Compression of Differential Expression Data with Deep Autoencoders

Tony Kabilan Okeke

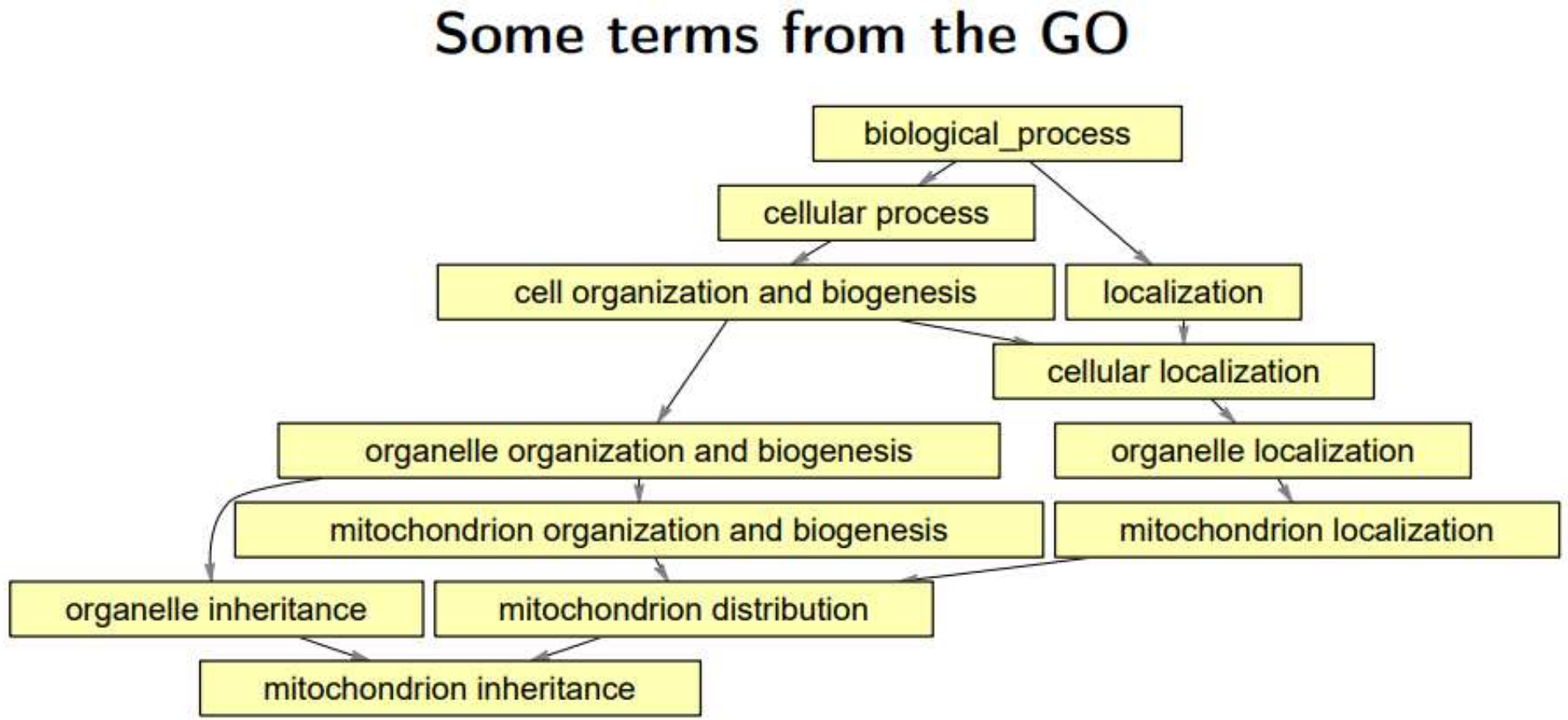
BMES 547: Course Project

Introduction

Problem Description

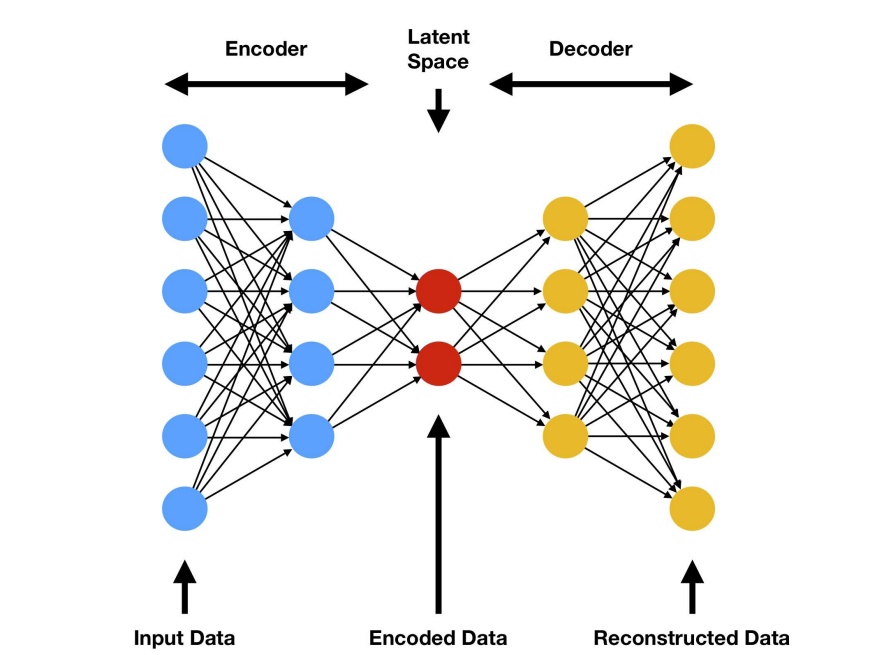
- **Differential Expression Analysis** is used to identify genes that have different levels of expression between >2 samples/conditions.
- **Gene Ontology Enrichment** analyzes functional annotations of differentially expressed genes.
- Extracting biologically significant information from genomic data is a difficult task.
  - The high dimensionality and complexity of genomic data makes it hard to build accurate and interpretable prediction models for protein function
- What data?
  - Microarrays
  - RNA-seq

Problem Description



Background: Autoencoders

- Autoencoders are a type of neural network that are used to learn efficient data encodings in an unsupervised manner.
- Two components
  - Encoder: compress input into latent space representation
  - Decoder: reconstruct original input from latent space
- Trained to minimize reconstruction error



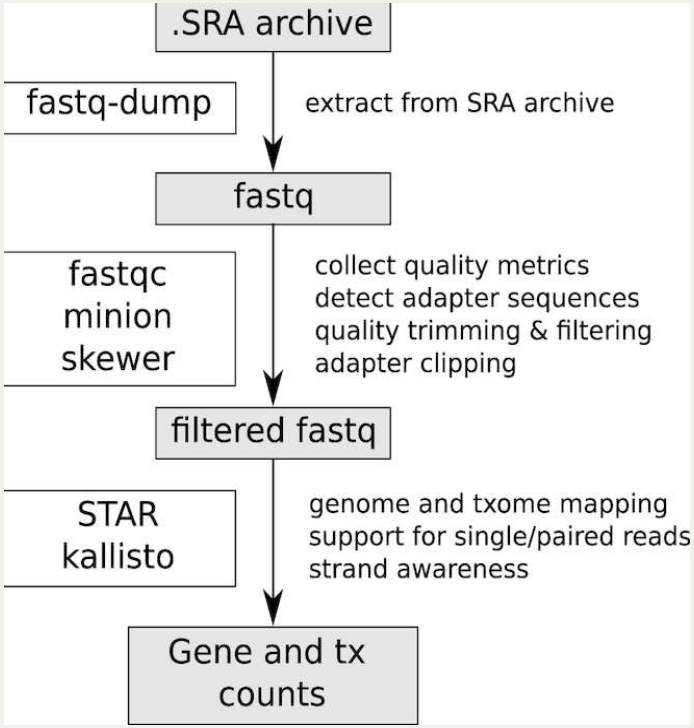
Goals

- Develop a deep autoencoder model to learn a compressed representation of differential expression data (i.e., log2 fold change values).
- Use the latent representation of the autoencoder to predict GO terms.
  - Identify genes responsible for GO term enrichment.
  - Couldn't complete due to time constraints.

The Datasets

Digital Expression Explorer 2

- Repository of uniformly processed RNA-seq data mined from public data obtained from the Sequence Read Archive.
- DEE2 provides 532711 processed RNA-seq runs for Mouse genes and 475547 processed RNA-seq runs for Human genes.



The Dataset

- Leveraged DEE2 to curate 11130 unique datasets with corresponding GEO series including over 350,000 individual sequencing runs (SRRs) from experiments on *Mus musculus*.
- Of these, 7130 datasets met the inclusion criteria for our analysis.
  - These datasets included 48,978 different gene transcripts.
- Differential expression anaysis was performed on each dataset and the resulting log2 fold change values were stored.
  - GO enrichment was then performed and enriched GO terms were stored.

Methods

Methods: Data Curation

- Filtered out datasets with no GEO accession - no metadata
- Analyzed GEO metadata to select datasets that include >1 groups
- Differential Expression & GO enrichment
  - Kept GO terms with \(p < 0.05\)
- Combined datasets into a single feature matrix
- One-hot encoded GO terms
  - 11,130 unique GO terms
- Train/Test/Validation split
- Data standardized

```r
deg <- limma::lmFit(counts, design) %>%
  limma::contrasts.fit(contrast) %>%
  limma::eBayes() %>%
  limma::topTable(number = Inf)

go <- GOfuncR::go_enrich(
    deg[c("symbol", "is_candidate")],
    organismDb = "org.Mm.eg.db",
    n_randsets = 100,
    silent = TRUE
  )$results
```

```python
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
X_val = scaler.transform(X_val)
```

Methods: Dimension Reduction

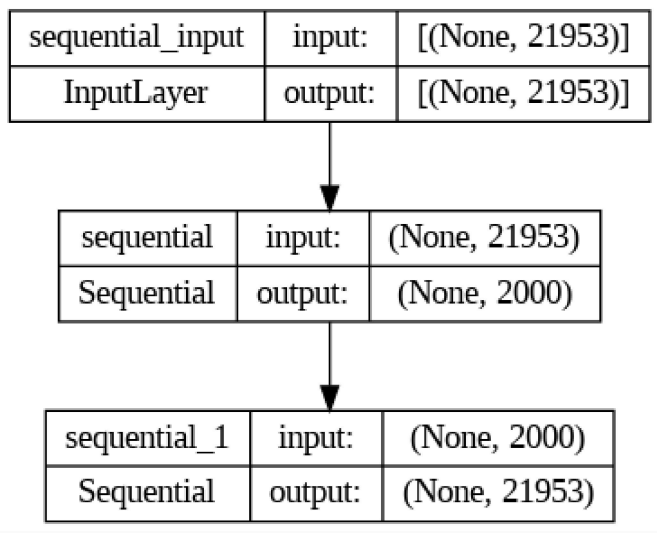
- `VarianceThreshold`
  - Several genes had very low variances across all datasets which impacted the performance of PCA.
- `PCA`
- `t-SNE`
  - Non-linear D.R. technique for high-dimensional data
  - Used in RNA-seq analysis to identify clusters of similar genes/cells

```
sel = VarianceThreshold(threshold=1)
sel.fit(X)

X_train = X_train[:, sel.get_support()]
X_val = X_val[:, sel.get_support()]

pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

tsne = TSNE(n_components=2, perplexity=50, n_
X_tsne = tsne.fit_transform(X)
```

Methods: Autoencoder

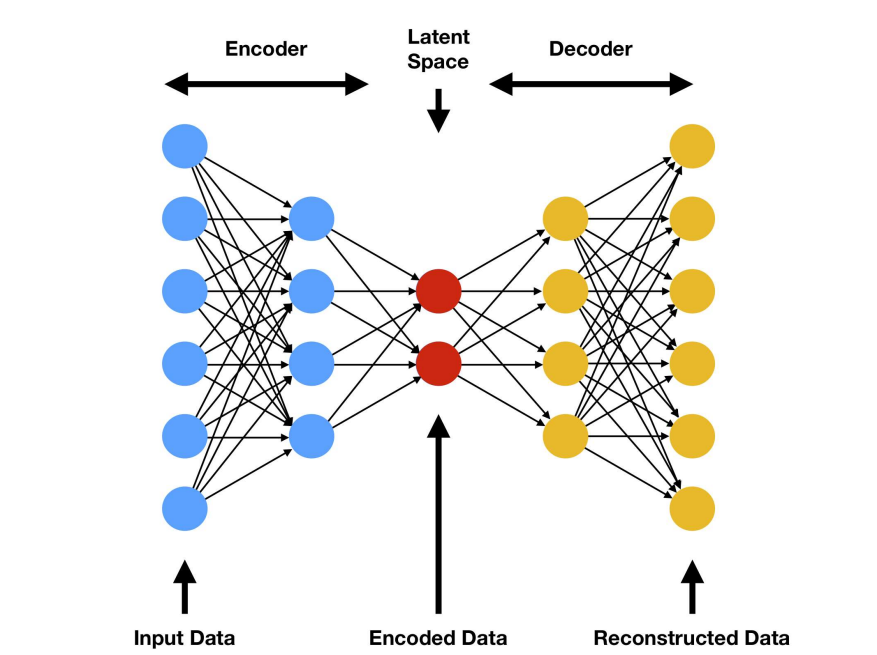
- Implemented the NN using `Keras` with `TensorFlow` backend
  - Used `Adam` optimizer
  - `MSE` loss function
  - `ReLU` activation
- Trained for 50 epochs
  - Evaluated `validation loss`
- `ReduceLROnPlateau` callback
  - Reduce learning rate when `val_loss` stops improving

```
encoder = Sequential()
for units in layer_sizes:
  encoder.add(Dense(units, activation="relu"))

decoder = Sequential()
for units in reversed(layer_sizes[:-1]):
  decoder.add(Dense(units, activation="relu"))
decoder.add(Dense(input_dim, activation="sigmoid"))

autoencoder = Sequential([encoder, decoder])
```



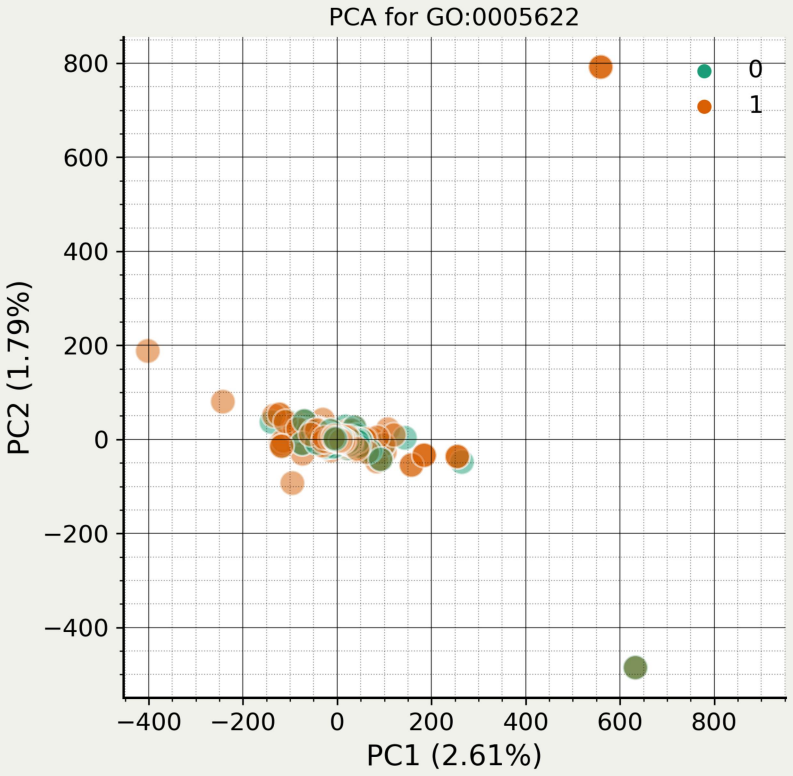
Methods: Optimization

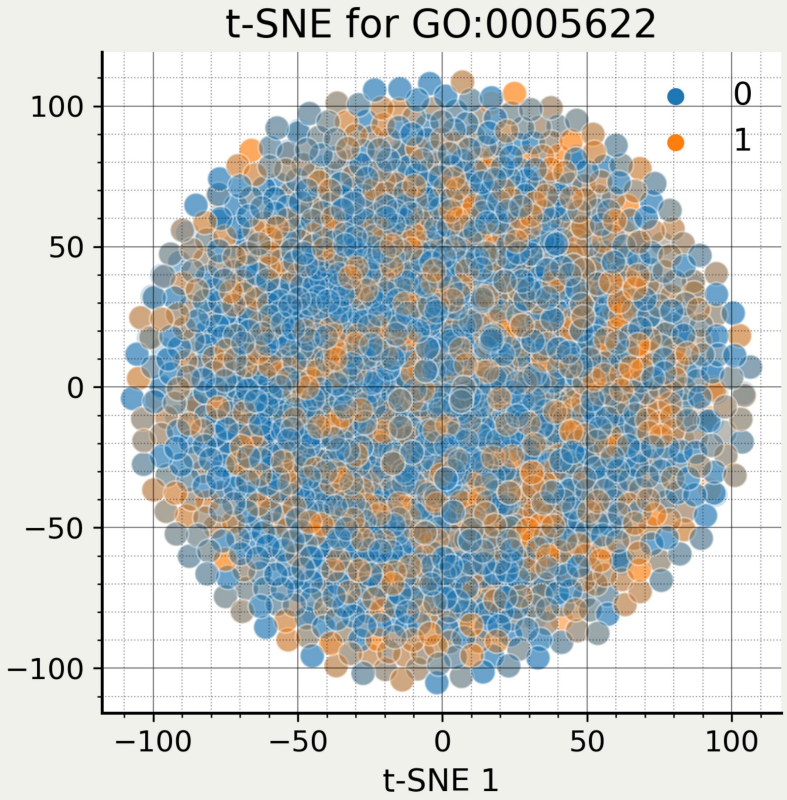
- How does total *Number of Neurons* affect performance?
- How does *Number of Hidden Layers* affect performance?
  - Distributed the neurons so that each layer has ~half the neurons of the previous layer
- Used *Random Search* to find optimal hyperparameters
  - `n_neurons` = 1, 10, 100, 200, 400, 600, 800, 1000, 2000
  - `n_layers` = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10



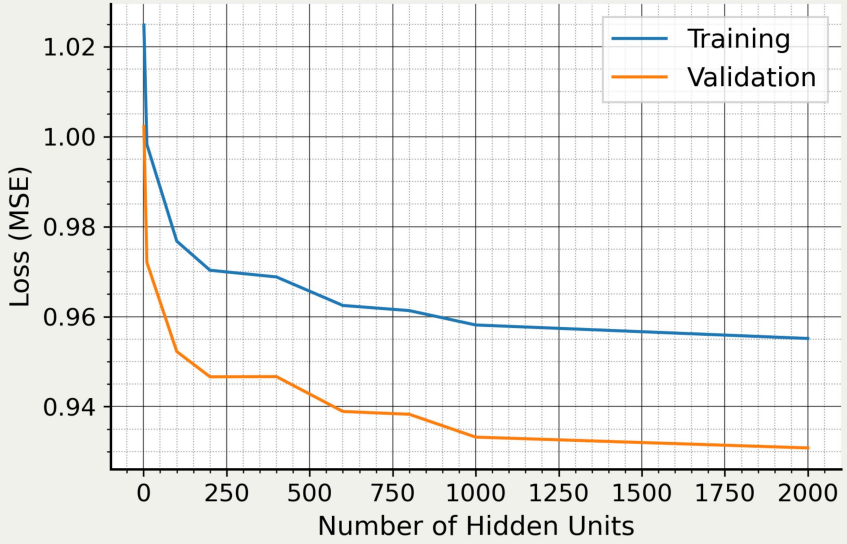
Results

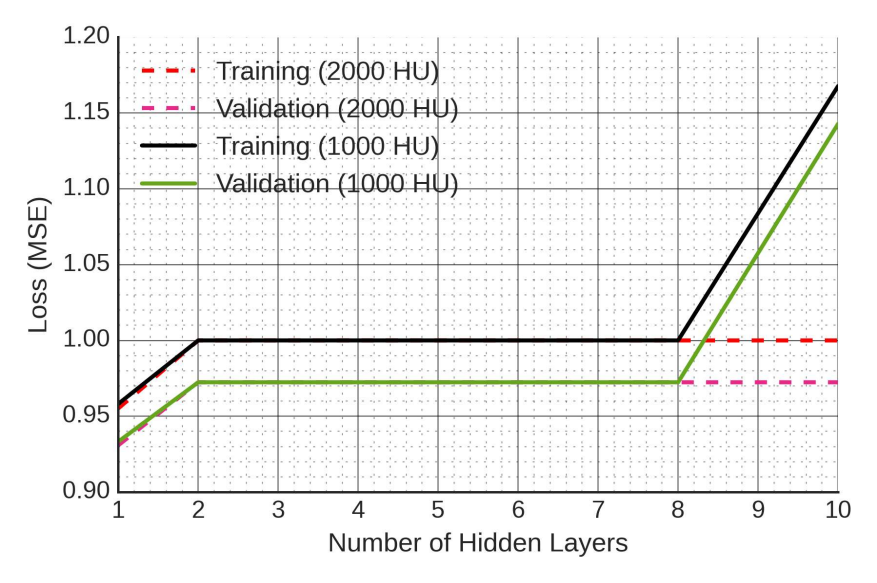
Results: Dimension Reduction





Results: Optimization & Performance





With Best Hyperparameters

Cross Validation Loss: \(0.9632\)

Discussion

- `Limitations`
  - Was only able to use 21953/48978 genes due to memory constraints with GPU training
  - Could not evaluate more hyperparameters
  - Was not able to evaluate performance on GO term prediction
- `Future Work`
  - Improve the performance of the autoencoder; try different architectures such as sparse and variational autoencoders
  - Find a larger / more balanced dataset

Thank You

References

- Gene Ontology
- Digital Expression Explorer 2
- Autoencoders

Cross-Validation Loss: 0.9632