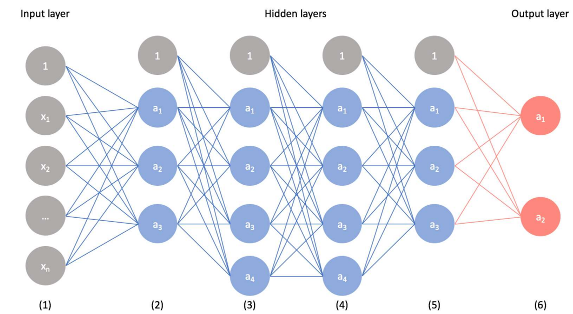
Computer Vision and Machine Learning

(Neural Network-3)

Bhabatosh Chanda
bchanda57@gmail.com

Neural network: An example



CNN or ConvNet: function

- ConvNet has two parts:
  - feature learning (Conv, ReLU and Pool) and
  - Classification (Fully-connected network and softmax).
- ConvNet architectures for images:
  - The explicit assumption that the inputs are images
  - Allows us to extract certain features
  - Large images do not fit into fully-connected structure
  - Pooling vastly reduces the number of parameters

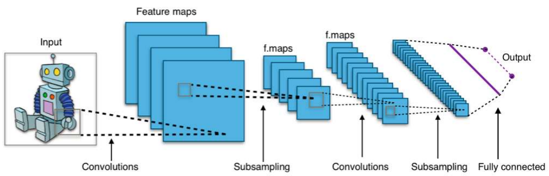
CNN or ConvNet: function

- ConvNet has two parts:
  - feature learning (Conv, ReLU and Pool) and
  - Classification (Fully-connected network and softmax).
- ConvNet architectures for images:
  - The explicit assumption that the inputs are images
  - Allows us to extract certain features
  - Large images do not fit into fully-connected structure
  - Pooling vastly reduces the number of parameters

Structure of CNN

- Input is passed through a series of
  - Convolutional layer(s),
  - Non-linear (squashing) operation,
  - Pooling (downsampling), and
  - Fully connected layers (optional)

  to obtain output.



Terms and standard variations

- Convolutional layer
  - Filter ≡ Neuron ≡ Kernel
  - Weights ≡ Parameters
  - Receptive field = size of the kernel
  - Output: Activation map ≡ Feature map
- Non-linear
  - Sigmoid OR tanh OR ReLU
- Pooling layer
  - Max OR Mean OR Median

Convolutional neural network (CNN)

Biological Connection -

- Hubel and Wiesel (1962) showed that some neuronal cells in the brain fired only in the presence of edges of a certain orientation.
  - Some neurons fire when exposed to vertical edges and some to horizontal or diagonal edges.
  - They are organized in a columnar architecture and produce visual perception.
- Convolutional operation (filter) can extract various features.
- Existence of feature may be confirmed by firing the neuron.

Convolution and Neural network

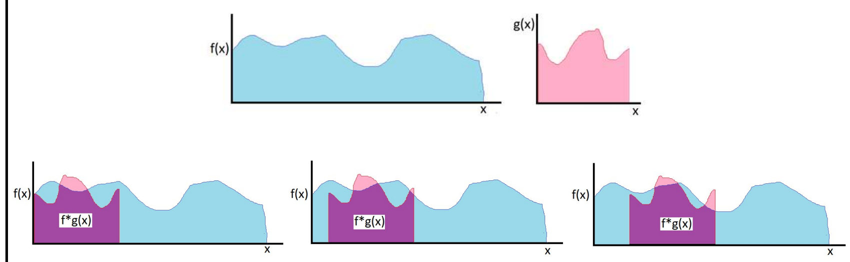
Convolution

- 1D continuous convolution: $f * g(x) = \int_{\alpha=-\infty}^{\infty} f(\alpha) g(x-\alpha) d\alpha$
- 1D discrete convolution: $f * g(x) = \sum_{j=0}^{M-1} f(j) g(x-j)$
- 2D continuous convolution: $f * g(x,y) = \int_{\beta=-\infty}^{\infty} \int_{\alpha=-\infty}^{\infty} f(\alpha,\beta) g(x-\alpha, y-\beta) d\alpha d\beta$
- 2D discrete convolution: $f * g(x,y) = \sum_{k=0}^{N-1} \sum_{j=0}^{M-1} f(j,k) g(x-j, y-k)$

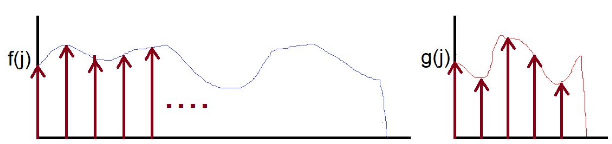
Properties of Convoluton

- Commutative: $f * g = g * f$
- Associative: $(f * g) * h = f * (g * h)$
- Homogeneous: $f * (\tau g) = \tau f * g$
- Distributive (over addition): $f * (g + h) = f * g + f * h$
- Shift-Invariant: $f * g(x - x_0, y - y_0) = (f * g)(x - x_0, y - y_0)$

1D continuous convolution: Example

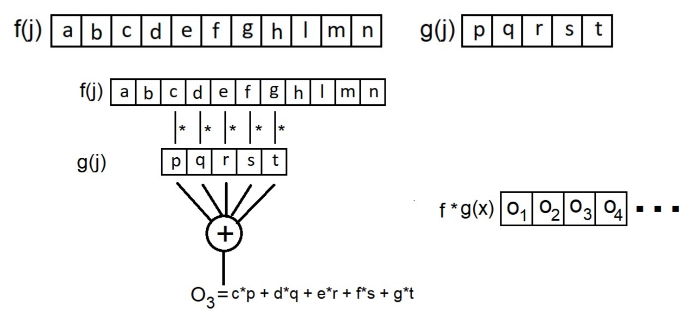


1D discrete convolution: Example

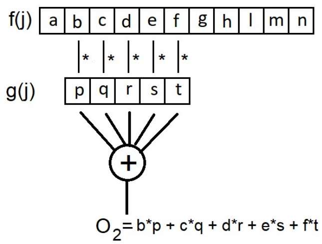


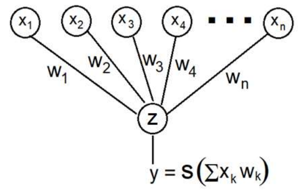
f(j) | a | b | c | d | e | f | g | h | l | m | n | g(j) | p | q | r | s | t |

1D discrete convolution: Example (cont.)

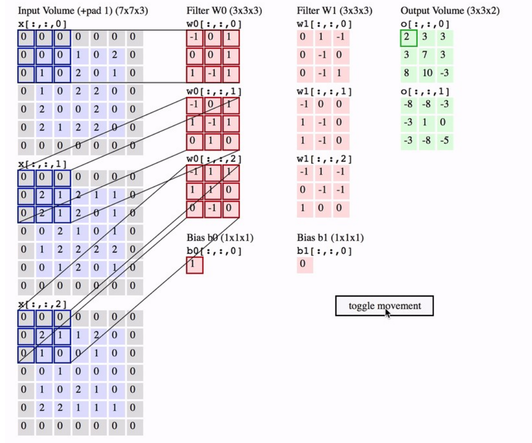


Convolution and neural network



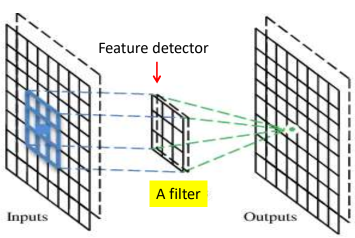


2D Convolution:



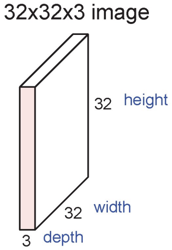
A convolutional layer

A convolutional layer has a number of filters that perform convolution operation.

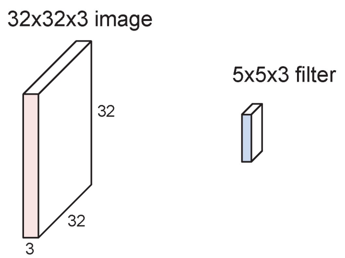


Convolutional layer: Operation & Size

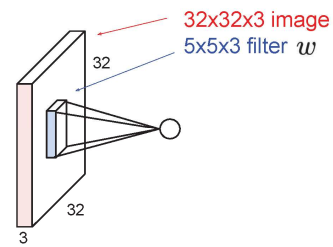
32x32x3 image



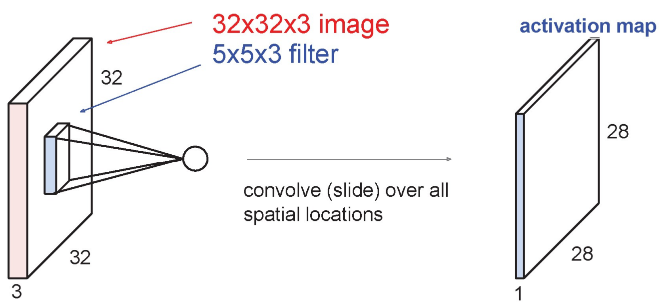
Convolutional layer: Operation & Size



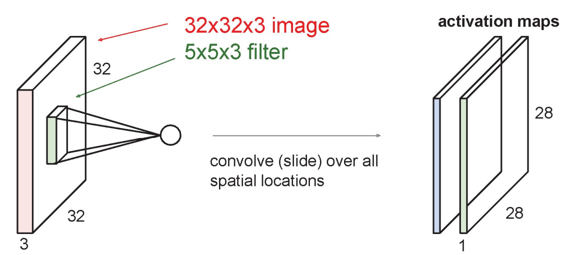
Convolutional layer: Operation & Size



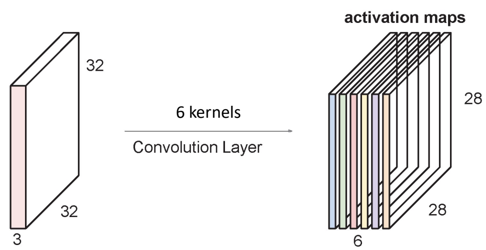
Convolutional layer: Operation & Size



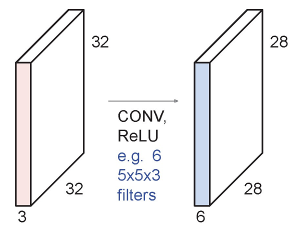
Convolutional layer: Operation & Size



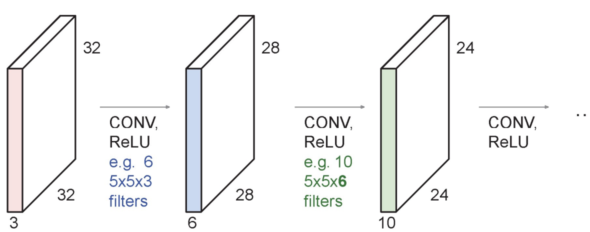
Convolutional layer: Operation & Size



Convolutional layer: Operation & Size

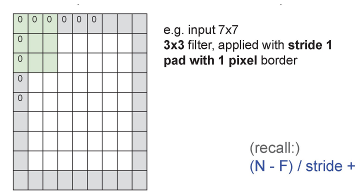


Convolutional layer: Operation & Size



Zero padding

- To prevent size reduction in Convolution Layer.
- If size of kernel is *KxK* (*K* odd), input image is padded with 0 (zero) around it by a thickness of *(k-1)/2*.

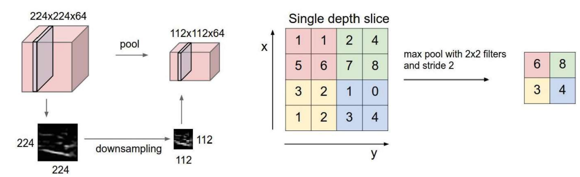


Training

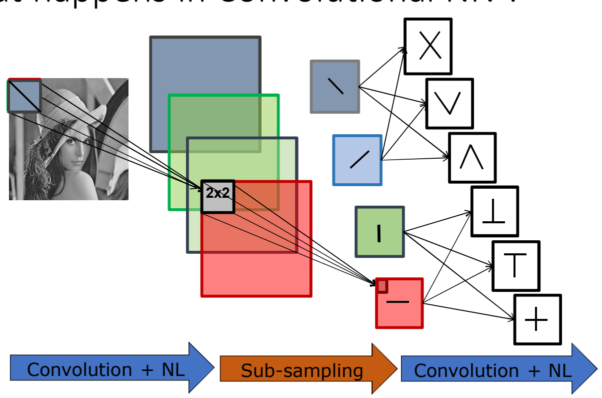
- Network training can be separated into 4 distinct steps
  - the forward pass,
  - the loss function,
  - the backward pass, and
  - the weight update.
- Training strategy and methods are same as neural network.

Pooling

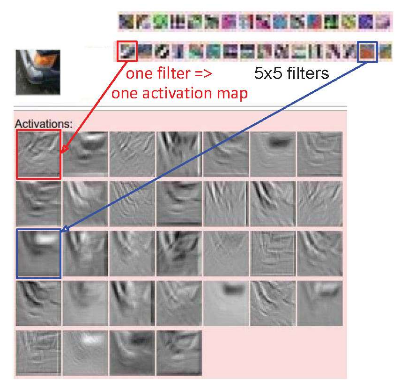
- To reduce the number of parameters.
- To increase effective size of receptive field.
- Sum or max or median over non-overlapping / overlapping regions
  - Invariant to small geometric transformation



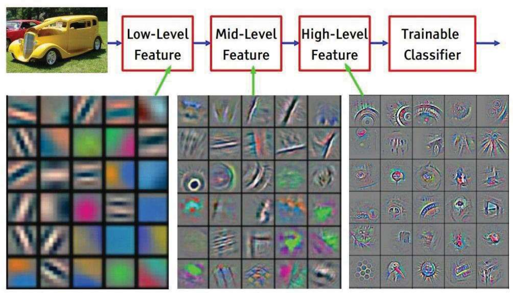
What happens in Convolutional NN ?



Kernel and extracted features



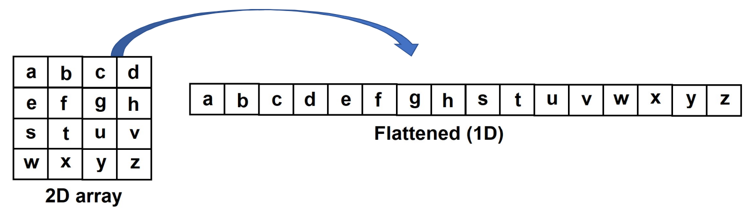
Kernel an extracted features



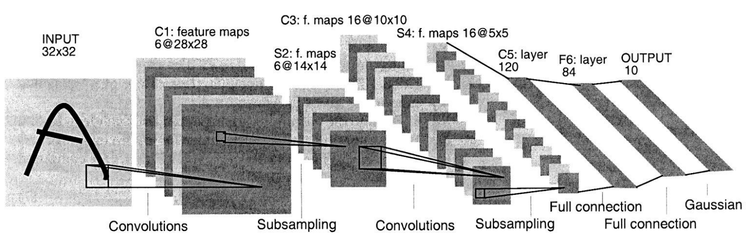
Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Flattening

- Converts 2D aray of Activation field or Feature map to linear (1D) array.
- To facilitate input to feed to fully connected neural network.



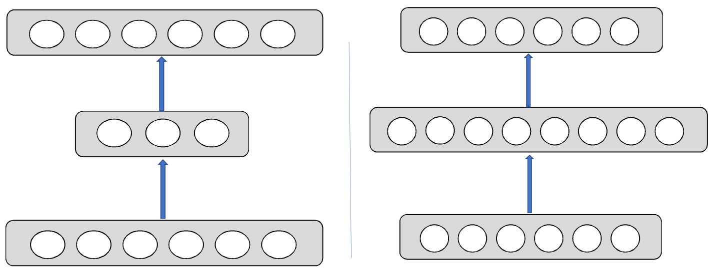
Fully connected neural network



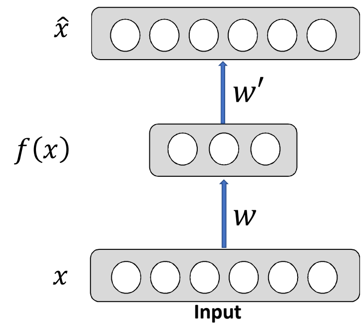
Autoencoder

- Autoencoder is an unsupervised learning technique.
  - Neural network (CNN) is exploited for **representation learning**.
- A neural network (CNN) architecture includes ***a bottleneck*** in the network that forces a **compressed** representation of input.
- It has two parts:
  - Encoder (from the input to bottleneck)
  - Decoder (beyond bottleneck to the output)

Undercomplete AE vs Overcomplete AE



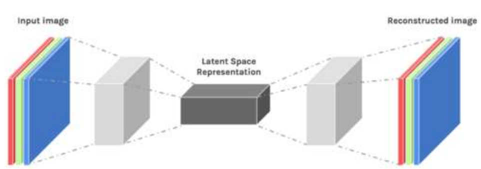
Undercomplete AE



- Hidden layer is **Undercomplete** if smaller than the input layer
  - Compresses the input
- Hidden nodes contain
  - Good features for the training distribution.
- Without bottleneck network could be trained easily to memorize data.

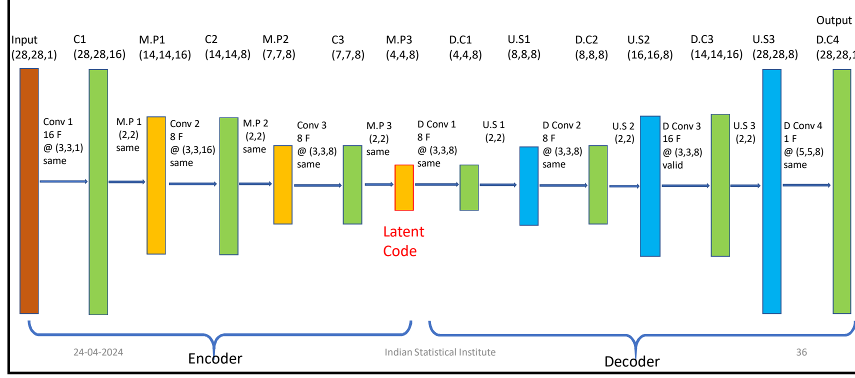
Bottleneck layer (undercomplete)

- Suppose input images are *nxn* and the latent space is *m* < *nxn*.
- Then the latent space is not sufficient to reproduce all images.
- Needs to learn an encoding that captures the important features in training data, sufficient for approximate reconstruction.



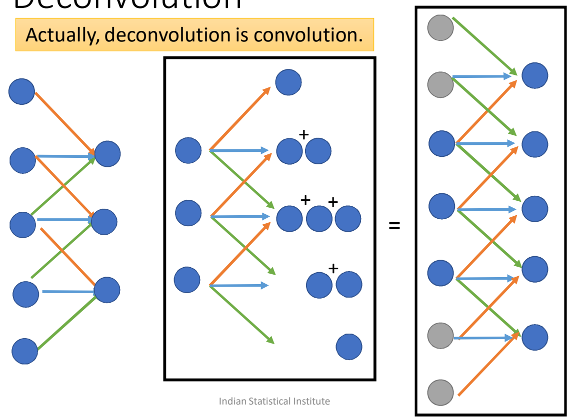
Convolutional AE

* Input values are normalized
* All of the conv layers activation functions are relu except for the last conv which is sigm

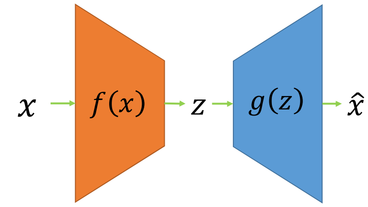


CNN - Deconvolution

Actually, deconvolution is convolution.



Autoencoder: Basic



Given data x (no labels) we would like to learn the functions f (encoder) and g (decoder) where:

$$f(x) = s(wx + b) = z$$

and

$$g(z) = s(w'z + b') = \hat{x}$$

such that $h(x) = g\big(f(x)\big) = \hat{x}$

where $h()$ is an **approximation** of the identity function.

(z is some **latent** representation or **code** and s is a non-linearity such as the sigmoid)

($\hat{x}$ is x's reconstruction)

Training the AE

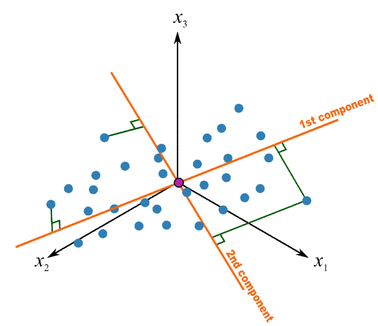
- We can train Autoencoder simply by using **Gradient descent** method like any other fully connected neural network (FC-NN).
- Loss or error function may be defined as

$$E(x) = ||x - \hat{x}||^2 = ||x - h(x)||^2 = ||x - g(f(x))||^2$$

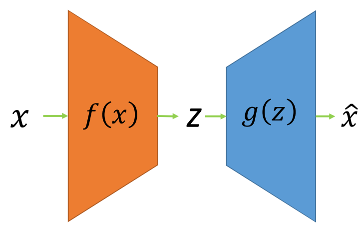
- So, still it needs to define a loss – this is an implicit supervision.

PCA – Principal component analysis

- Statistical approach for data compression and visualization
- Invented by Karl Pearson in 1901
- Weakness: linear components only.



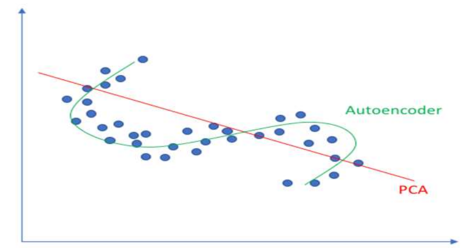
Autoencoder and PCA



- Traditionally an autoencoder is used for dimensionality reduction and feature learning.
- Unlike the **PCA** now we can use activation functions to achieve non-linearity.
- It has been shown that an AE without activation functions achieves the **PCA** capacity.

AE vs. PCA: Representation visualization

Linear vs nonlinear dimensionality reduction



PCA attempts to discover a lower dimensional hyperplane which describes the original data, autoencoders learn nonlinear manifold.

Properties of Autoencoders

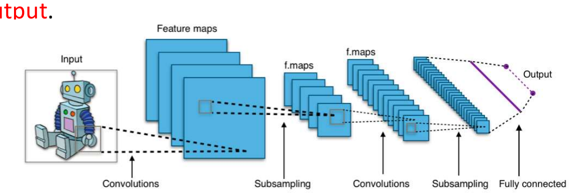
- **Data-specific**: Autoencoders are only able to compress data similar to what they have been trained on.
- **Lossy:** The decompressed outputs will be degraded compared to the original inputs.
- **Learned automatically from examples:** It is easy to train specialized instances of the algorithm that will perform well on a specific type of input.

https://www.edureka.co/blog/autoencoders-tutorial/

Structure of CNN for classification

- Input is passed through a series of
  - Convolutional layer(s),
  - nonlinear (squashing) operation,
  - pooling (downsampling), and
  - fully connected layers (optional)

  to obtain the output.



Problem with standard autoencoders

- Standard autoencoders learn to generate compact representations and reconstruct their inputs well,
  - but asides from a few tasks like feature extraction and denoising, their applications are fairly limited.
- The fundamental problem with autoencoders, for generation, is
  - the latent space they convert their inputs into, where their encoded vectors lie, may not allow easy interpretation.

Denoising autoencoders

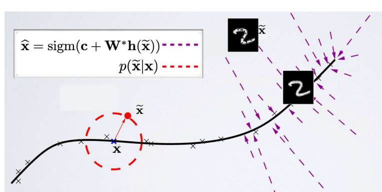
- Basic autoencoder trains to minimize the loss between *x* and the reconstruction *g(f(x))*.
- Denoising autoencoders train to minimize the loss between *x* and *g(f(x+w))*, where *w* is random noise.
- Same possible architectures, different training data.

- Kaggle has a dataset on damaged documents.

https://blog.keras.io/building-autoencoders-in-keras.html

Denoising autoencoders

- Denoising autoencoders can't simply memorize the input output relationship.
- Intuitively, a denoising autoencoder learns a projection from a neighborhood of training data back onto the training data.

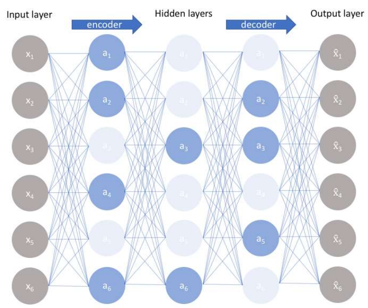


https://ift6266h17.files.wordpress.com/2017/03/14_autoencoders.pdf

Sparse autoencoder

- Sparse autoencoders provides an alternative method for creating a bottleneck.
  - It does not require a reduction in the number of nodes in hidden layers apriori.
  - Encoder and decoder rely on activating a small number of neurons in the hidden layer.
  - Individual nodes of a trained model that are activated are *data-dependent*.
    - Different inputs result in activation of different nodes.
- Undercomplete autoencoder use the entire network for every observation, while a sparse autoencoder use only a selective portion of the network.

Sparse autoencoder: architecture



Sparse autoencoder: sparsity

- Loss function is constructed to penalize activation of hidden neurons within a layer.
  - In other words, only a subset of neurons are activated for an input.
  - Set of neurons to be activated in a hidden layer depends on the input.
  - A neuron is said to be active if its output is close to 1.
- Such sparsity may be imposed in two different ways.
  - **L1 regularization**
  - **KL-divergence**

Sparsity: L1 regularization

- By adding a term to the loss function $L(x, \hat{x})$ to penalize the absolute value of the activation vector ***a*** in the layer ***h*** for ***i***-th observation

$$L(x, \hat{x}) + \gamma \sum_i |a_i^{(h)}|$$

where γ is the controlling parameter.

Thank you!

Any question?