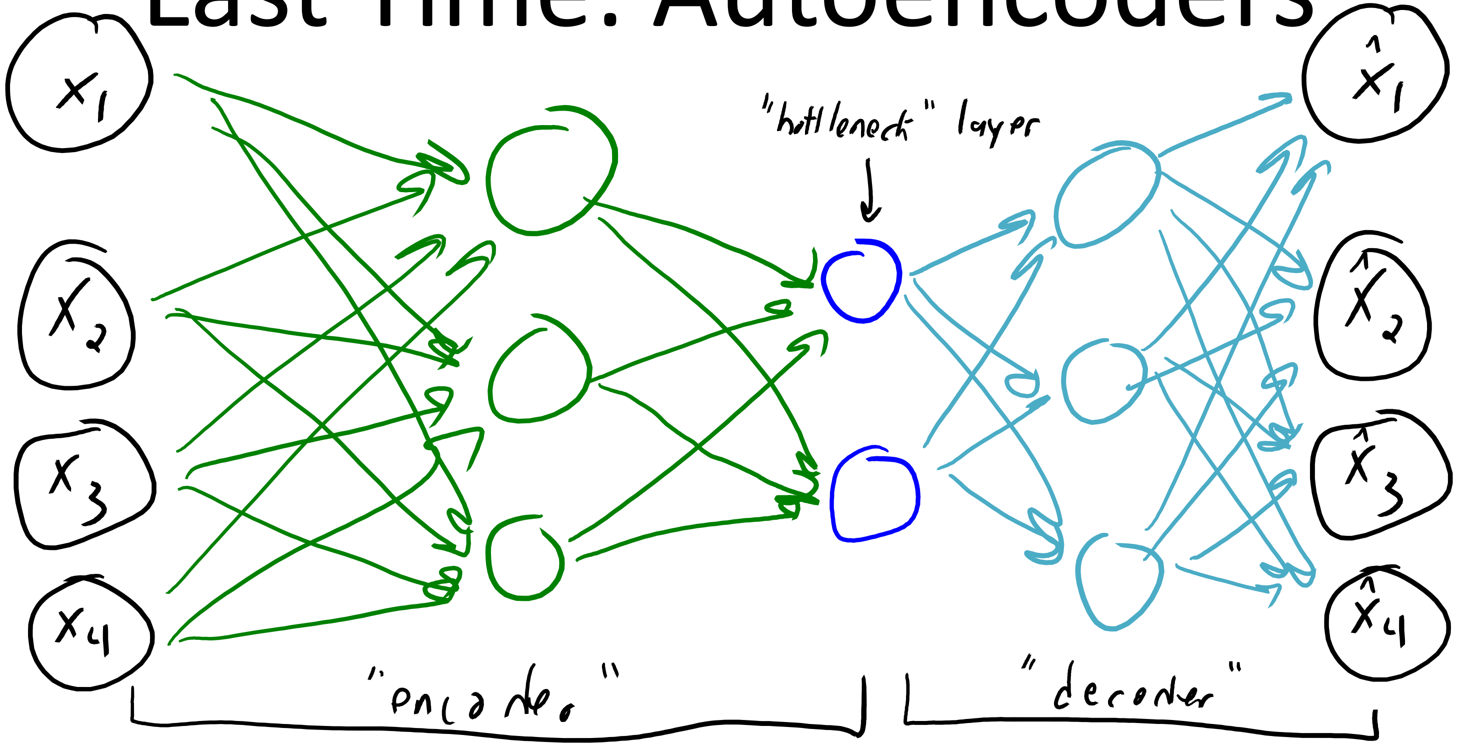
# CPSC 440: Machine Learning

Autoencoders

Winter 2022

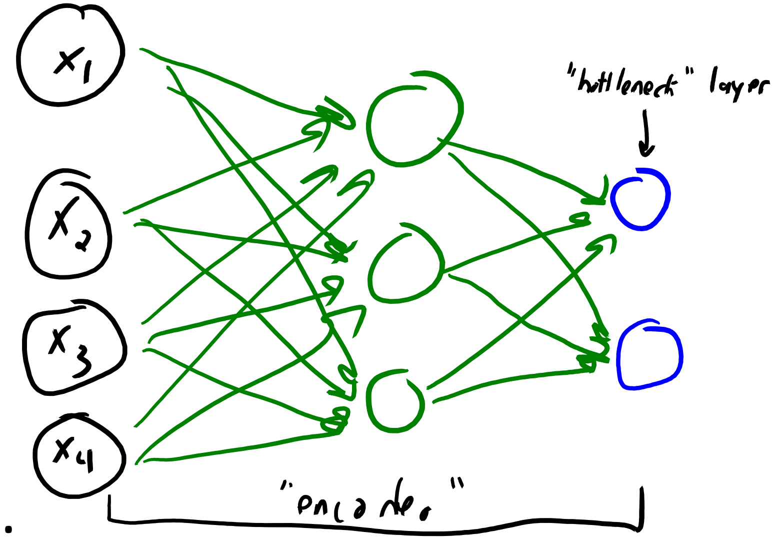
# Last Time: Autoencoders




- Autoencoders are neural networks with same input and output.
  - Includes a bottleneck layer: with dimension 'k' smaller than input 'd'.
  - First layers "encode" the input into bottleneck.
  - Last layers "decode" the bottleneck into a (hopefully valid) input.
- This is unsupervised, and is a non-linear generalization of PCA.

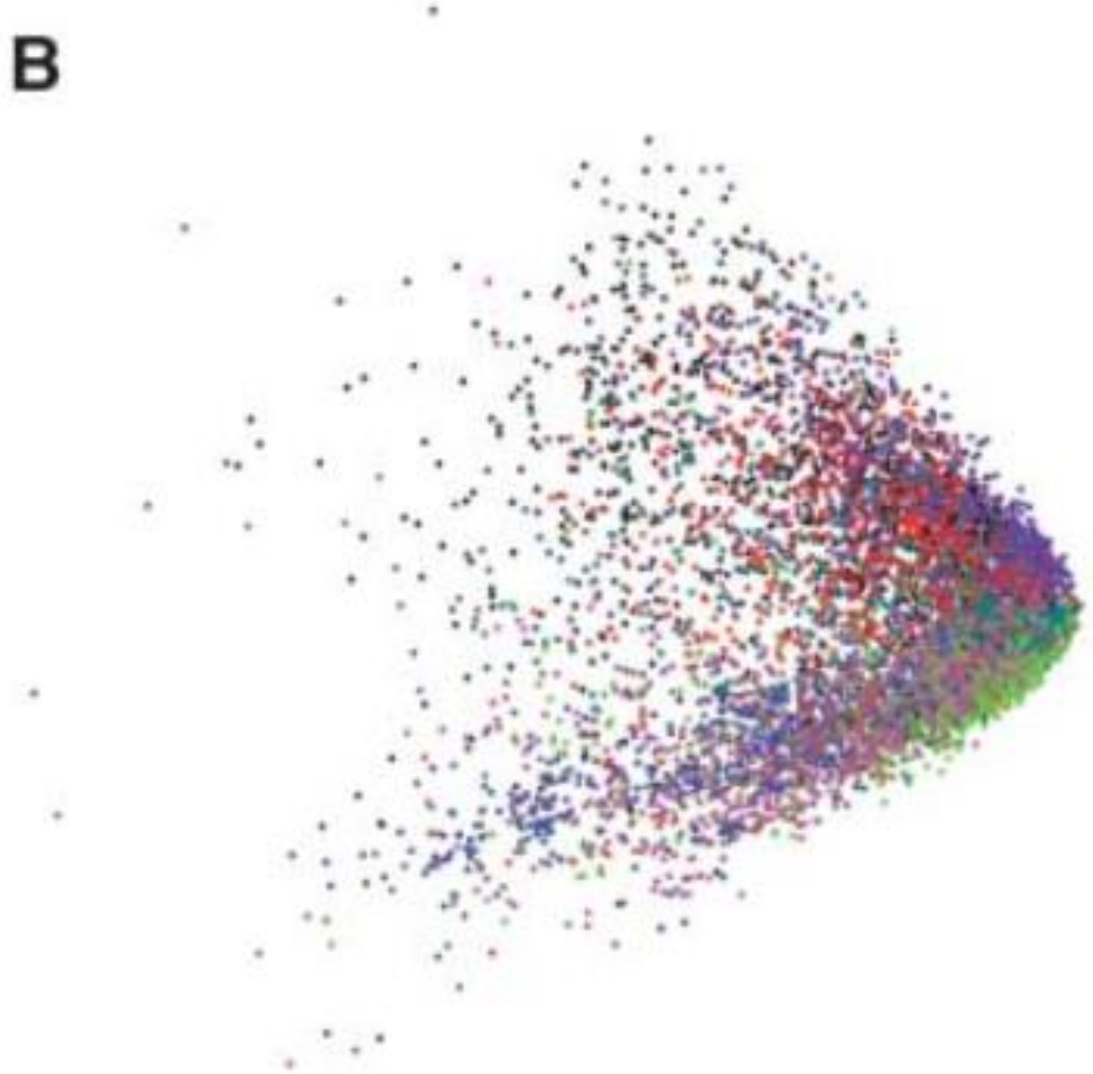
  (I am not showing the 'h' functions to keep the diagram simple.)

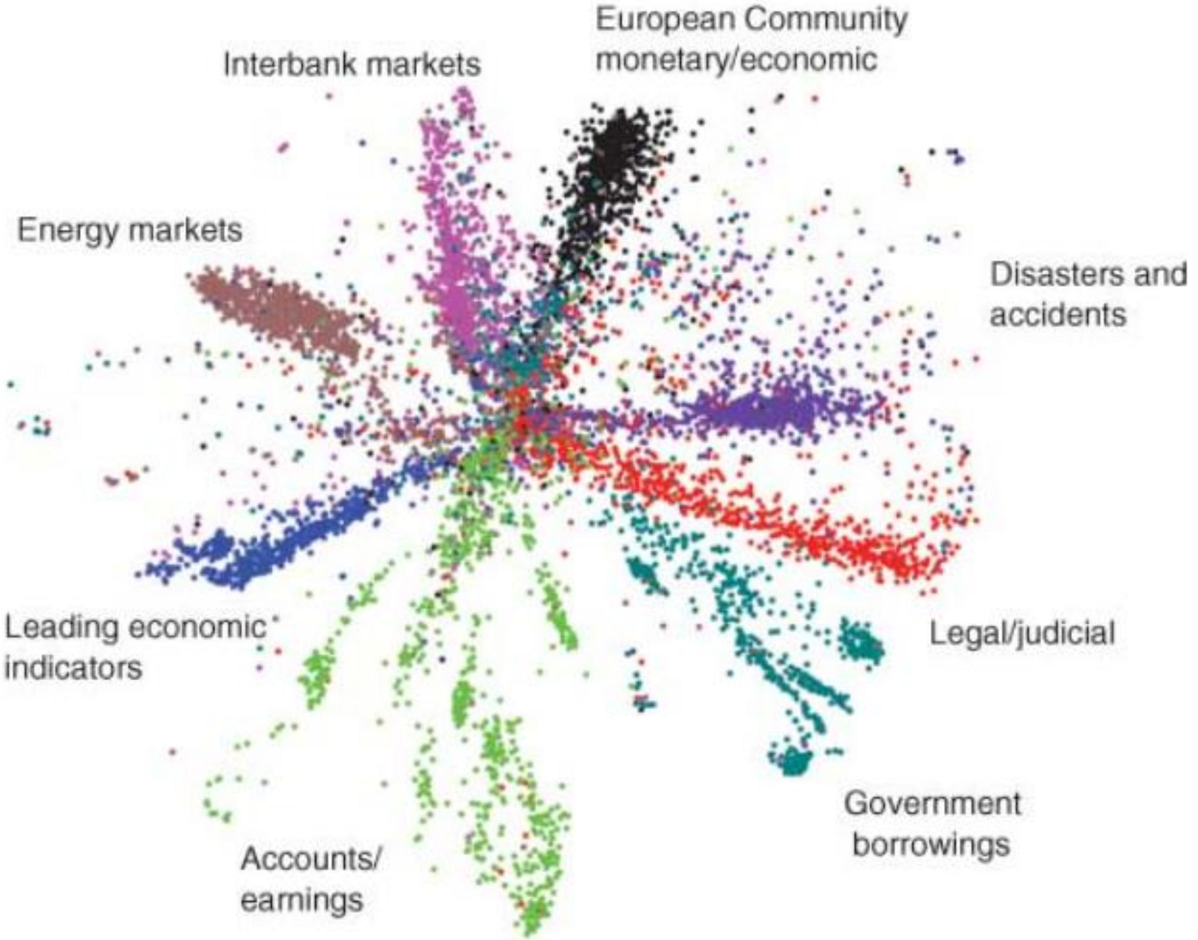
# Encoder as Learning a Representation




- Consider the encoder part of the network:
  - Takes features '$x^i$' and makes low-dimensional '$z^i$'.
- Ways you could use the encoder:
  - Use $z^i$ as compressed input (reduce memory needed).
  - Set bottleneck size to 2, and plot the $z^i$ to visualize the data.
  - Try to interpret what the bottleneck features $z^i$ mean.
  - Use the $z^i$ as features for supervised learning.
    - For the special case of PCA and regression with L2 loss, this is called "partial least squares".
  - You could add a supervised '$y^i$' to final layer of trained autoencoder, fit with SGD.
    - This is called "unsupervised pre-training".
    - If you use unlabeled data to do this initialization, an example of "self-supervised" learning.
      - Usually it is easier to get a lot of unlabeled data than it is to get labeled data.

# PCA vs. Deep Autoencoder (Document Data)



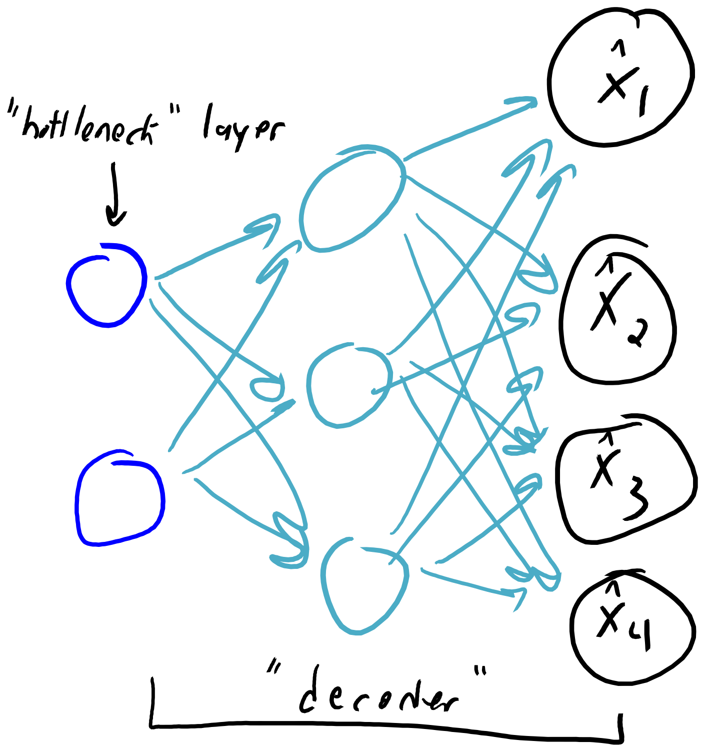




https://www.cs.toronto.edu/~hinton/science.pdf

(these days I would recommend t-SNE for making visualizations like this)

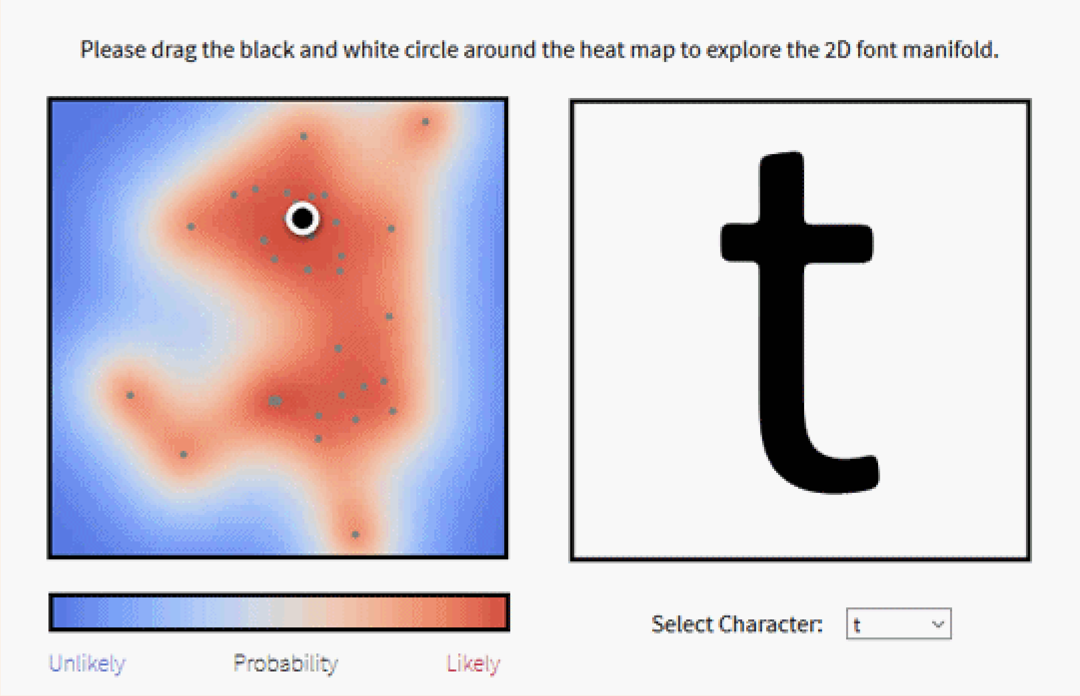
# Decoder as Generative Model

- Consider the decoder part of the network:
  - Takes low-dimensional '$z^i$' and makes features '$\hat{x}^i$'.
- Can be used for outlier detection:
  - Check distance to original features to detect outliers.
- Can be used to generate new data:
  - The 'z' close to training examples should generate new valid samples.
  - But this is not density estimation, since we are not modeling p(z) yet.



# Font Manifold

- Going from encoding to decoding for different fonts:



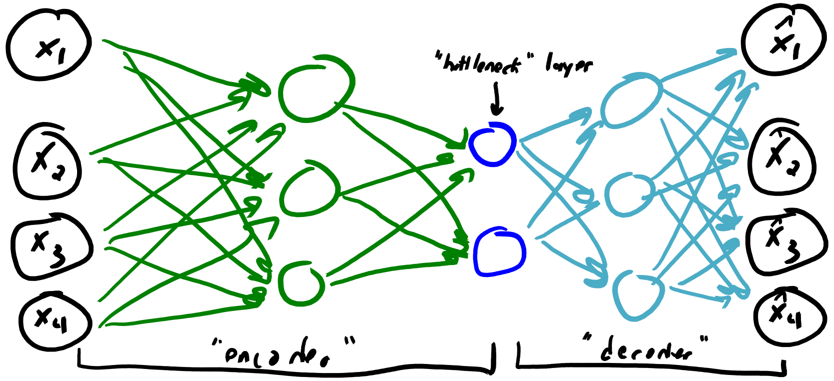

- Demo here.
  - The above was generated by a Gaussian process and not an autoencoder.
  - But the decoder part of autoencoders is trying to do something like this.

http://entangled.systems/fragments/20160729-learning-a-manifold-of-fonts-machine-learning-research-from-2014-by-dr-neill-campbell-provides-an-interactive-exploration-of.html

# Neural Networks with Multiple Outputs

- Previous neural networks we have seen only have 1 output 'y'.
- In autoencoders, we have 'd' outputs (one for each feature).

$$\left.\begin{aligned} \hat{x}_1 &= v_1^T h(W^3 h(W^2 h(W^1 x))) \\ \hat{x}_2 &= v_2^T h(W^3 h(W^2 h(W^1 x))) \\ &\vdots \\ \hat{x}_d &= v_d^T h(W^3 h(W^2 h(W^1 x))) \end{aligned}\right\} \hat{x} = V h(W^3 h(W^2 h(W^1 x)))$$




- For training, we add up the loss across all 'j':

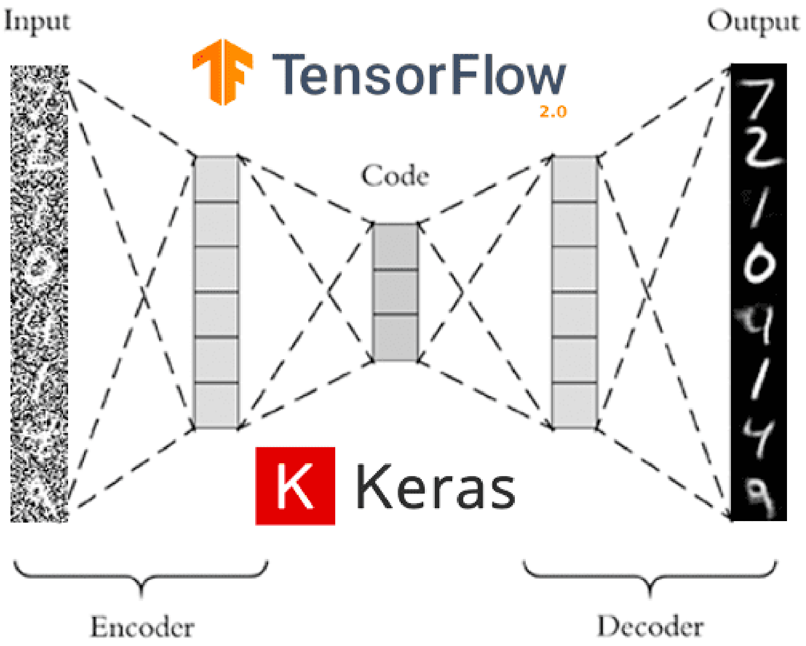
$$f(W^1, W^2, V) = \sum_{i=1}^n \sum_{j=1}^d \underbrace{(\hat{x}_j^i - x_j^i)^2}_{\text{squared error for continuous } x_j^i} \qquad \hat{x}_j^i = v_j^T h(W^3 h(W^2 h(W^1 x^i)))$$

$$f(W^1, W^2, V) = \sum_{i=1}^n \sum_{j=1}^d \underbrace{\log(1 + \exp(-\hat{x}_j^i x_j^i))}_{\text{logistic loss for binary } x_j^i \in \{-1, +1\}}$$

- Fit with SGD (sampling random 'i'), and usual deep learning tricks can be used.
  - Even though network has multiple outputs, 'f' is a scalar so AD works as before.
  - For images, may want to use convolution layers.

# Denoising Autoencoders

- A common variation on autoencoders is denoising autoencoders:
  - Use "corrupted" inputs, and learn to reconstruct uncorrupted originals.




  - "Learn a model that removes the noise". Easy to get lots of training data.
    - You can apply the model to denoise new images.
    - Do not necessarily need a "bottleneck" layer.

https://www.pyimagesearch.com/2020/02/24/denoising-autoencoders-with-keras-tensorflow-and-deep-learning/

# Image Colourization

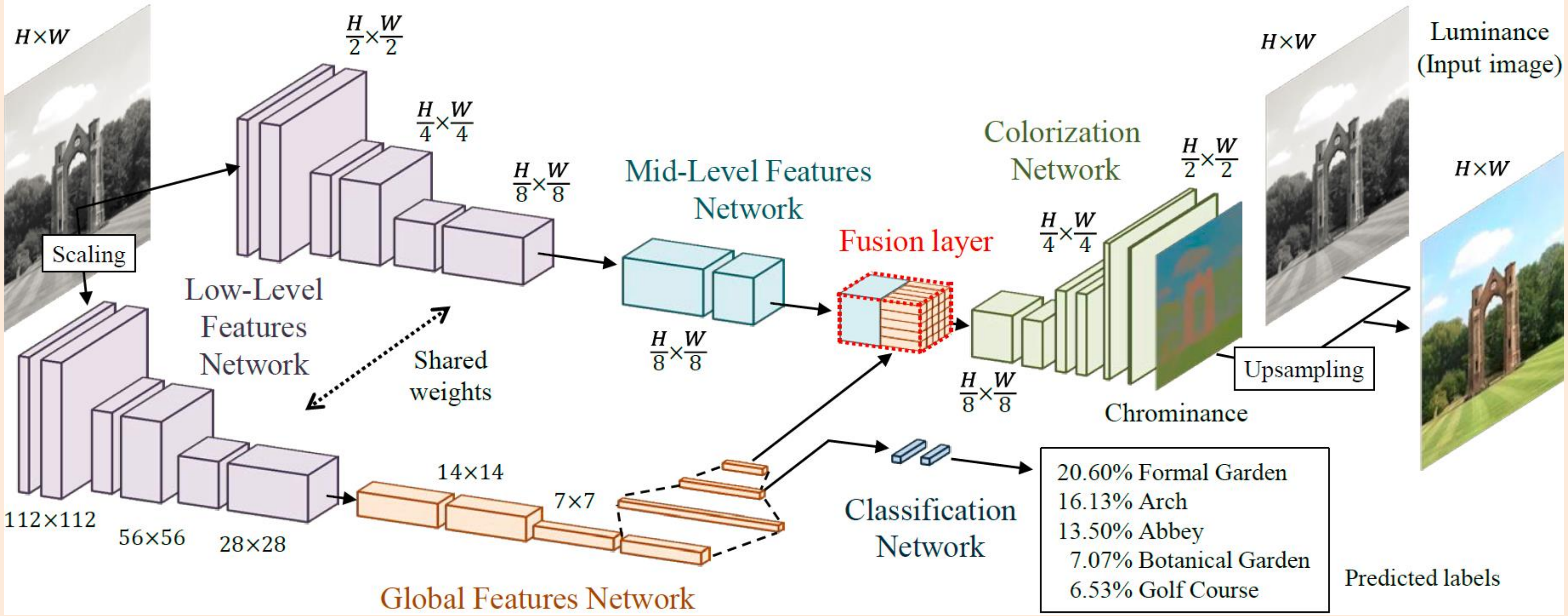

Colorado National Park, 1941 | Textile Mill, June 1937 | Berry Field, June 1909 | Hamilton, 1936

- Gallery: http://iizuka.cs.tsukuba.ac.jp/projects/colorization/extra.html
- Video: https://www.youtube.com/watch?v=ys5nMO4Q0iY

http://iizuka.cs.tsukuba.ac.jp/projects/colorization/en/

# Image Colourization

- Instead of noisy inputs, you use de-coloured inputs:




- Another application is super-resolution:
  - Learn to output a high-resolution image based on low-resolution images.

http://iizuka.cs.tsukuba.ac.jp/projects/colorization/en/

# Next Topic: Multi-Label Classification

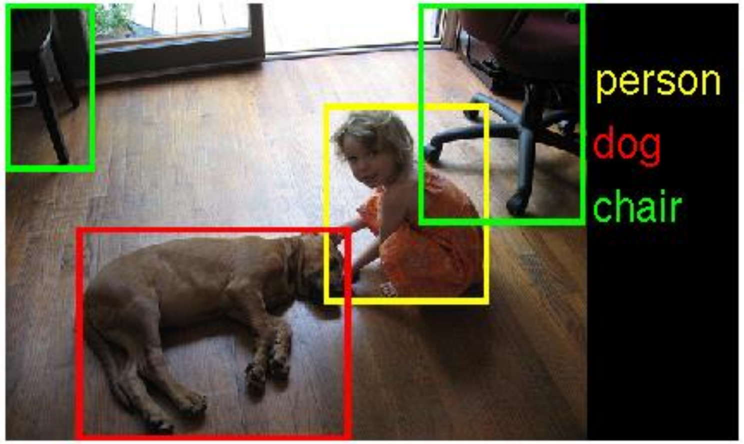
# Motivation: Multi-Label Classification

- Consider multi-label classification:

$$X = \underbrace{\left.\begin{bmatrix} \quad & \quad & \quad \\ & & \\ & & \end{bmatrix}\right\} n}_{d}$$

$$Y = \begin{array}{c} \begin{array}{ccccc} \text{cat} & \text{dog} & \text{person} & \text{chair} & \text{mouse} \end{array} \\ \underbrace{\left.\begin{bmatrix} 1 & -1 & 1 & 1 & 1 \\ -1 & -1 & 1 & -1 & -1 \\ -1 & -1 & -1 & 1 & -1 \\ 1 & 1 & -1 & 1 & -1 \\ -1 & -1 & 1 & 1 & -1 \\ 1 & -1 & 1 & -1 & 1 \end{bmatrix}\right\} n}_{k} \end{array}$$

- Which of the 'k' objects are in this image?
  - There may be more than one "correct" class label.




http://image-net.org/challenges/LSVRC/2013/

# Independent Classifier Approach

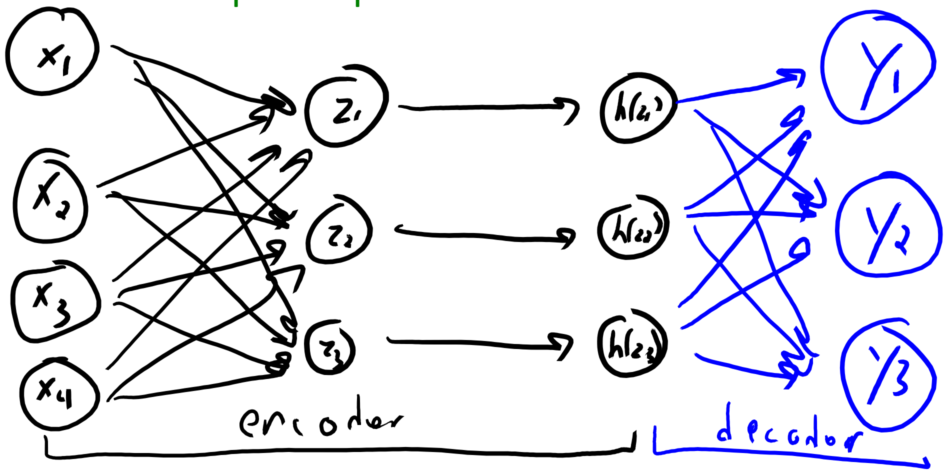
- One way to build a multi-label classifier:
  - Train a classifier for each label.
    - Train a neural network that predicts +1 if the image contains a dog, and -1 otherwise.
    - Train a neural network that predicts +1 if the image contains a cat, and -1 otherwise.
    - …
  - To make predictions for the ‘k’ classes, concatenate predictions of the ‘k’ models.

- Can think of this as a “product of independent classifiers”.

- Drawbacks:
  - Lots of parameters: k*(number of parameters for base classifier).
  - Each classifier needs to “relearn from scratch”.
    - Each classifier needs to learn its own Gabor filters, how corners and light works, and so on.
    - A lot of visual features for “dog” might also help us predict “cat”.

# Encoding-Decoding for Multi-Label Classification

- Multi-label classification with an encoding-decoding approach:
  - Input is connected to a hidden layer.
  - Hidden layer is connected to multiple output units.



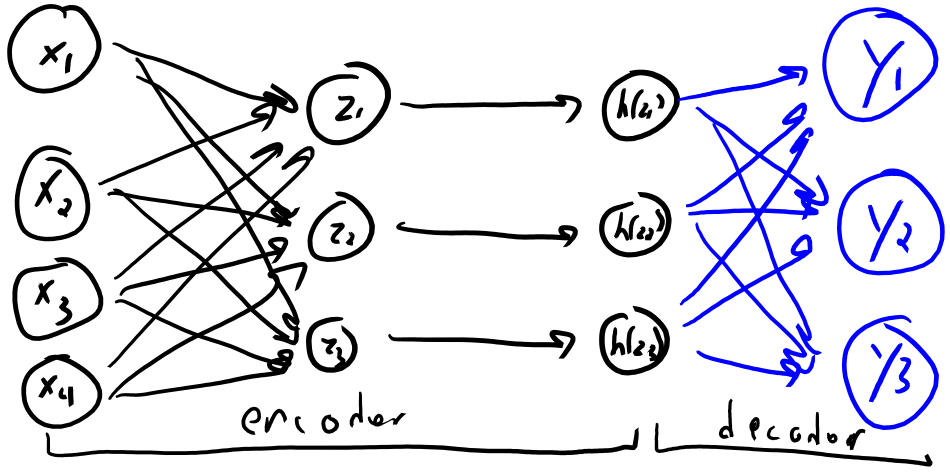

$$\hat{y}_1 = v_1^T h(Wx)$$
$$\hat{y}_2 = v_2^T h(Wx)$$
$$\hat{y}_3 = v_3^T h(Wx)$$

- Prediction: compute hidden layer, compute activations, compute output:

$$\hat{y} = V h(Wx)$$

- Number of parameters and cost is O(dm + mk) for ‘k’ classes and ‘m’ hidden units.
  - If we trained a separate network for each class, number of parameters and cost would be O(kdm) (for ‘W’ for each class)
- Might have multiple layers, convolution layers, and so on. And no need to have a “bottleneck” layer.

# Encoding-Decoding for Multi-Label Classification




- We usually assume that the classes are independent given last layer:

$$p(y_1, y_2, \dots, y_k \mid x_1, x_2, \dots, x_d, W, V) = p(y_1 \mid x_1, x_2, \dots, x_d, W, v_1)\, p(y_2 \mid x_1, x_2, \dots, x_d, W, v_2) \cdots p(y_k \mid x_1, x_2, \dots, x_d, W, v_k)$$

with: $p(y_1 = 1 \mid x, W, v) = \underbrace{\dfrac{1}{1 + \exp(-v_1^T h(Wx))}}_{\theta_1}$ $\quad p(y_2 = 1 \mid x, W, v) = \underbrace{\dfrac{1}{1 + \exp(-v_2^T h(Wx))}}_{\theta_2}$ $\quad \dots$

  - Conditioned on features/parameters, this is ultimately a fancy product of Bernoullis model:
    - $p(y_1, y_2, \dots, y_k \mid x, W, V) = p(y_1 \mid x, W, V)p(y_2 \mid x, W, V) \cdots p(y_k \mid x, W, V)$, where $p(y_c = 1 \mid x, W, V) = \theta_c$.
    - This makes decoding and other inference problems easy: you do inference on each $y_c$ independently.

# Encoding-Decoding for Multi-Label Classification

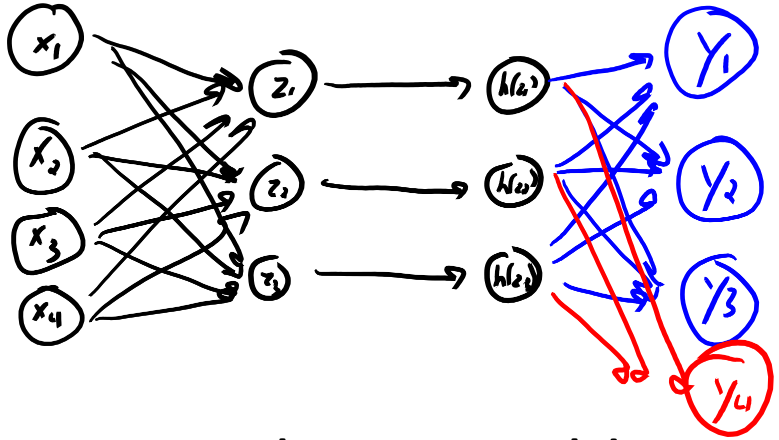
- The negative log-likelihood we optimize for MLE:

$$f(W, V) = \sum_{i=1}^{n} \sum_{c=1}^{k} \log(1 + \exp(-y_c^i v_c^T h(Wx^i)))$$

- Use backpropagation or AD to compute gradient, train by SGD.
  - You randomly sample a training example 'i' and compute gradient for all labels.
  - The updates of 'W' lead to features that are useful across classes.
  - The updates of 'V' focus on getting the class labels right given the features.
- Important:
  - We assumed independence of labels given the last layer.
  - But the last layer can reflect dependencies.
    - If "dog" and "human" are frequently together, this should be reflected in the hidden layer.
      - For example, $\theta_{human}$ might be higher when the features give a high value for $\theta_{dog}$.

# Pre-Training for Multi-Label Classification

- Consider a scenario where we get a new class label.
  - For example, we get new images that contain horses (not seen in training).




- Instead of training from scratch, we could:
  - Add an extra set of weights $v_{k+1}$ to the final layer for the new class.
  - Train these weights with the encoding weights 'W' fixed.
    - This is a simple/convex logistic regression problem.
    - If we already have "features" that are good for many classes, we may be able to learn a new class with very-few training examples!

# Pre-Training for Multi-Label Classification

- Using an existing network for new problems is called "pre-training"
  - Typically, we start with a network trained on a large dataset.
  - We use this network to give us features to fit a smaller dataset.
    - "Few-shot learning".

- Depending the setup, you may also update 'W' and the other '$v_c$'.
  - Useful if you have a lot of data on the new class.
  - In this case, would typically mix in new examples with old ones.

- Increasing trend in vision and language to using pre-training a lot.
  - No need to learn everything about language for every language task!

# Summary

- Autoencoders:
  - Neural network where the output is the input.
  - Encode data into a bottleneck layer, then decode predict original input.
  - Can be used for visualization, compression, outlier detection, pre-training.
- Denoising autoencoders train to uncorrupt/enhance images.
  - Can be used for removing noise, adding colour, super-resolution, and so on.
- Multi-label classification:
  - Classification with more than one label per example.
- Encoding-Decoding approach to multi-label classification:
  - Have all classes shared the same hidden layer(s).
  - Reduces number of parameters.
  - Models dependencies between classes, while keeping inference easy.
- Pre-training:
  - Use parameters from model trained a on large diverse dataset, to initialize SGD for new dataset.

- Next time: helping teach fish to drive?