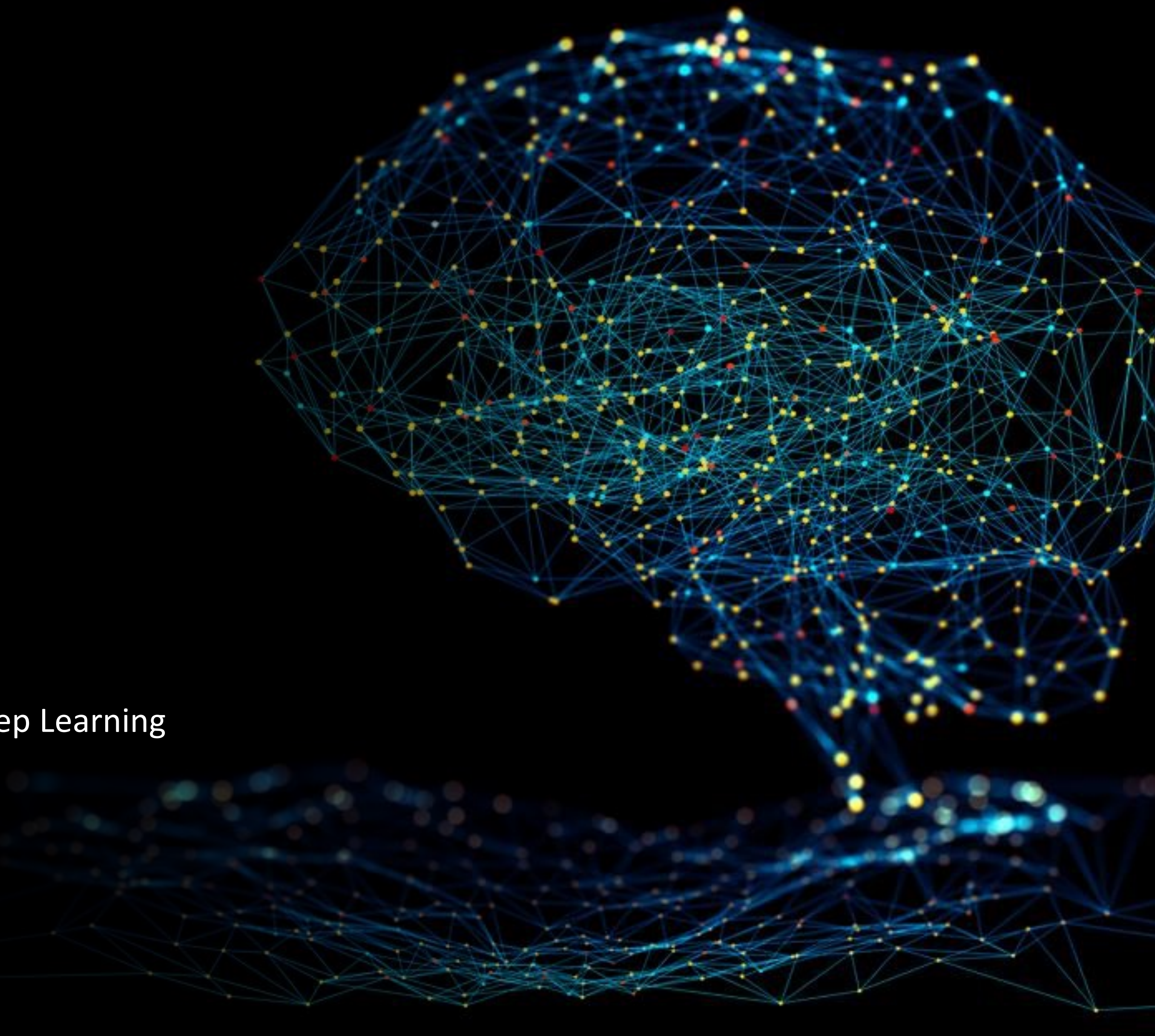
Target-Guided AutoEncoder

Niclas Wölner-Hanssen, VT 2023, STAN47 – Deep Learning

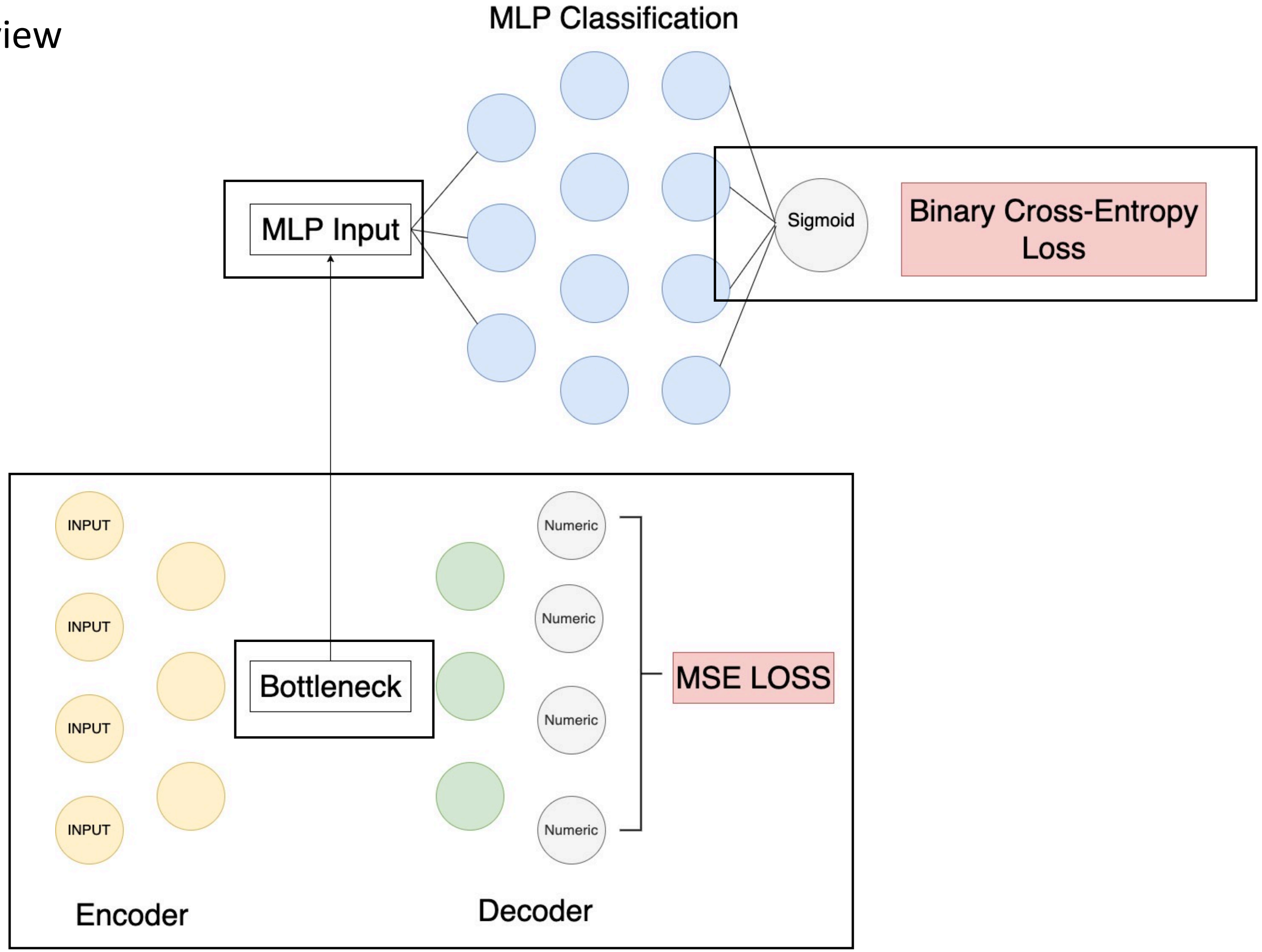


Targeted-Guided AutoEncoder

- Feature extraction/ dimension reduction technique that is guided to construct latent variables that are informative.
- Often times the indirect purpose of dimension reduction techniques such as PCA and AE is to reduce the number of dimensions to make better predictions (overcome multicollinearity, overfitting etc.)
- However, these transformed new features, the latent variables, are not necessarily predictive.
- No information about the target, the thing we try to predict, is introduced in the transformation of the features into the latent variables when using PCA or Simple AutoEncoder.

Targeted-Guided AutoEncoder

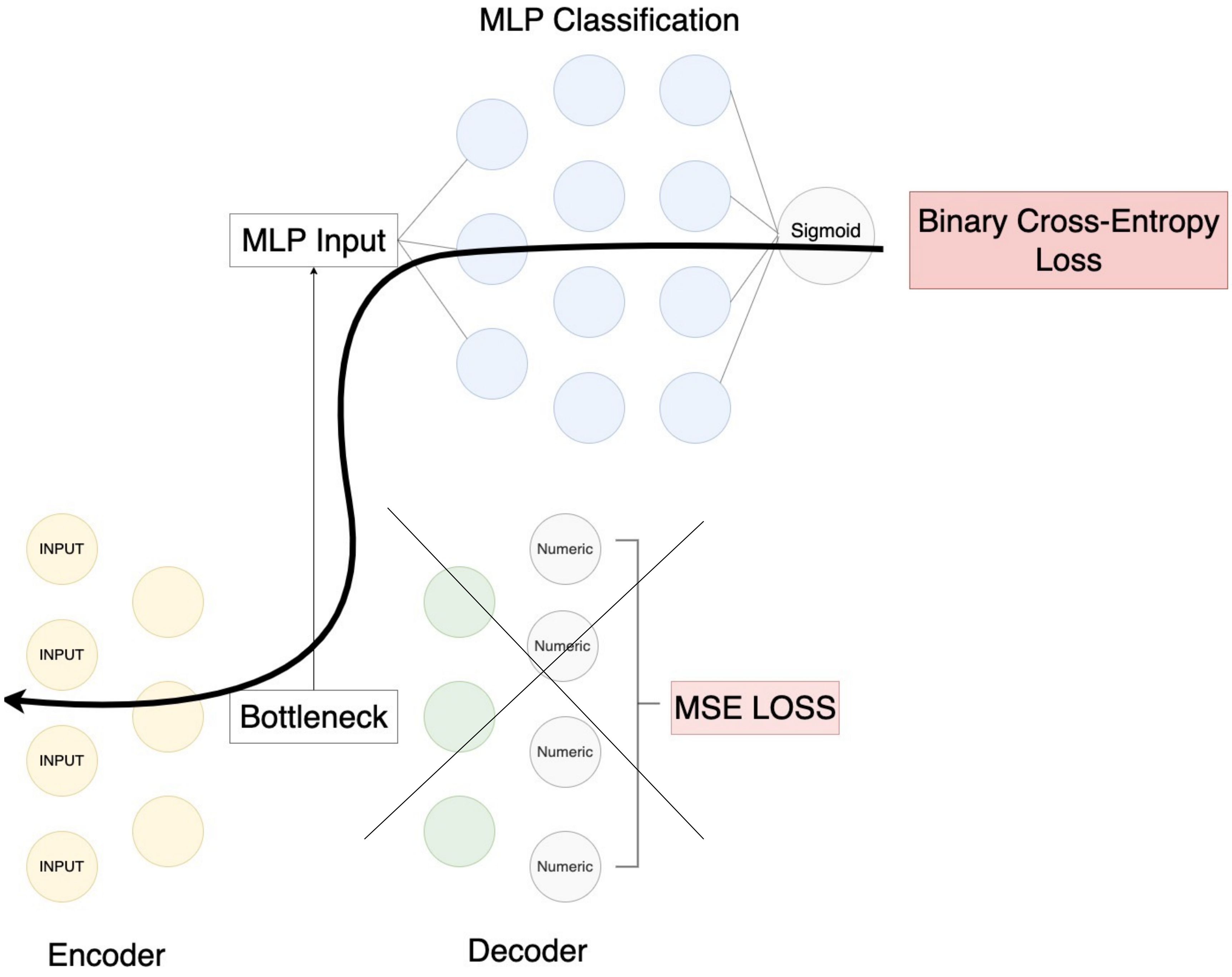
Model Overview



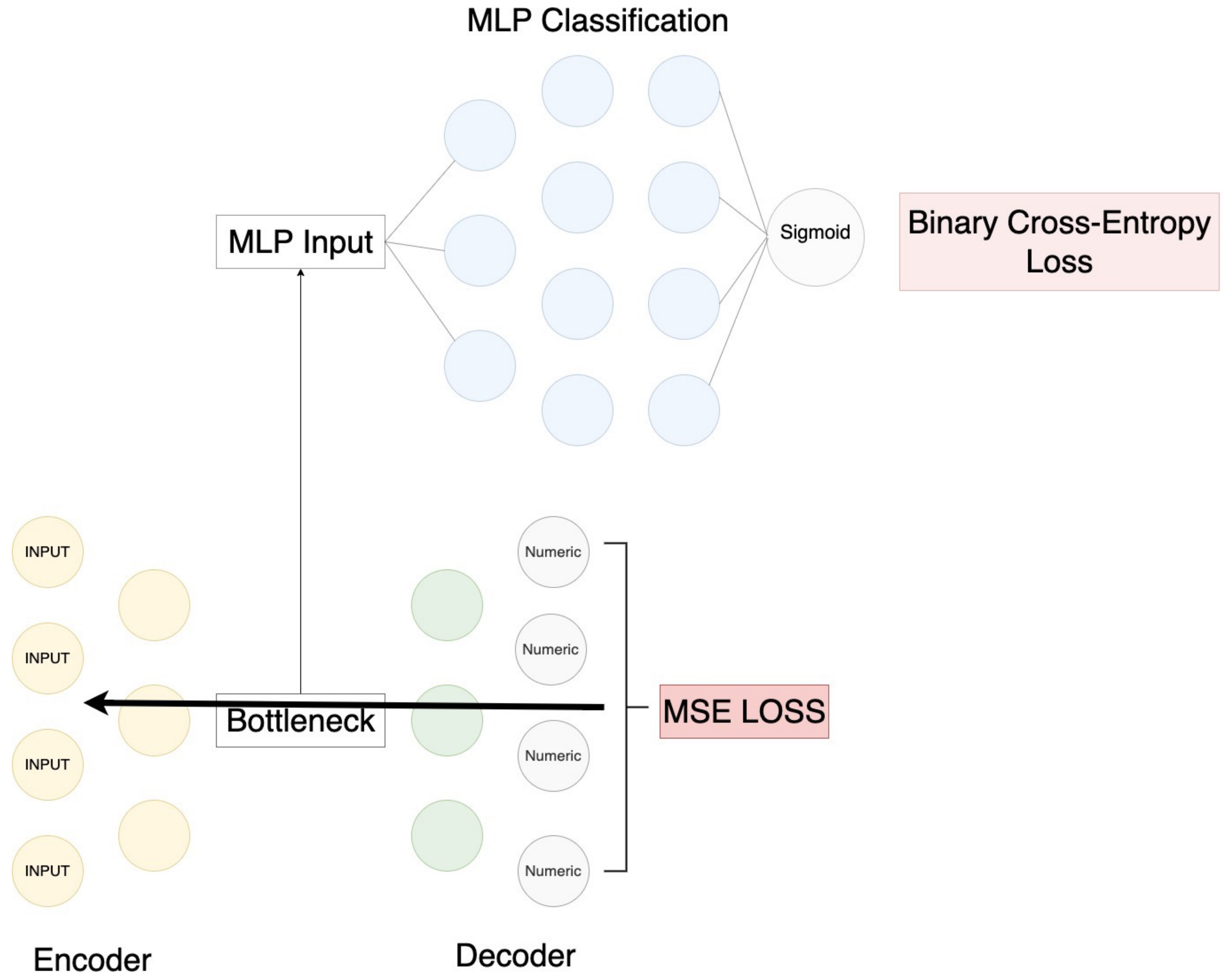
Target-Guided AutoEncoder

- By including a seperate MLP branch in the overall model architecture, the target information (i.e y_train) is introduced to the shared bottleneck layer of the autoencoder.
- During training, the gradients of the MLP loss with respect to the weights of the MLP are propagated backwards through the shared bottleneck layer of the autoencoder, allowing the model to update the bottleneck representation in a way that is informed by the classification task.
- Bottomline is that the inclusion of the MLP branch enables the model to learn the bottleneck representation that is not only optimized for reconstruction of the input data but also takes the target information into account.

Backprop

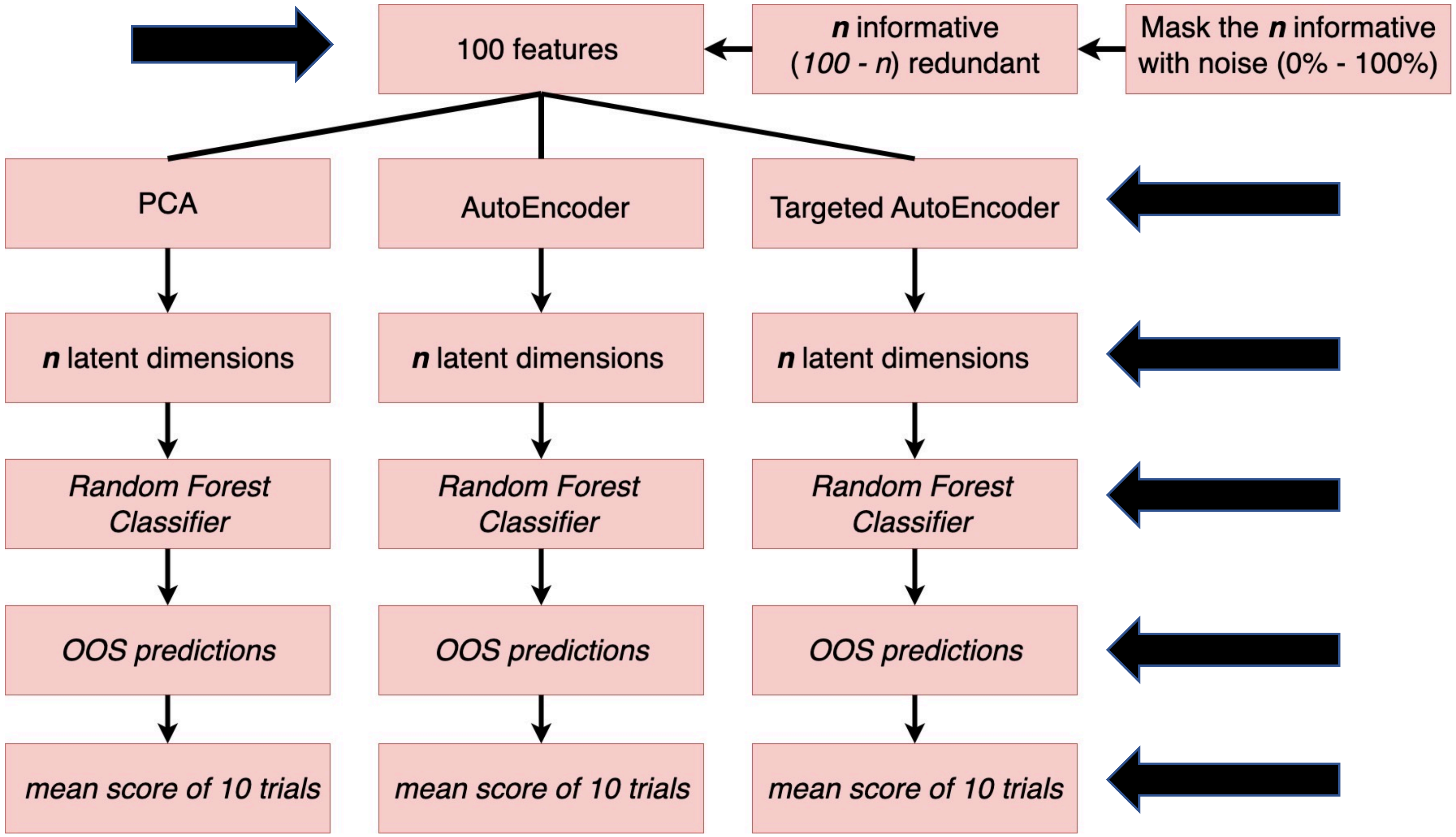

MLP Classification
MLP Input
Sigmoid
Binary Cross-Entropy Loss
INPUT
INPUT
INPUT
INPUT
Bottleneck
Numeric
Numeric
Numeric
Numeric
MSE LOSS
Encoder
Decoder

Backprop



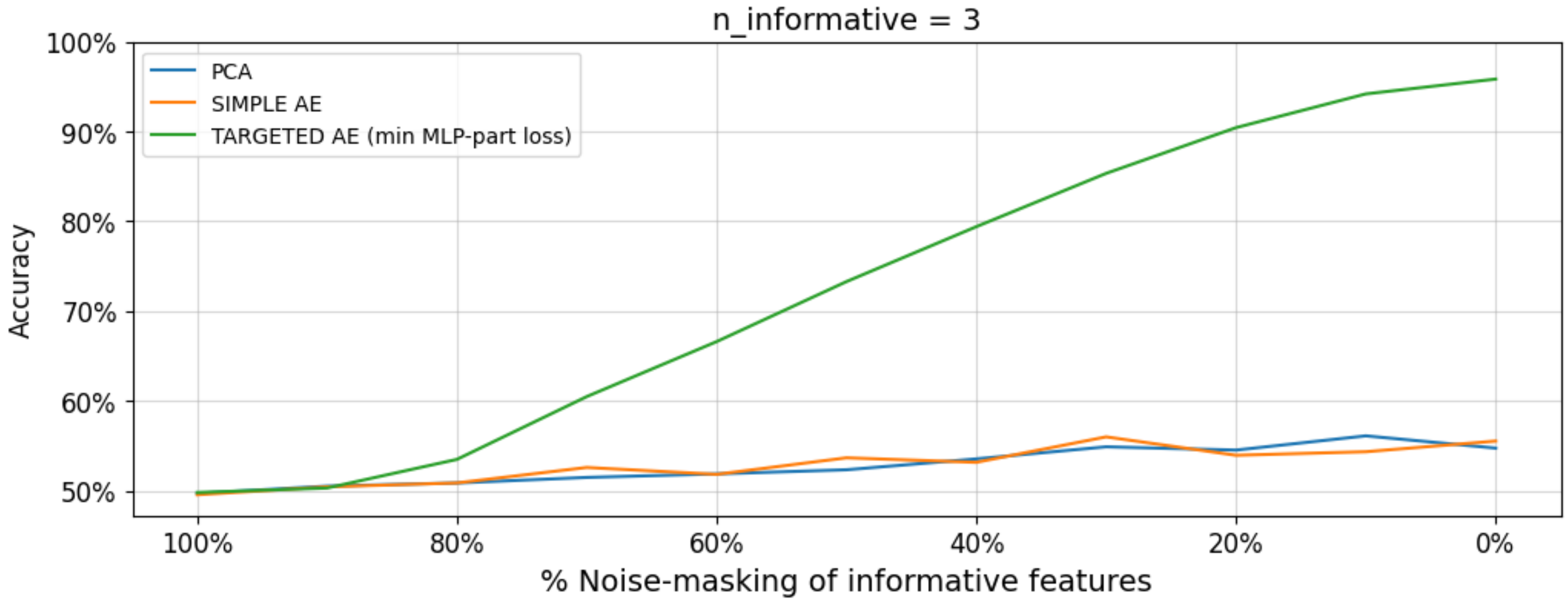
MLP Classification
MLP Input
Sigmoid
Binary Cross-Entropy Loss
INPUT
INPUT
INPUT
INPUT
Bottleneck
Numeric
Numeric
Numeric
Numeric
MSE LOSS
Encoder
Decoder

Testing



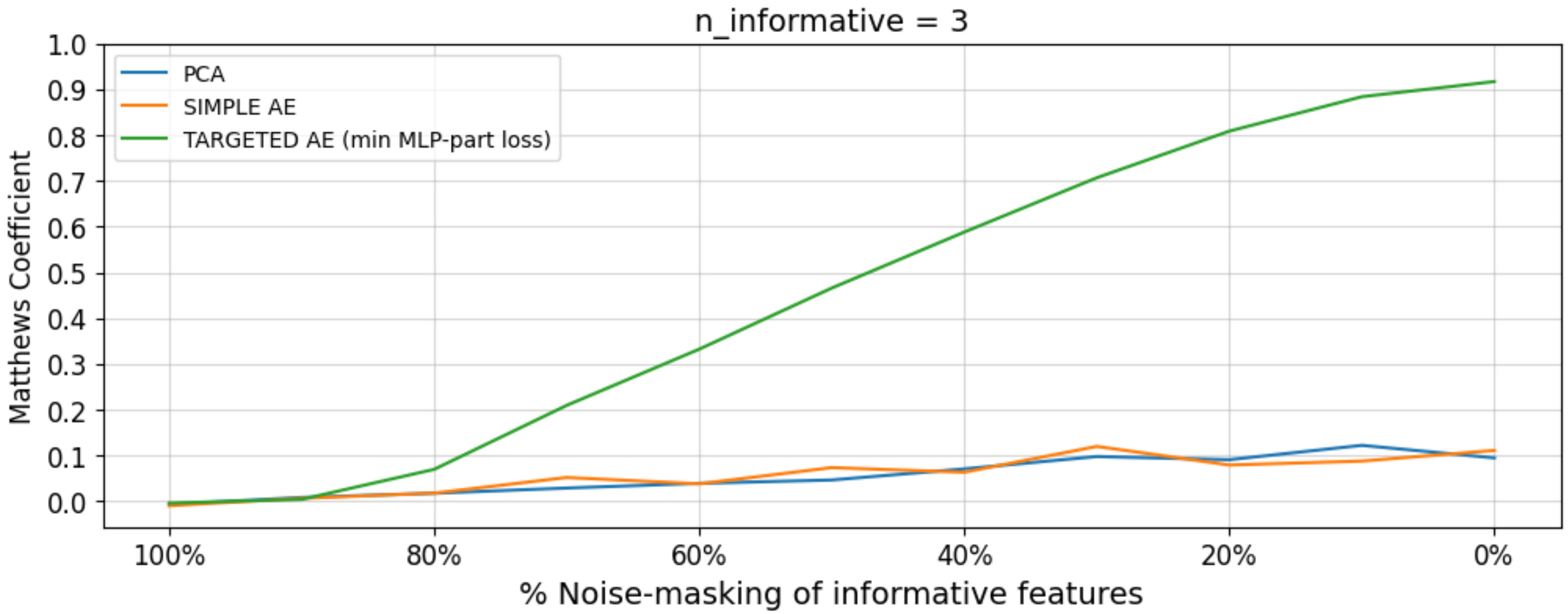
*RF: n_estimators =500, max_depth = 5, max_samples = 100

Testing



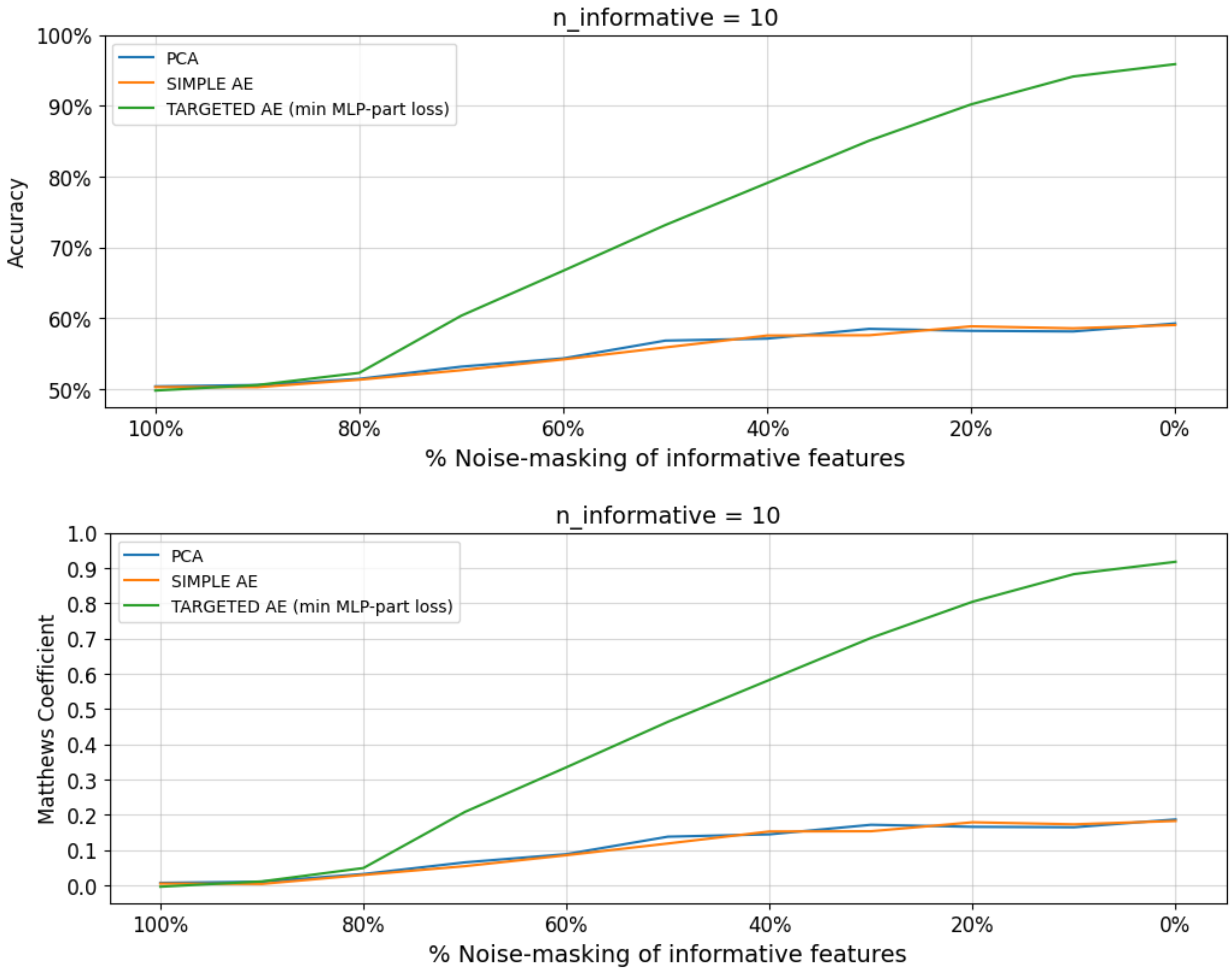
- 100 features (IID Std Normal, 10K samples)
- The sum of the first 3 features makes up the binary target:
  - y>0 = 1
    y<0 = 0
- The other features are thus just noise.
- In the plot the 3 informative features are after the construction of the target masked with noise. Going from complete noise (100%) to no-noise (0%)(x-axis).

Testing

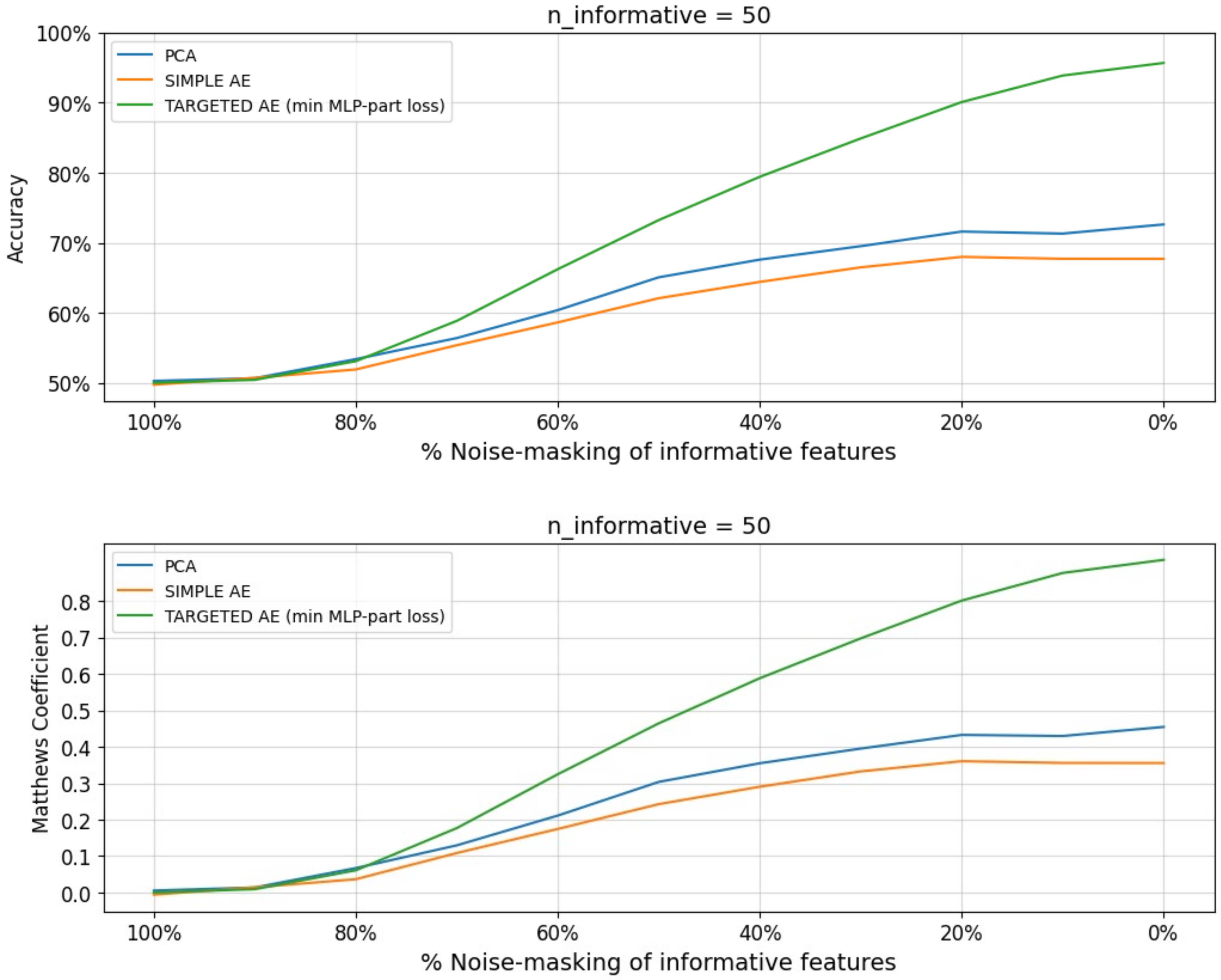


- Matthews correlation coefficient

Testing


n_informative = 10
PCA
SIMPLE AE
TARGETED AE (min MLP-part loss)
Accuracy
100%
90%
80%
70%
60%
50%
100%
80%
60%
40%
20%
0%
% Noise-masking of informative features
n_informative = 10
PCA
SIMPLE AE
TARGETED AE (min MLP-part loss)
Matthews Coefficient
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
100%
80%
60%
40%
20%
0%
% Noise-masking of informative features

Testing

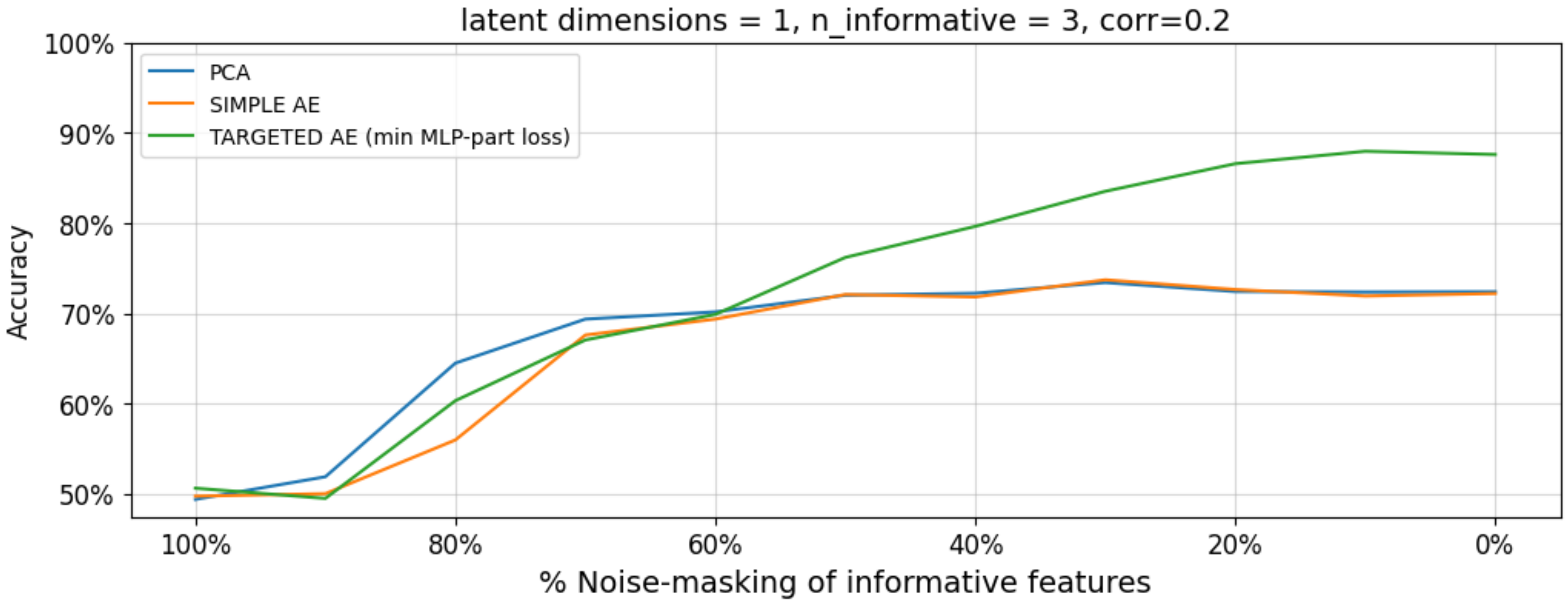


n_informative = 50
Accuracy
100%
90%
80%
70%
60%
50%
PCA
SIMPLE AE
TARGETED AE (min MLP-part loss)
100%
80%
60%
40%
20%
0%
% Noise-masking of informative features
n_informative = 50
Matthews Coefficient
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
PCA
SIMPLE AE
TARGETED AE (min MLP-part loss)
100%
80%
60%
40%
20%
0%
% Noise-masking of informative features

Target-Guided AutoEncoder

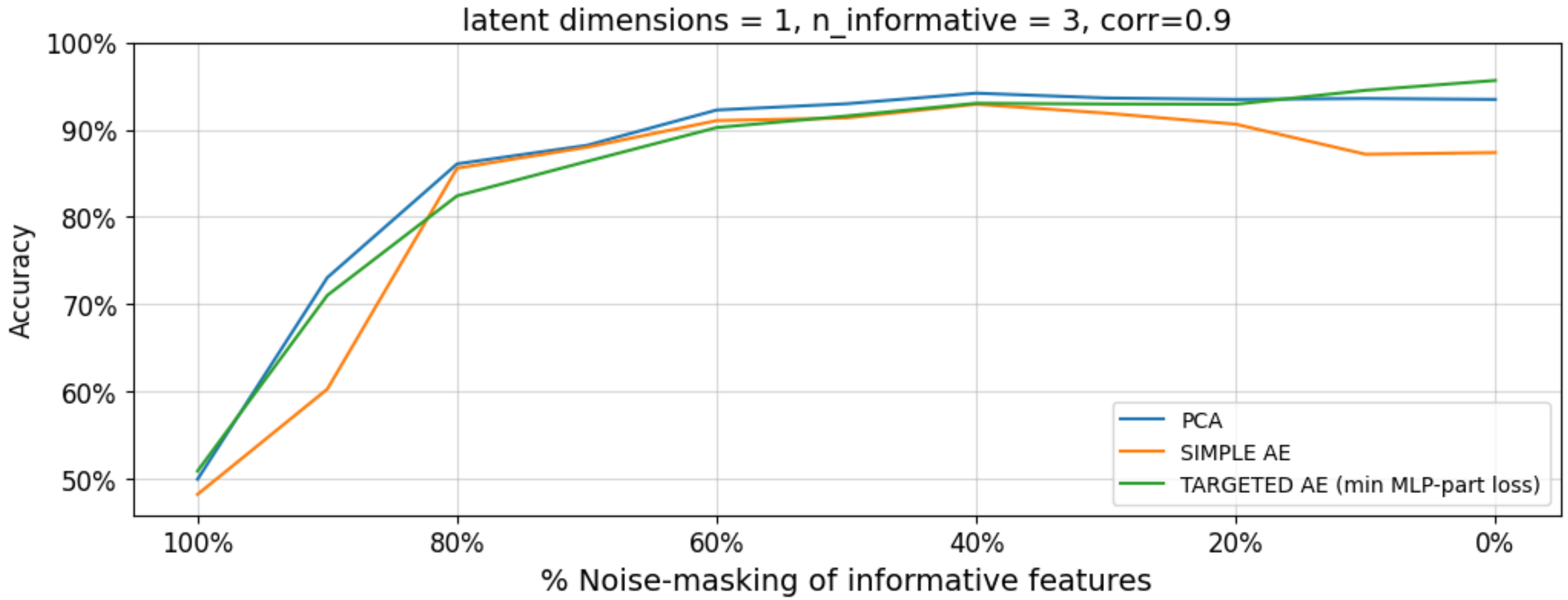
- So far, the samples has been IID. However, PCA is based on the covariance/correlation matrix and generally the higher the correlation between the features the more information (i.e. variance) of these it can store in fewer dimensions.
- For this reason, lets introduce some correlation between the features.

Testing



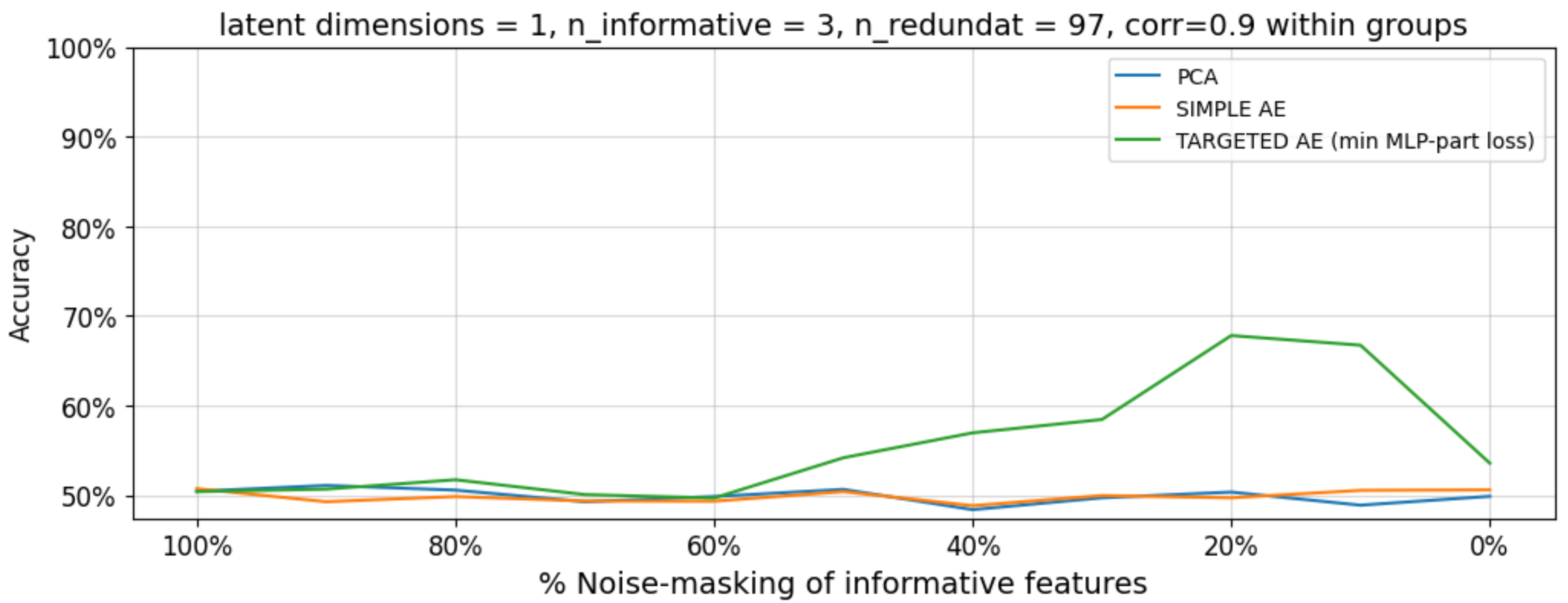
- The target is constructed as the sign of the sum of the 3 informative features, the other 97 features are ”redundant”.
- However, the correlation between all features is approx 0.2 when noise = 0%.
- PCA: First principal component
- AE & Targeted AE: 1 latent dimension

Testing



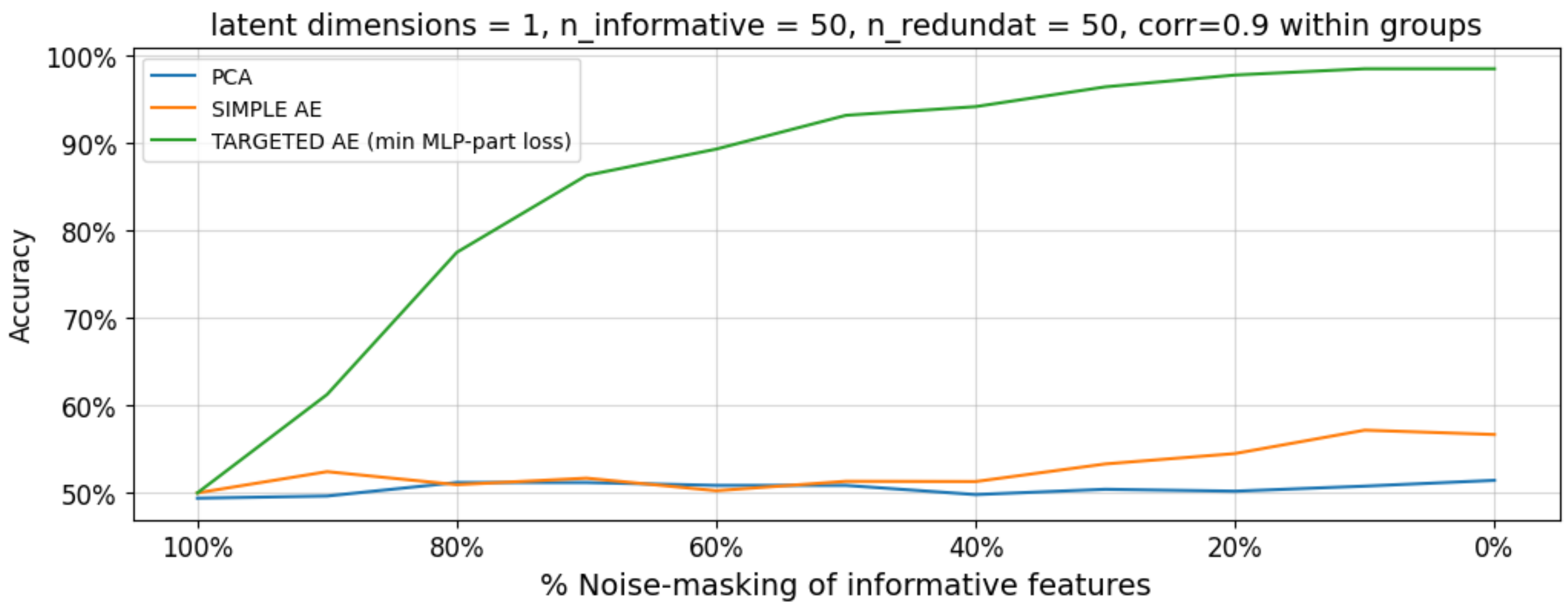
- The target is constructed as the sign of the sum of the 3 informative features, the other 97 features are ”redundant”.
- However, the correlation between all features is approx 0.9 when noise = 0%.
- PCA: First principal component
- AE & Targeted AE: 1 latent dimension

Testing



- Within the 3 informative features the correlation is 0.9 when noise = 0%.
- Within the 97 redundant features the correlation is 0.9 when noise = 0%.
- Between these groups the correlation is approx 0.
- PCA: First principal component
- AE & Targeted AE: 1 latent dimension

Testing



- Within the 50 informative features the correlation is 0.9 when noise = 0%.
- Within the 50 redundant features the correlation is 0.9 when noise = 0%.
- Between these groups the correlation is approx 0.
- PCA: First principal component
- AE & Targeted AE: 1 latent dimension

Conclusion:

Testing

- The Target-Guided AutoEncoder tend to at least never underperform PCA and simple AE, and most often it outperforms in terms of informative quality in constructing latent variables.
- The results also shows that PCA and Simple AutoEncoder seems to from time to time miss the informative information in the features, whereas the Target-Guided AutoEncoder finds these with realtive ease most of the time.

Code

In [94]:

```python
# input regularization
input_x = Input(shape=(x_inputs,))
input_x = GaussianNoise(0.03)(input_x)
input_x = BatchNormalization()(input_x)
# encoder level 1
e = Dense(x_inputs, activation='tanh',kernel_initializer=RandomNormal(mean=0.0, stddev=0.01))(input_x) #layer 1
e = BatchNormalization()(e)
e = Dropout(dropout_rate)(e)
# encoder level 2
e = Dense(int(x_inputs/2),activation='tanh')(e) #layer 2
e = BatchNormalization()(e)
e = Dropout(dropout_rate)(e)
# bottleneck
bottleneck = Dense(latent_dimensions)(e) #Bottleneck

# decoder, level 1
d = Dense(int(x_inputs/2),activation='tanh')(bottleneck) #layer 1
d = BatchNormalization()(d)
d = Dropout(dropout_rate)(d)
# decoder level 2
d = Dense(x_inputs,activation='tanh')(d)  #layer 2
d = BatchNormalization()(d)
d = Dropout(dropout_rate)(d)
# output layer
decoder_output = Dense(x_inputs, activation='linear',name = 'decoder_output')(d)

merged = concatenate([bottleneck])

####MLP Classification

mlp = Dense(merged.shape[1], activation='tanh')(merged) #Input Layer MLP
mlp = BatchNormalization()(mlp)
mlp = Dropout(dropout_rate)(mlp)

mlp = Dense(x_inputs*3, activation='tanh')(mlp) #layer 1
mlp = BatchNormalization()(mlp)
mlp = Dropout(dropout_rate)(mlp)

mlp = Dense(x_inputs*3, activation='tanh')(mlp) #layer 2
mlp = BatchNormalization()(mlp)
mlp = Dropout(dropout_rate)(mlp)

mlp = Dense(x_inputs*2, activation='tanh')(mlp) #layer 3
mlp = BatchNormalization()(mlp)
mlp = Dropout(dropout_rate)(mlp)

mlp_output = Dense(1, activation = 'sigmoid', name = 'mlp_output')(mlp) #Output Layer MLP

# define autoencoder model
model   = Model(inputs= input_x,    outputs=[decoder_output,mlp_output])
encoder = Model(inputs= input_x    , outputs=bottleneck)
decoder = Model(inputs= bottleneck, outputs=decoder_output)
MLP     = Model(inputs= merged     , outputs=mlp_output)

# compile model
model.compile(optimizer = tf.keras.optimizers.Adam(learning_rate = 1e-3),
              loss=    {'decoder_output': 'mse', 'mlp_output': 'binary_crossentropy'},
              metrics={'decoder_output': 'mse', 'mlp_output': AUC()},
              sample_weight_mode={'decoder_output': None, 'mlp_output': 'temporal'},
              weighted_metrics={'mlp_output': 'binary_crossentropy'}
             )
```

Encoder

Decoder

Code

```
es = EarlyStopping(monitor = 'val_mlp_output_loss',min_delta = 1e-4, patience = 20,
                                    baseline = None, restore_best_weights = True, verbose = 0)



history = model.fit(

    x = X_train,
    y = {'decoder_output': X_train, 'mlp_output': y_train},
    epochs=200,
    batch_size=100,
    verbose=0,
    shuffle=True,
    validation_split = 0.2,
    callbacks = [ckp, es])
```

Thank you!