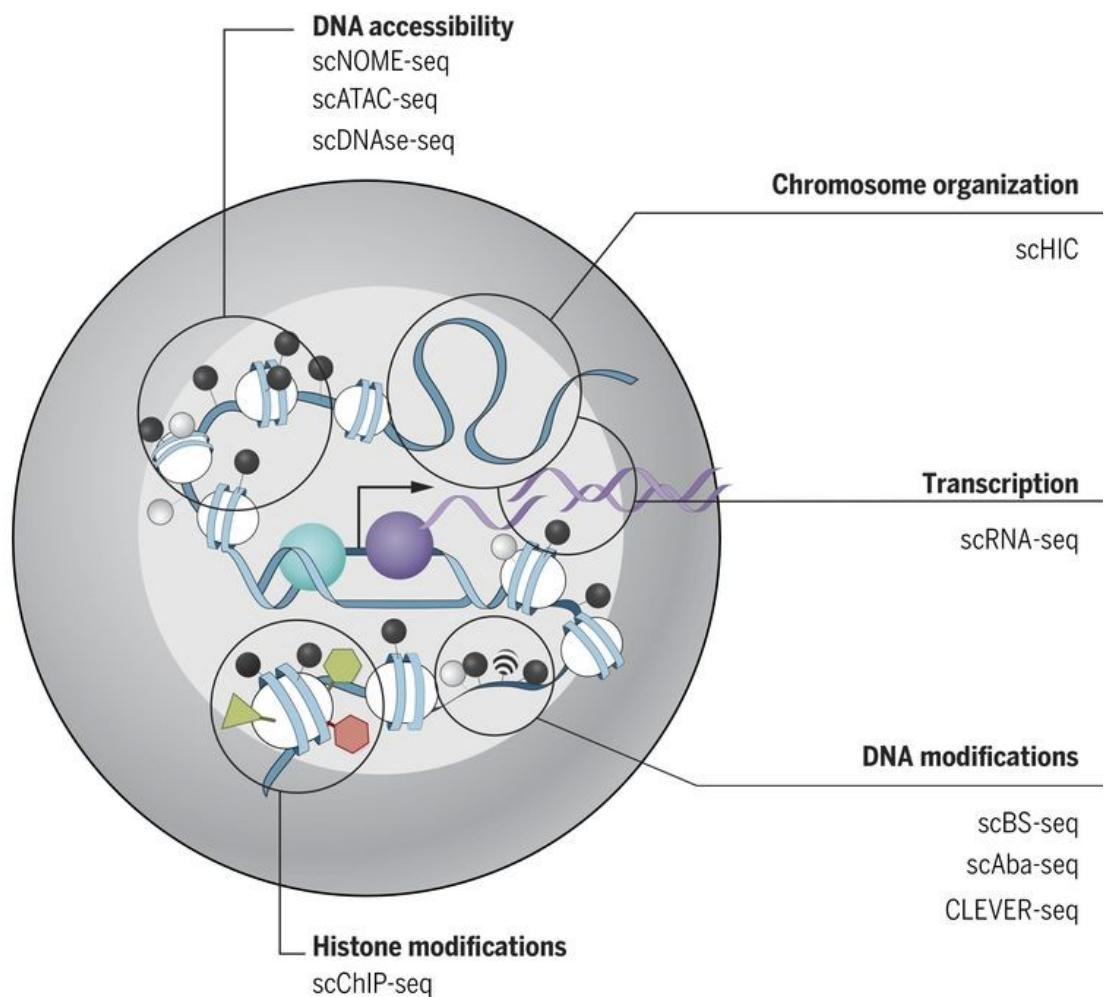
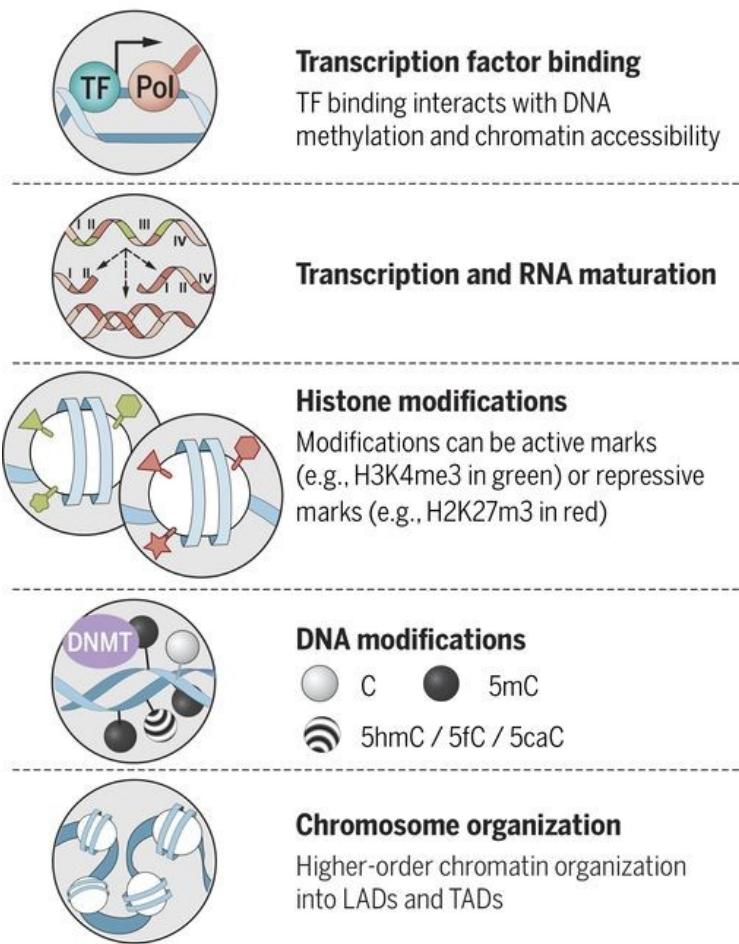


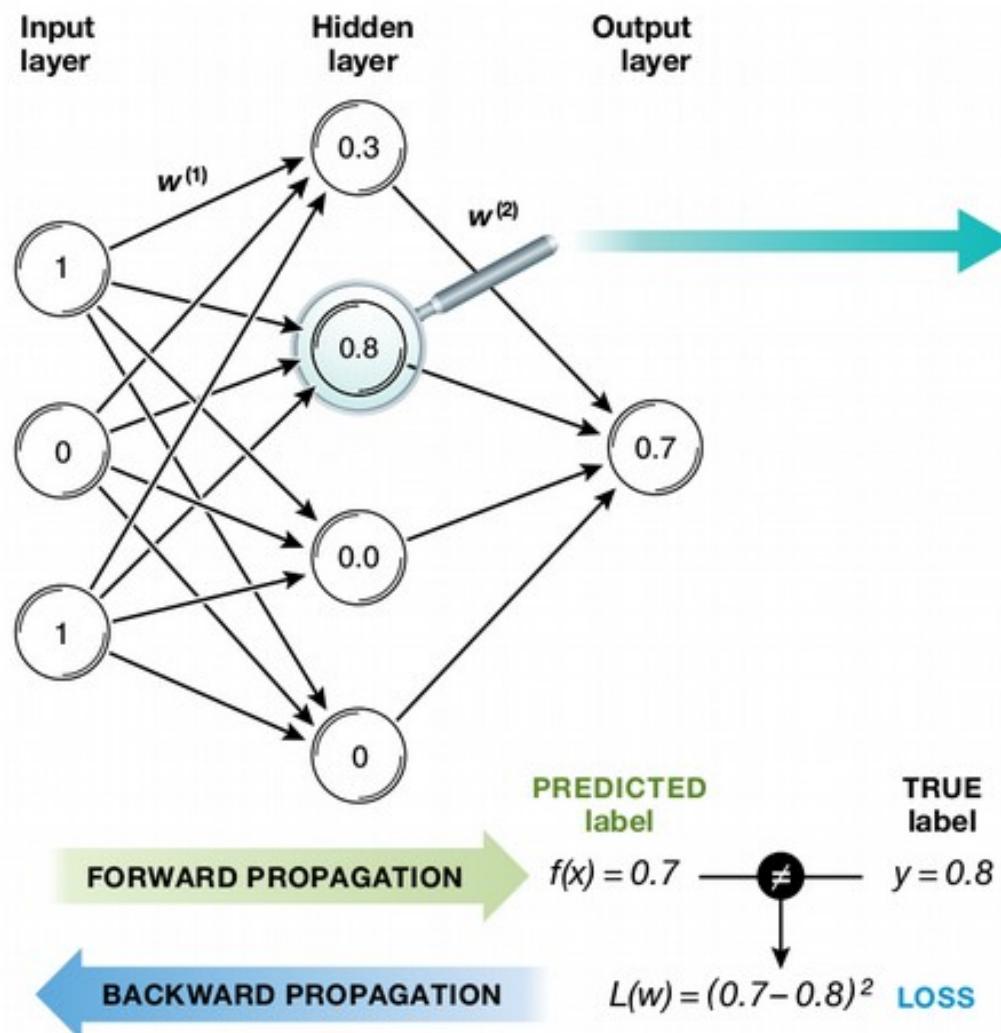
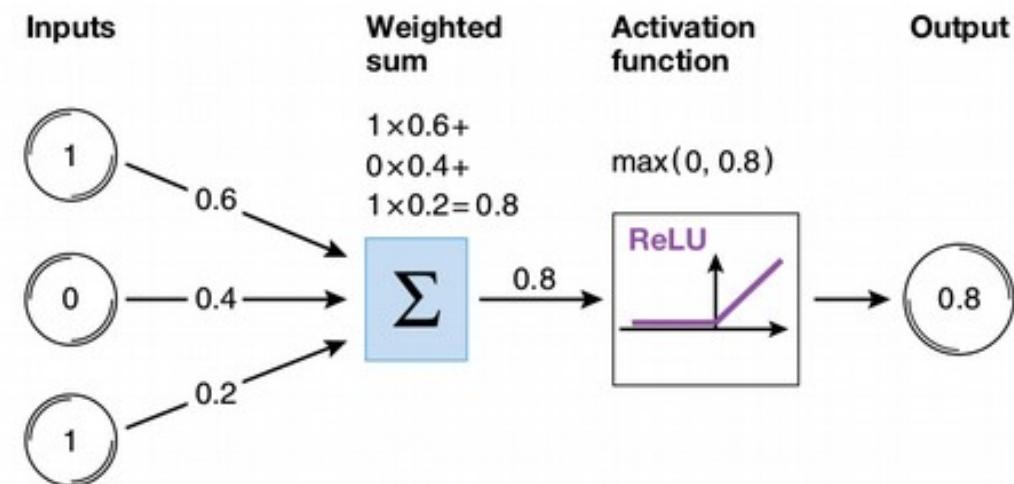
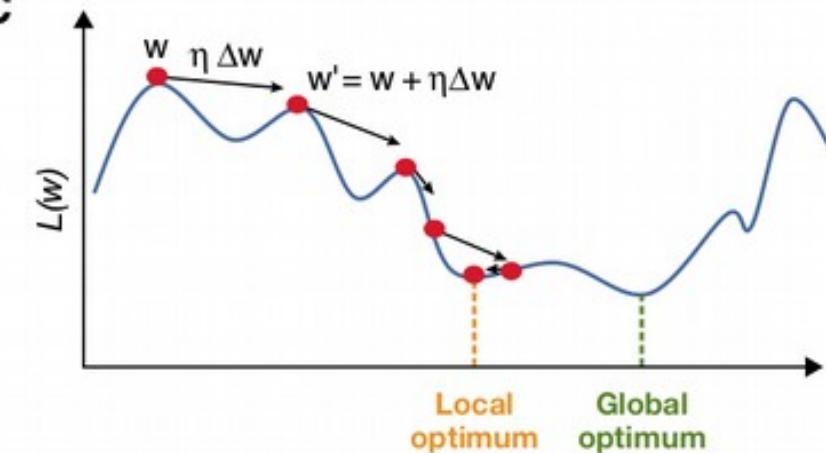
Deep Learning for OMICs Integration

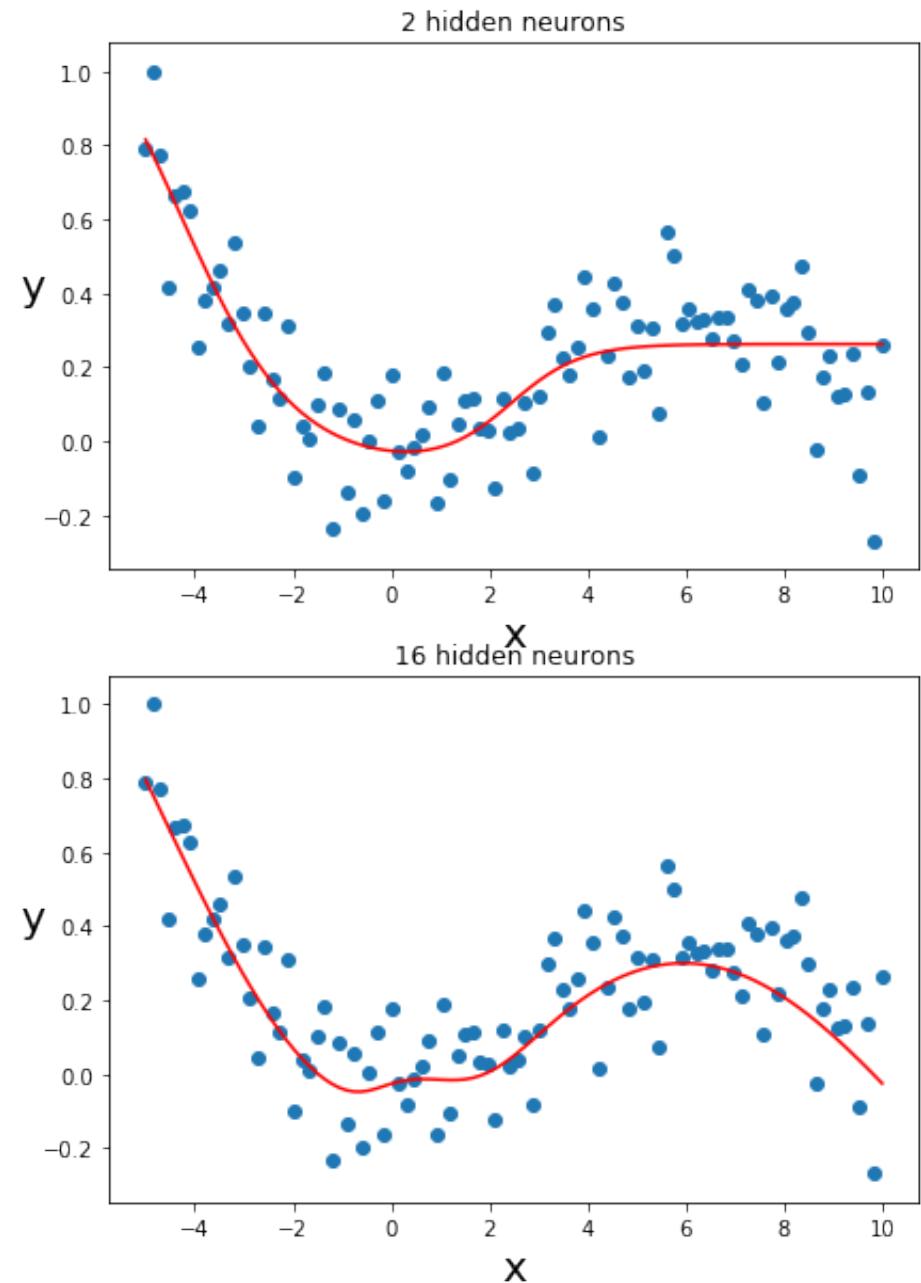
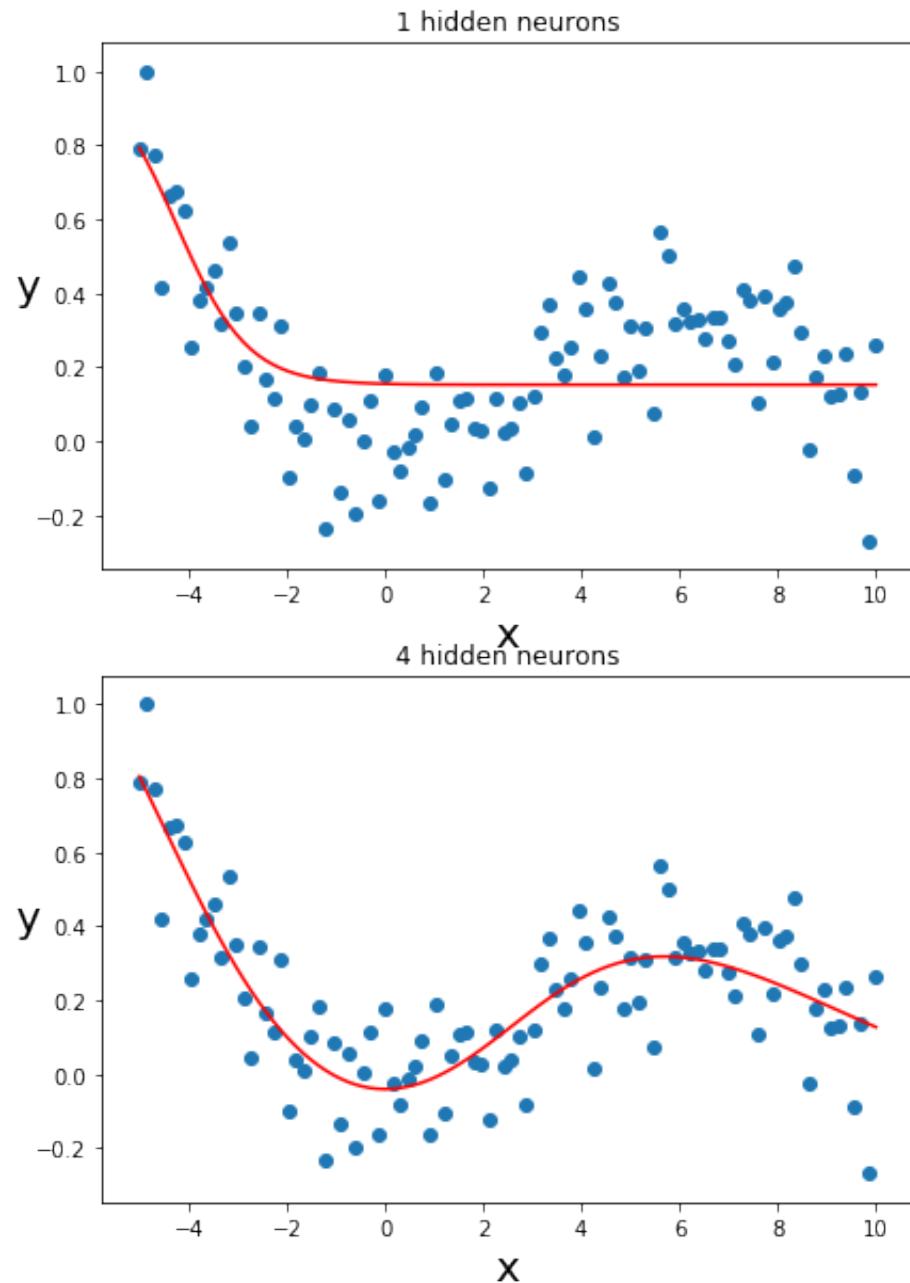
OMICs Integration and Systems Biology course

Nikolay Oskolkov, NBIS SciLifeLab

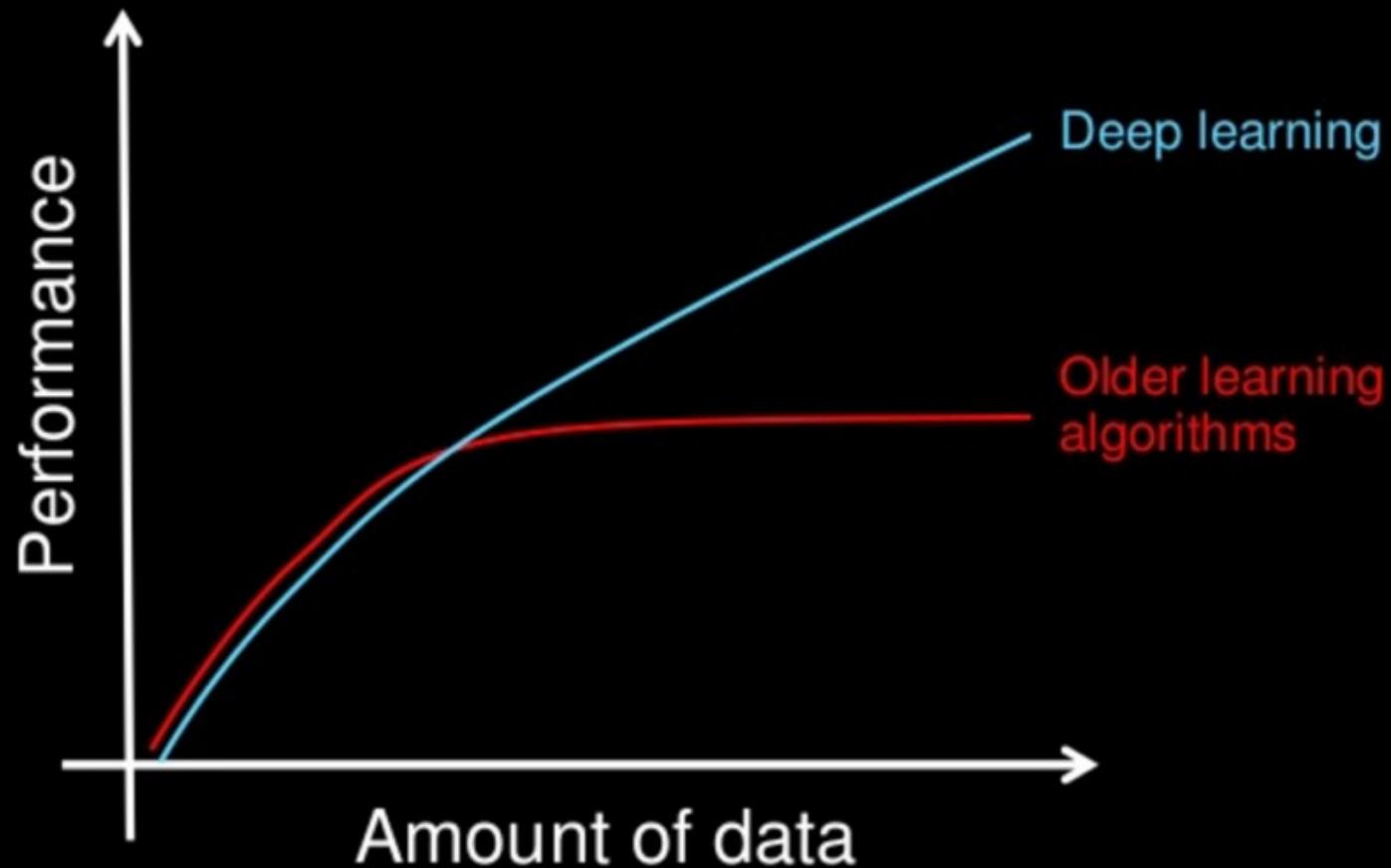
Lund, 5.10.2020



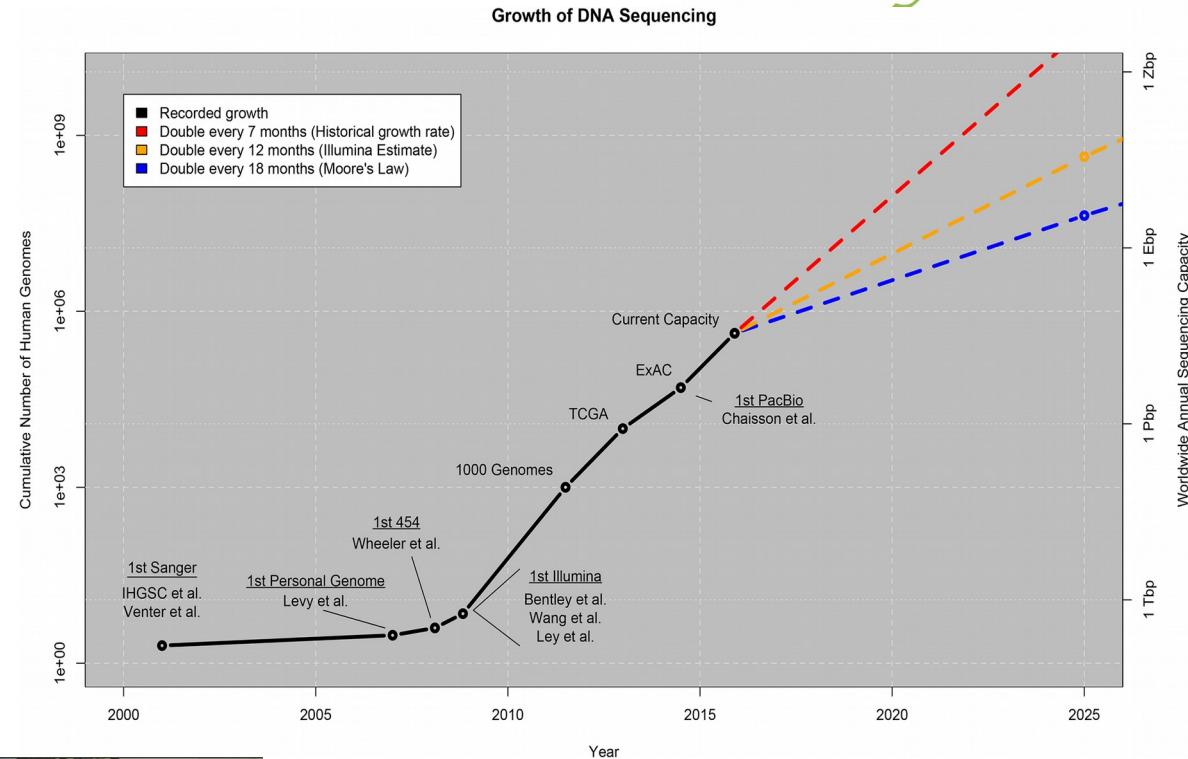
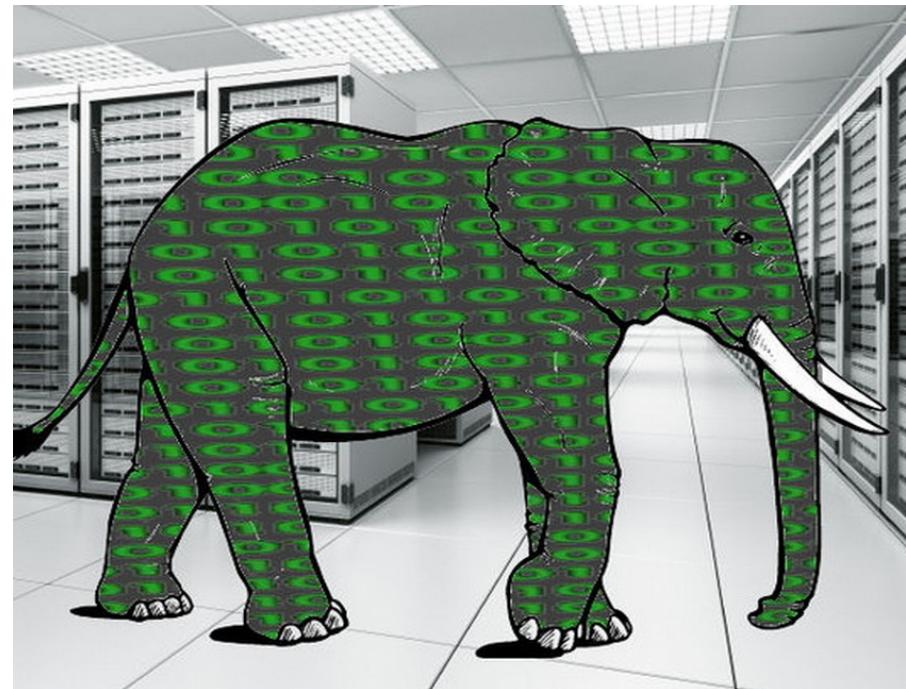
A**B****C**



Why deep learning

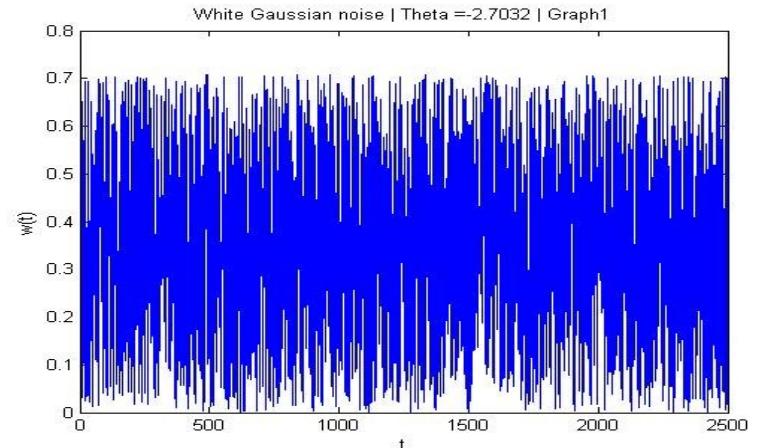


How do data science techniques scale with amount of data?



I have 500 TB of data on my disk, this is big.

I have Big Data, I want to run Deep Learning on my Big Data

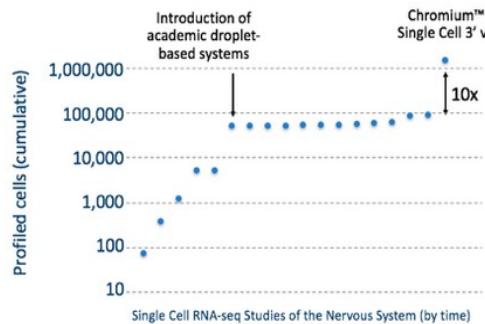


CAREERS BLOG 10X UNIVERSITY

10X GENOMICS SOLUTIONS & PRODUCTS RESEARCH & APPLICATIONS EDUCATION & RESOURCES

< Back to Blog

< Newer Article Older Article >



Our 1.3 million single cell dataset is ready 0 KUDOS



POSTED BY: grace-10x, on Feb 21, 2017 at 2:28 PM

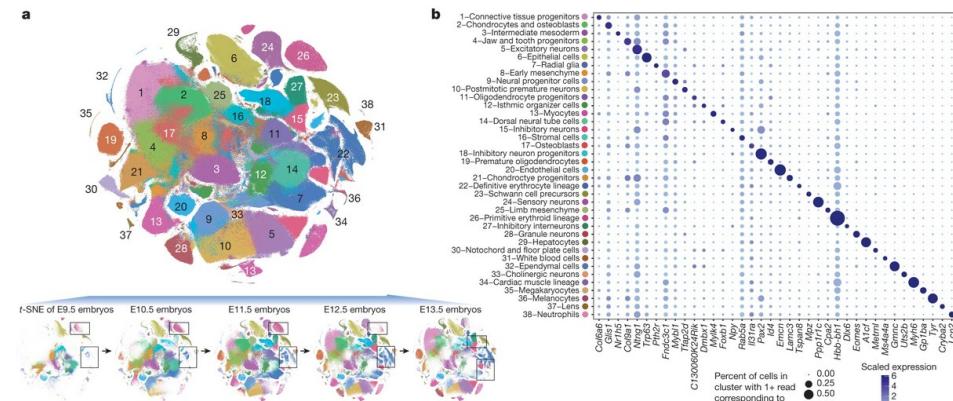
At ASHG last year, we announced our 1.3 Million Brain Cell Dataset, which is, to date, the largest dataset published in the single cell RNA-sequencing (scRNA-seq) field. Using the Chromium™ Single Cell 3' Solution (v2 Chemistry), we were able to sequence and profile 1,308,421 individual cells from embryonic mice brains. Read more in our application note [Transcriptional Profiling of 1.3 Million Brain Cells with the Chromium™ Single Cell 3' Solution](#).

**Watch out Underfitting!
Paradise for Deep Learning!**

MENU nature

Fig. 2: Identifying the major cell types of mouse organogenesis.

From: [The single-cell transcriptional landscape of mammalian organogenesis](#)



a, t-SNE visualization of 2,026,641 mouse embryo cells (after removing a putative doublet cluster), coloured by cluster identity (ID) from Louvain clustering (in **b**), and annotated on the basis of marker genes. The same t-SNE is plotted below, showing only cells from each stage (cell numbers from left to right: n = 151,000 for E9.5; 370,279 for E10.5; 602,784 for E11.5; 468,088 for E12.5; 434,490 for E13.5). Primitive erythroid (transient) and definitive erythroid (expanding) clusters are boxed. **b**, Dot plot showing expression of one selected marker gene per cell type. The size of the dot encodes the percentage of cells within a cell type in

BioTuring™ Solutions Resources

Explore **4,000,000 CELLS** at ease with **Bioturing Browser**
A next-generation platform to re-analyze published single-cell sequencing data

EXPLORER NOW

Single Cell Analysis

5,500,000 cells will be indexed into BioTuring Single-cell Data Repository this September

by biomembers • August 30, 2019

Human Cell Atlas, single-cell data

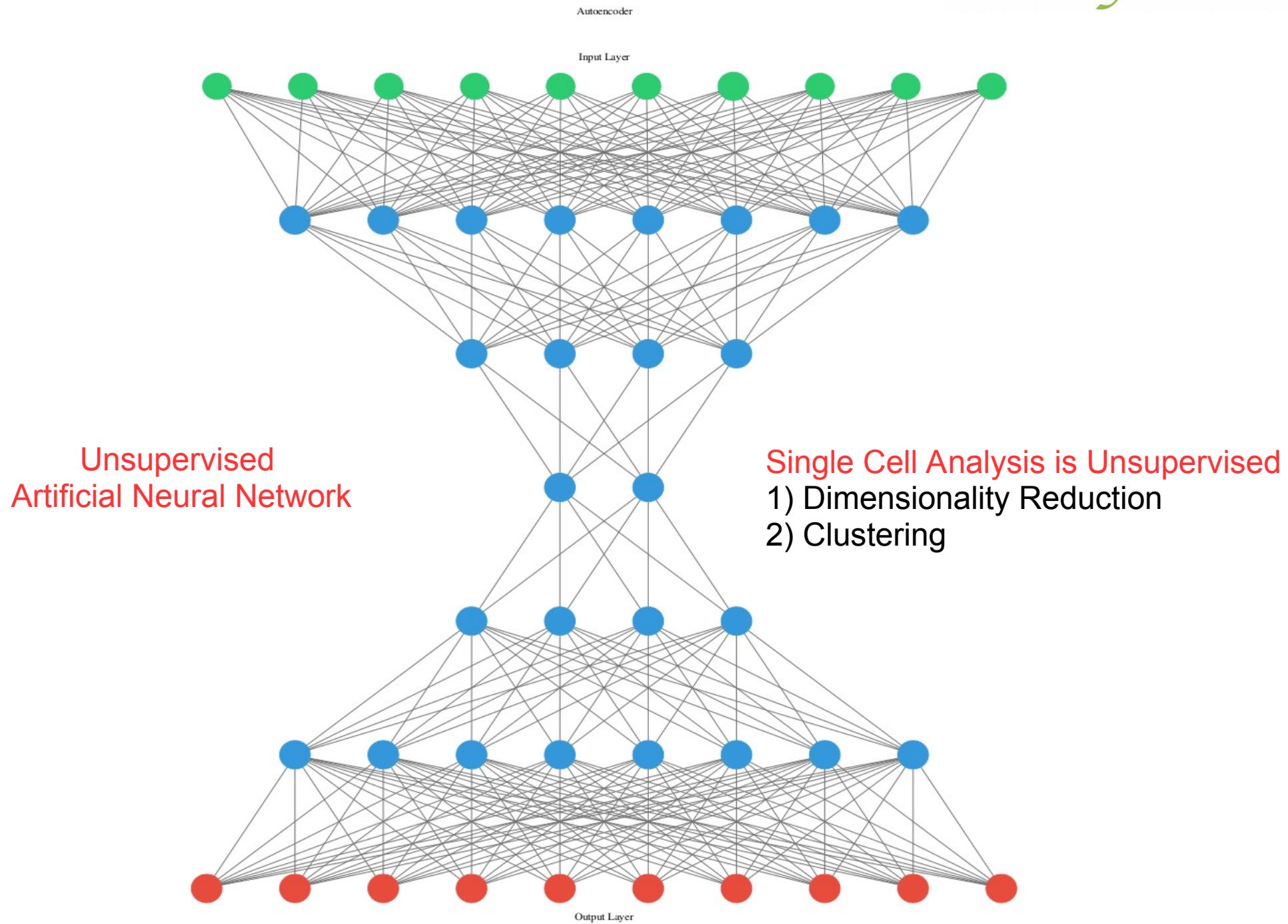
We are glad to announce that we will upsize the current single-cell database in **BioTuring Single-cell Browser** to 5,500,000 cells this September. With this release, we will double the current number of publications indexed in BioTuring Single-cell Browser, and cross the number of cells hosted on available public single-cell data repositories like [Human Cell Atlas \(HCA\)](#) and [Broad Institute's Single-cell Portal](#).

Search

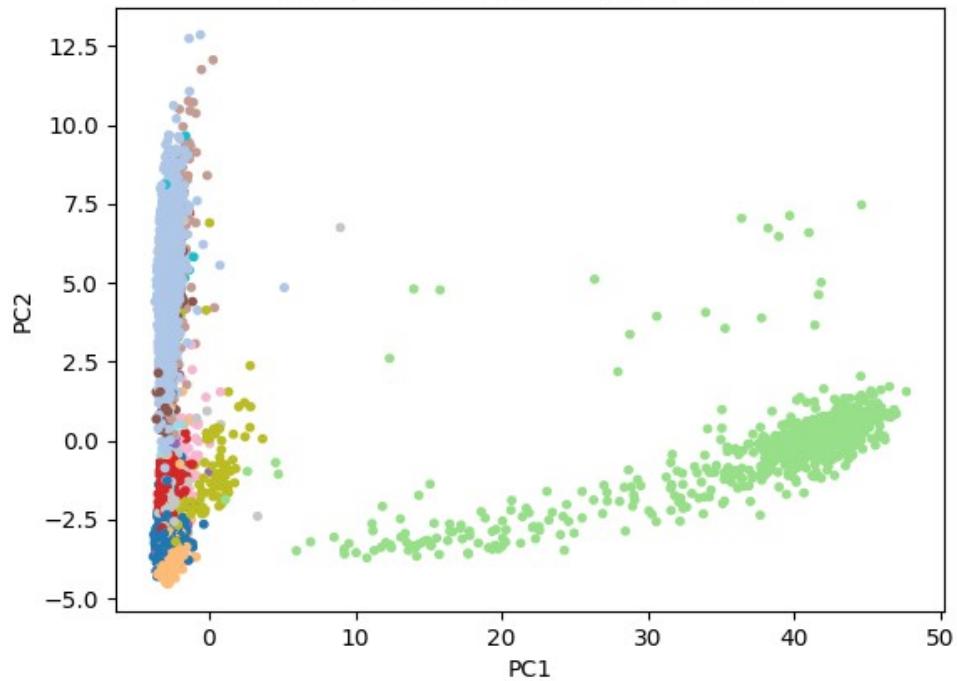
RECENT POSTS

A new tool to interactively visualize single-cell objects (Seurat, Scanpy, SingleCellExperiments, ...)
September 26, 2019

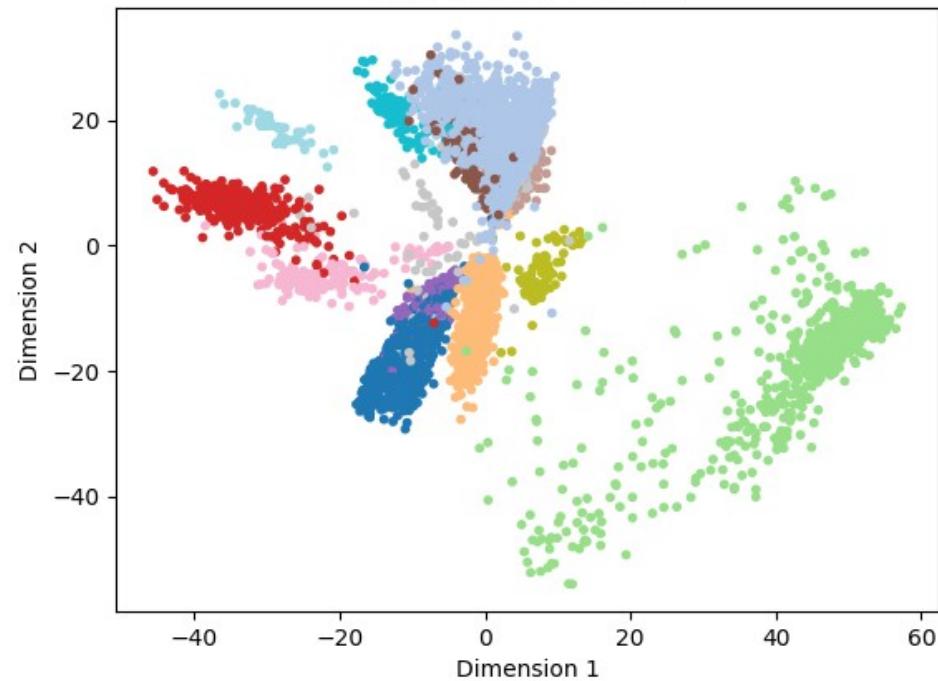
5,500,000 cells will be indexed into BioTuring Single-cell Data Repository this September
August 30, 2019



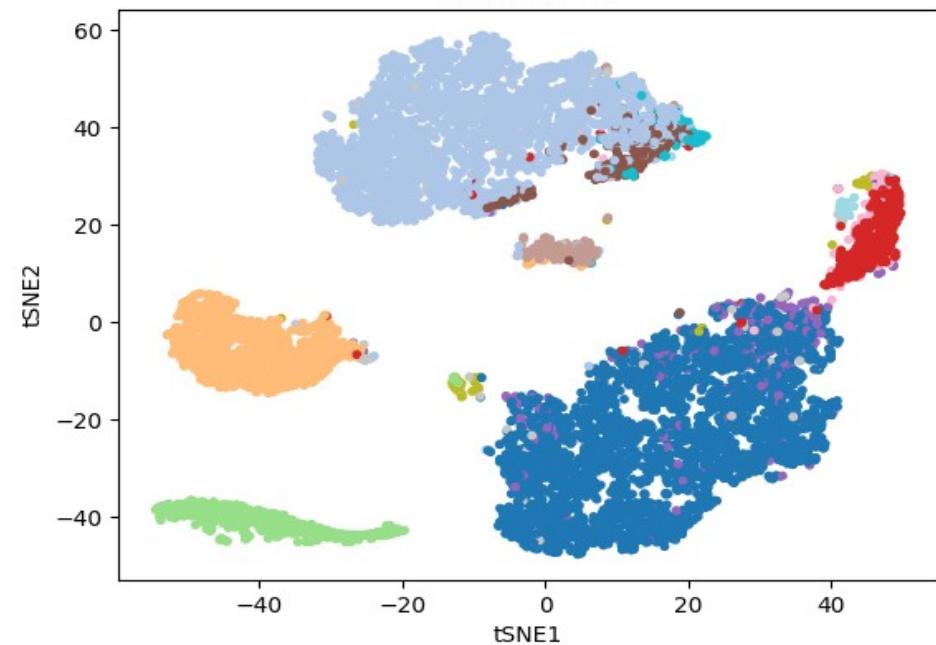
Principal Component Analysis (PCA)



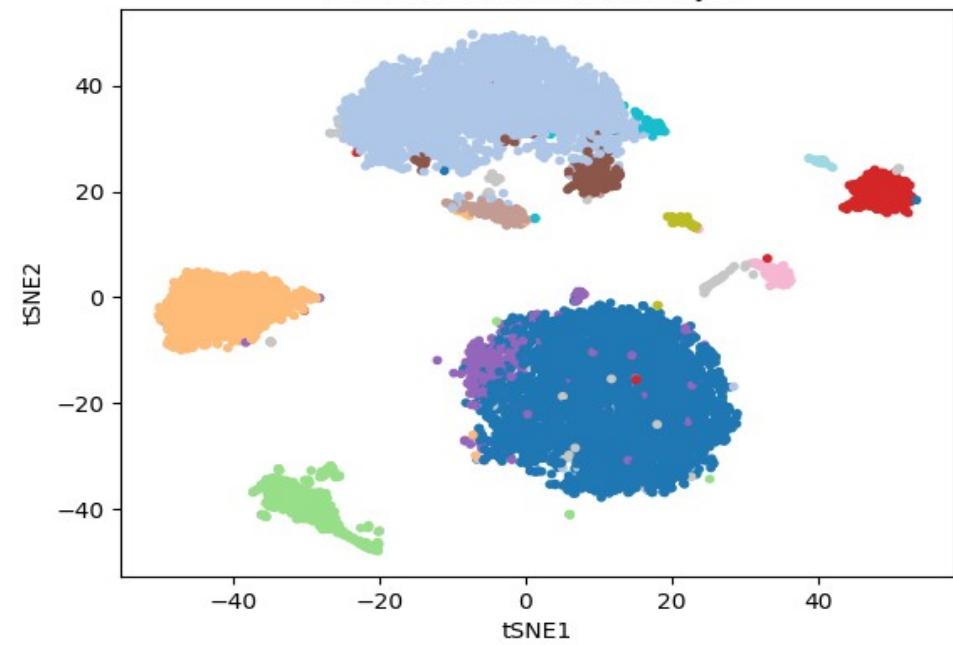
Autoencoder: 8 Layers



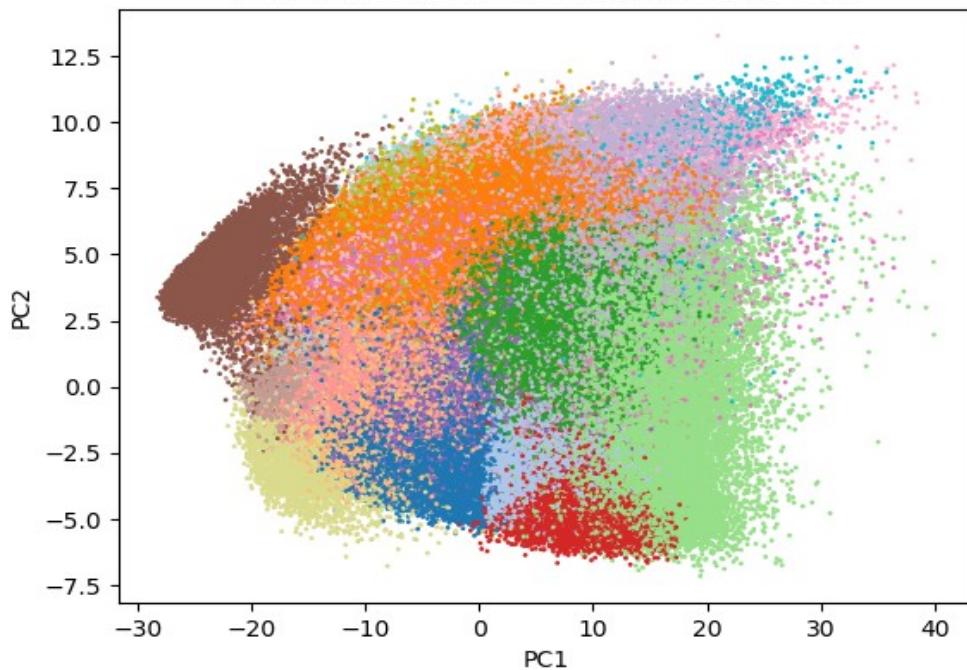
tSNE on PCA



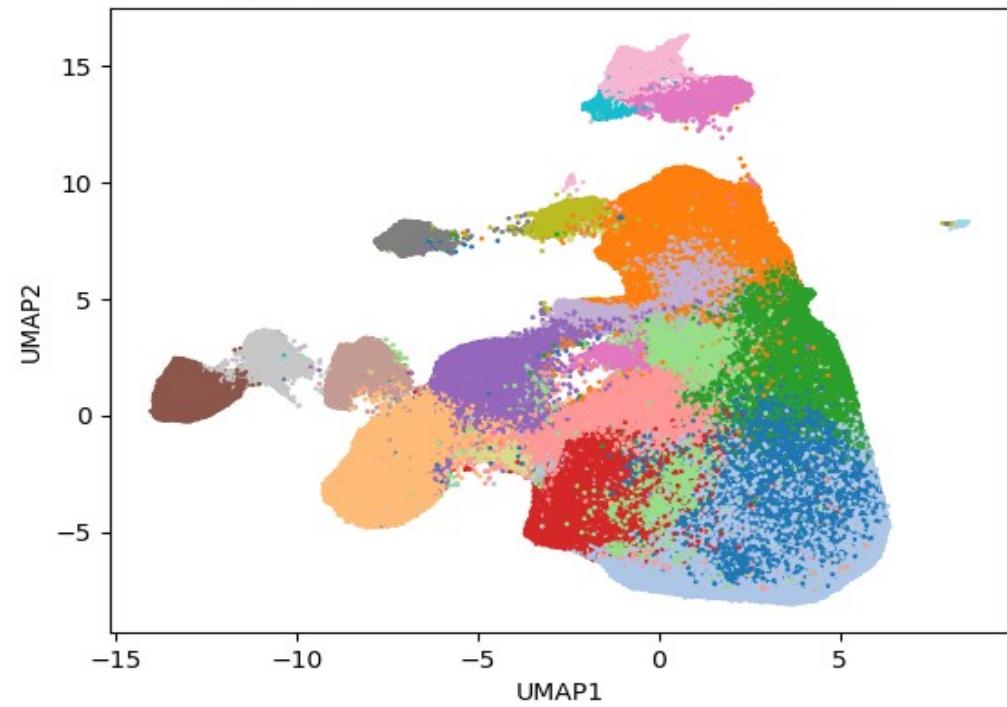
tSNE on Autoencoder: 8 Layers



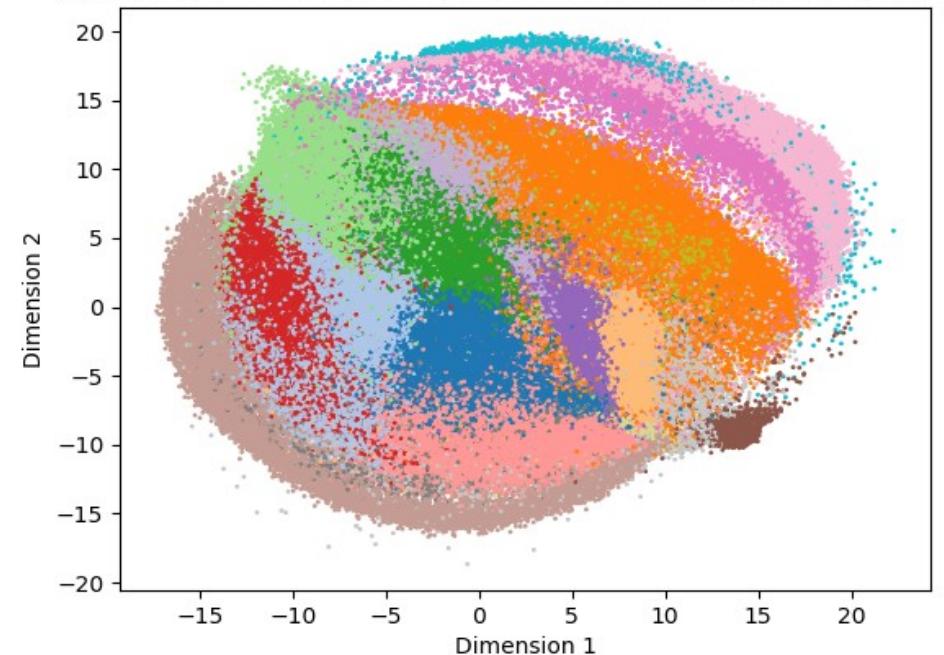
PCA, 10X Genomics 1.3M Mouse Brain Cells



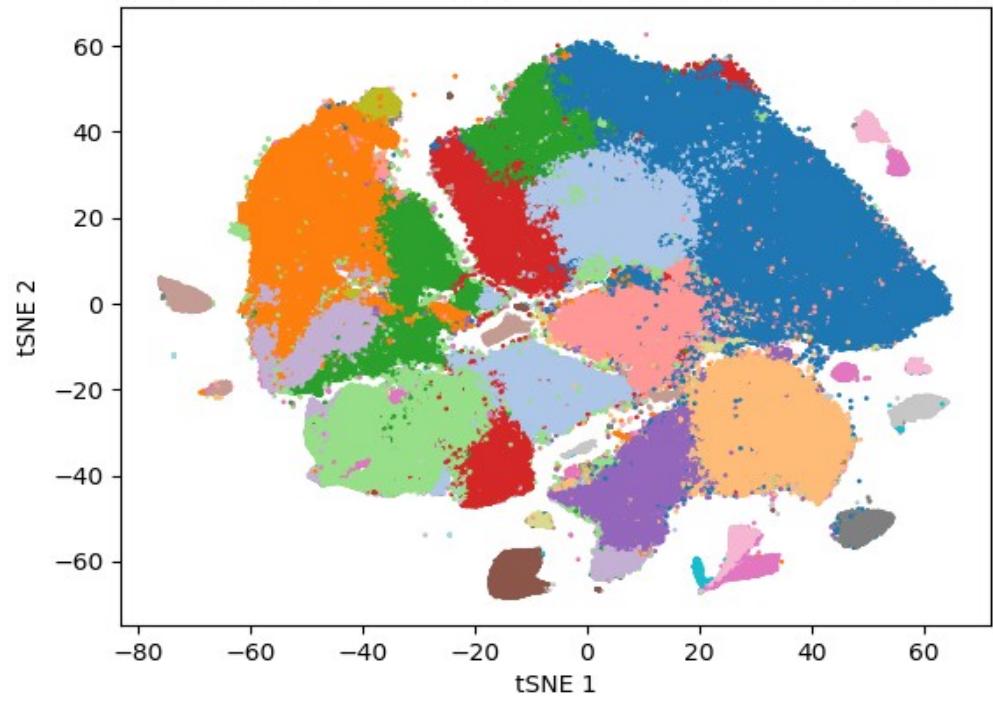
UMAP 10X Genomics 1.3M Mouse Brain cells

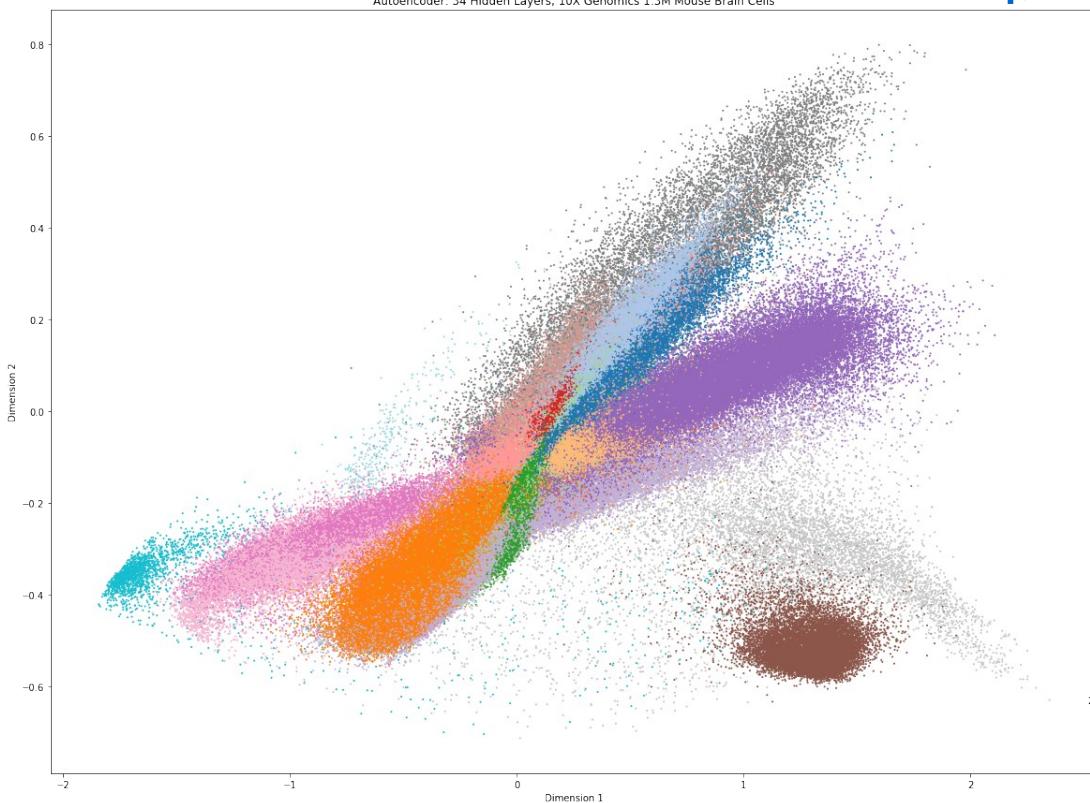


Autoencoder 10 Hiden Layers, 10X Genomics 1.3M Mouse Brain cells



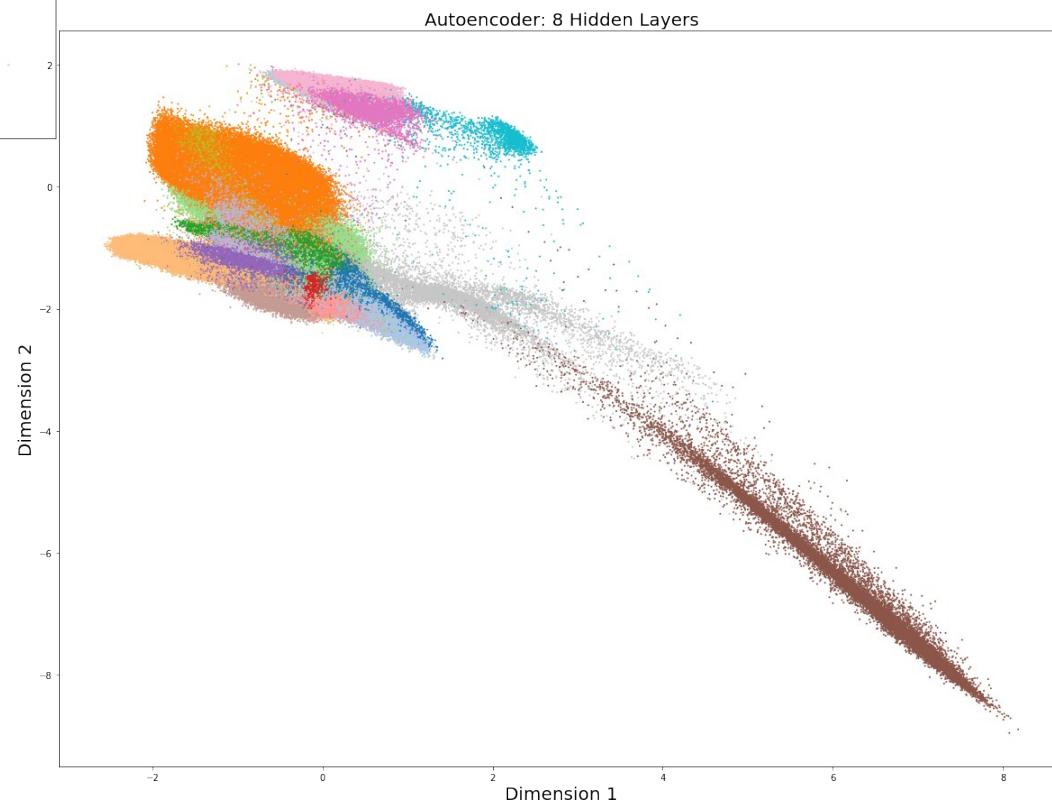
tSNE perplexity = 350, 10X Genomics 1.3M Mouse Brain cells

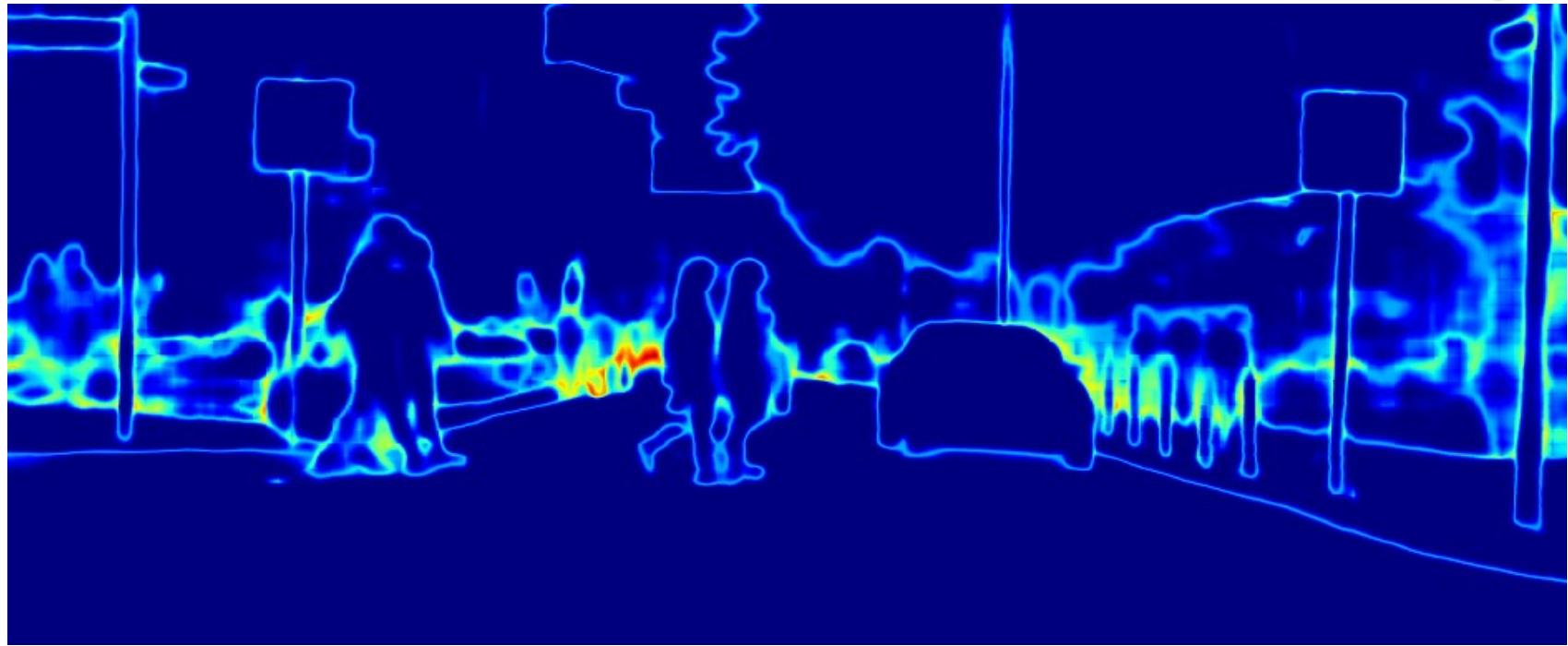




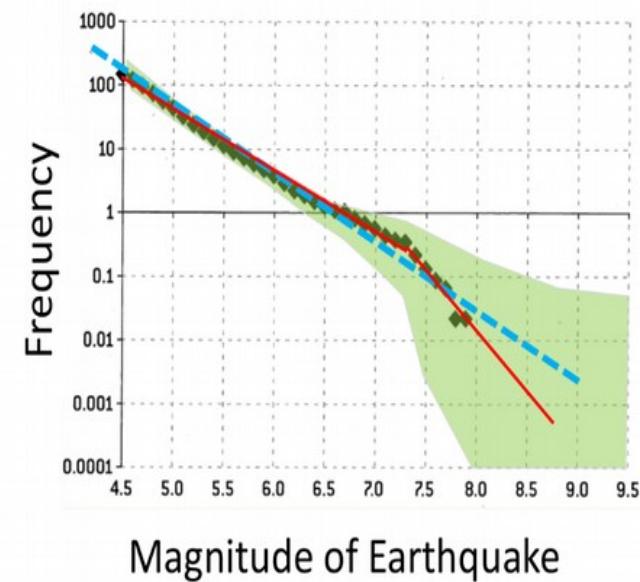
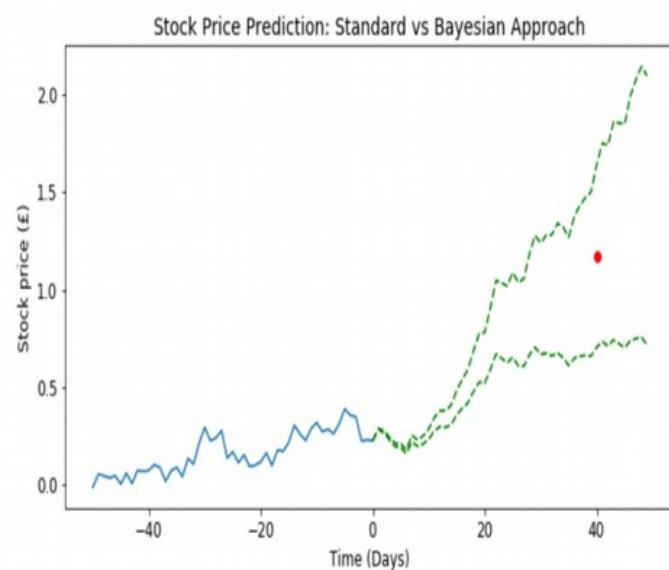
Autoencoders are good for non-linear pre- dimension reduction, the bottleneck can be fed to tSNE / UMAP

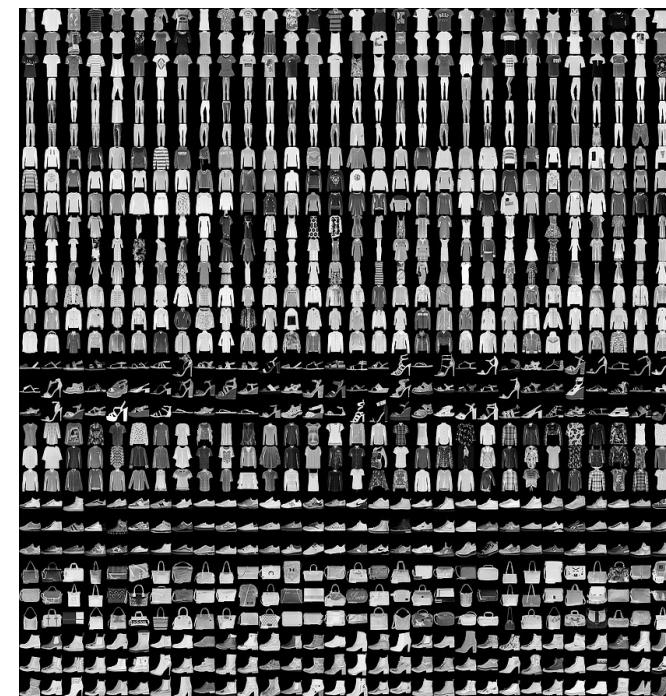
Autoencoder itself perhaps is not that great for visualization of scOmics





Intelligence is to know how much you do not know





```
In [24]: # normalize inputs from 0-255 to 0-1.0
X_train = X_train.reshape(X_train.shape[0], 1, 28, 28).astype('float32')
X_test = X_test.reshape(X_test.shape[0], 1, 28, 28).astype('float32')
X_train = X_train / 255.0
X_test = X_test / 255.0

In [25]: # one hot encode outputs
y_train = np_utils.to_categorical(y_train)
y_test = np_utils.to_categorical(y_test)
num_classes = y_train.shape[1]
print(num_classes)
10

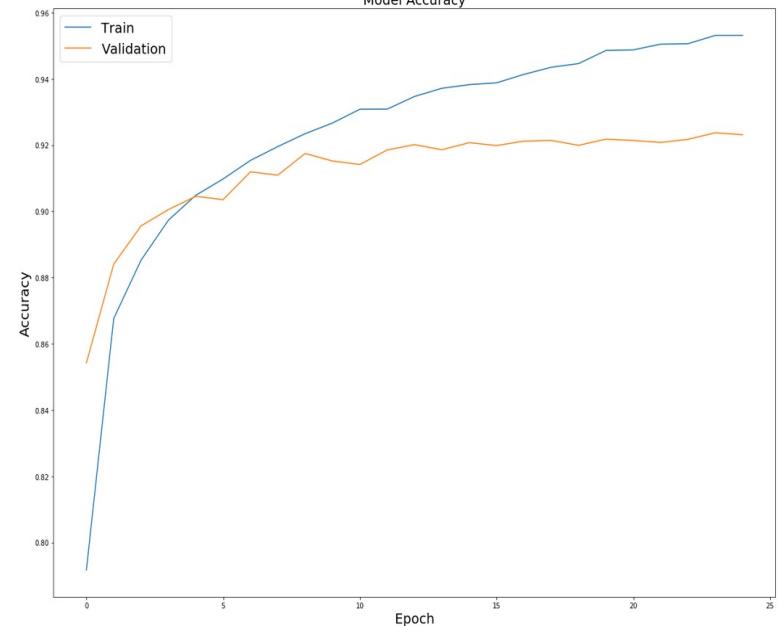
In [27]: # Create the model
model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=(1, 28, 28), padding='same', activation='relu',
               kernel_constraint=maxnorm(3)))
model.add(Dropout(0.2))
model.add(Conv2D(32, (3, 3), padding='same', activation='relu',
               kernel_constraint=maxnorm(3)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(512, activation='relu', kernel_constraint=maxnorm(3)))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

# Compile model
epochs = 25
lr_rate = 0.01
decay = lr_rate/epochs
sgd = SGD(lr=lr_rate, momentum=0.9, decay=decay, nesterov=False)
model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])
print(model.summary())

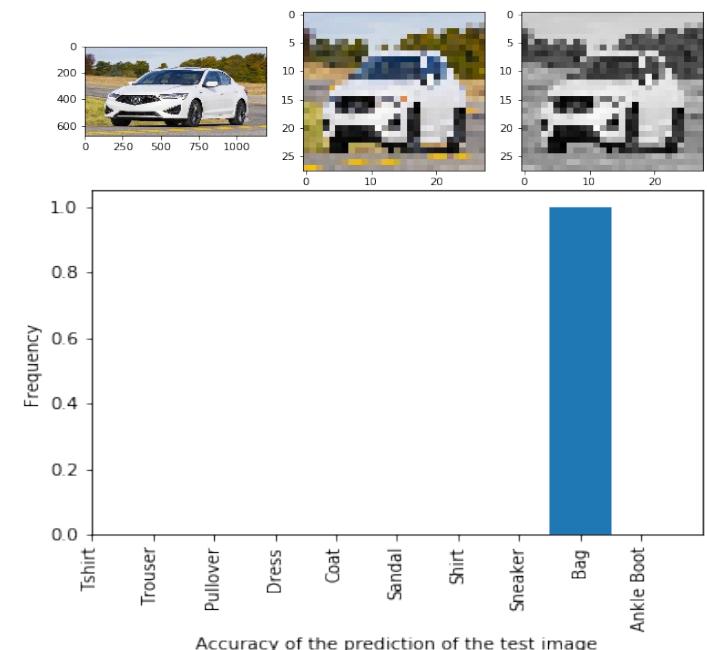
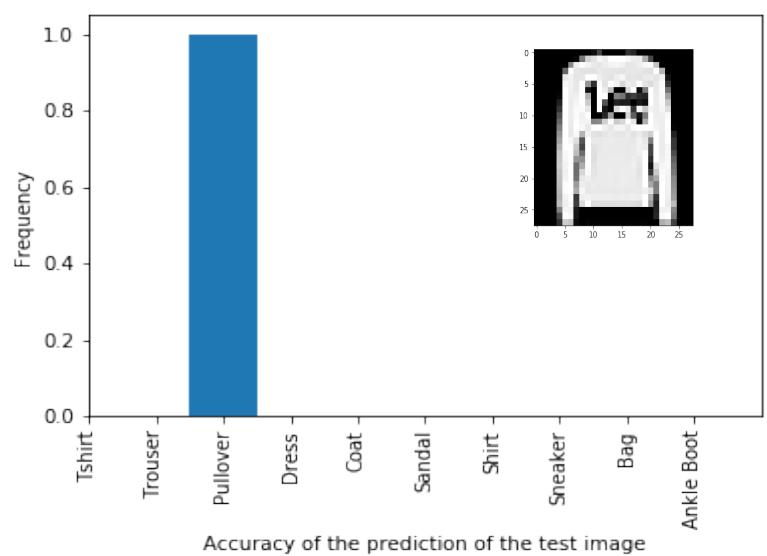
Layer (type) Output Shape Param #
=====conv2d_8 (Conv2D) (None, 32, 28, 28) 320
dropout_7 (Dropout) (None, 32, 28, 28) 0
conv2d_9 (Conv2D) (None, 32, 28, 28) 9248
max_pooling2d_4 (MaxPooling2D) (None, 32, 14, 14) 0
flatten_4 (Flatten) (None, 6272) 0
dense_7 (Dense) (None, 512) 3211776
dropout_8 (Dropout) (None, 512) 0
dense_8 (Dense) (None, 10) 5130
=====
Total params: 3,226,474
Trainable params: 3,226,474
Non-trainable params: 0
None

In [28]: # Fit the model
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=epochs, batch_size=32,
          history = model.fit(X_train, y_train, epochs=epochs, verbose = 1, validation_split = 0.25,
          batch_size = 32, shuffle = True)

Train on 45900 samples. validate on 15000 samples
Epoch 1/25
45900/45900 [=====] - 1158s 26ms/step - loss: 0.5762 - acc: 0.7917 - val_loss: 0.39
73 - val_acc: 0.8542
Epoch 2/25
45900/45900 [=====] - 1124s 25ms/step - loss: 0.3643 - acc: 0.8676 - val_loss: 0.31
25 - val_acc: 0.8841
Epoch 3/25
45900/45900 [=====] - 1158s 26ms/step - loss: 0.3129 - acc: 0.8853 - val_loss: 0.28
25 - val_acc: 0.8956
Epoch 4/25
45900/45900 [=====] - 1609s 36ms/step - loss: 0.2813 - acc: 0.8973 - val_loss: 0.27
27 - val_acc: 0.9005
Epoch 5/25
45900/45900 [=====] - 902s 20ms/step - loss: 0.2618 - acc: 0.9048 - val_loss: 0.258
8 - val_acc: 0.9045
Epoch 6/25
45900/45900 [=====] - 936s 21ms/step - loss: 0.2451 - acc: 0.9098 - val_loss: 0.256
4 - val_acc: 0.9035
Epoch 7/25
```



Prediction



PyMC3, Edward, TensorFlow Probability

```
In [8]: x_train = x_train.reshape((x_train.shape[0],D))
x_test = x_test.reshape((x_test.shape[0],D))
print(x_train.shape)
print(x_test.shape)

(60000, 784)
(10000, 784)

In [9]: from keras.utils import to_categorical
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
print(y_train.shape)
print(y_test.shape)

(60000, 10)
(10000, 10)

In [10]: ed.set_seed(314159)
N = 100 # number of images in a minibatch.
D = D # number of features.
K = 10 # number of classes.

# Create a placeholder to hold the data (in minibatches) in a TensorFlow graph.
x = tf.placeholder(tf.float32, [None, D])
# Normal(0,1) priors for the variables. Note that the syntax assumes TensorFlow 1.1.
w = Normal(loc=tf.zeros([D, K]), scale=tf.ones([D, K]))
b = Normal(loc=tf.zeros(K), scale=tf.ones(K))
# Categorical likelihood for classification.
y = Categorical(tf.matmul(x, w) + b)

In [11]: # Construct the q(w) and q(b). In this case we assume Normal distributions.
qw = Normal(loc=tf.Variable(tf.random_normal([D, K])), scale=tf.nn.softplus(tf.Variable(tf.random_normal([D, K]))))
qb = Normal(loc=tf.Variable(tf.random_normal([K])), scale=tf.nn.softplus(tf.Variable(tf.random_normal([K]))))

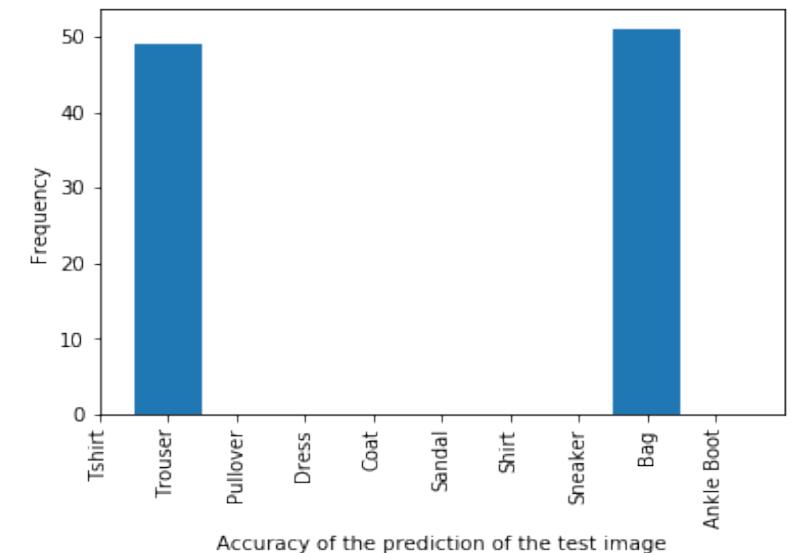
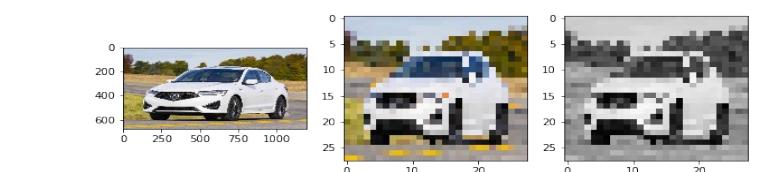
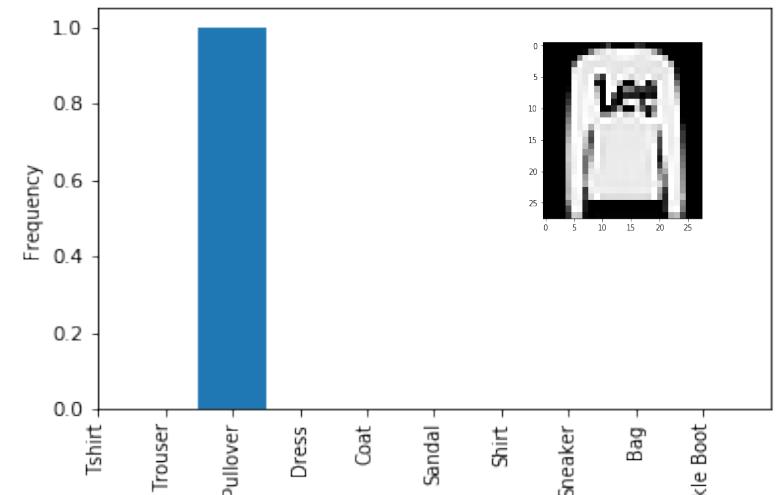
In [12]: def generator(arrays, batch_size = N):
    starts = [0] * len(arrays) # pointers to where we are in iteration
    while True:
        batches = []
        for i, array in enumerate(arrays):
            start = starts[i]
            stop = start + batch_size
            diff = stop - array.shape[0]
            if diff >= 0:
                batch = array[start:stop]
                starts[i] += batch_size
            else:
                batch = np.concatenate((array[start:], array[:diff]))
                starts[i] = diff
            batches.append(batch)
        yield batches
    cifar10 = generator([x_train, y_train], N)

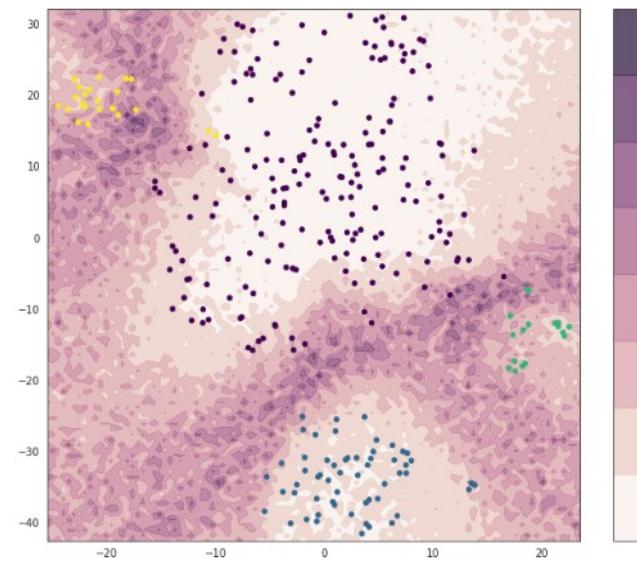
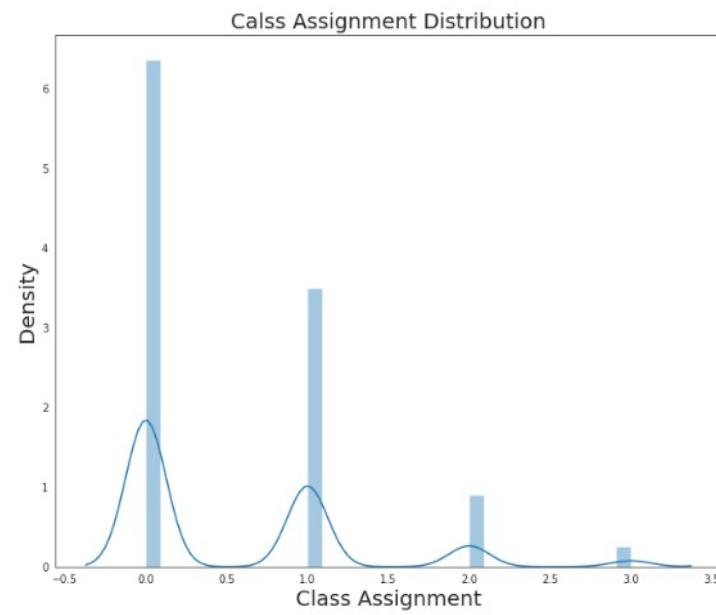
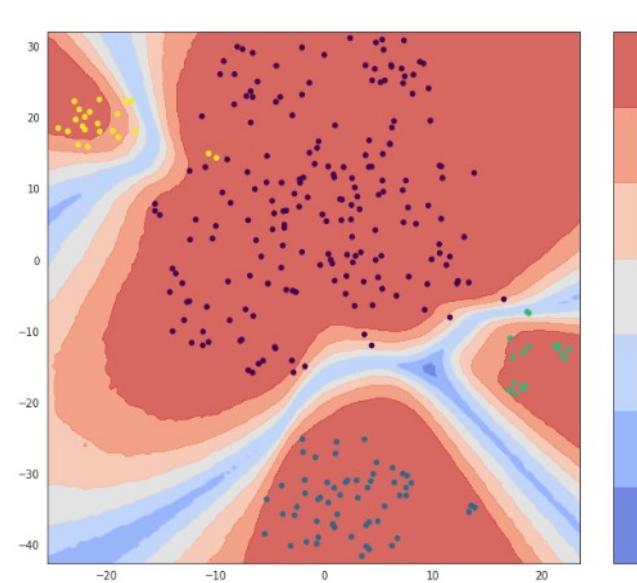
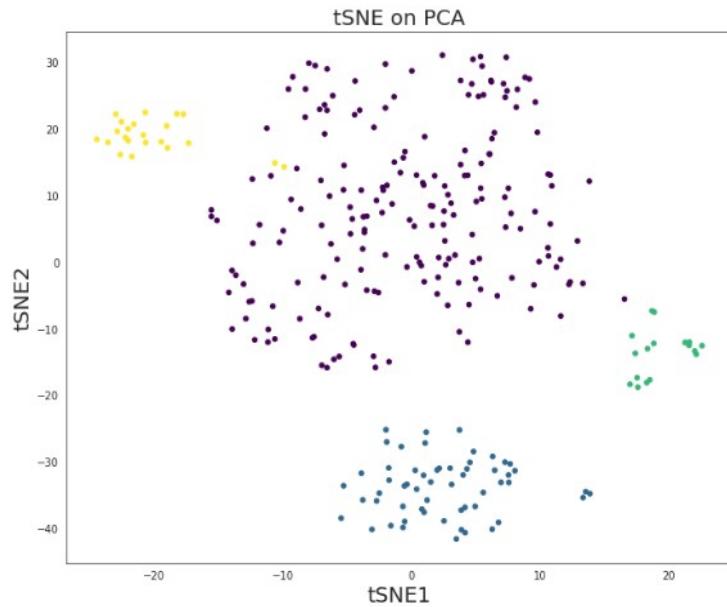
In [13]: # We use a placeholder for the labels in anticipation of the training data.
y_ph = tf.placeholder(tf.int32, [None])
# Define the VI inference technique, ie. minimise the KL divergence between q and p.
infERENCE = ed.KLqp({w: qw, b: qb}, data={y: y_ph})
# Initialise the inference algorithm.
infERENCE.initialize(n_iter=50000, n_print=100, scale=(float(x_train.shape[0]) / N))
# We will use an interactive session.
sess = tf.InteractiveSession()
# Initialise all the variables in the session.
tf.global_variables_initializer().run()
# Let the algorithm build the graph by feeding the data in minibatches and update the VI inference using each new batch.
for _ in range(inference.n_iter):
    X_batch, Y_batch = next(cifar10)
    #X_batch = X_batch.reshape(N, -1)
    # TensorFlow method gives the label data in a one hot vector format. We convert that into a single label.
    Y_batch = np.argmax(Y_batch, axis=1)
    info_dict = inference.update(feed_dict={x: X_batch, y_ph: Y_batch})
    inference.print_progress(info_dict)
50000/50000 [100%] Elapsed: 221s | Loss: 85453.266

In [14]: # Generate samples the posterior and store them.
n_samples = 100
prob_lst = []
samples = []
w_samples = []
b_samples = []
for _ in range(n_samples):
    w_samp = qw.sample()
    b_samp = qb.sample()
    w_samples.append(w_samp)
    b_samples.append(b_samp)
    # Also compute the probability of each class for each (w,b) sample.
    prob = tf.nn.softmax(tf.matmul(x_test, w_samp) + b_samp)
    prob_lst.append(prob.eval())
    sample = tf.concat([tf.reshape(w_samp, [-1]), b_samp], 0)
    samples.append(sample.eval())

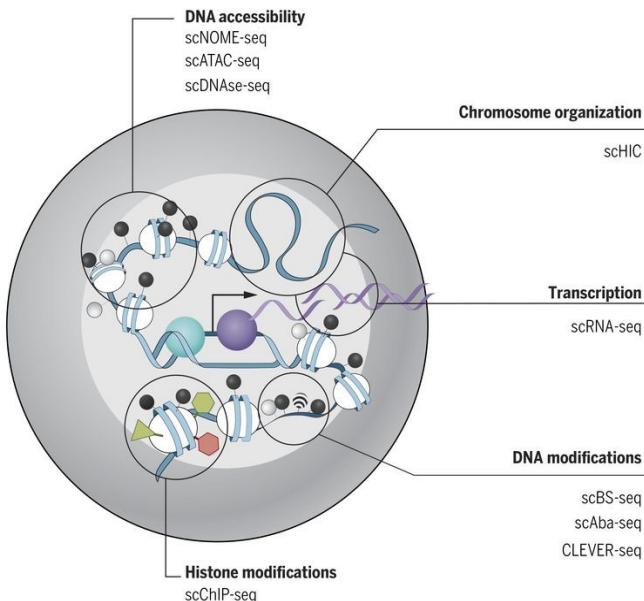
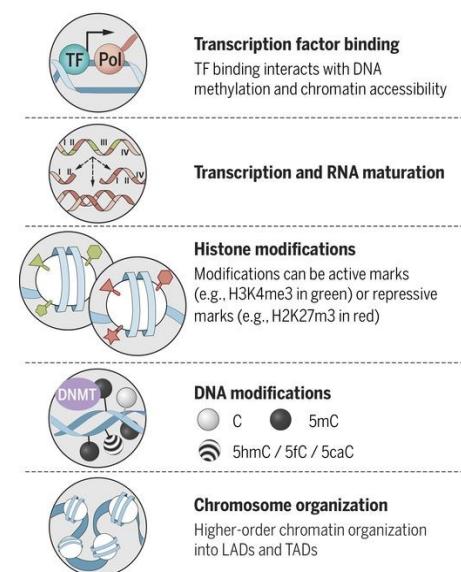
In [15]: # Compute the accuracy of the model.
# For each sample we compute the predicted class and compare with the test labels.
# Predicted class is defined as the one which has maximum probability.
# We perform this test for each (w,b) in the posterior giving us a set of accuracies.
# Finally we make a histogram of accuracies for the test data.
accy_test = []
for prob in prob_lst:
    y_trn_prd = np.argmax(prob, axis=1).astype(np.float32)
    acc = (y_trn_prd == np.argmax(y_test, axis=1)).mean() * 100
    accy_test.append(acc)

plt.hist(accy_test)
plt.title("Histogram of prediction accuracies in the CIFAR10 test data")
plt.xlabel("Accuracy")
plt.ylabel("Frequency")
plt.show()
```

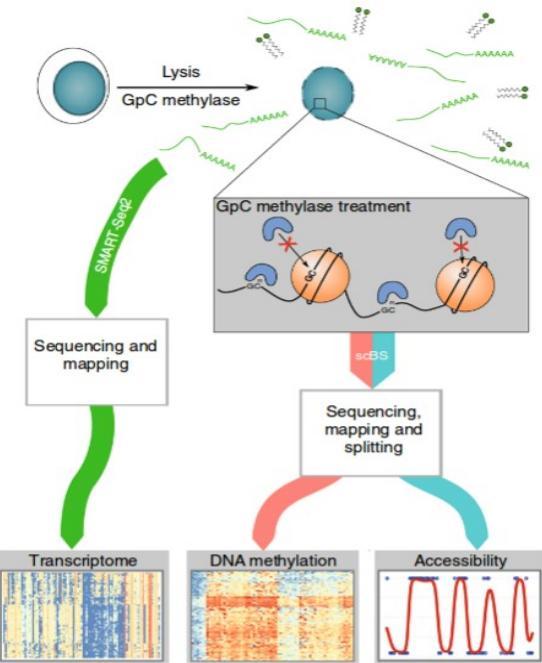




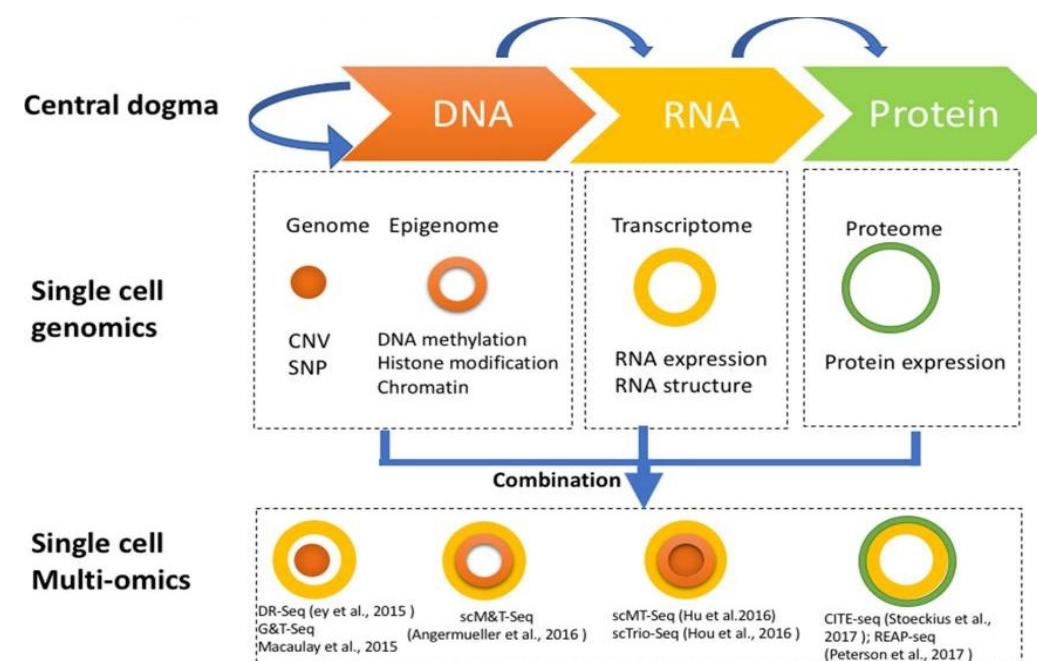
Bartoschek et al. 2018, Nature Communications, 9, 5150



Kelsey et al., 2017, Science 358, 69-75



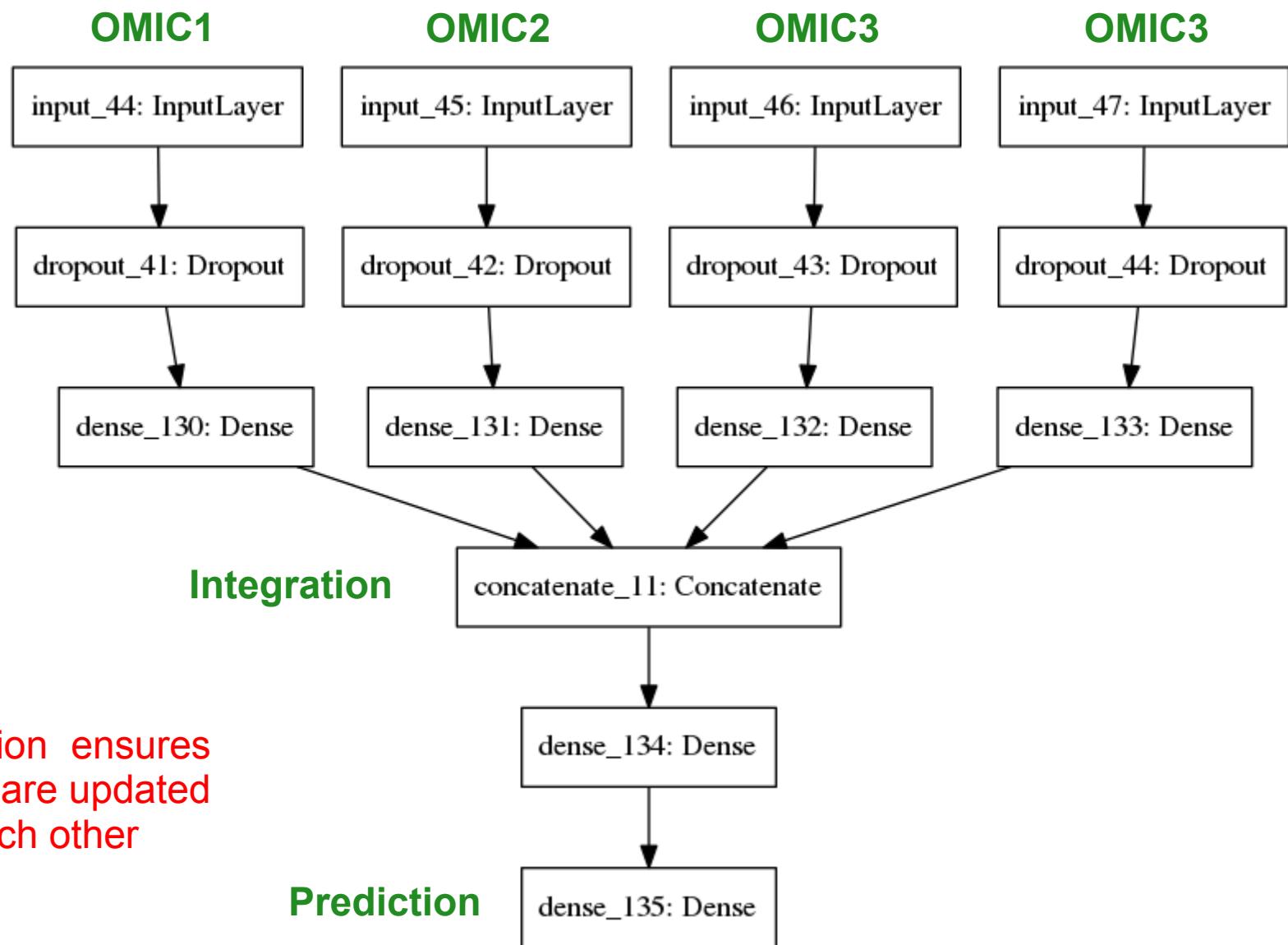
Clark et al., 2018, Nature Communications 9, 781
 scNMT-seq enables joint profiling of chromatin accessibility, DNA methylation and transcription in single cells



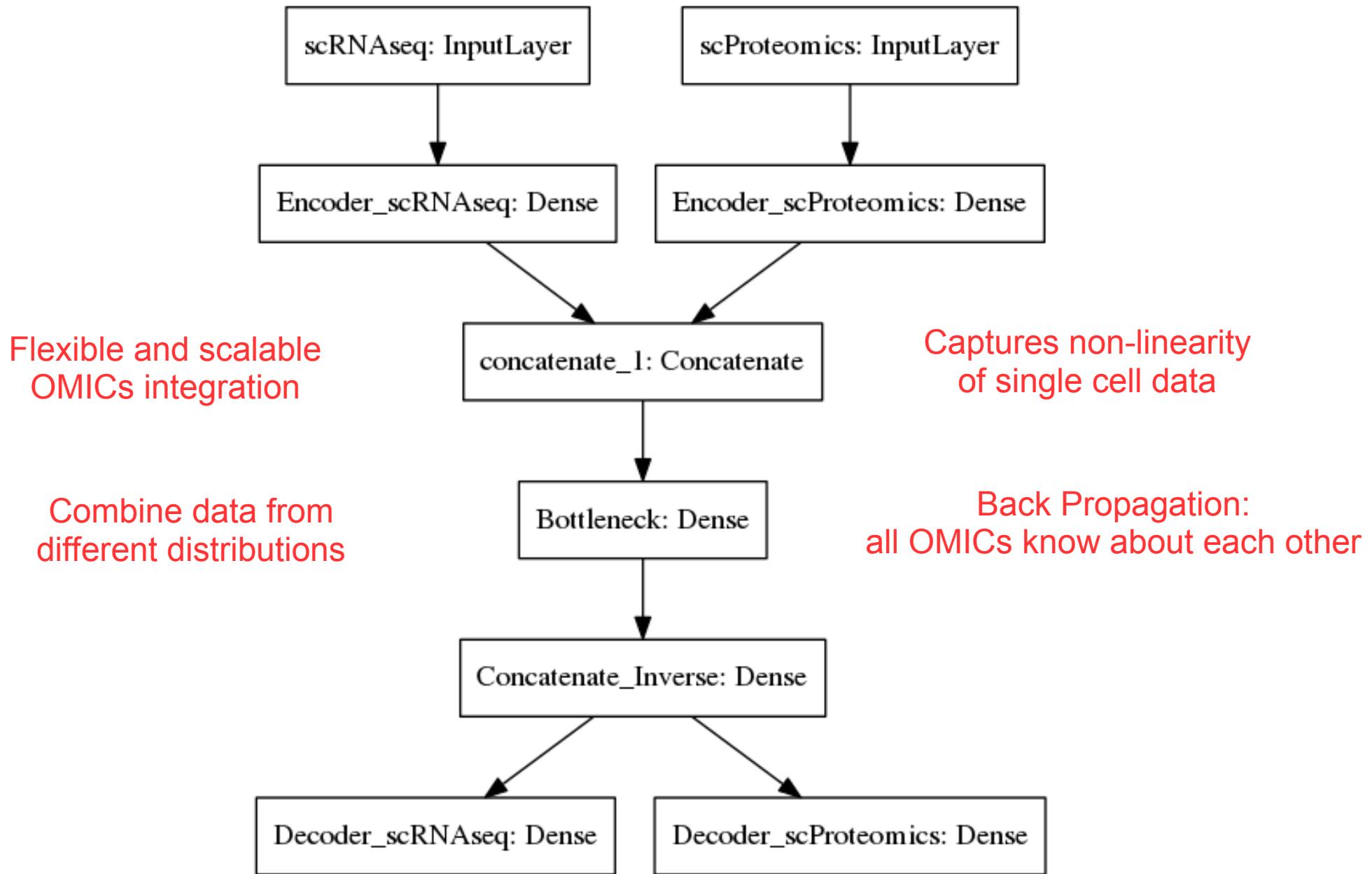
Ultimate Goal:
Model Behavior of Biological Cells

Regularization**Conversion to
nonparametric
space****Integration**

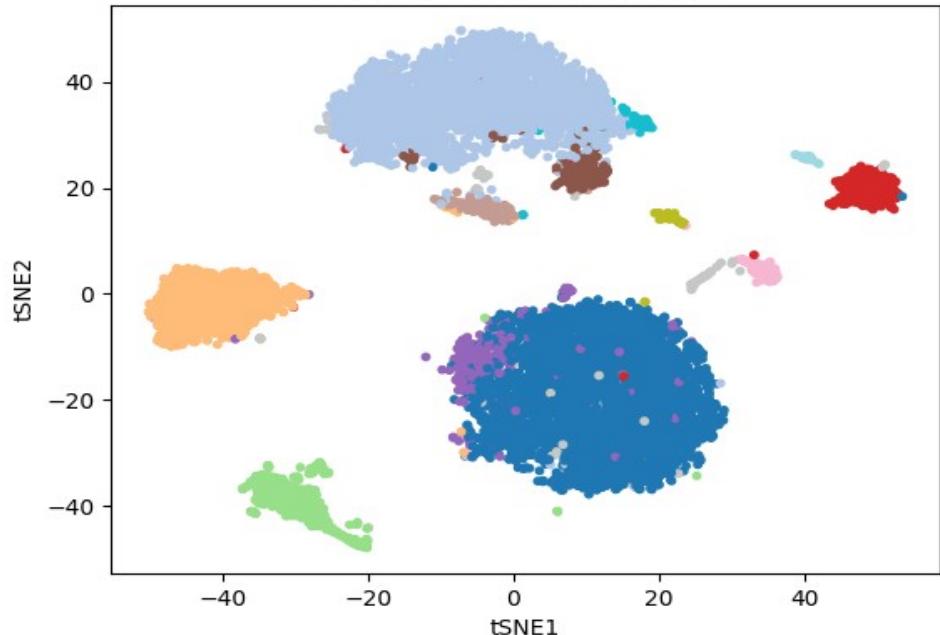
Back propagation ensures
that all weights are updated
in context of each other

Prediction

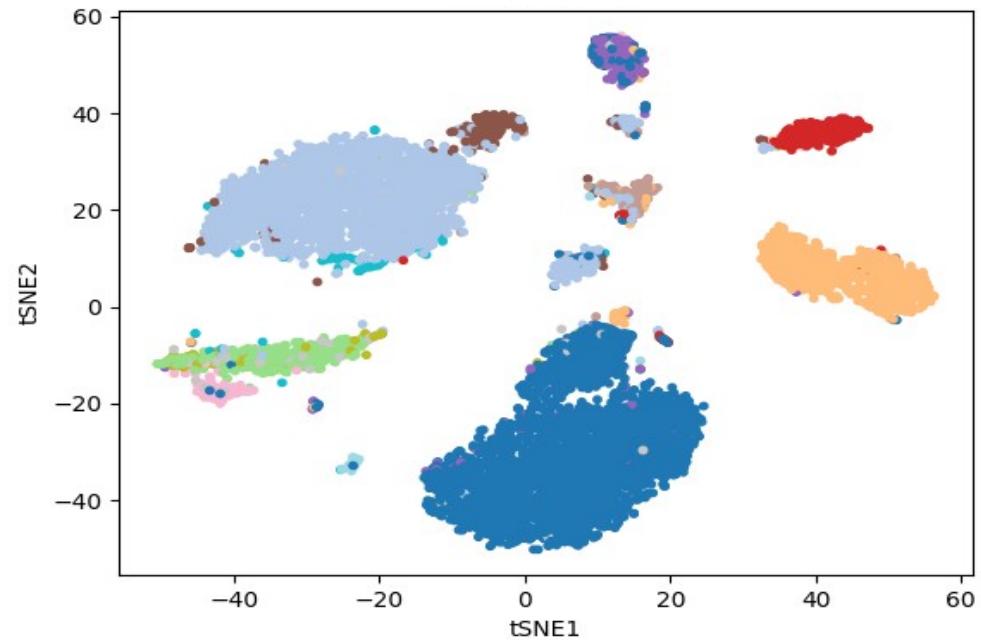
CITE-seq: Data Integration Autoencoder



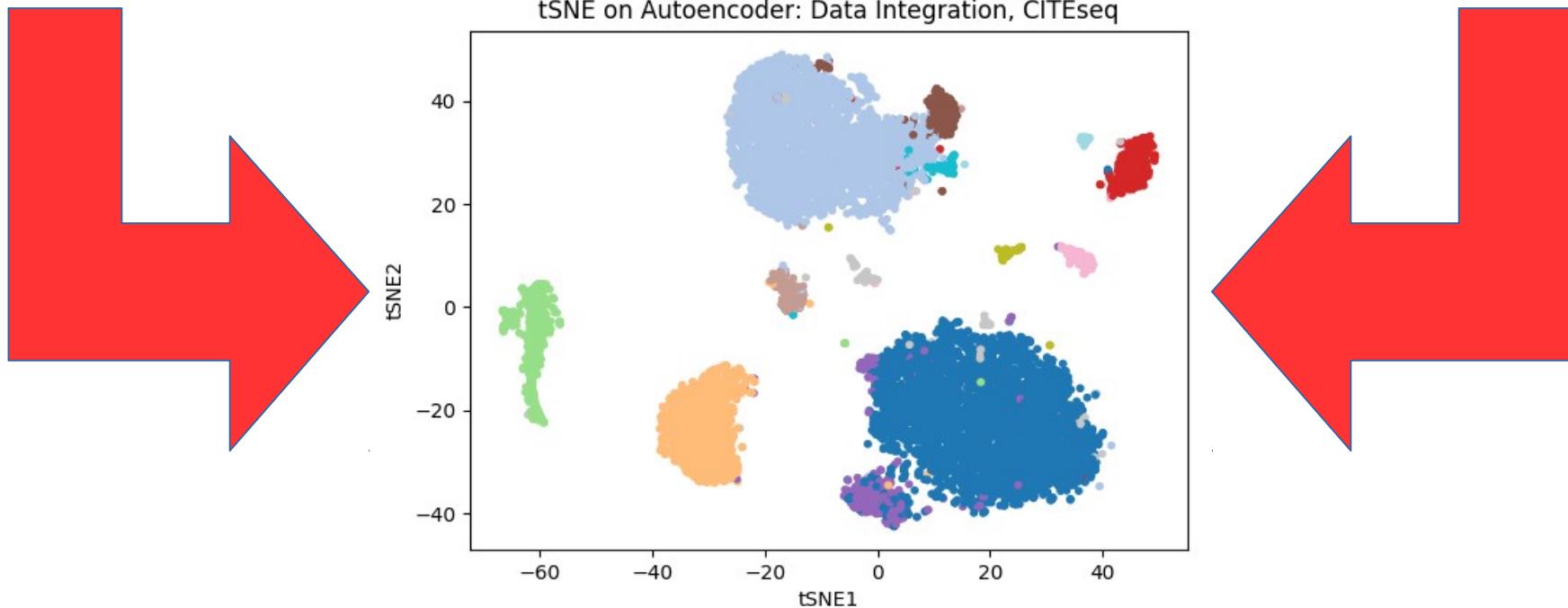
scRNAseq

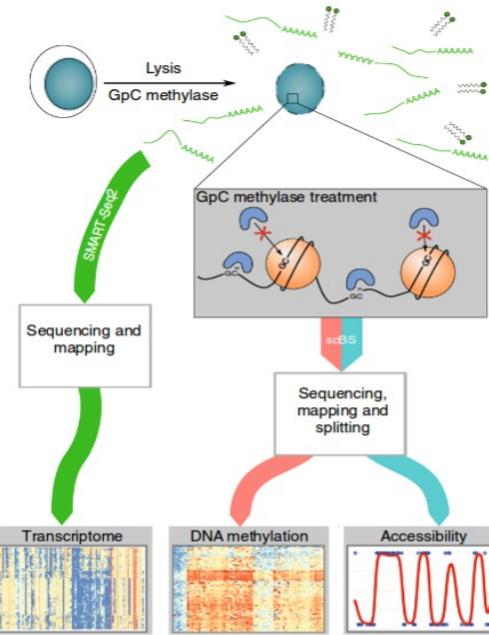


scProteomics

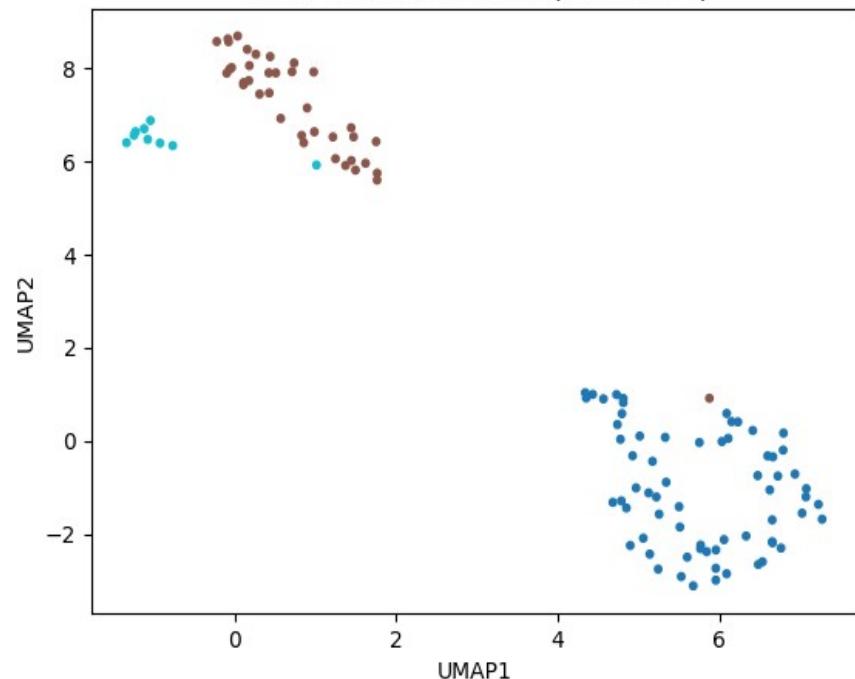


tSNE on Autoencoder: Data Integration, CITEseq

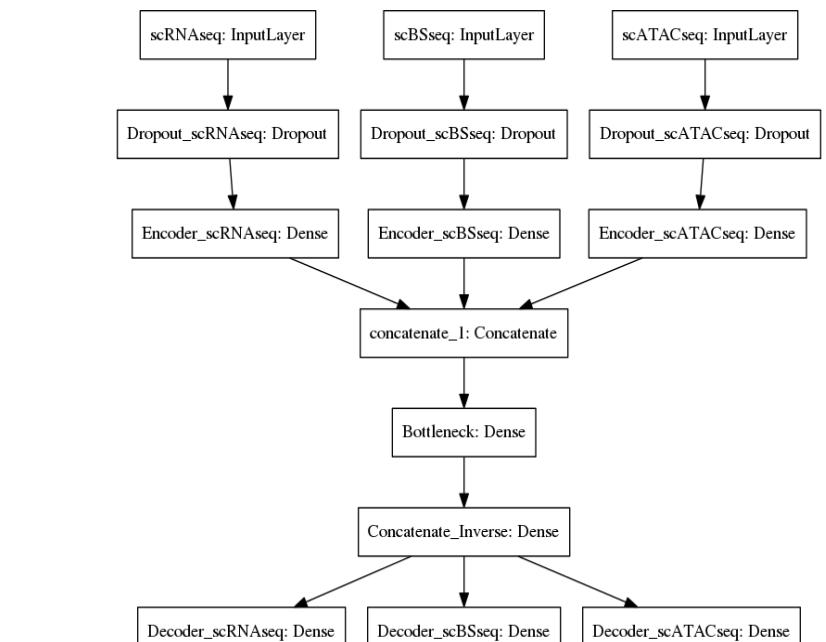




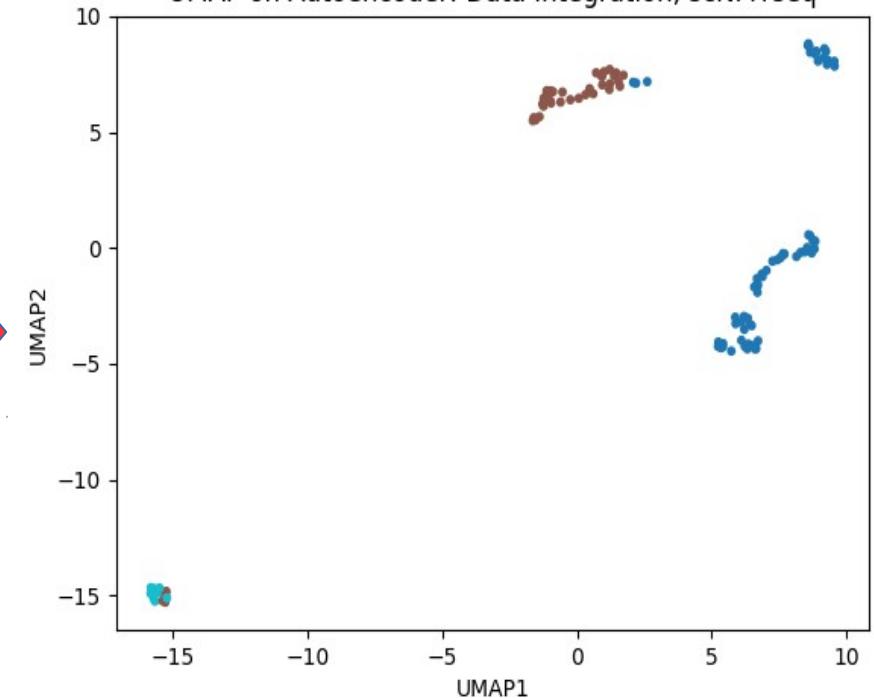
scNMTseq: Clark et al., 2018, Nature Communications 9, 781
 UMAP on PCA: scNMTseq, scRNAseq



From Single
To
Multi-OMICs



UMAP on Autoencoder: Data Integration, scNMTseq





National Bioinformatics Infrastructure Sweden (NBIS)

SciLifeLab



*Knut och Alice
Wallenbergs
Stiftelse*



LUNDS
UNIVERSITET