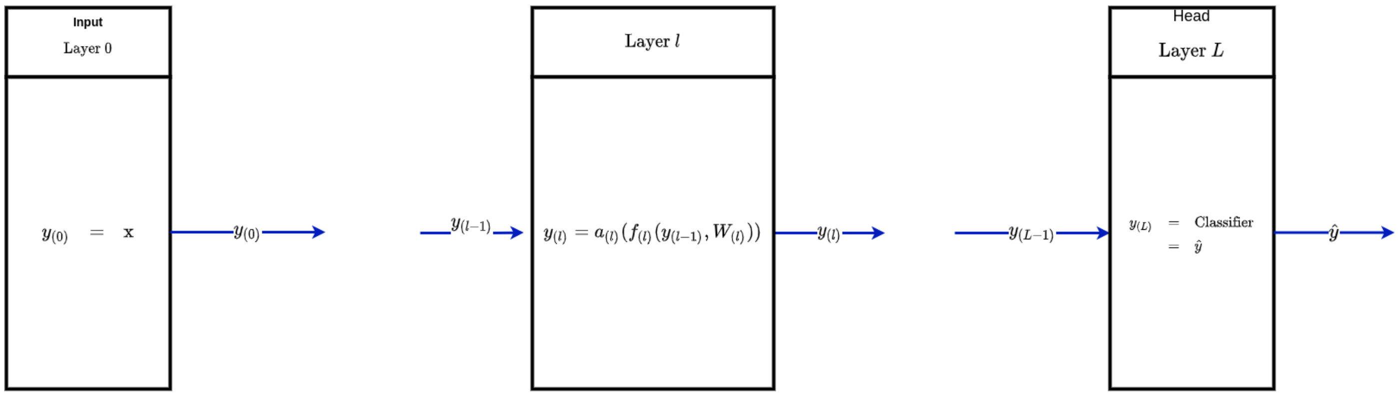
Interpreting Representations: Preview

We have described an L layer (Sequential) Neural Network as

- a sequence of transformations of the input
    - each transformation a *layer* $1 \le l \le (L-1)$, producing a new *representation* $\mathbf{y}_{(l)}$
- that feed the final representation $\mathbf{y}_{(L-1)}$ to a *head* (classifier, regressor)

Layers


Input
Layer 0
$y_{(0)} = \mathbf{x}$
$y_{(0)}$
$y_{(l-1)}$
Layer l
$y_{(l)} = a_{(l)}(f_{(l)}(y_{(l-1)}, W_{(l)}))$
$y_{(l)}$
$y_{(L-1)}$
Head
Layer L
$y_{(L)} = \text{Classifier}$
$= \hat{y}$
$\hat{y}$

Is it possible to *interpret* each representation $\mathbf{y}_{(l)}$?

- What do the new "synthetic features" mean ?
- Is there some structure among the new features ?
    - e.g., does each feature encode a "concept"

We will briefly introduce the topic of Interpretation.

A deeper dive will be the subject of a later lecture.

Our goal, for the moment, is to motivate Autoencoders.

Interpretation: Examine the weights

Perhaps the most obvious may to obtain insight into the working of a Neural Network is to examine the weights.

- When the weights are used in a dot product
- They can be interpreted as "patterns" that a layer is trying to match

The linear models of Classical Machine Learning motivate this idea.

Linear Regression

- $\hat{\mathbf{y}} = \mathbf{\Theta}^T \cdot \mathbf{x}$
- Prediction $\hat{\mathbf{y}}$, given features $\mathbf{x}$, is linear in parameters Θ.

Logistic Regression

- $\hat{\mathbf{s}} = \mathbf{\Theta}^T \cdot \mathbf{x}$
- Score $\hat{\mathbf{s}}$, which is turned into a probability via the sigmoid function σ
$$\hat{\mathbf{p}} = \sigma(\hat{\mathbf{s}})$$
  is linear in $\mathbf{\Theta}$

Let's examine the role of Θ_j in the dot product.

Consider one *numeric* feature $\mathbf{x}_j^{(\mathbf{i})}$ for example i.

- A unit increase in $\mathbf{x}_j^{(\mathbf{i})}$
- Holding constant the values for all other features,
- Increases $(\Theta^T \cdot \mathbf{x}^{(\mathbf{i})})$ by Θ_j

So Θ_j may be interpreted as the sensitivity of the dot product to a unit change in feature j

$$\Theta_j = \frac{\partial}{\partial \mathbf{x}_j}(\Theta^T \cdot \mathbf{x})$$

That is: how much does the prediction or score depend on the value of the feature.

Suppose instead that $\mathbf{x}_j$ corresponds to the binary feature (indicator/dummy variable)

- Is c_1

Then the dot product formula indicates that

- Θ_j is the *increment* to $(\Theta^T \cdot \mathbf{x})$
- Arising from $\mathbf{x}_j^{(\mathbf{i})} = 1$
- Compared to $\mathbf{x}_j^{(\mathbf{i})} = 0$

That is: how much the presence of feature $\mathbf{x}_j$ increases the prediction or score.

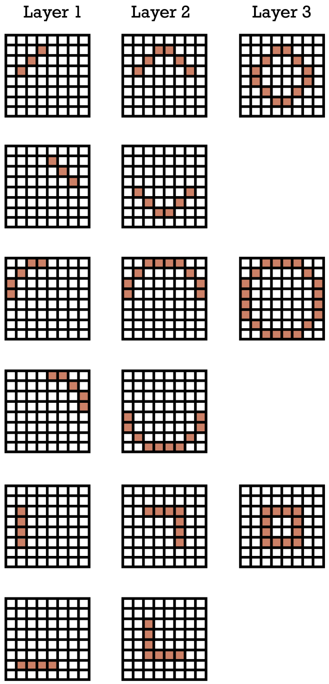
This idea is even more appealing when the original input $\mathbf{x^{(i)}}$ is an image.

- We may be able to relate weights to recognizable sub-images of the input

In Convolutional Layers, there is some evidence that

- The first layer recognizes features (matches patterns) for *primitive* concepts
- The second layer recognizes features that are *combinations* of primitive concepts (layer 1 concepts)
- The l recognizes features that are *combinations* of layer $(l-1)$ concepts

Features by layer



Although simple, it may be naive to hope that this technique will provide insight into multi-layer Neural Networks

- The layers $1 \le l \le (L-1)$ preceding the head Regression/Classification layer L
- Are *transforming* input $\mathbf{x}$ into synthetic features $\mathbf{y}_{(L-1)}$
- That are extremely useful for prediction
- But which may no longer be interpretable

For example

- Do we recognize the digit "0"
- Because of interpretable features like the doughnut shape
- Or because of the *ratio* of dark to light pixels ?

We will make further attempts at interpretability that work

- *Not* by interpreting the weights
- Instead: by finding groups of inputs
- And relating them to synthetic features in some layer

Interpretation: Clustering of examples

One way to try to interpret $\mathbf{y}_{(l)}$ is relative to a dataset
$\langle \mathbf{X}, \mathbf{y} \rangle = \{ \mathbf{x^{(i)}}, \mathbf{y^{(i)}} | 1 \le i \le m \}$

By passing each example $\mathbf{x^{(i)}}$ through the layers to obtain $\mathbf{y}_{(l)}^{\mathbf{(i)}}$

- We create a mapping from examples to layer l representations
$$\langle \mathbf{X}, \mathbf{y}_{(l)} \rangle = \{ \mathbf{x^{(i)}}, \mathbf{y}_{(l)}^{\mathbf{(i)}} \mid 1 \le i \le m \}$$

Mapping inputs to layer l representations

Let's create a scatter plot of each example's representation $\mathbf{y}_{(l)}^{(\mathbf{i})}$

- In $n_{(l)}$-dimensional space
- Labeling each point
- With the target $\mathbf{y}^{(\mathbf{i})}$
- Or with a set of input attributes, e.g., $(\mathbf{x}_j^{(\mathbf{i})}, \mathbf{x}_{j'}^{(\mathbf{i})})$

Perhaps clusters of examples will appear.

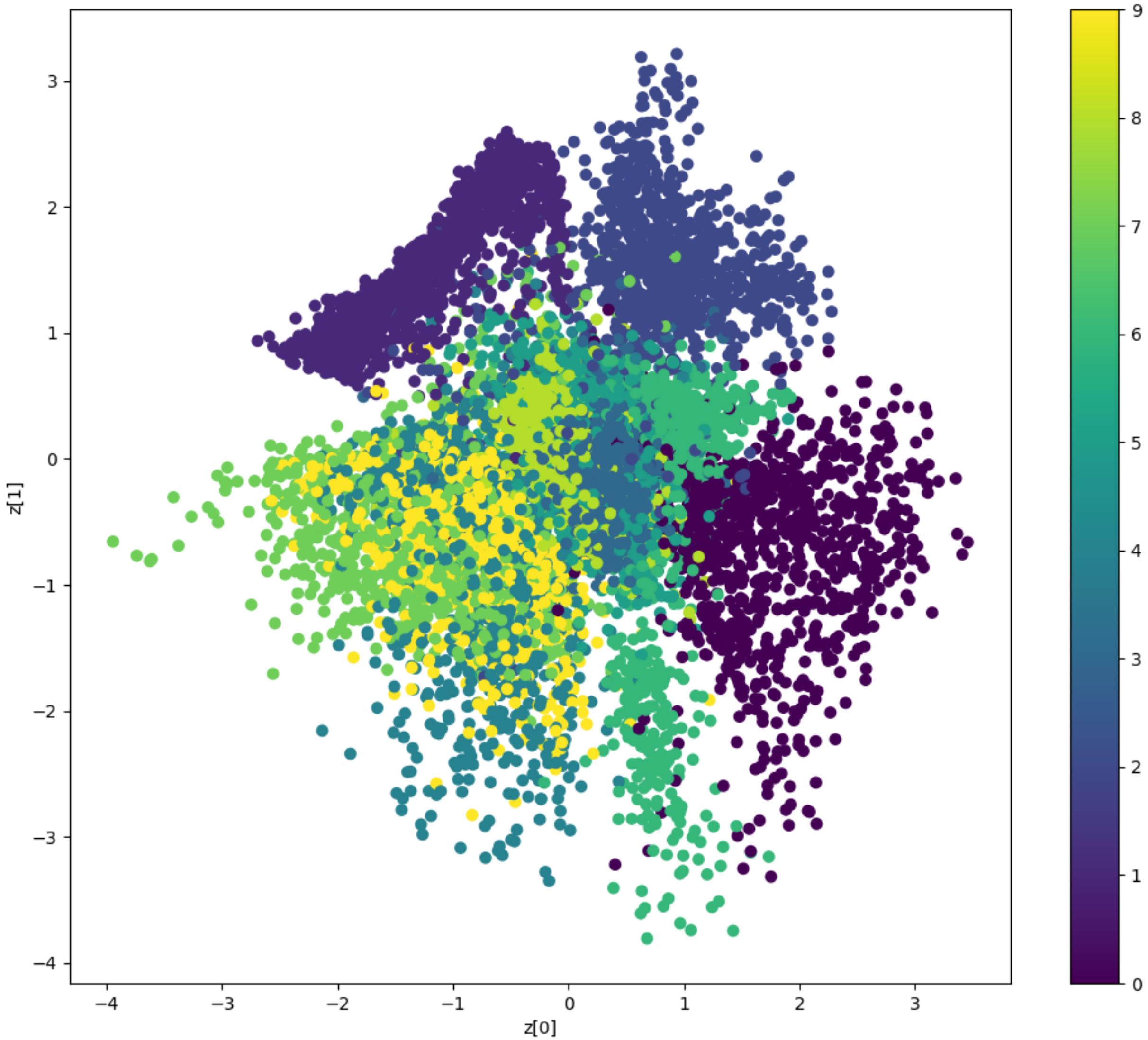
If all points in the cluster have the same label

- We might be able to identify the representation with a target or set of input features

Here is an example of the representation of the MNIST digits in an intermediate layer of a particular network

- The output of the Encoder half of an Autoencoder
- Which we will study in a subsequent lecture

MNIST clustering produced by a VAE



z[1]
z[0]
3
2
1
0
−1
−2
−3
−4
−4
−3
−2
−1
0
1
2
3
9
8
7
6
5
4
3
2
1
0

- Each point is an example $\mathbf{x^{(i)}}$
- With coordinates chosen from two of the synthetic features in $\mathbf{y}_{(l)}$
- The color corresponds to the label $\mathbf{y^{(i)}}$ (i.e., the digit that is represented by the image)

You can see that some digits form tight clusters.

By understanding

- The commonality of examples within a cluster
- How the digit label's vary as a synthetic feature varies

we might be able to infer meaning of the synthetic features.

The first two synthetic features in $\mathbf{y}_{(l)}$ of MNIST may correspond to properties of those digits

- digits with "tops"
- digits with "curves"

Note

This is not too different from trying to interpret Principal Components.

Interpretation: Examining the latent space

Suppose we could *invert* the representation $\mathbf{y}_{(l)}$ to obtain a value $\mathbf{x}$ that lies in the input domain.

Then

- By perturbing individual synthetic features $\mathbf{y}_{(l),j}$ in a given representation
    - Perturb $\mathbf{y}_{(l)}$ to obtain $\mathbf{y}'_{(l)}$
- And examining the effect on the inverted value $\mathbf{x}'$
- We might be able to assign meaning to the layer l feature $\mathbf{y}_{(l),j}$

Note that the inverted value $\mathbf{x}'$ **is not necessarily** (and probably not) a value in training set $\mathbf{X}$!

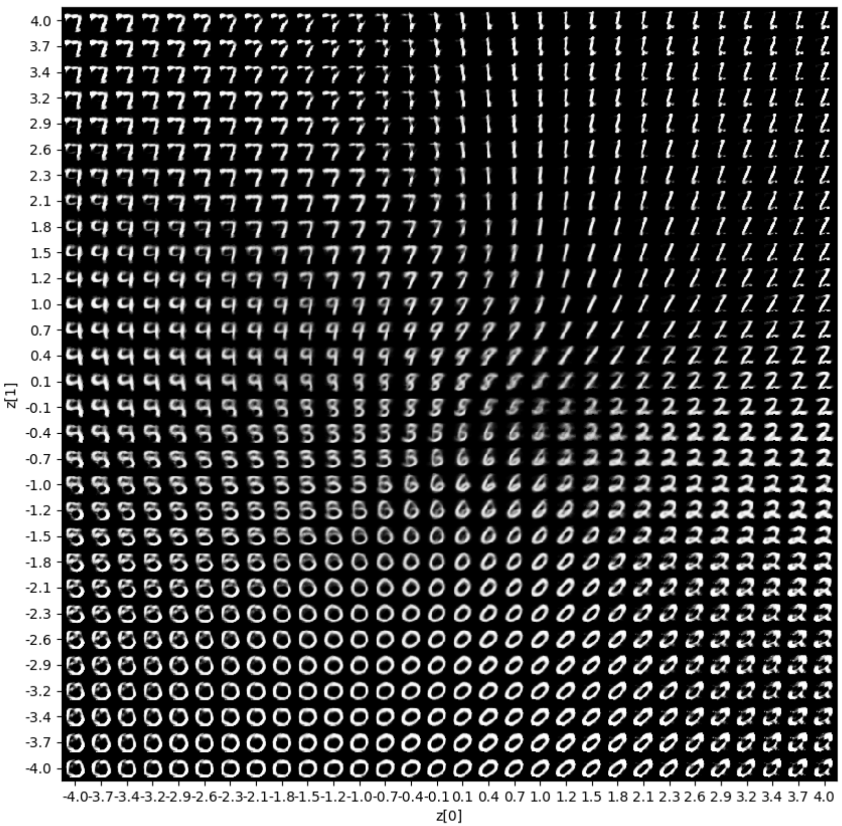
- It is merely a value obtained by the mathematical inversion of a function
- Especially since the perturbed $\mathbf{y}'$ may not be the mapping of any example $\mathbf{x^{(i)}} \in \mathbf{X}$

Invert layer l representation

Here are the inverted images obtained by perturbing two synthetic features in $\mathbf{y}_{(l)}$

- Horizontal axis perturbs one feature
- Vertical axis perturbs a second feature

MNIST clustering produced by a VAE



Some observations (with possible interpretation)

- Does the synthetic feature on the horizontal axis control slant ?
    - Examine 0's along bottom row
- Does the synthetic feature on the vertical axis control "curviness" ?
    - Examine the 2's column at the right edge, from bottom to top

There is *no reason to expect* that the inversion of an arbitrary representation *looks like* a digit but it does !

Perhaps

- The mapping from inputs to representations is such that similar inputs have very similar representations
- Or we impose some constraints on the inversion to force the inverted value to look like a digit

In order for this method to work, we must be able to *invert* $\mathbf{y}_{(l)}$.

We will show how to do this in a later lecture.

Deja vu: have we seen this before ?

These two methods of interpretation have been encountered in an earlier lecture

- mapping original features $\mathbf{x}^{(\mathbf{i})}$ to synthetic features $\tilde{\mathbf{x}}^{(\mathbf{i})}$
- inverting synthetic feature $\tilde{\mathbf{x}}^{(\mathbf{i})}$ to obtain original feature $\mathbf{x}^{(\mathbf{i})}$

Principal Component Analysis (PCA) !

PCA is an Unsupervised Learning task that can be used for

- dimensionality reduction
- clustering

The key to it's intepretability was the simplicity of transforming and inverting

$$\begin{aligned}
\mathbf{X} &= U\Sigma V^T && \text{SVD decomposition of } \mathbf{X} \\
\tilde{\mathbf{X}} &= \mathbf{X}V && \text{transformation to synthetic features} \\
\mathbf{X} &= \tilde{\mathbf{X}}V^T && \text{inverse transformation to original features}
\end{aligned}$$

The transformation V via matrix multiplication is *linear*.

We will explore *non-linear*, *invertible* transformations during our study of Autoencoders.

Conclusion

Neural Networks have the reputation of being magical but opaque.

We hope this brief introduction to interpretation provides some hope that we can understand their inner workings.

A separate lecture will explore this topic in greater depth.

In [4]:

```python
print("Done")
```

```
Done
```