================] - 0s 582us/step
Arch: 1ukryta -> MSE: 5928.8434, MAE: 45.8481

--- Trening modelu: 2ukryte ---
32/32 [==============================] - 0s 542us/step
Arch: 2ukryte -> MSE: 5884.9454, MAE: 45.3581

--- Trening modelu: 3ukryte ---
32/32 [==============================] - 0s 672us/step
Arch: 3ukryte -> MSE: 5902.0042, MAE: 45.5717
```








```
Najlepsza architektura wg MSE: 2ukryte {'mse': 5884.945396345052, 'mae': 45.358130662204935}
32/32 [==============================] - 0s 599us/step
```



Out[6]:

	mse	mae
1ukryta	5928.843382	45.848133
2ukryte	5884.945396	45.358131
3ukryte	5902.004159	45.571734