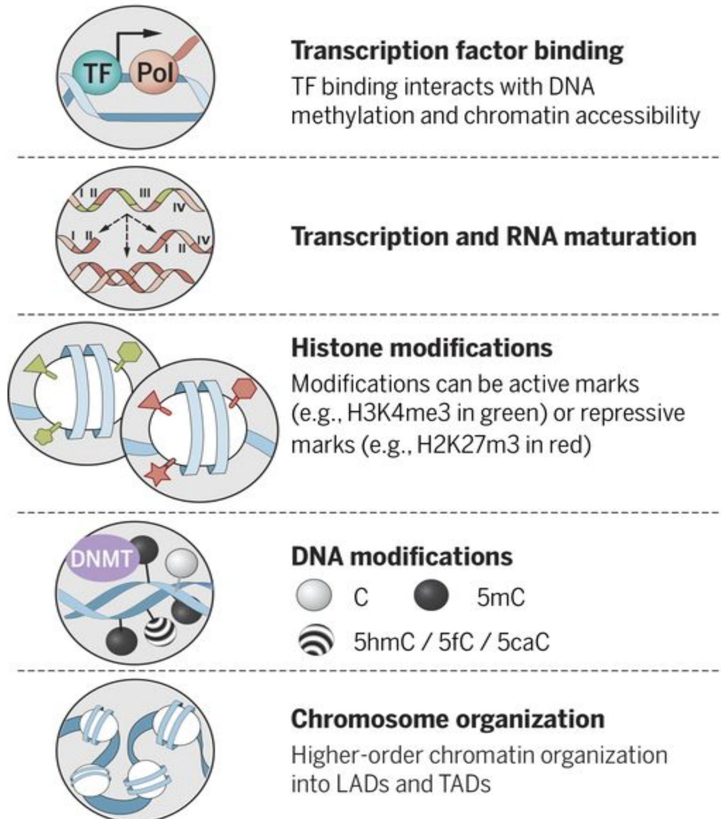
NBIS

SciLifeLab

Deep Learning for Omics Integration

Omics Integration and Systems Biology course

Nikolay Oskolkov, Lund University, NBIS SciLifeLab, Sweden



DNA accessibility
scNOME-seq
scATAC-seq
scDNAse-seq

Chromosome organization
scHIC

Transcription
scRNA-seq

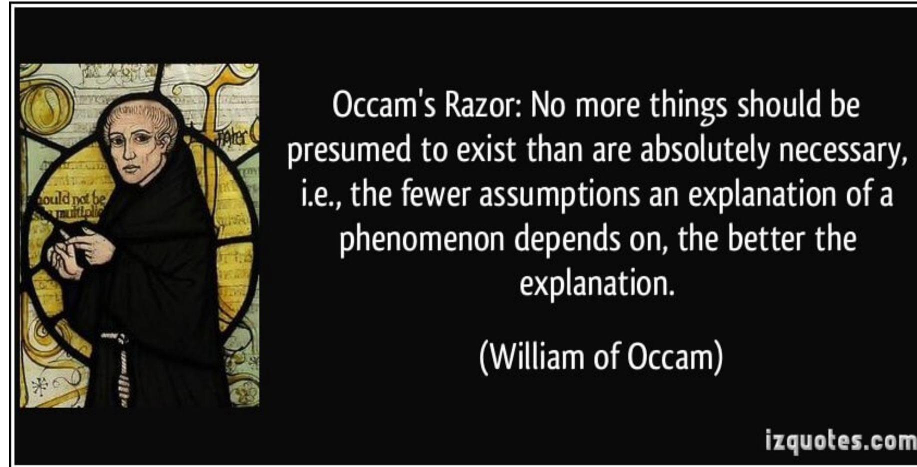
DNA modifications
scBS-seq
scAba-seq
CLEVER-seq

Histone modifications
scChIP-seq

Image adapted from Kelsey et al., Science 2017

Challenges of Deep Learning in Life Sciences

- Difficult to apply to real Life Science projects (NGS: tabular data)
- Lack of data in Life Sciences (exceptions: single cell, microscopy)
- Simpler (than Deep Learning) methods often perform better



Why don't neural networks always work?

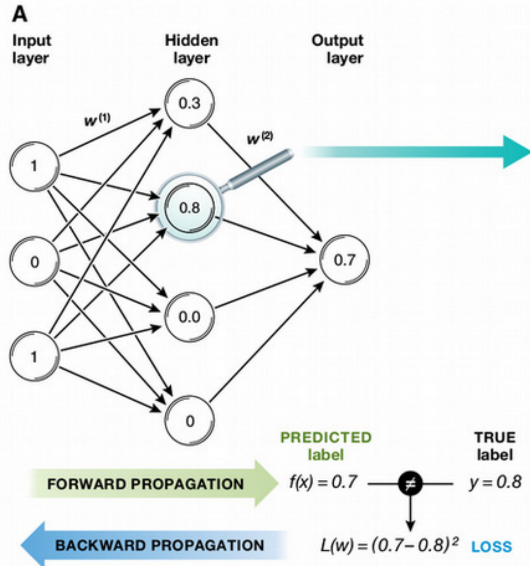
Geoffrey Hinton: Nobel prize in physics 2024

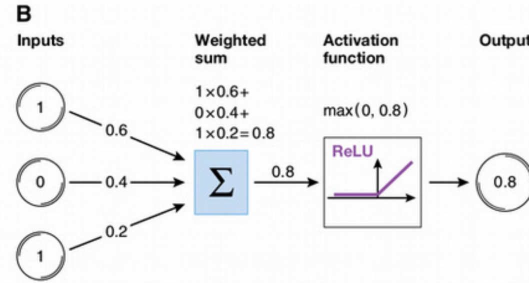
Artificial Neural Networks: general principles

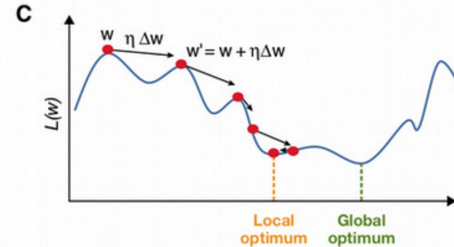
NBIS

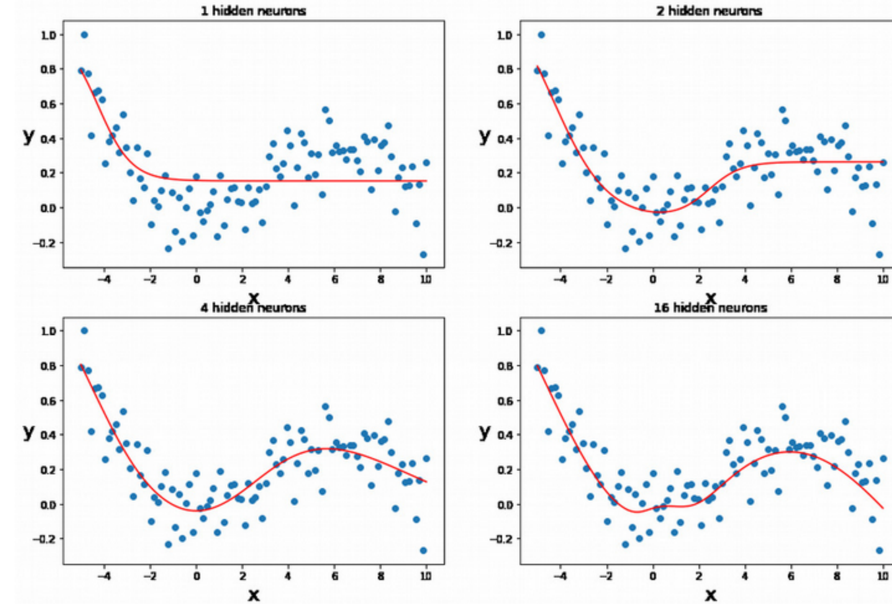
SciLifeLab

- ANN: a mathematical function Y = f(X) with a special architecture
- Can be **non-linear** depending on **activation function**
- Backward propagation **(gradient descent)** for minimizing error
- Universal Approximation Theorem



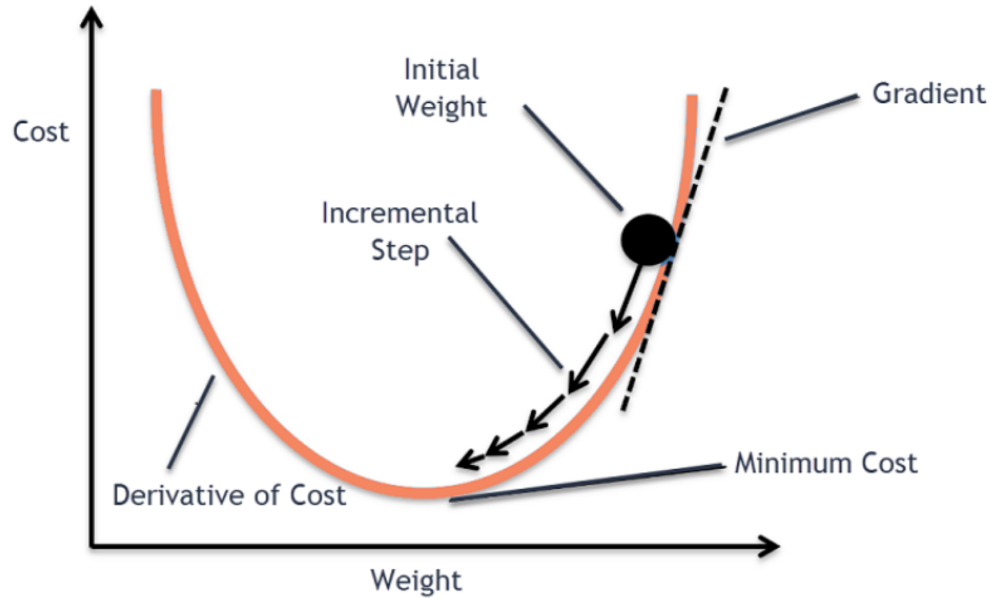






Gradient descent: backward propagation

NBIS SciLifeLab



$$y_i = \alpha + \beta x_i + \epsilon, \;\; i = 1 \ldots n$$

$$E(\alpha, \beta) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \alpha - \beta x_i)^2$$

$$\hat{\alpha}, \hat{\beta} = \text{argmin } \mathrm{E}(\alpha, \beta)$$

$$\frac{\partial E(\alpha, \beta)}{\partial \alpha} = -\frac{2}{n} \sum_{i=1}^{n} (y_i - \alpha - \beta x_i)$$

$$\frac{\partial E(\alpha, \beta)}{\partial \beta} = -\frac{2}{n} \sum_{i=1}^{n} x_i (y_i - \alpha - \beta x_i)$$

Numeric implementation of gradient descent:

$$\alpha_{i+1} = \alpha_i - \eta \frac{\partial E(\alpha, \beta)}{\partial \alpha} \Bigg|_{\alpha=\alpha_i, \beta=\beta_i}$$

$$\beta_{i+1} = \beta_i - \eta \frac{\partial E(\alpha, \beta)}{\partial \beta} \Bigg|_{\alpha=\alpha_i, \beta=\beta_i}$$

Coding gradient descent from scratch in R

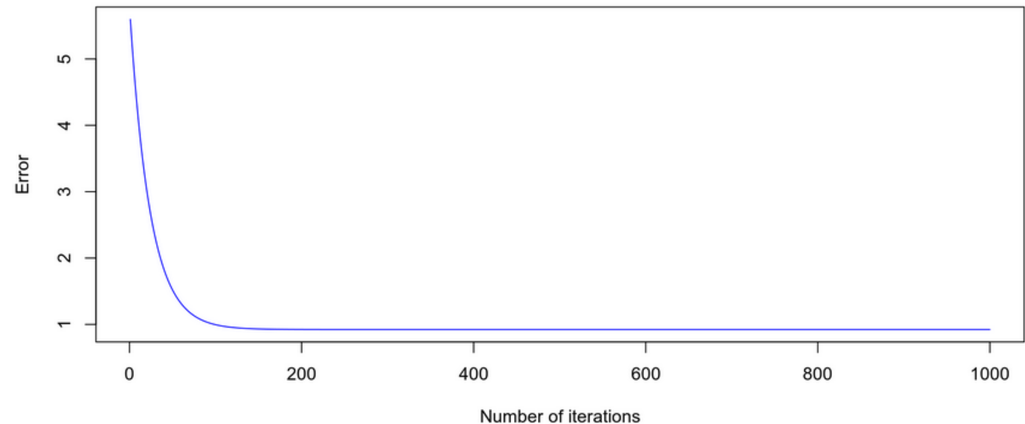
```r
n <- 100 # sample size
x <- rnorm(n) # simulated expanatory variable
y <- 3 + 2 * x + rnorm(n) # simulated response variable
summary(lm(y ~ x))
```

```
Call:
lm(formula = y ~ x)

Residuals:
    Min      1Q  Median      3Q     Max 
-1.9073 -0.6835 -0.0875  0.5806  3.2904 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept)  2.89720    0.09755   29.70   <2e-16 ***
x            1.94753    0.10688   18.22   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.9707 on 98 degrees of freedom
Multiple R-squared:  0.7721,	Adjusted R-squared:  0.7698 
F-statistic:   332 on 1 and 98 DF,  p-value: < 2.2e-16
```

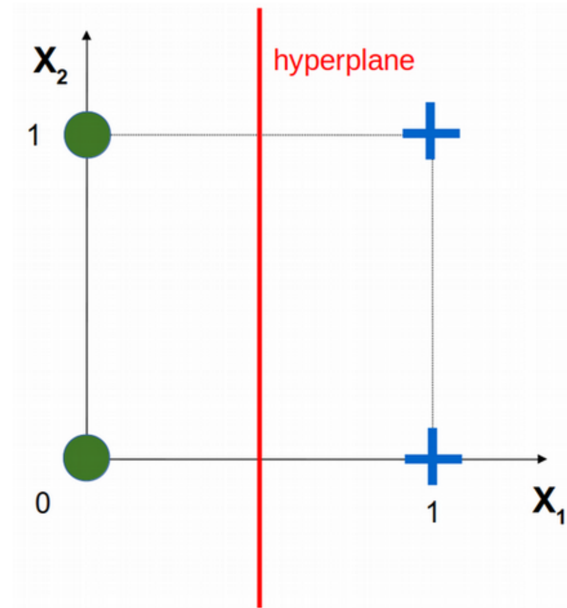
Let us now reconstruct the intercept and slope from gradient descent

```r
alpha <- vector(); beta <- vector()
E <- vector(); dEdalpha <- vector(); dEdbeta <- vector()
eta <- 0.01; alpha[1] <- 1; beta[1] <- 1 # initialize alpha and beta
for(i in 1:1000)
{
  E[i] <- (1/n) * sum((y - alpha[i] - beta[i] * x)^2)
  dEdalpha[i] <- - sum(2 * (y - alpha[i] - beta[i] * x)) / n
  dEdbeta[i] <- - sum(2 * x * (y - alpha[i] - beta[i] * x)) / n

  alpha[i+1] <- alpha[i] - eta * dEdalpha[i]
  beta[i+1] <- beta[i] - eta * dEdbeta[i]
}
print(paste0("alpha = ", tail(alpha, 1),", beta = ", tail(beta, 1)))
```

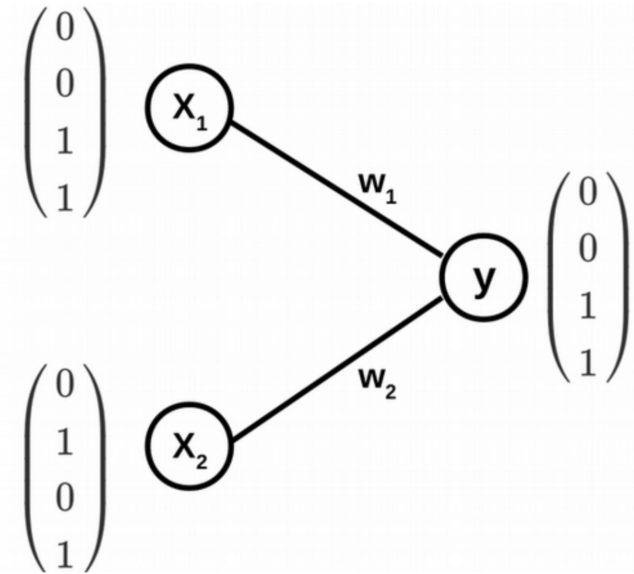
```
[1] "alpha = 2.89719694937354, beta = 1.94752837381973"
```



ANN from scratch: problem formulation



x_1	x_2	d (true) y (pred)
0	0	0 - circle
0	1	0 - circle
1	0	1 - cross
1	1	1 - cross




```r
d <- c(0, 0, 1, 1)  # true labels
x1 <- c(0, 0, 1, 1) # input variable x1
x2 <- c(0, 1, 0, 1) # input variable x2

data.frame(x1 = x1, x2 = x2, d = d)
```

x1	x2	d
0	0	0
0	1	0
1	0	1
1	1	1

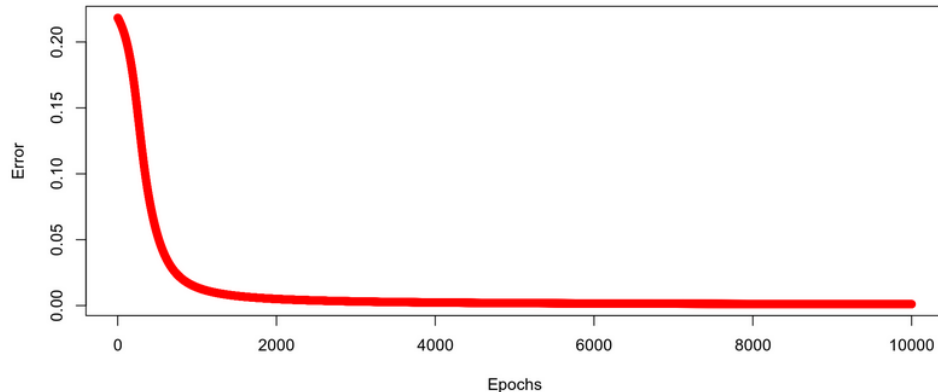
$$y(w_1, w_2) = \phi(w_1 x_1 + w_2 x_2)$$

$$\phi(s) = \frac{1}{1 + e^{-s}} \quad - \text{sigmoid}$$

$$\phi'(s) = \phi(s)\,(1 - \phi(s))$$

Coding ANN from scratch in R

```r
phi <- function(x){return(1/(1 + exp(-x)))} # activation function

mu <- 0.1; N_epochs <- 10000
w1 <- 0.1; w2 <- 0.5; E <- vector()
for(epochs in 1:N_epochs)
{
  #Forward propagation
  y <- phi(w1 * x1 + w2 * x2 - 3) # we use a fixed bias -3

  #Backward propagation
  E[epochs] <- (1 / (2 * length(d))) * sum((d - y)^2)
  dE_dw1 <- - (1 / length(d)) * sum((d - y) * y * (1 - y) * x1)
  dE_dw2 <- - (1 / length(d)) * sum((d - y) * y * (1 - y) * x2)
  w1 <- w1 - mu * dE_dw1
  w2 <- w2 - mu * dE_dw2
}
plot(E ~ seq(1:N_epochs), xlab="Epochs", ylab="Error", col="red")
```



$$E(w_1,w_2) = \frac{1}{2N}\sum_{i=1}^{N}(d_i - y_i(w_1,w_2))^2$$

$$w_{1,2} = w_{1,2} - \mu\frac{\partial E(w_1,w_2)}{\partial w_{1,2}}$$

$$\frac{\partial E}{\partial w_1} = -\frac{1}{N}\sum_{i=1}^{N}(d_i - y_i) * y_i * (1 - y_i) * x_{1i}$$

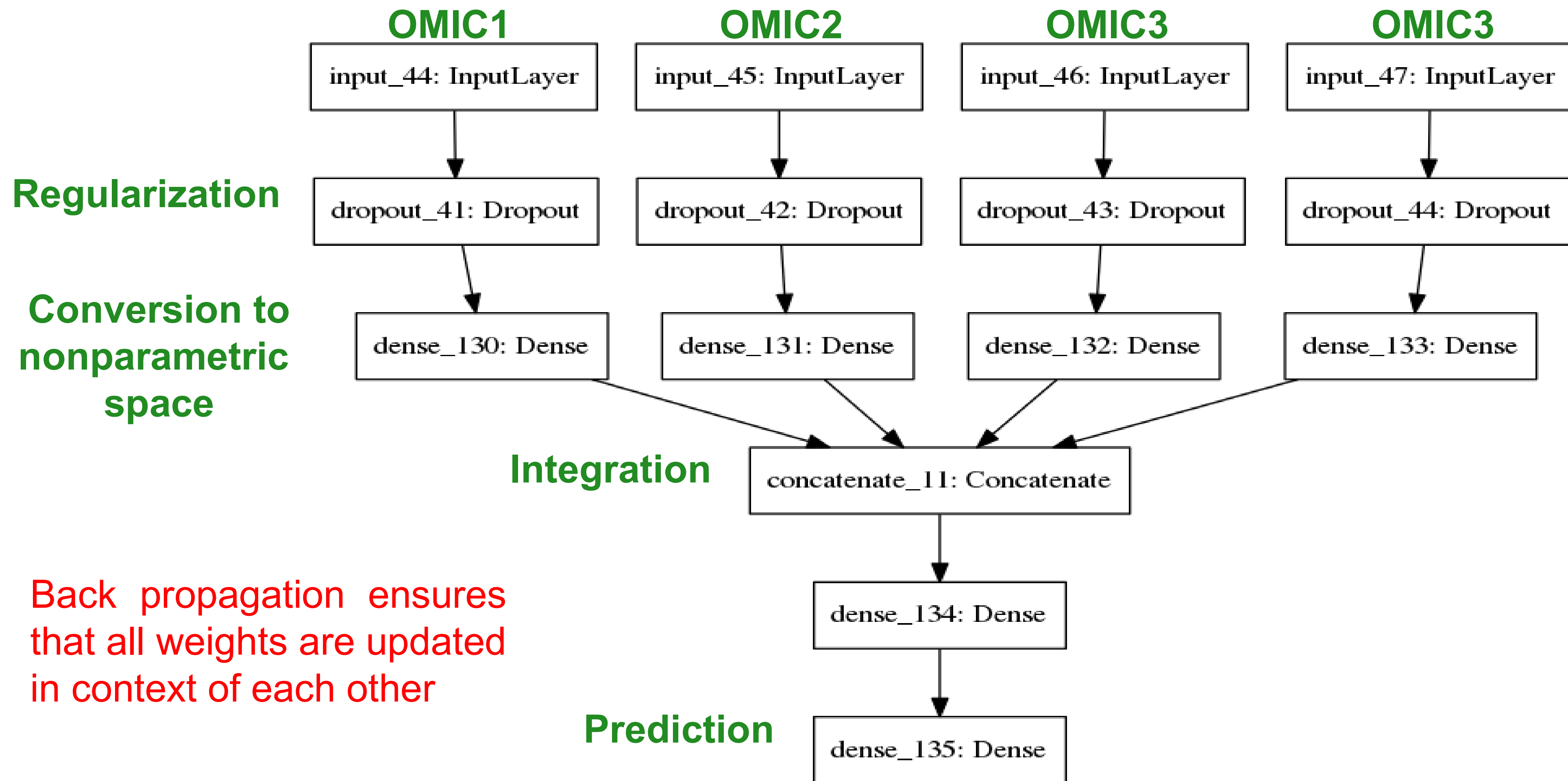
$$\frac{\partial E}{\partial w_2} = -\frac{1}{N}\sum_{i=1}^{N}(d_i - y_i) * y_i * (1 - y_i) * x_{2i}$$

```r
y
```

```
[1] 0.04742587 0.05752359 0.95730271 0.96489475
```

We nearly reconstruct true labels d = (0, 0, 1, 1)

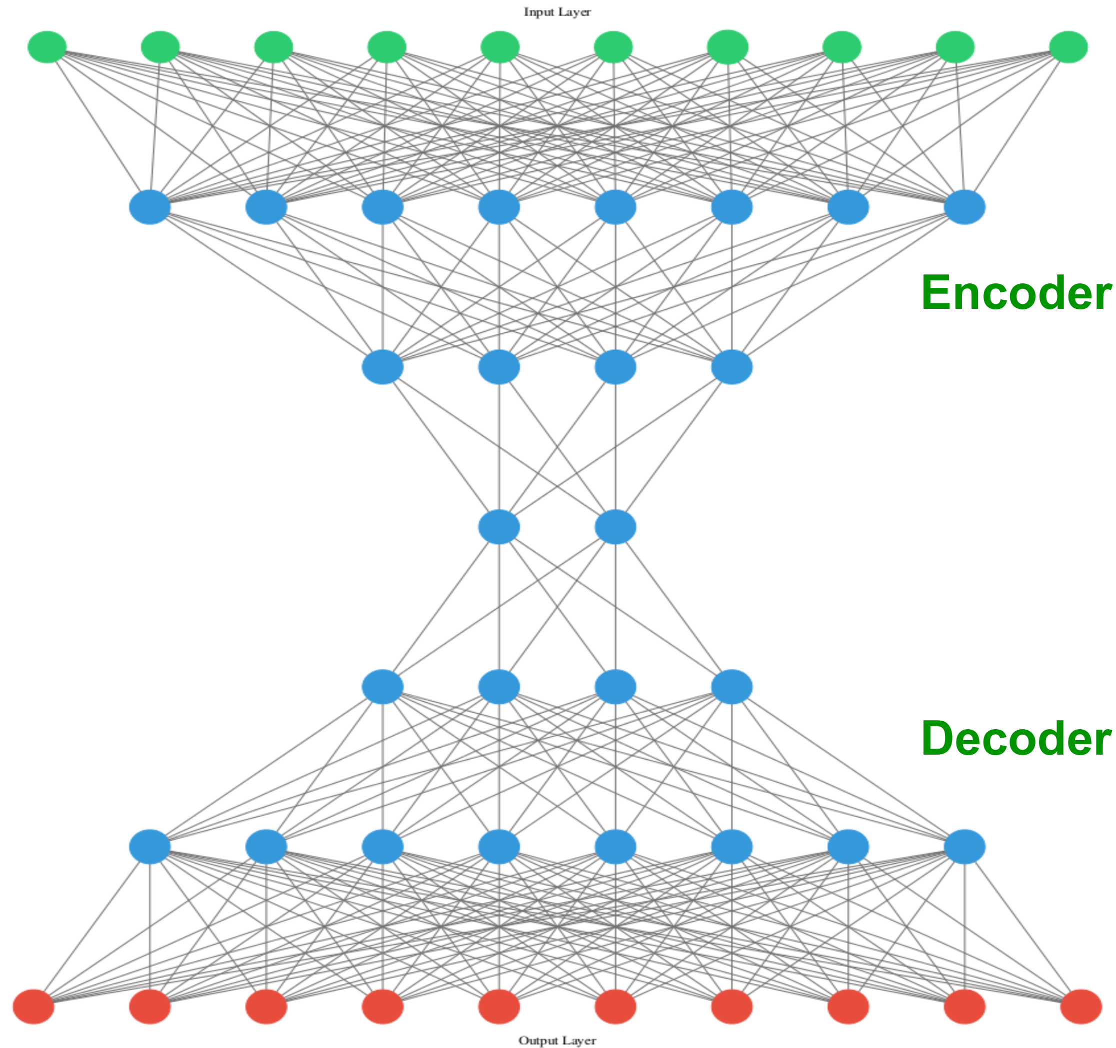
Supervised deep learning Omics integration



NBIS

Autoencoder: unsupervised neural network

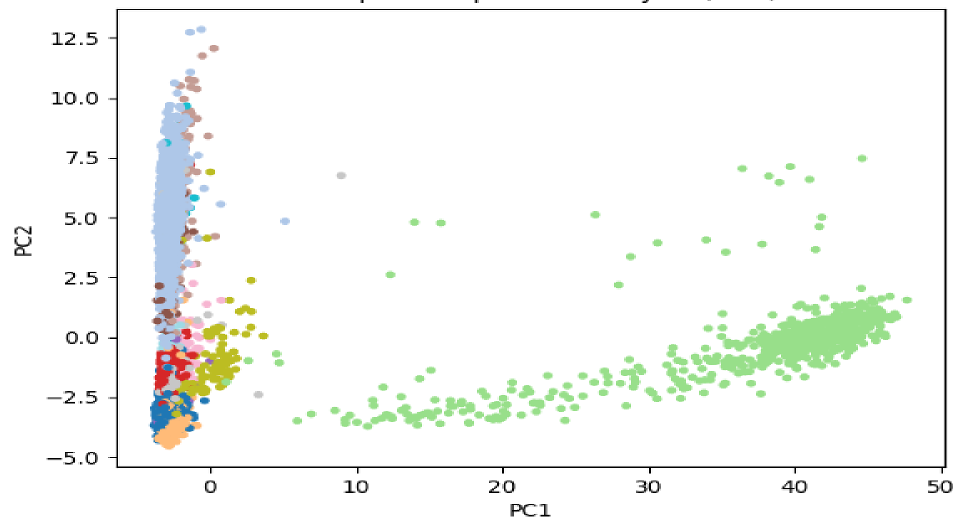
SciLifeLab

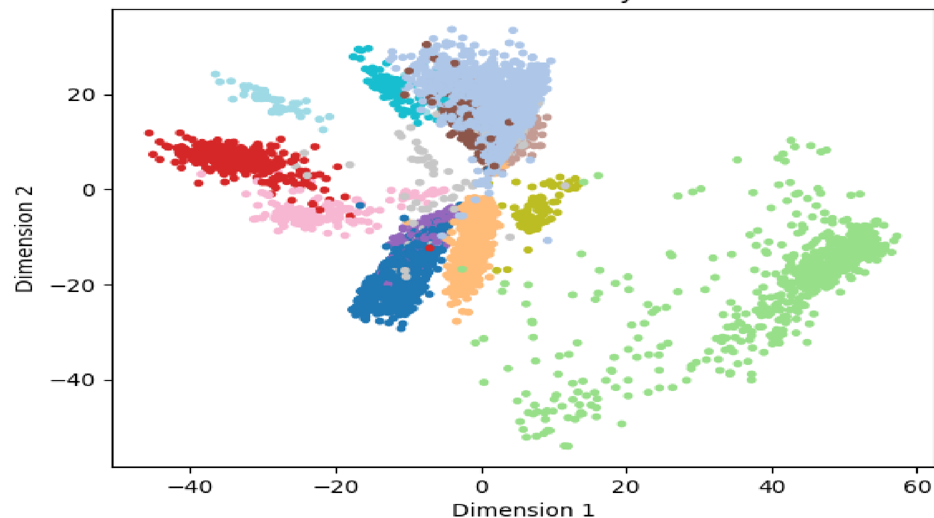


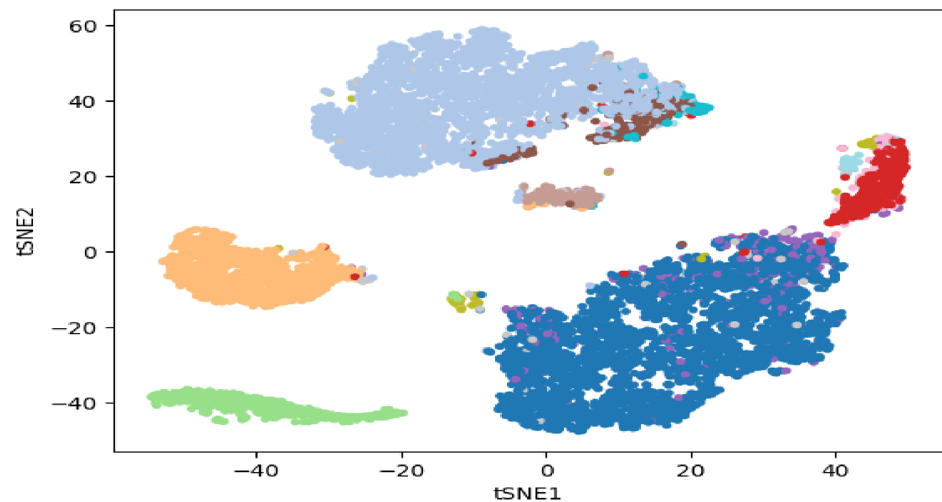
Encoder

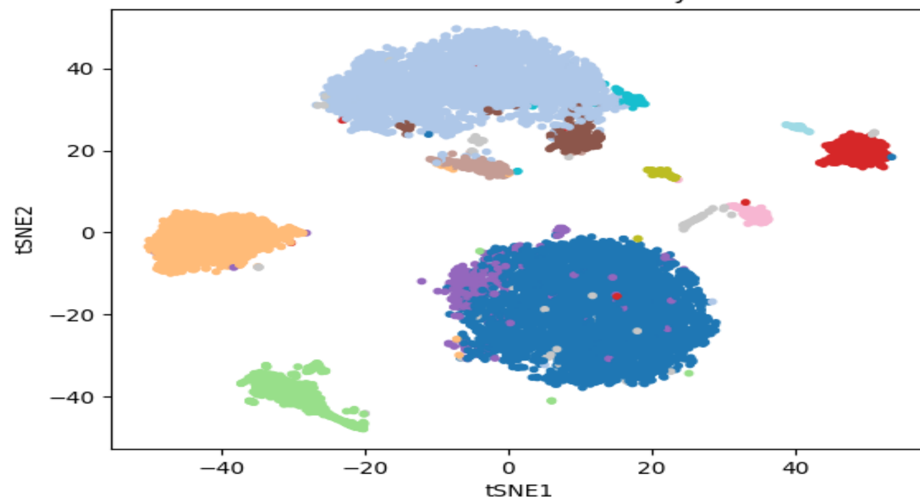
Decoder

Autoencoder dimension reduction: 8617 PBMC cells

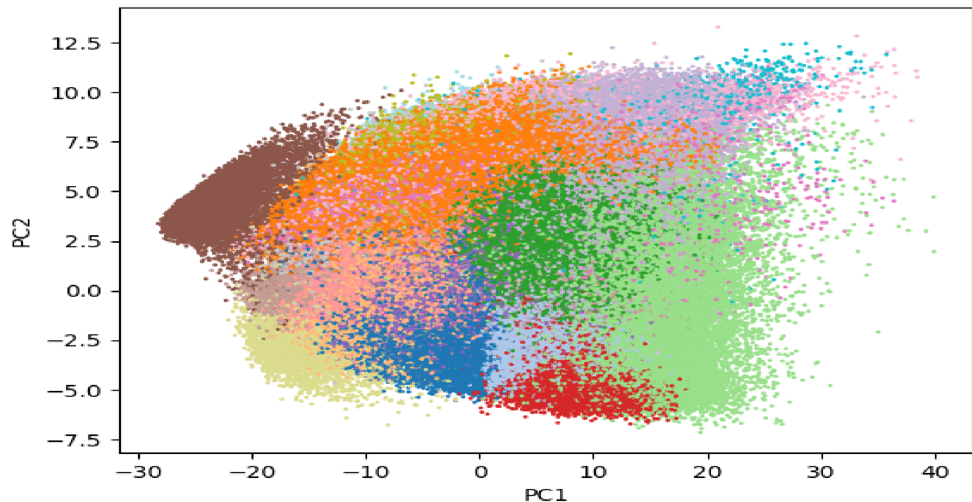


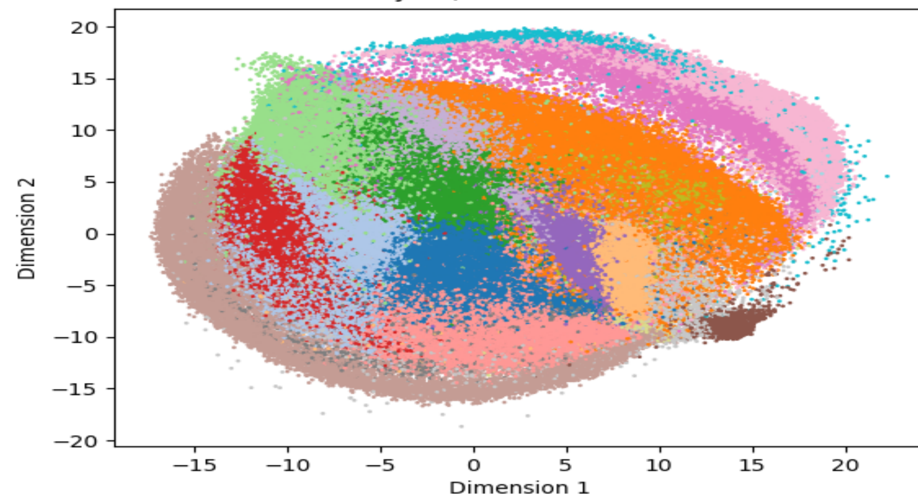


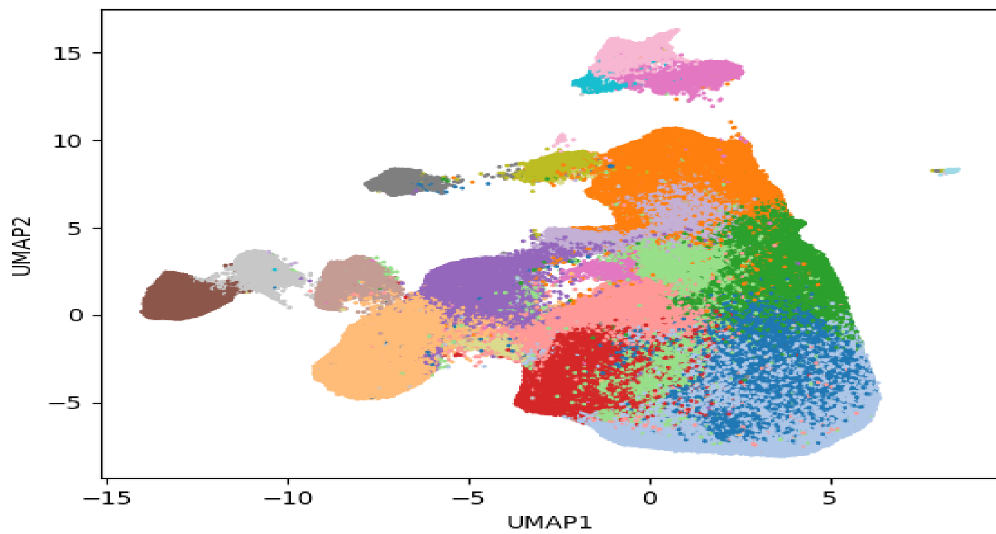


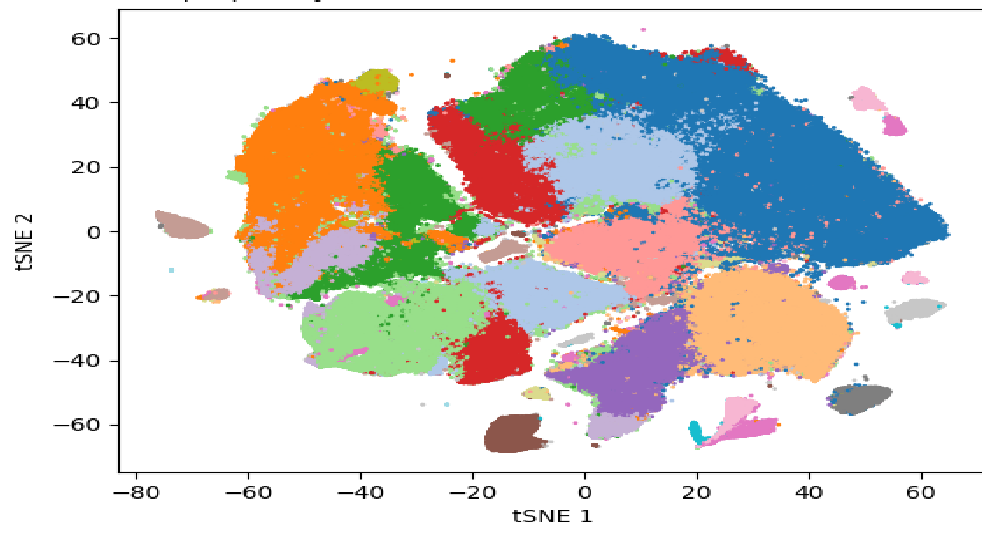


10X Genomics Mouse Brain: scRNAseq, 1.3M cells

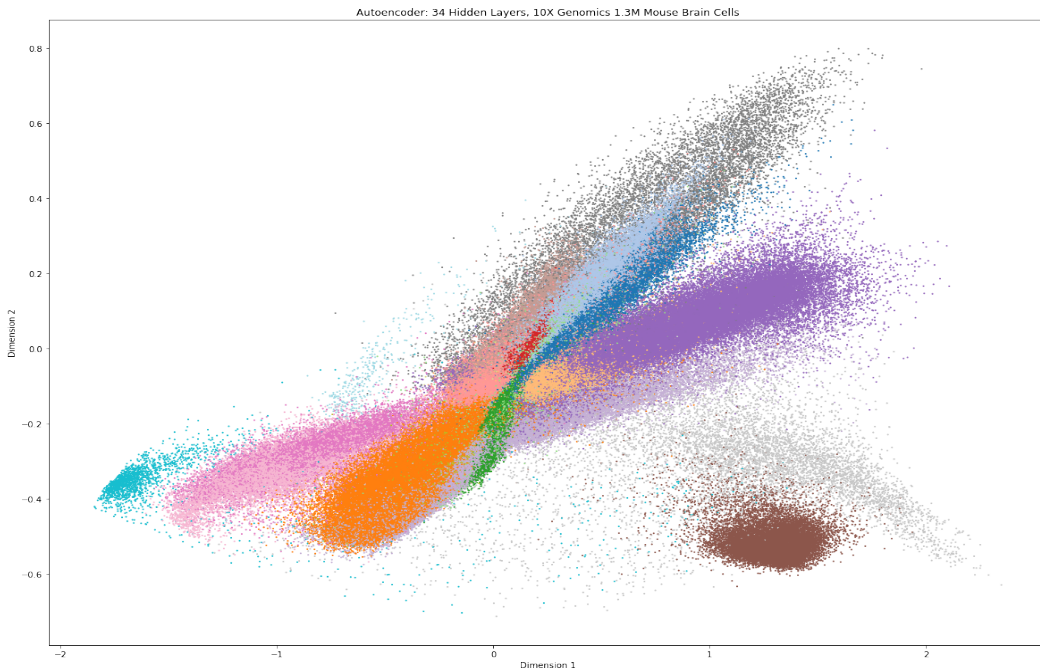




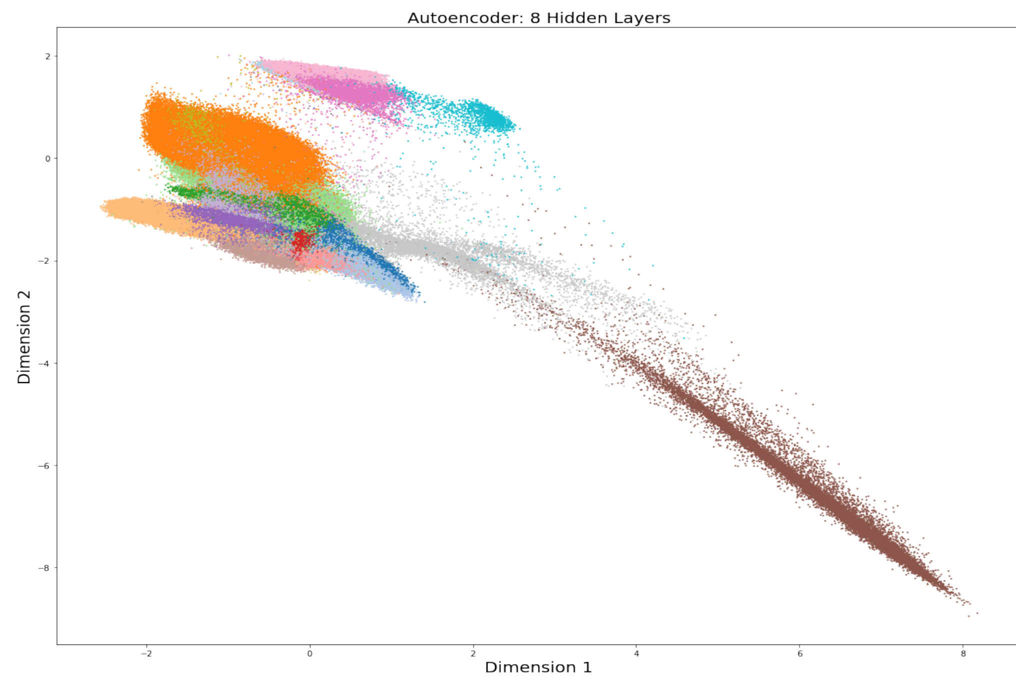




10X Genomics mouse brain: scRNAseq, 1.3M cells

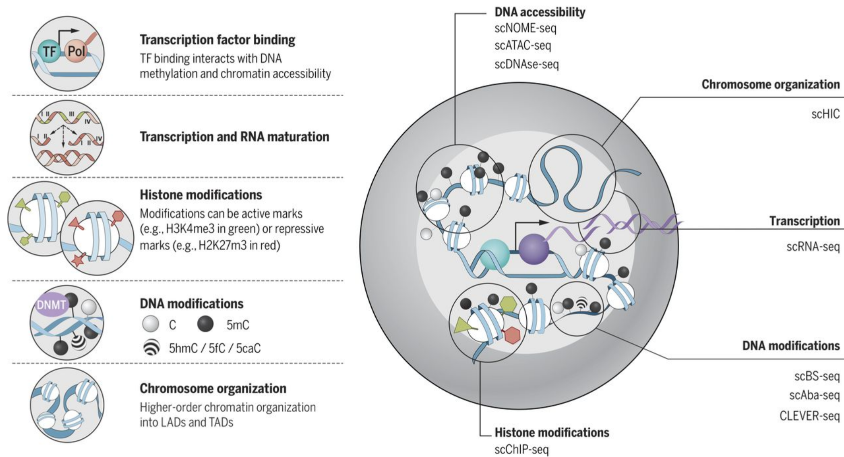


Autoencoders can be promising for non-linear data pre-processing, the bottleneck can potentially be fed to tSNE / UMAP

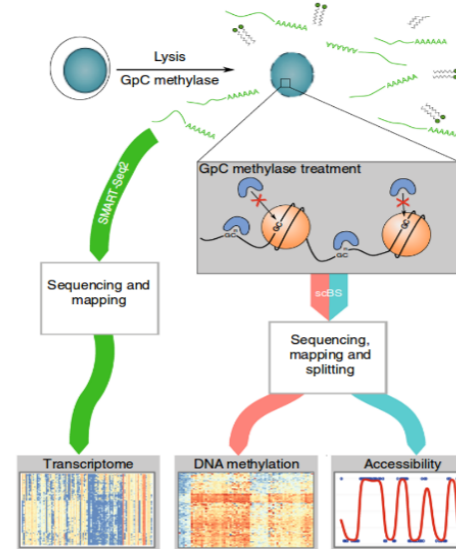


Autoencoders themselves are perhaps not optimal for visualization of scOmics

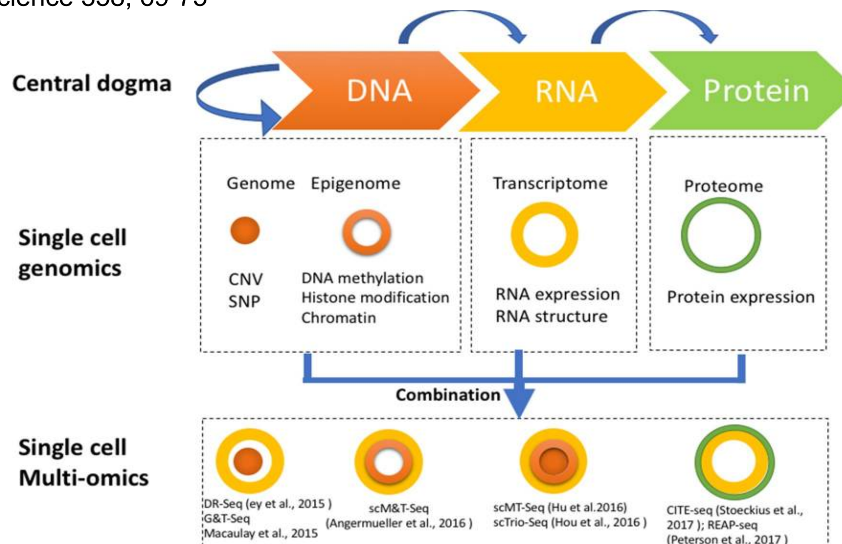
Multi-Omics in Single Cell Genomics



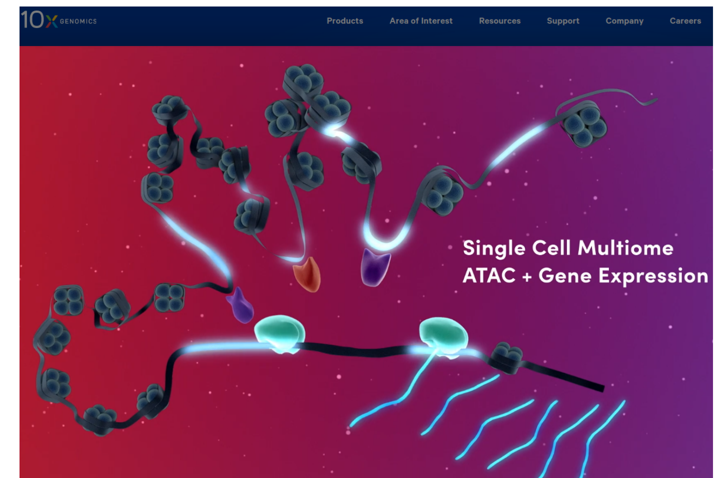
Kelsey et al., 2017, Science 358, 69-75



Clark et al., 2018, Nature Communications 9, 781



Hu et al., 2018, Frontier in Cell and Developmental Biology 6, 1-13



Autoencoder for data integration

Input Layer

Autoencoder: unsupervised neural network

Single Cell analysis is unsupervised
1) Dimensionality Reduction
2) Clustering

Output Layer

scRNAseq: InputLayer

scProteomics: InputLayer

Encoder_scRNAseq: Dense

Encoder_scProteomics: Dense

concatenate_1: Concatenate

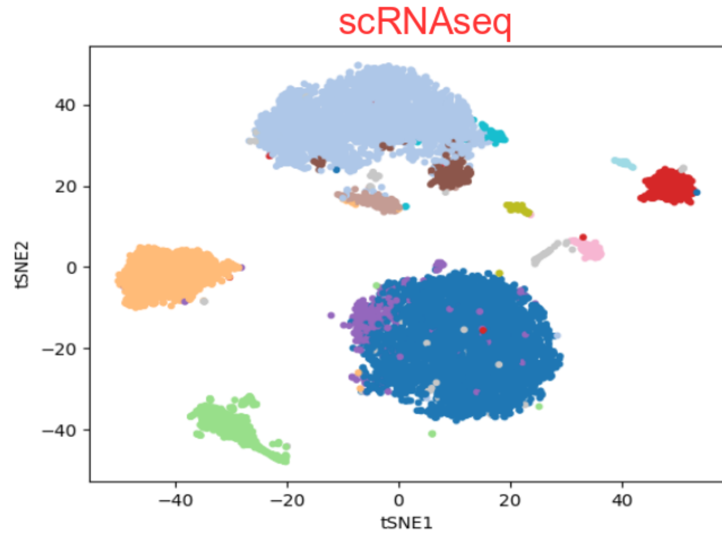
Bottleneck: Dense

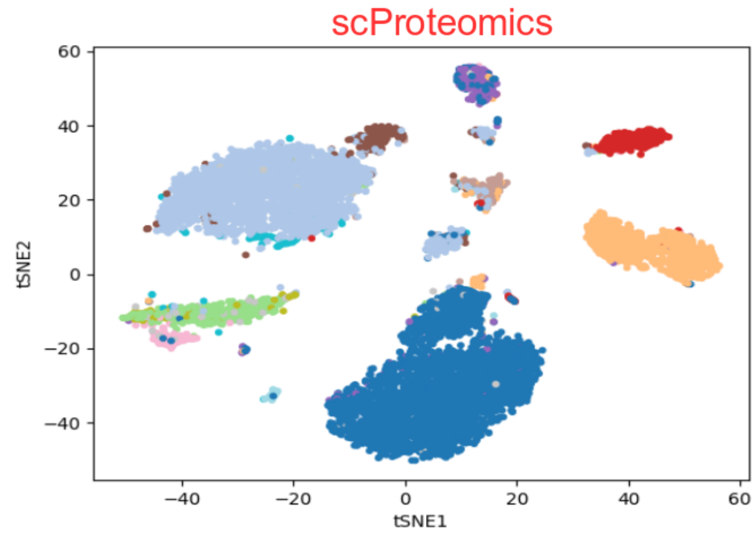
Concatenate_Inverse: Dense

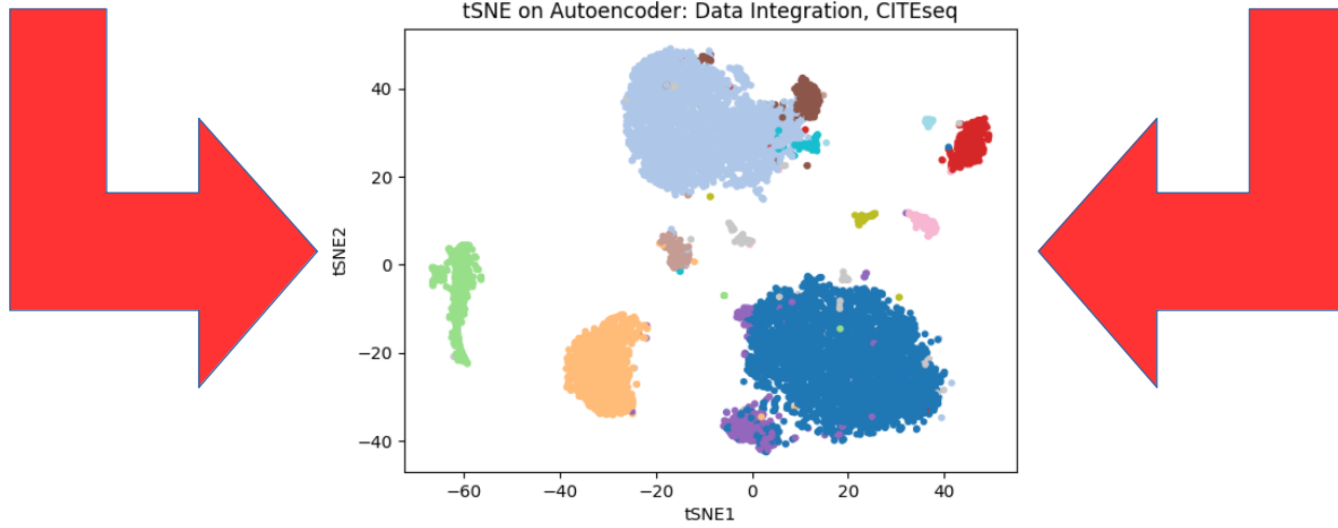
Decoder_scRNAseq: Dense

Decoder_scProteomics: Dense

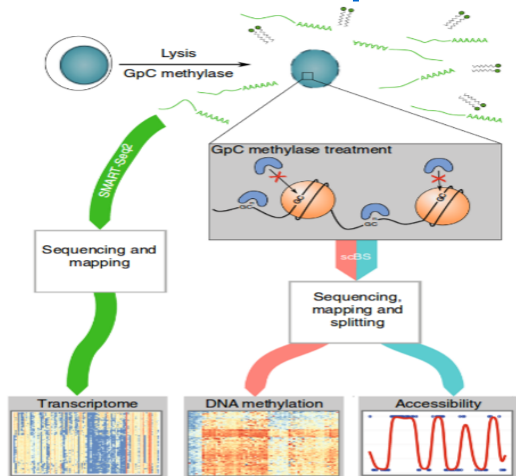
Autoencoder for data integration: CITEseq



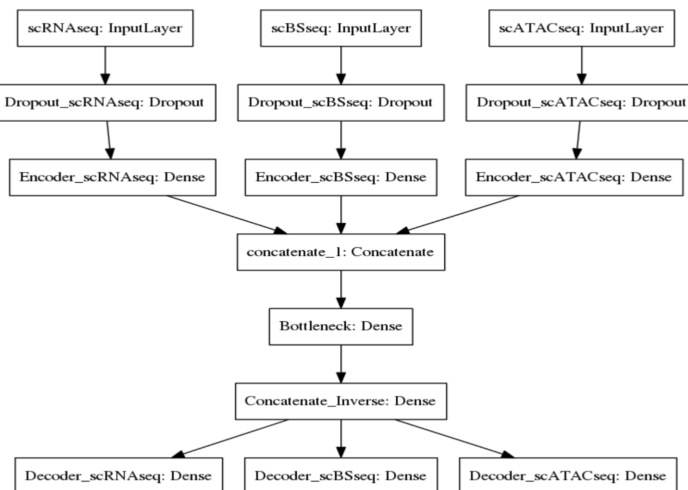


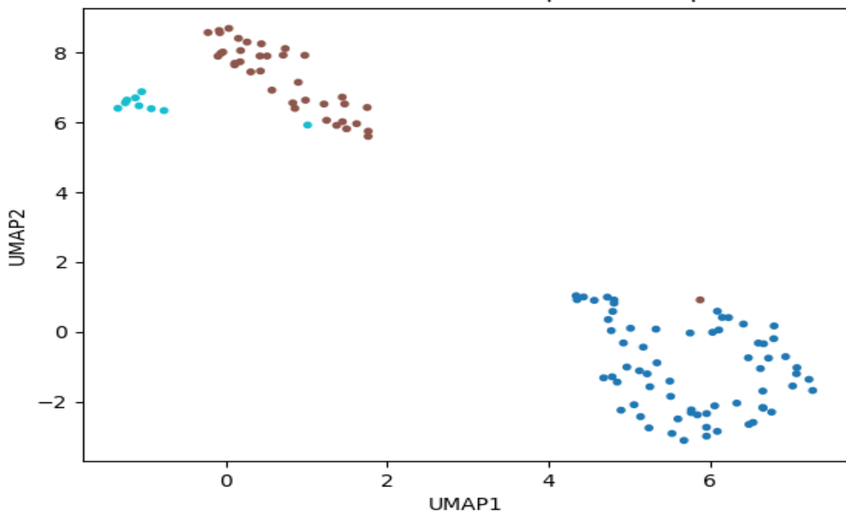


scNMTseq: scRNAseq + scBSseq + scATACseq

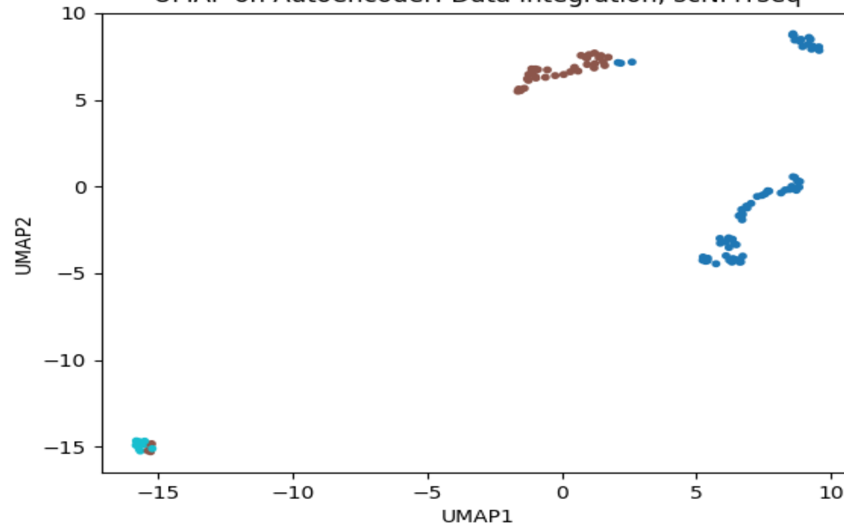


scNMTseq: Clark et al., 2018, Nature Communications 9, 781





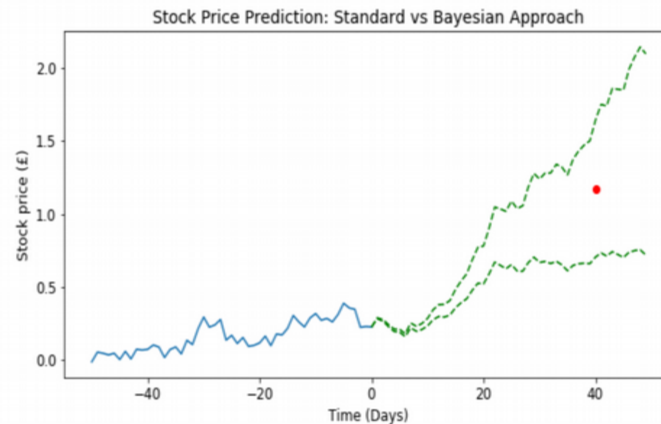
From Single
To
multi-Omics

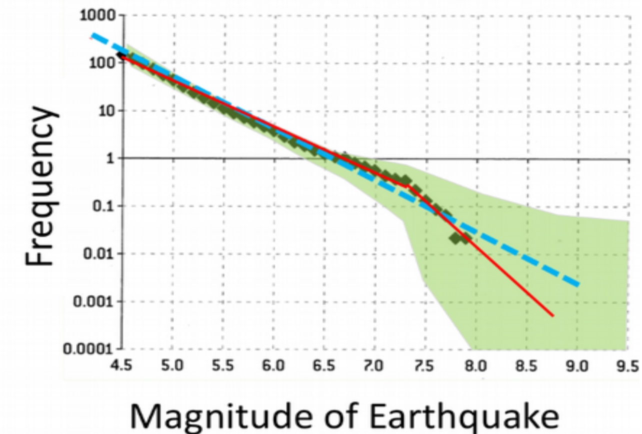


When Deep Learning is not good enough

NBIS SciLifeLab

Intelligence is to know how much you do not know



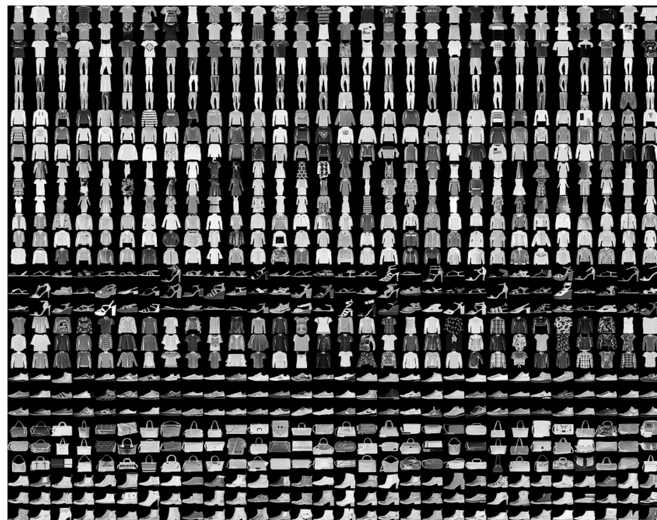


NBIS

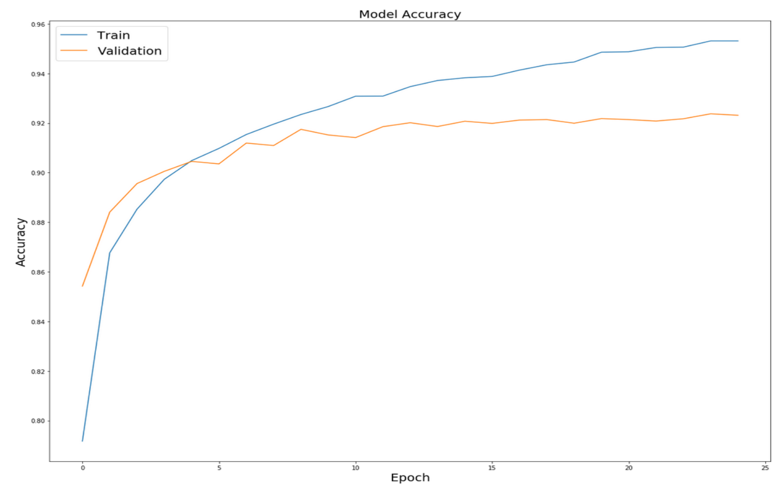
Frequentist image recognition

SciLifeLab

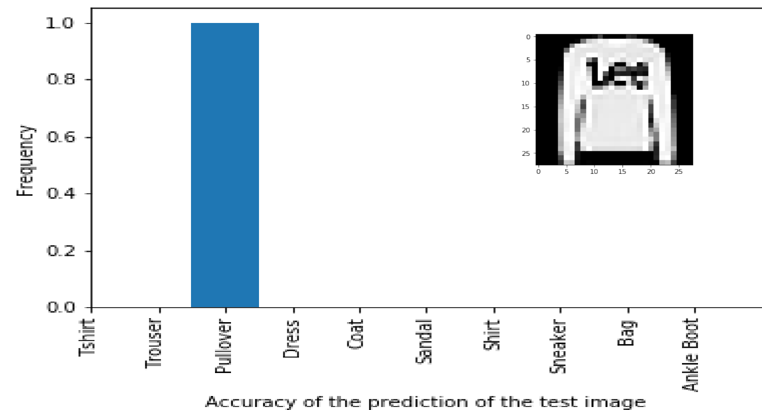
Fashion MNIST

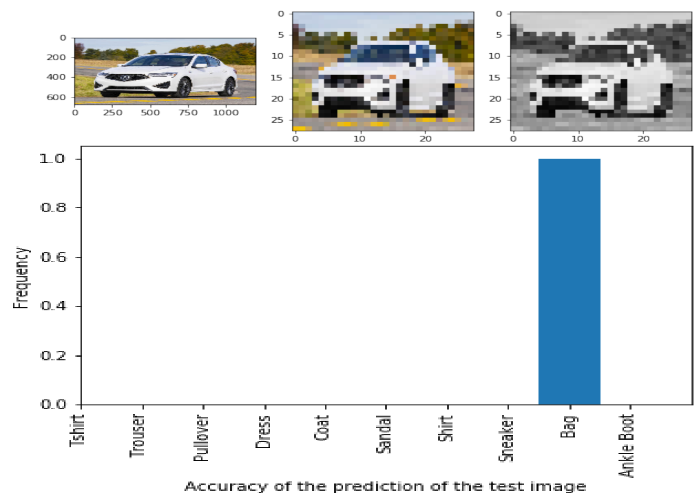


```
In [24]: # normalize inputs from 0-255 to 0.0-1.0
X_train = X_train.reshape(X_train.shape[0], 1, 28, 28).astype('float32')
X_test = X_test.reshape(X_test.shape[0], 1, 28, 28).astype('float32')
X_train = X_train / 255.0
X_test = X_test / 255.0

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_test.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
lrate = 0.01
decay = lrate/epochs
sgd = SGD(lr=lrate, 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 (MaxPooling2 (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 45000 samples, validate on 15000 samples
Epoch 1/25
45000/45000 [==============================] - 1158s 26ms/step - loss: 0.5762 - acc: 0.7917 - val_loss: 0.39
73 - val_acc: 0.8542
Epoch 2/25
45000/45000 [==============================] - 1124s 25ms/step - loss: 0.3643 - acc: 0.8676 - val_loss: 0.31
27 - val_acc: 0.8841
Epoch 3/25
45000/45000 [==============================] - 1158s 26ms/step - loss: 0.3129 - acc: 0.8853 - val_loss: 0.28
25 - val_acc: 0.8956
Epoch 4/25
45000/45000 [==============================] - 1609s 36ms/step - loss: 0.2813 - acc: 0.8973 - val_loss: 0.27
27 - val_acc: 0.9005
Epoch 5/25
45000/45000 [==============================] - 902s 20ms/step - loss: 0.2618 - acc: 0.9048 - val_loss: 0.258
8 - val_acc: 0.9045
Epoch 6/25
45000/45000 [==============================] - 936s 21ms/step - loss: 0.2451 - acc: 0.9098 - val_loss: 0.256
4 - val_acc: 0.9035
Epoch 7/25
```



Prediction



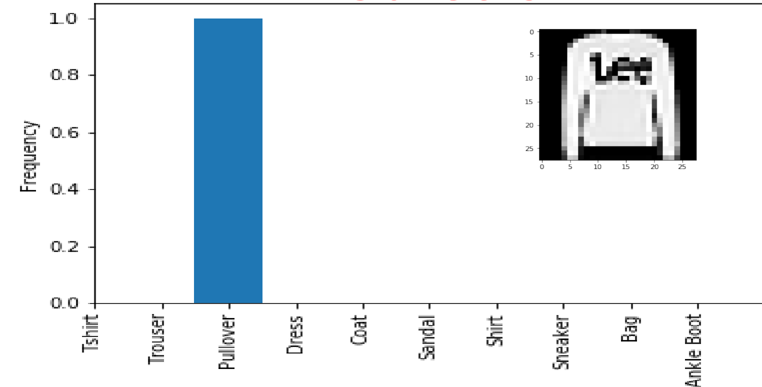


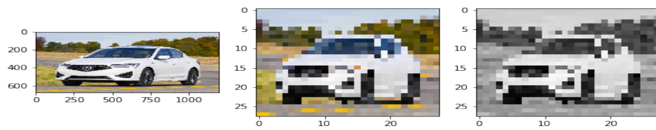
Bayesian image recognition

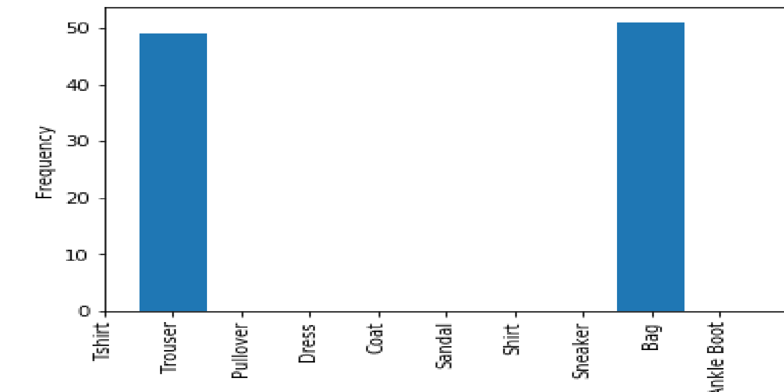
PyMC3, Edward, TensorFlow Probability

```
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 classication.
         y = Categorical(tf.matmul(x, w) + b)

In [11]: # Contruct 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 traning data.
         y_ph = tf.placeholder(tf.int32, [N])
         # 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})
         # Initialse the infernce variables
         inference.initialize(n_iter=50000, n_print=100, scale={y: float(x_train.shape[0]) / N})
         # We will use an interactive session.
         sess = tf.InteractiveSession()
         # Initialise all the vairables in the session.
         tf.global_variables_initializer().run()
         # Let the training begin. We load the data in minibatches and update the VI infernce 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 vetor 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 probabiliy 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 as 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")
```

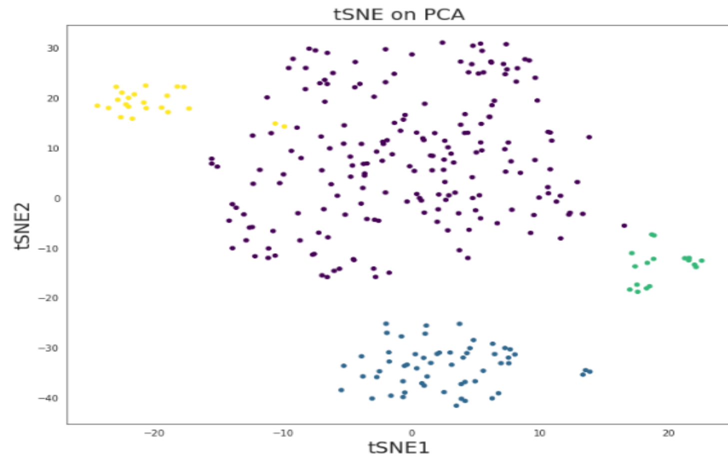
Prediction

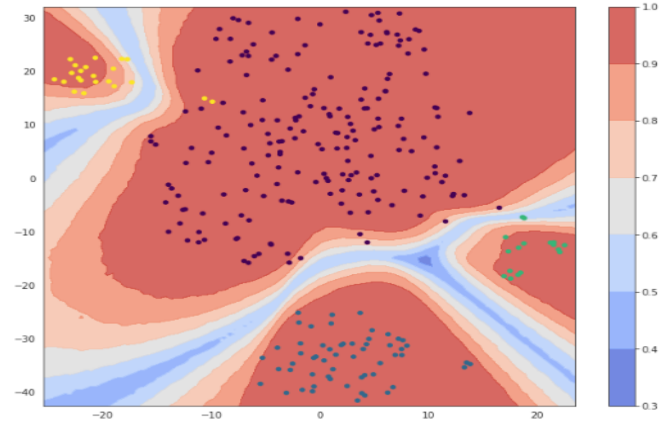


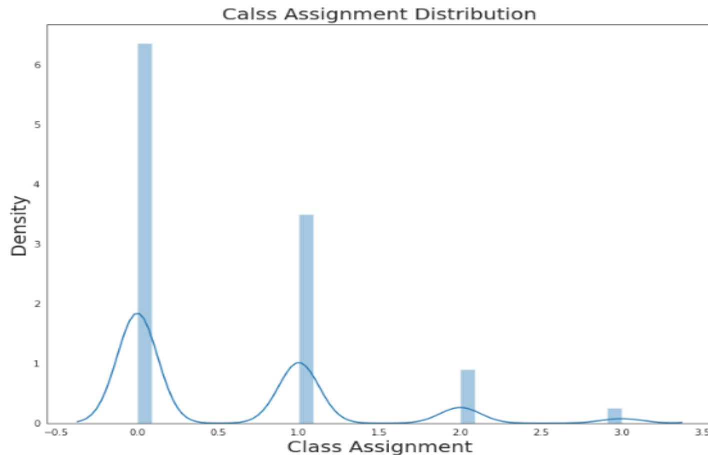


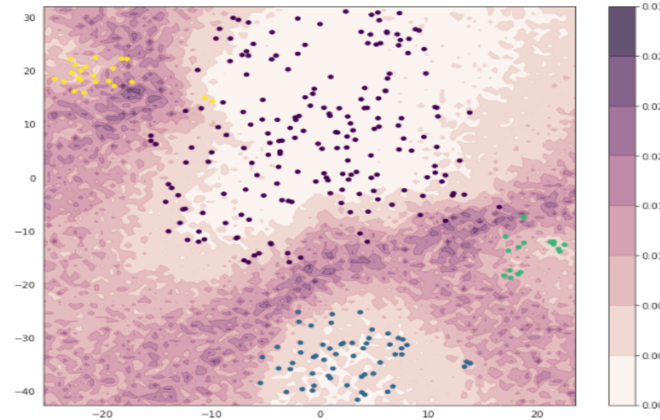


Bayesian Deep Learning for single cell









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

Uncertainties are critical for Clinical Diagnostics

NBIS

National Bioinformatics Infrastructure Sweden (NBIS)

SciLifeLab

Knut och Alice Wallenbergs Stiftelse

LUNDS UNIVERSITET