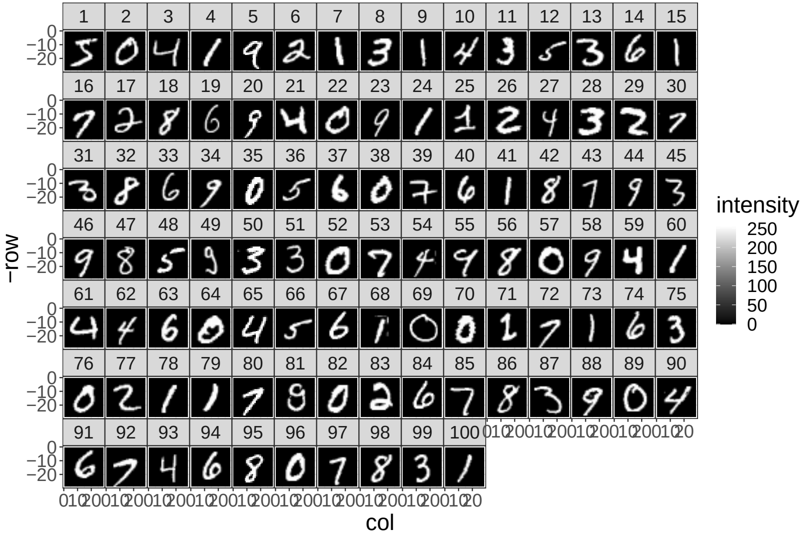
Auto-encoders

Toby Dylan Hocking

Motivation: MNIST digits data



Set of digits is represented as a matrix

- Each digit image in MNIST data set is a matrix of 28×28 pixel intensity values, $x_i \in \{0, \ldots, 255\}^{784}$.
- Each of the images is a row in the data matrix.
- Each of the columns is a pixel.
- All images on last slide represented by a data matrix with $n = 100$ rows/images and $p = 784$ columns/pixels.

Background/motivation: non-linear dimensionality reduction

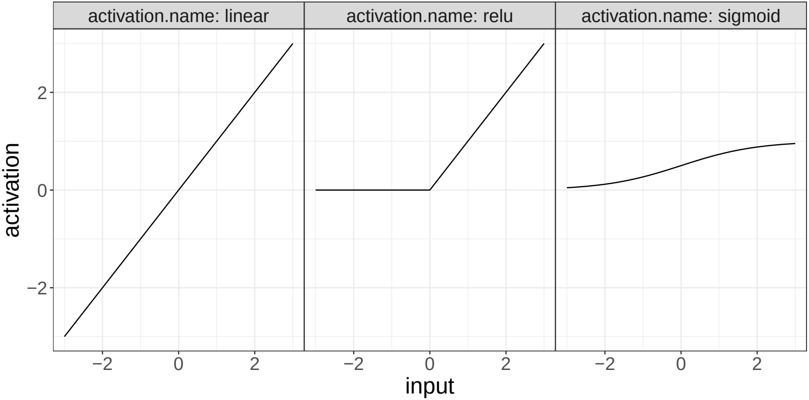
- High dimensional data are difficult to visualize.
- For example each observation/example in the MNIST data is of dimension 28 x 28 = 784 pixels.
- We would like to map each observation into a lower-dimensional space for visualization / understanding patterns in the data.
- Principal Components Analysis (PCA) is a linear dimensionality reduction method, which is computed using the Singular Value Decomposition (SVD).
- Auto-encoders are non-linear, which means they can be more accurate than PCA, in terms of reconstruction error.

Auto-encoders are a type of neural network

- A neural network with L layers is a function $f(x) = f_{L-1}[...f_1(x)]$.
- Each function $f_l(z) = \sigma_l(W_l z)$ consists of multiplication by a matrix W_l followed by an activation function σ_l.
- The number of layers L, the sizes of the weight matrices W_l, and the activation functions σ_l are all hyper-parameters that must be chosen prior to learning.
- Number of units/features in each layer determines weight matrix sizes. To compute u_{l+1} units from u_l units the W_l matrix must be of size $u_{l+1} \times u_l$.
- The name "auto" is not an abbreviation of automatic; it means that the input feature vector x is also used as the output.
- Auto-encoders have a middle "code" layer which is the low dimensional embedding (typically size 2 for visualization), and intermediate layer sizes are typically symmetric, for example $L = 5$ layers for a data set with $p = 100$ features (100,50,2,50,100).

Activation function choices

- linear: $\sigma(z) = z$.
- relu: $\sigma(z) = z$ if $z \geq 0$ else 0.
- sigmoid: $\sigma(z) = 1/(1 + e^{-z})$.



Auto-encoder learning algorithm

- The goal of learning is to find a low dimensional mapping of the data which is able to reconstruct the original data.
- This is measured by the mean squared reconstruction error, $\text{MSE}(f) = \sum_{i=1}^{n} [f(x_i) - x_i]^2$.
- The values in the weight matrices W_l are the model parameters which are learned using the Stochastic Gradient Descent (SGD) algorithm.
- The batch size hyper-parameter is the number of observations for which the MSE and its gradient are computed and summed during each iteration (step or update to weight matrices).
- Each iteration of SGD updates the weight matrices W_l in order to get better predictions (reduce MSE).
- An "epoch" involves one or more gradient descent iterations (computes gradient with respect to each observation once).

Details of Stochastic Gradient Descent (SGD)

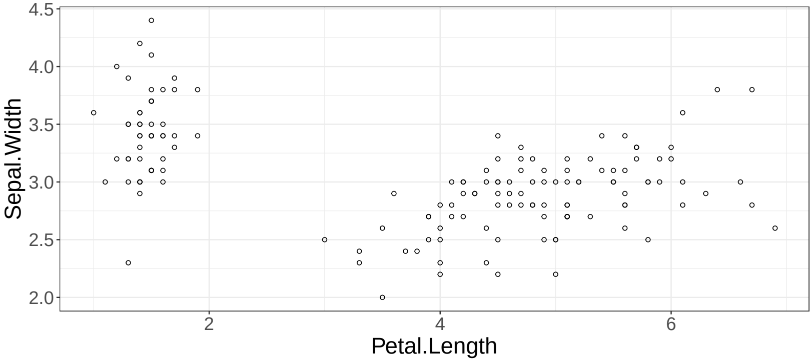
- The algorithm starts with arbitrary/random weight matrices W_l close to zero.
- The update (one step/iteration) is $W \leftarrow W - \alpha G$ where W are the weights, $\alpha > 0$ is a learning rate/step size hyper-parameter, and G is the gradient.
- The gradient is the direction of steepest descent, so the loss/MSE is guaranteed to decrease if the step size is small enough.
- But if the step size is too small then many iterations are required to get a small loss/MSE (too slow).
- If step size is too big then loss/MSE can increase, so you want to choose an intermediate step size; best step size depends on the problem and data.

Example: 2d iris data

- Simple example: iris.
- One row for each flower (only 6 of 150 shown below).
- One column for each measurement/dimension.

```
##   Sepal.Width Petal.Length
## 1         3.5          1.4
## 2         3.0          1.4
## 3         3.2          1.3
## 4         3.1          1.5
## 5         3.6          1.4
## 6         3.9          1.7
```

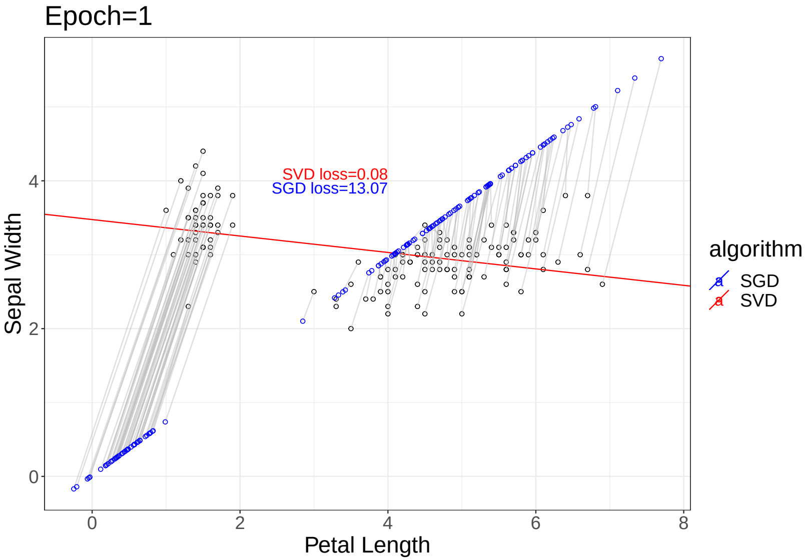
Example: 2d iris data



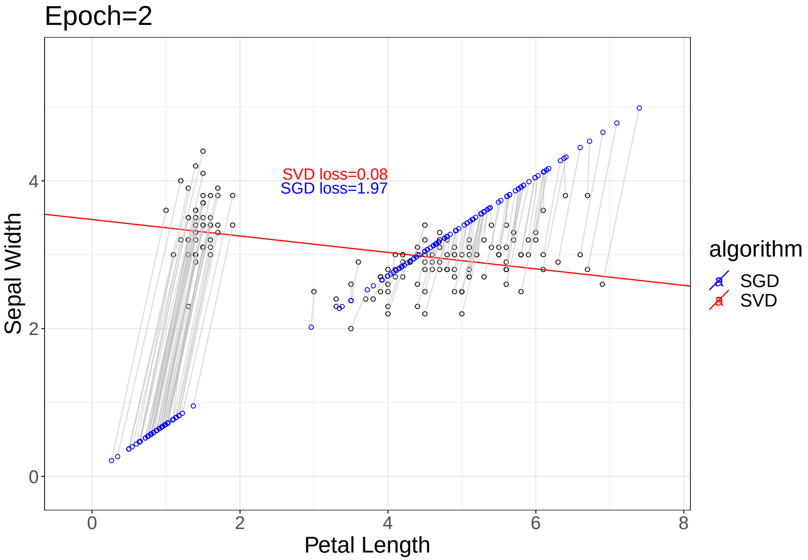
Auto-encoder neural network architecture

- In this example the number of units in each layer is (2, 1, 2).
- Input/output layers have two units.
- Code layer has one unit.
- First function has two weights $W_1 \in \mathbb{R}^{1\times 2}$.
- Second function has two weights $W_2 \in \mathbb{R}^{2\times 1}$.
- Linear activation function, so same model as PCA: low-dimensional embedding is a linear combination of input features.
- Learning algorithm iteratively searches for best linear model.

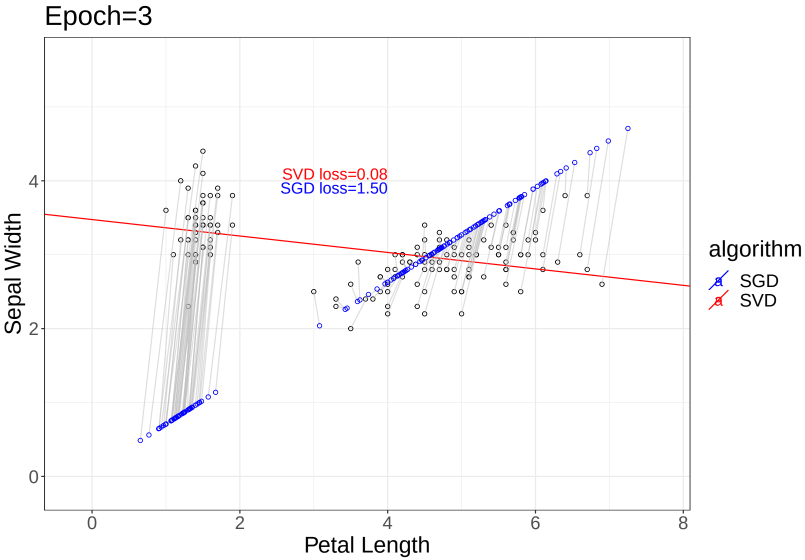
Visualization of predicted values



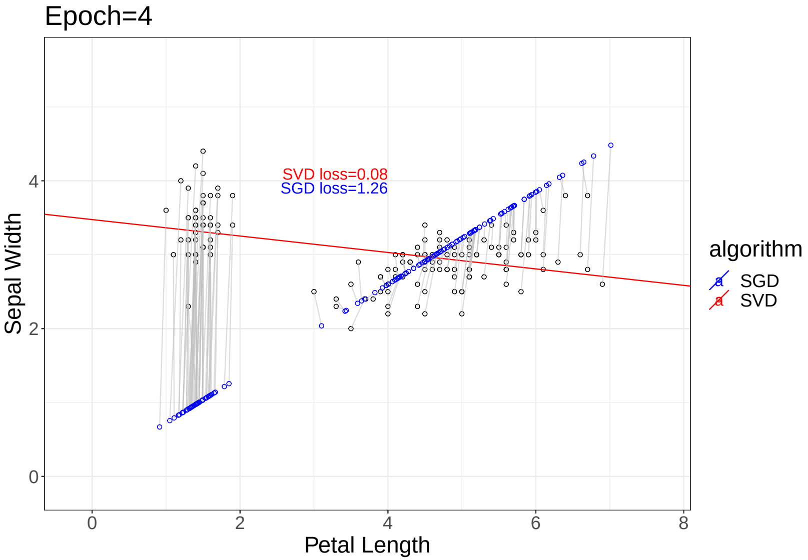
Visualization of predicted values



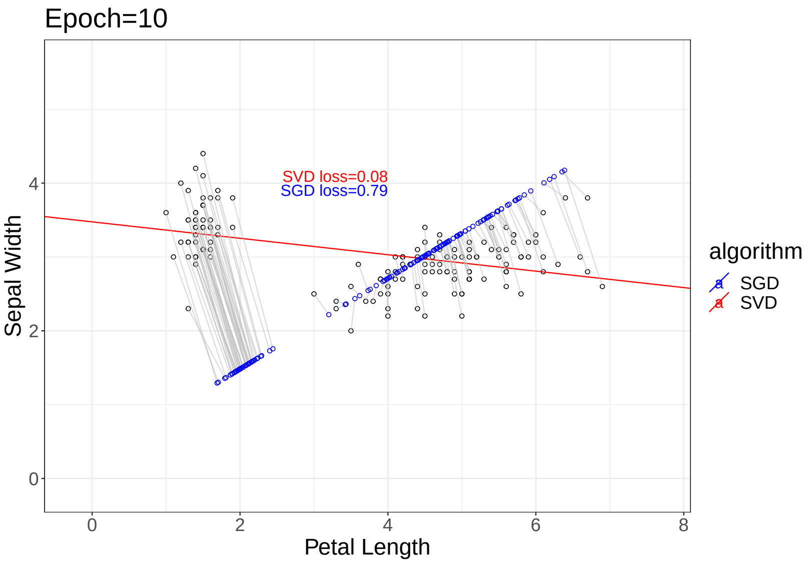
Visualization of predicted values



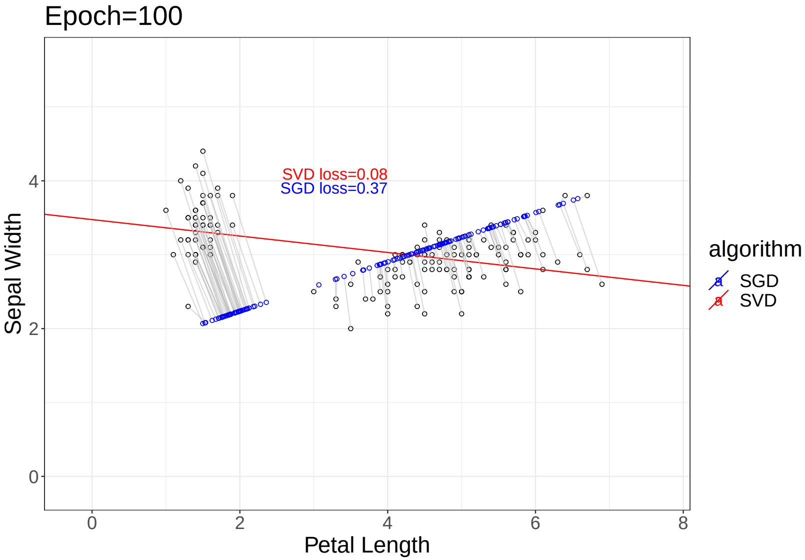
Visualization of predicted values



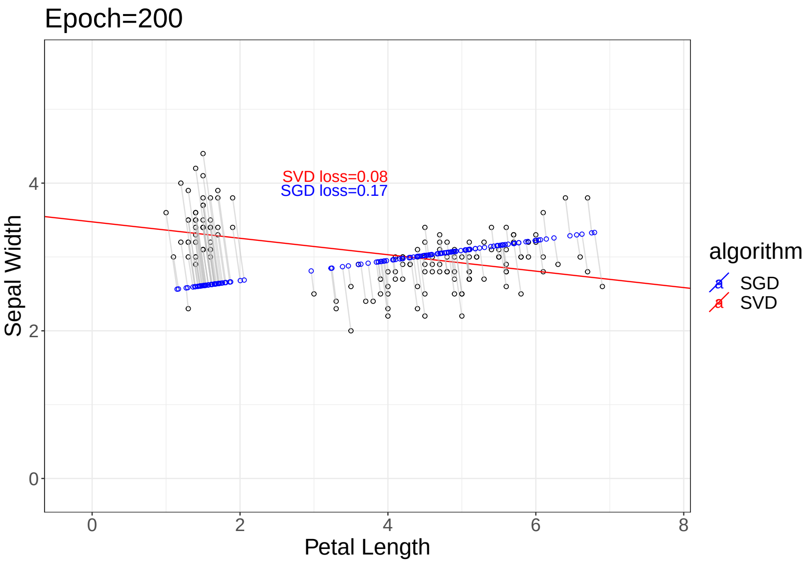
Visualization of predicted values



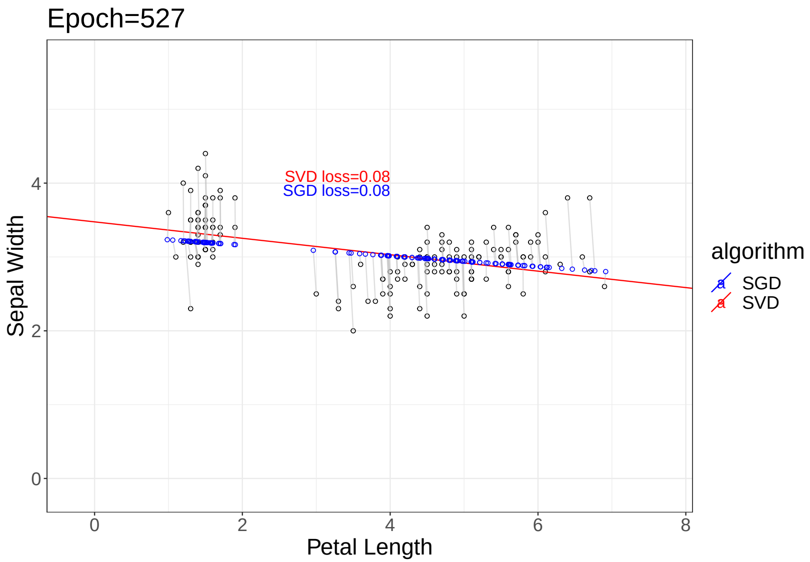
Visualization of predicted values



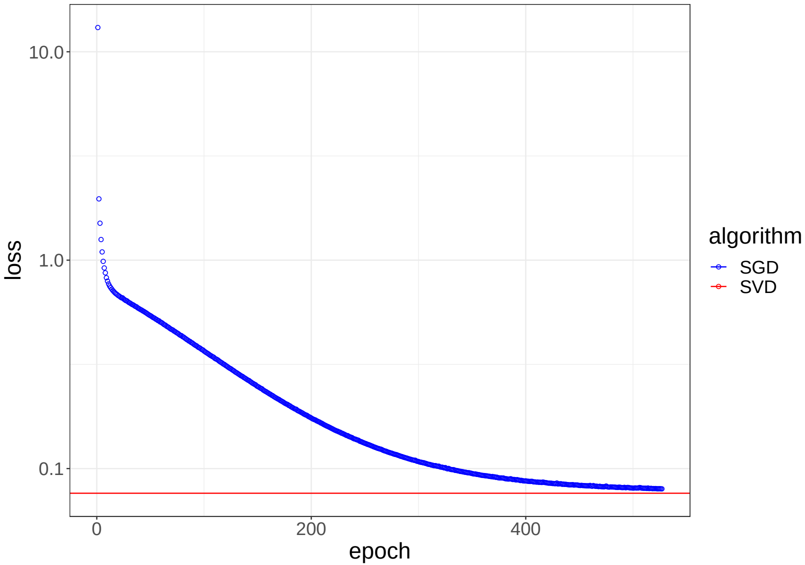
Visualization of predicted values



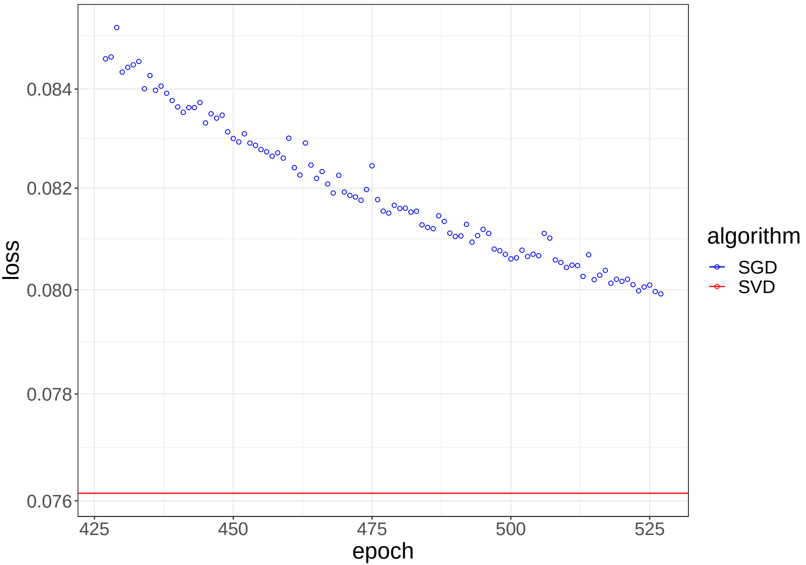
Visualization of predicted values



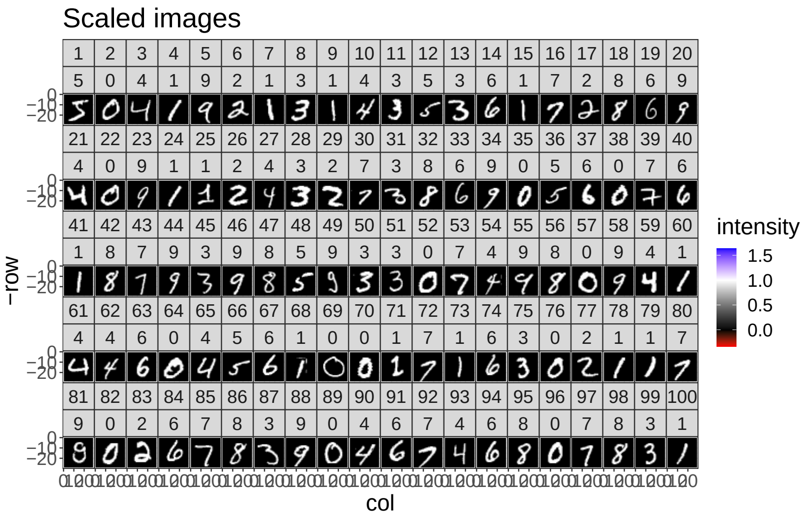
Loss decreases with number of epochs



Zoom to last 100 epochs



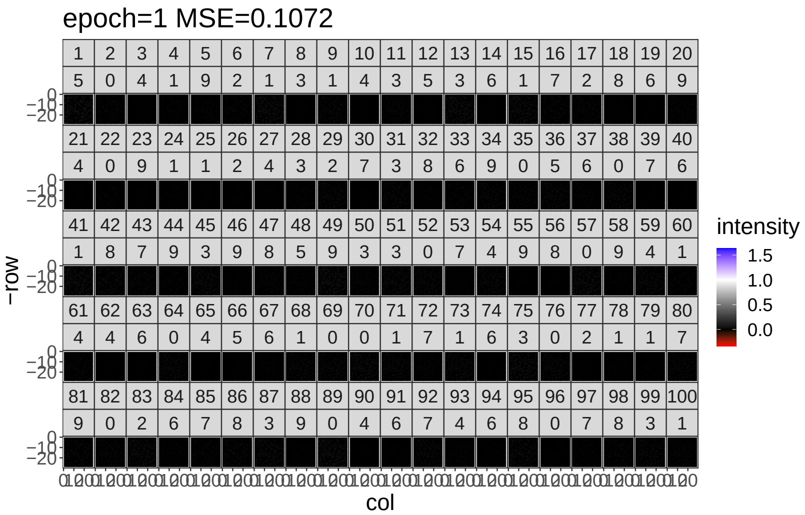
Actual image data



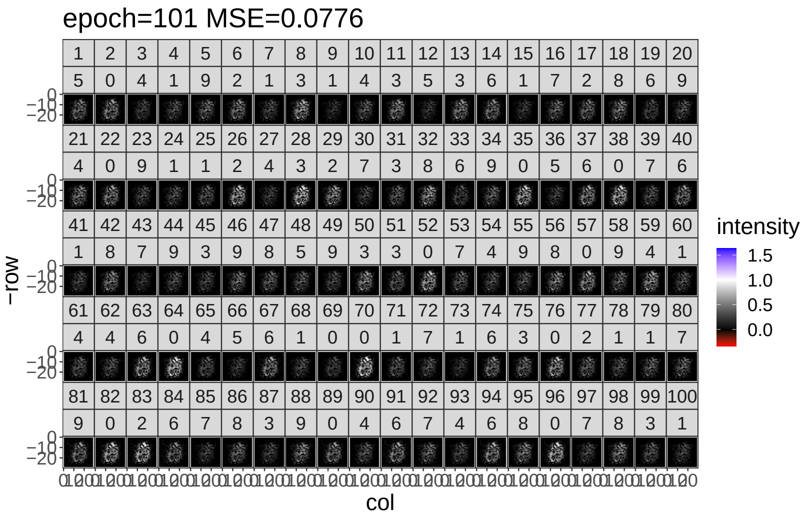
Auto-encoder for image data

- Each image is represented by a vector of 784 pixel intensity values, so this is the number of units in the first/last layer.
- The code layer will have 2 units for visualization purposes (two axes on a scatterplot).
- There is a choice of the number of intermediate layers; here we choose one layer with 100 units (on each side of the code layer).
- Overall model architecture, in terms of number of units/features per layer, is (784,100,2,100,784).
- Weight matrix sizes are therefore $W_1 \in \mathbb{R}^{100\times784}, W_2 \in \mathbb{R}^{2\times100}, W_3 \in \mathbb{R}^{100\times2}, W_4 \in \mathbb{R}^{784\times100}$.
- To low-dimensional embedding for an image x is computed via $f_2[f_1(x)] = \sigma_2[W_2\sigma_1(W_1x)]$.

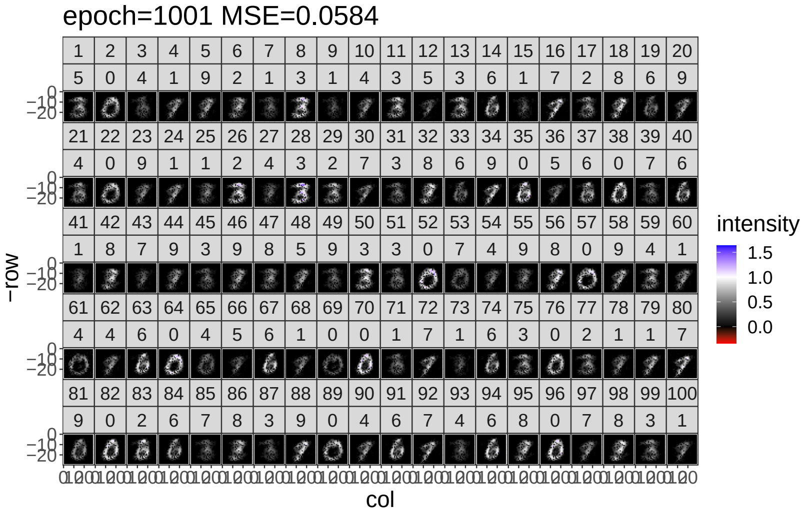
Reconstruction improves with epochs of learning



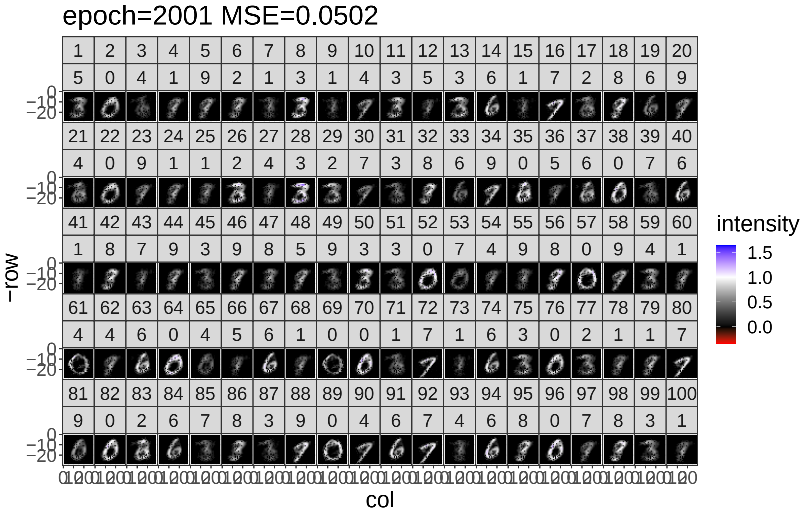
Reconstruction improves with epochs of learning



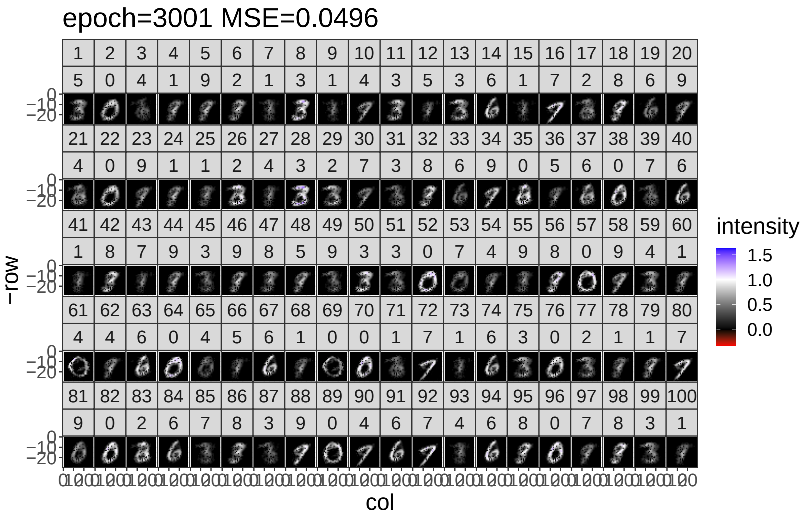
Reconstruction improves with epochs of learning



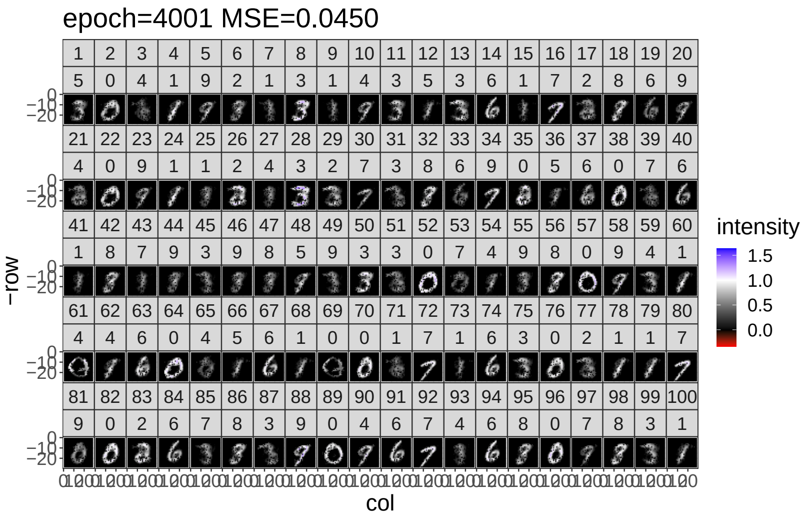
Reconstruction improves with epochs of learning



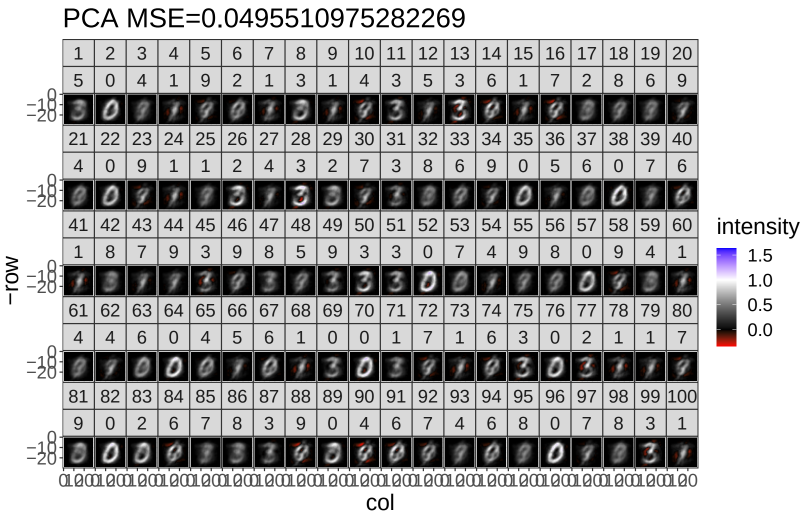
Reconstruction improves with epochs of learning



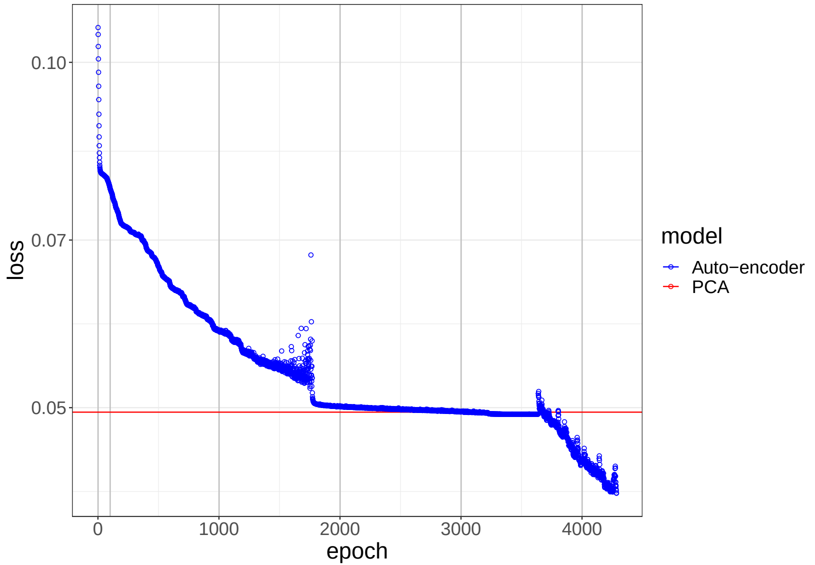
Reconstruction improves with epochs of learning



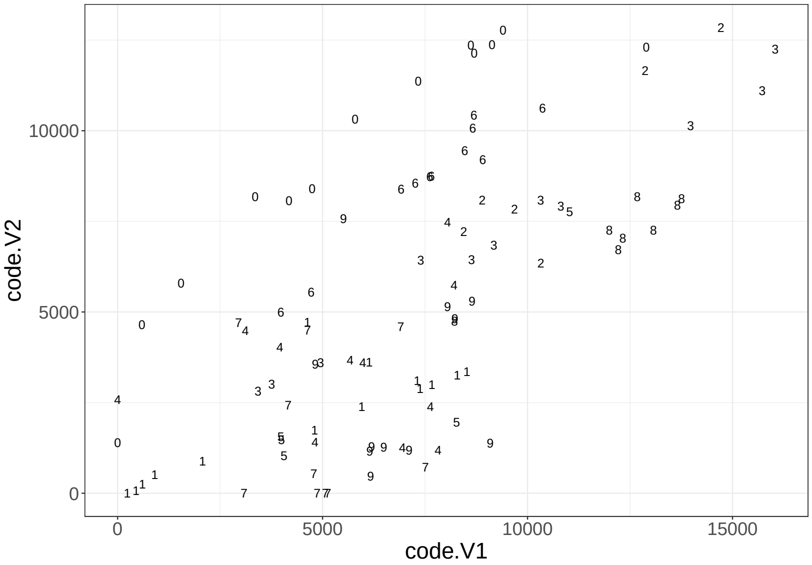
Reconstruction of PCA



Loss versus number of epochs



Plot code layer variables instead of PCs



Possible exam questions

- What choices do you need to make in the auto-encoder in order to have the result be the same as PCA?
- What is the total number of parameters for an auto-encoder of a data set with $p = 100$ features, if we use 2 code units and 10 intermediate units? (assume only weight matrices, no bias/intercept to learn)