Lecture 8: Neural Networks and Unsupervised Learning

Matt Ellis and Mike Smith

Session outline

Neural networks

- Recap
- Convolutional operations and networks

Unsupervised learning: Dimensional reduction

- Principal component analysis (PCA)
- Auto-encoders

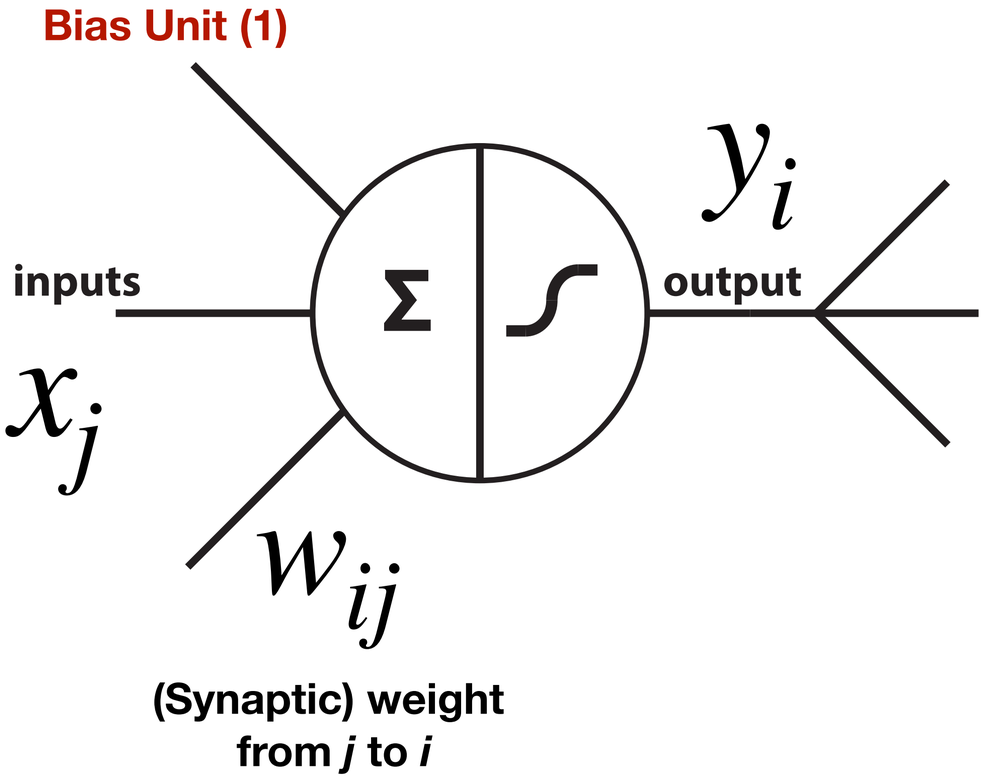
Learning Objectives

By the end of this session you should be able to:

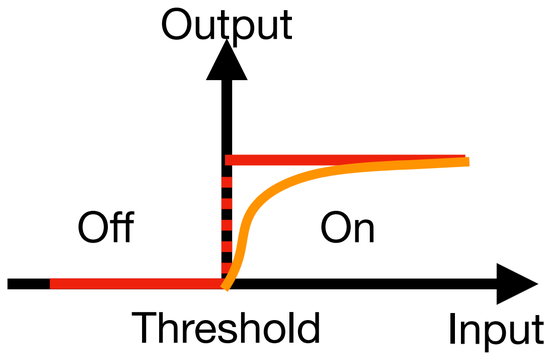
1. Explain the structure and training of convolutional neural networks.
2. Explain the method of PCA for dimensionality reduction.
3. Recall the criterion function for the 1st principal component.
4. Define the structure and method of auto-encoder neural networks.

Neural Networks

Artificial Neurons



Activation function

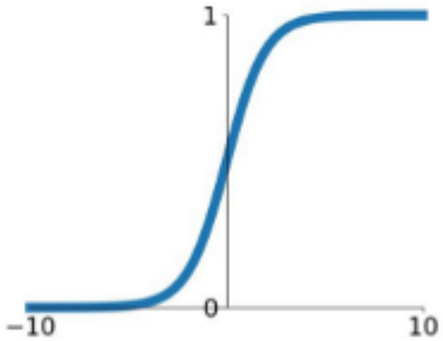


$$y_i = f\left(\sum_j w_{ij} x_j\right)$$

Common activation functions

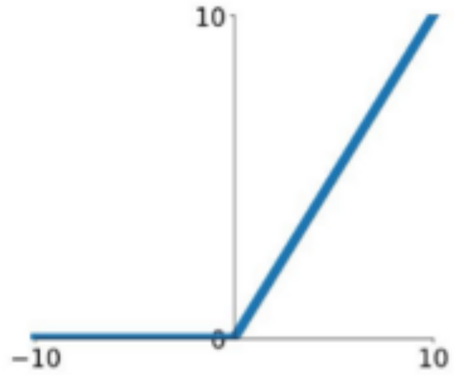
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



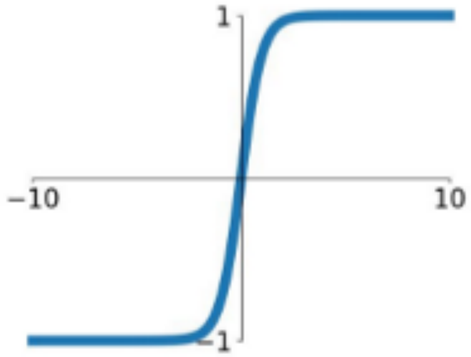
ReLU

$$\max(0, x)$$

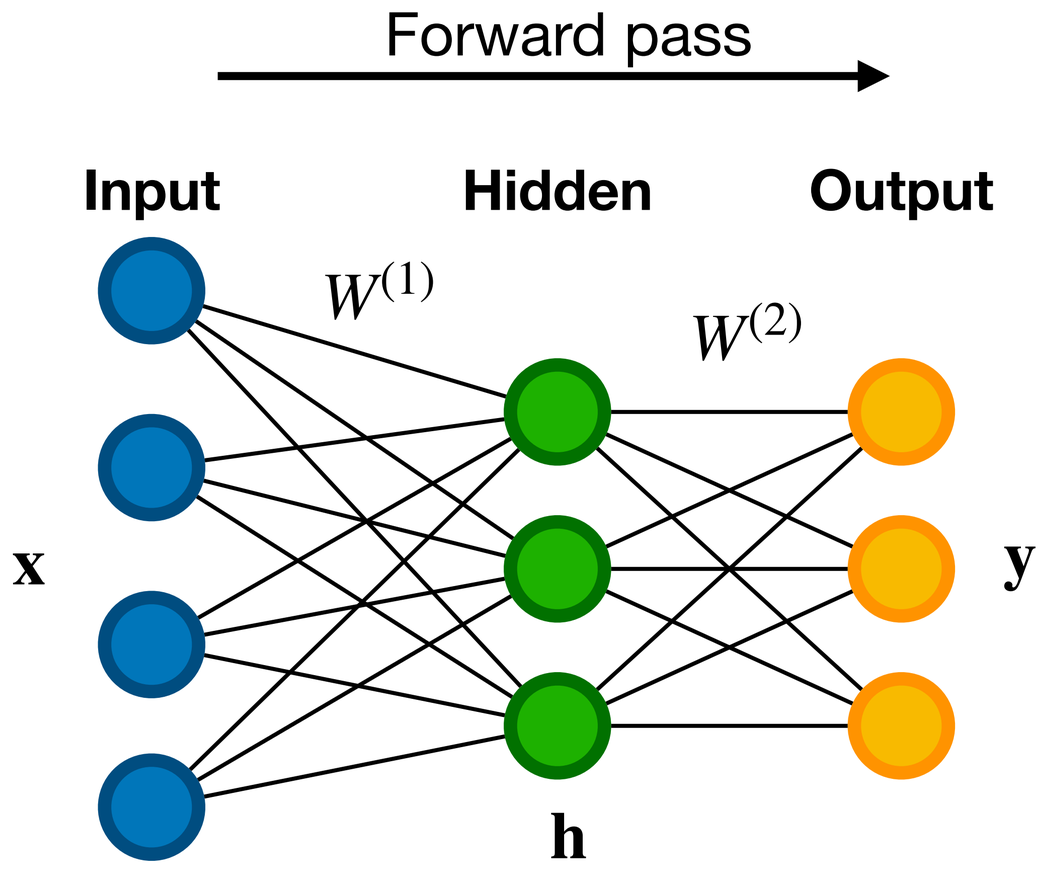


tanh

$$\tanh(x)$$



Computation in neural networks



Make predictions (decisions)
Plug in x to get y

Hidden layer neurons:

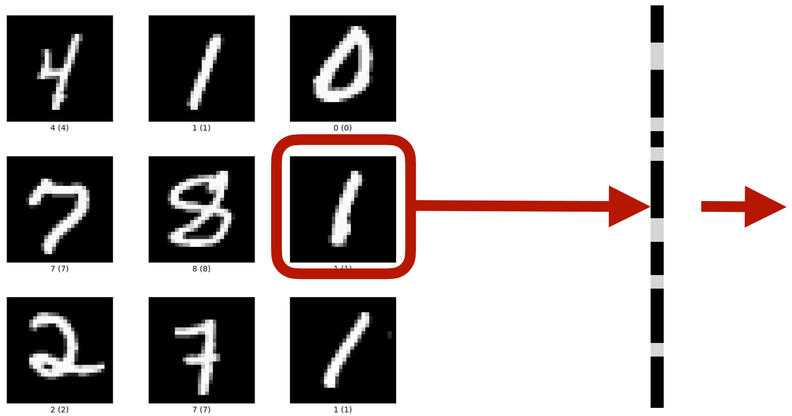
$$\mathbf{h} = f\left(W^{(1)}\mathbf{x} + b^{(1)}\right)$$

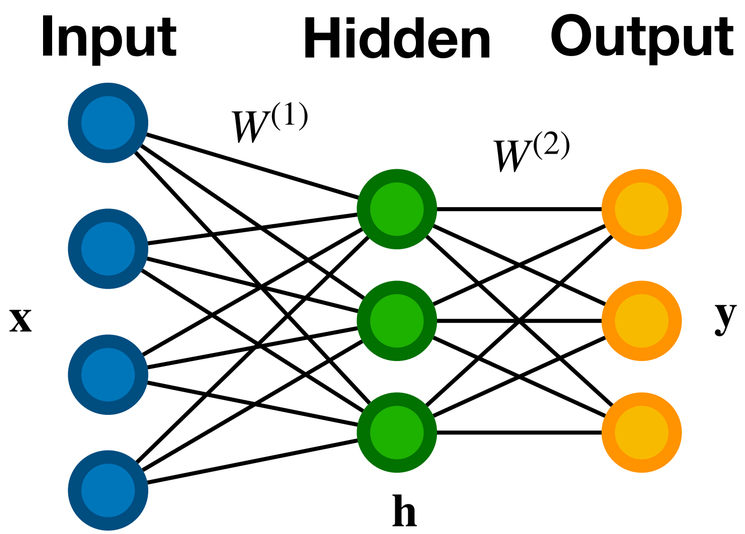
Output layer neurons:

$$\mathbf{y} = f\left(W^{(2)}\mathbf{h} + b^{(2)}\right)$$

Compute gradients of the cost (error or loss) w.r.t weights to find optimal values.

Image classification with neural networks





NN predicts class:
1 output neuron per class
(one hot encoding)

$$c = \arg\max_i \left(y_i\right)$$

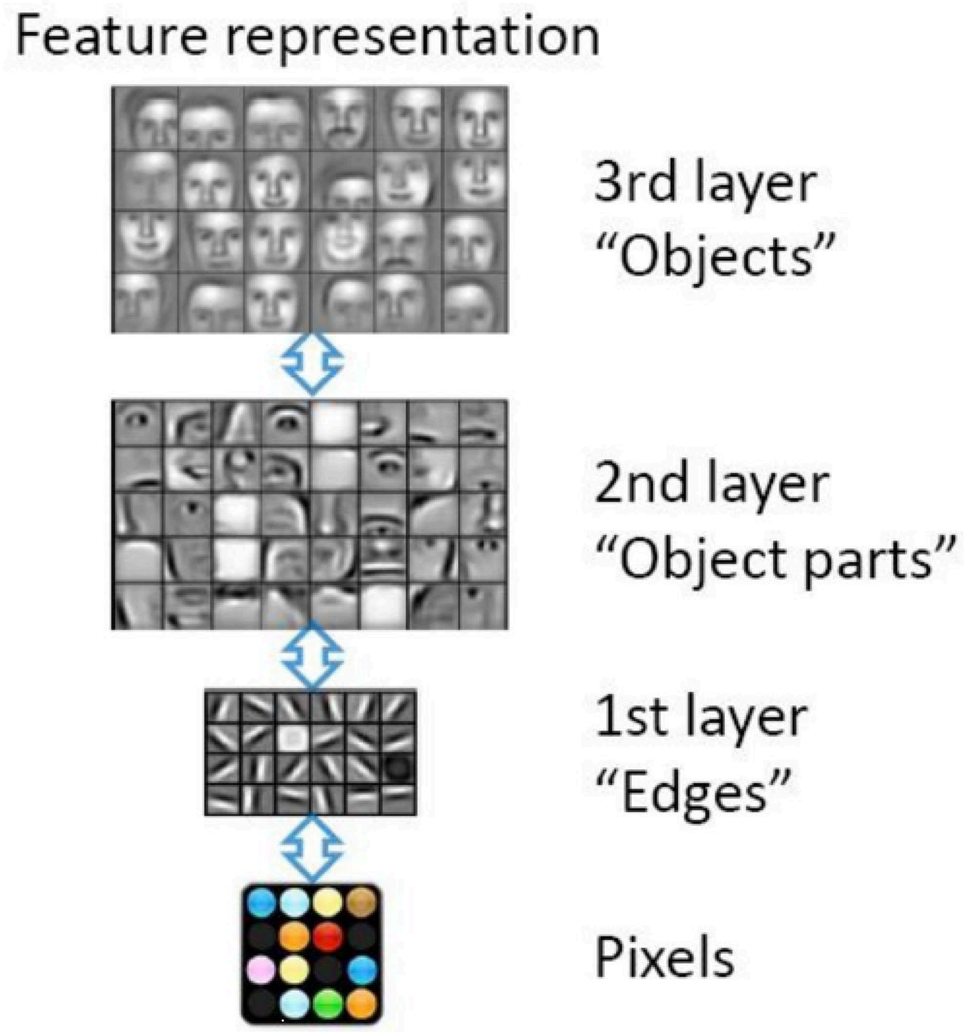
Flatten to a 1D array/vector.

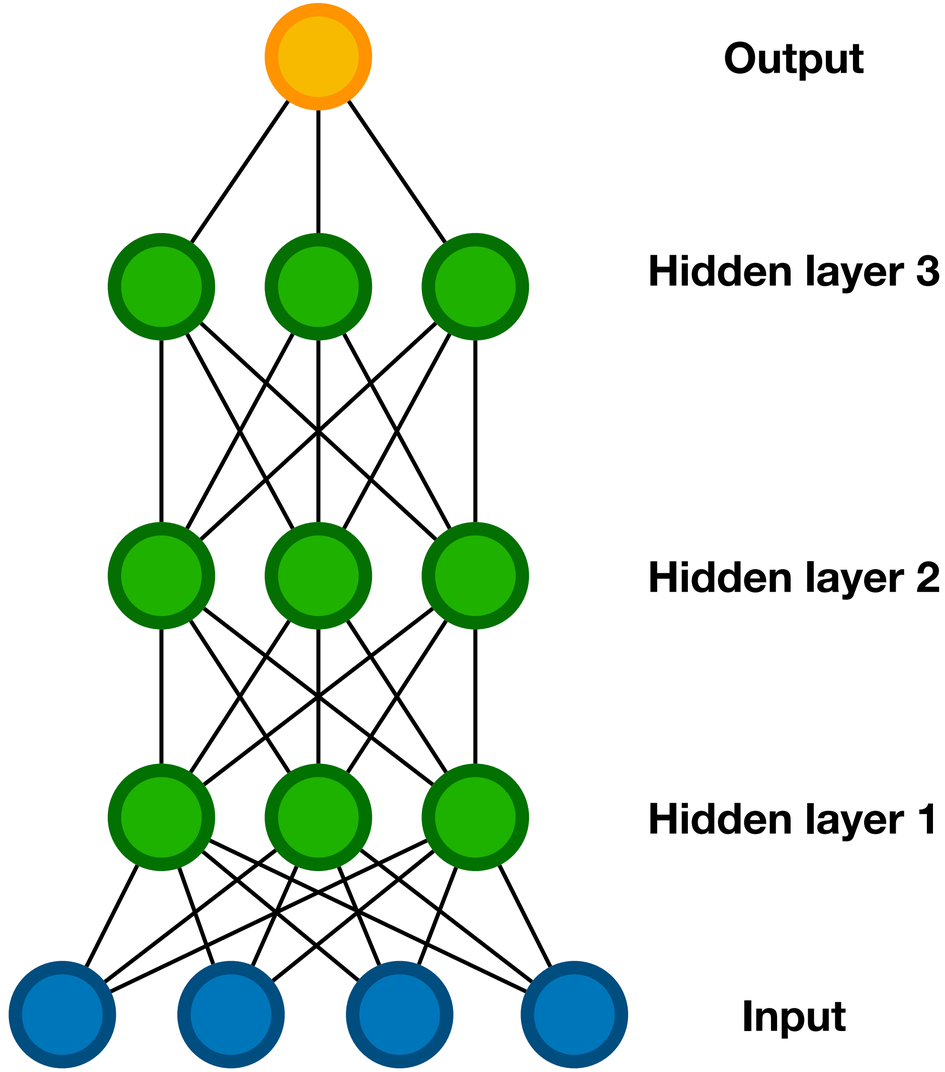
N x N image to a N^2 x 1 vector.

Some models convert $\mathbf{y}$ into a probability, e.g softmax

Cross entropy loss is suitable for multi class predictions.

Different levels of abstraction





Example from Honglak Lee (NeurIPS 2010)

Batches!

Predict for multiple inputs at once, update weights after each batch.

1 sample	N_B samples	All samples
Stochastic	Mini-Batch	Batch
More ‘noise’ Quick updates	Compromise	Smooth Slow updates

The convention used for PyTorch is number of samples as first dimension.

X.shape = (N_B = batch_size, N_{in} = input_features)

This means we have to modify our linear operation to make sure the matrix dimensions match!

$$\mathbf{y} = \mathbf{x}W^T + \mathbf{b}$$

(N_B x N_{out}) = (N_B x N_{in}) @ (N_{in} x N_{out})

Using torch.nn to create models

If we want to build our own models in PyTorch we can create classes inheriting from the **torch.nn.Module** class.
Internally this class can then hold various layers and operations.

```
class logistic_regression(nn.Module):
    def __init__(self, in_features, out_features, bias=True):
        super().__init__()
        self.lin = nn.Linear( in_features, out_features, bias)
        self.act_func = nn.Sigmoid()

    def forward(self, x):
        return self.act_func(self.lin(x))
```

Example for a multi-layer network

```
class neural_network(nn.Module):
    def __init__(self, in_features, hidden_features, out_features, bias=True):
        super().__init__()
        self.lin1 = nn.Linear( in_features, hidden_features, bias)
        self.act_func1 = nn.ReLU()
        self.lin2 = nn.Linear( hidden_features, out_features, bias)
        self.act_func2 = nn.Sigmoid()

    def forward(self, x):
        h = self.act_func1(self.lin1(x))
        return self.act_func2(self.lin2(h))
```

Simplify using nn.Sequential

If we are chaining together layers, we can use the built in Sequential class:

```
model = nn.Sequential(
    nn.Linear(in_features, hidden_features),
    nn.ReLU(),
    nn.Linear(hidden_features, out_features),
    nn.Sigmoid()
)
```

In each case we can use the model to predict using:

```
y_approx = model(x)
```

Fully connected layers

What if our network was bigger?

- Input image: 200 x 200 pixels, first hidden layer: 500 neurons

Q: How many weights from input to first hidden layer?

- 200 x 200 x 500 = 20 million

Q: Why might a FC layer be problematic for images?

- Lots of weights = long computation.
- Needs lots of training data to avoid over-fitting.
- Small shift in weights can lead to a large change in prediction.
- Not making use of geometry.

Convolutional Neural Networks

Imagenet Large Scale Visual Recognition Challenge

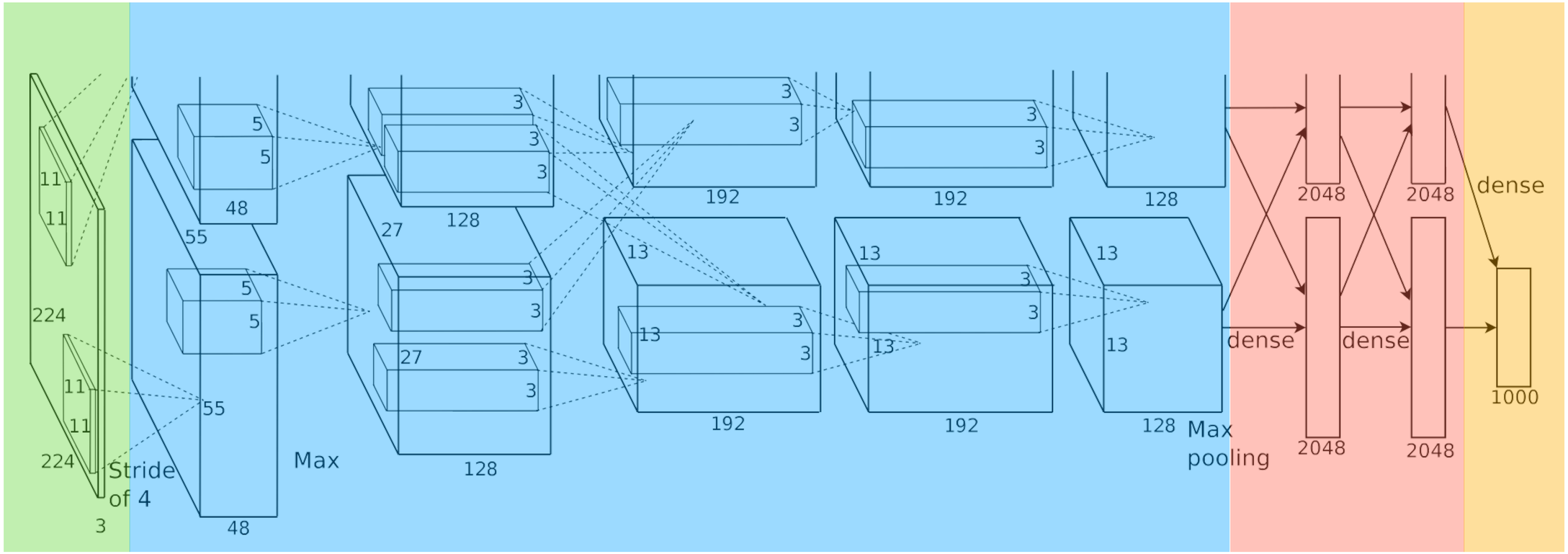
Alexnet

Input image (pixels)

5 Convolutional layers with max pooling

Fully connected layers

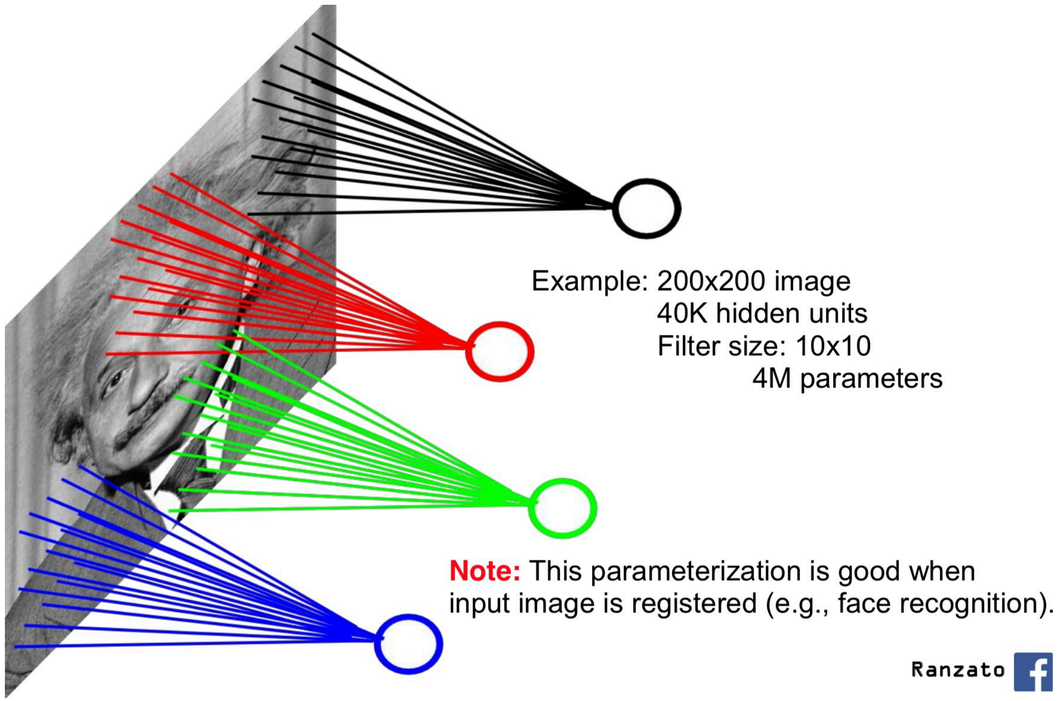
1000-way softmax

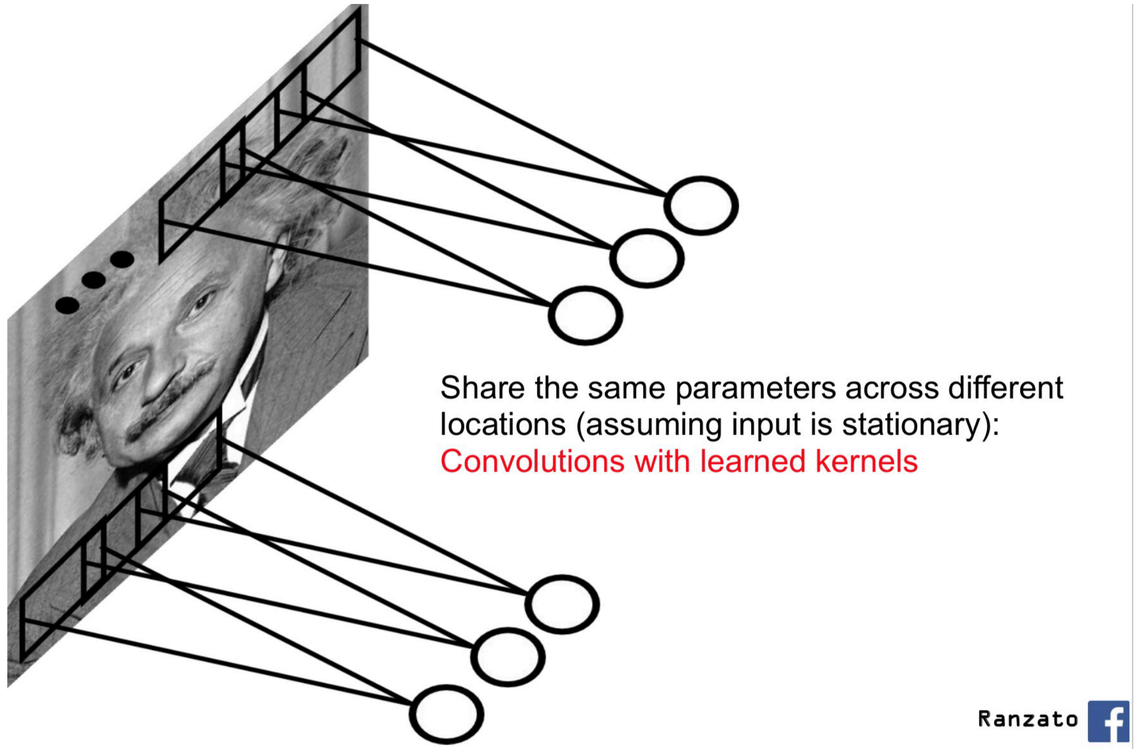


Convolutional neural network

Locally connected layers: look for local features in small regions of the image

Weight sharing: detect the same local features across the whole image





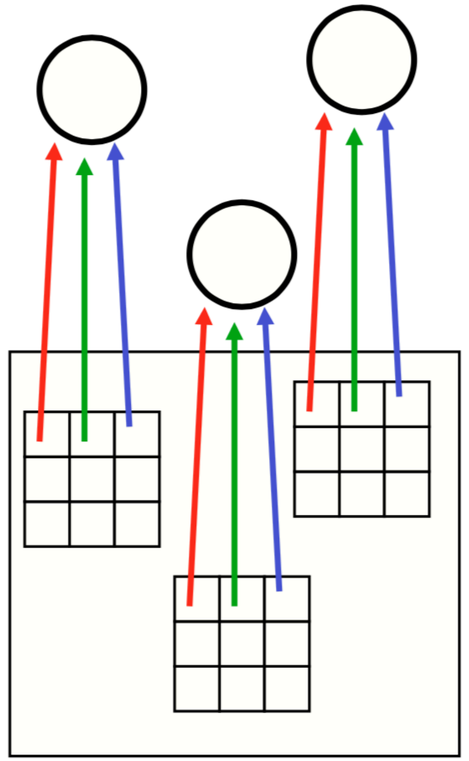
Weight sharing

The red connections all have the same weight.

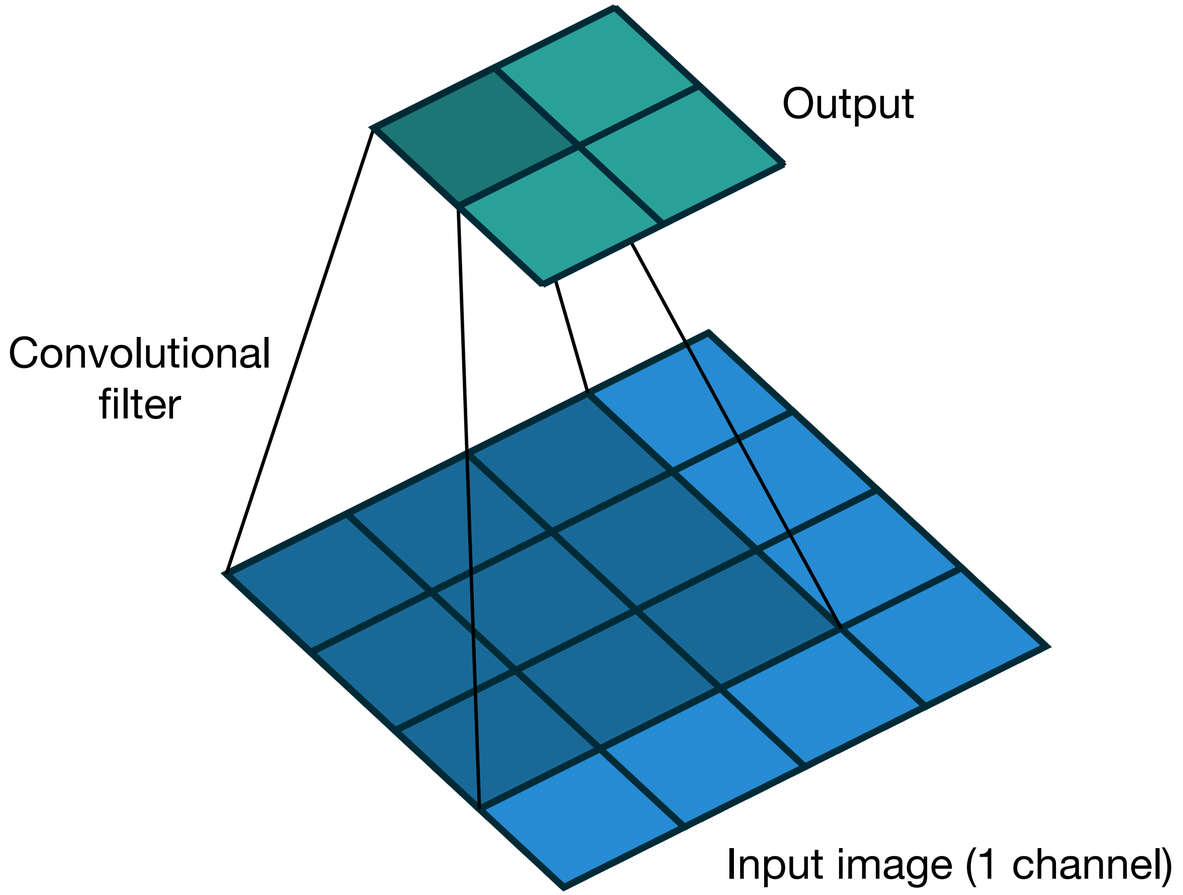
Each neuron in the higher layer detects the **same** feature, but in a **different** location in the lower layer.

Detection - the output (activation) is high if the feature is present.

Feature - something in the image (shape, blob, line) that we want to detect.



Convolutional filters



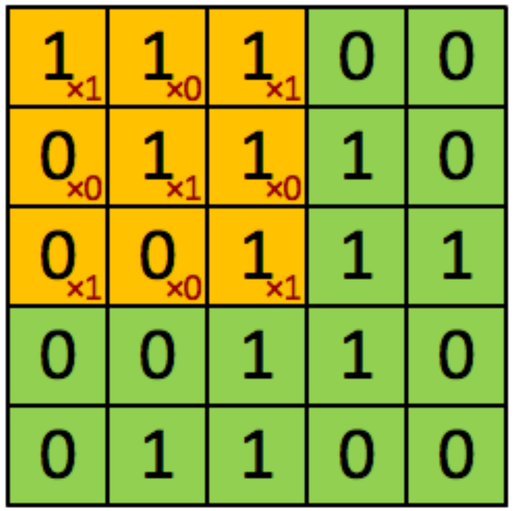
Convolution filter is applied as a moving window over the 2D input image.

$$y_{ij} = b + \sum_{k=0}^{F-1} \sum_{l=0}^{F-1} w_{kl}\, x_{i+k,j+l}$$

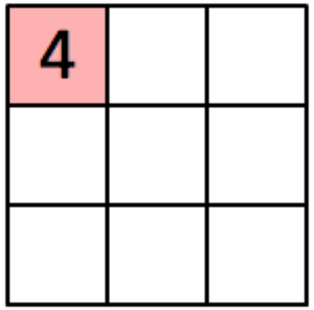
Vincent Dumoulin, Francesco Visin - A guide to convolution arithmetic for deep learning

Forward pass example

The filter/kernel (yellow) contains the trainable weights. Here filter size is 3 x 3.



Image



Convolved Feature

Exercise

Input image: 7 x 7

Filter size: 3 x 3

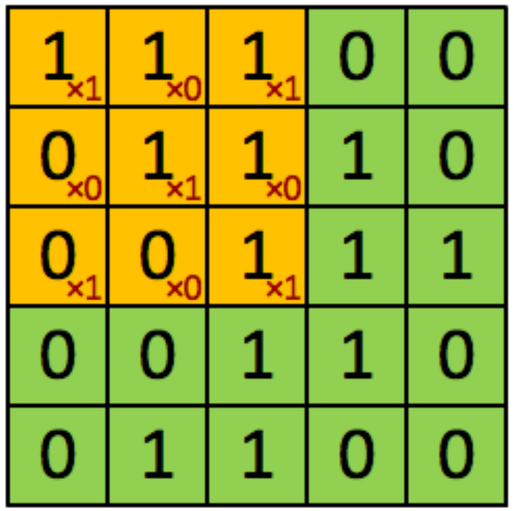
How many units in the output?

How many trainable weights?

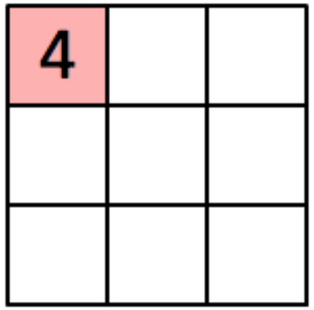
What are the values of the bottom output row?

Forward pass example

The filter/kernel (yellow) contains the trainable weights. Here filter size is 3 x 3.



Image



Convolved Feature

Exercise

Input image: 7 x 7

Filter size: 3 x 3

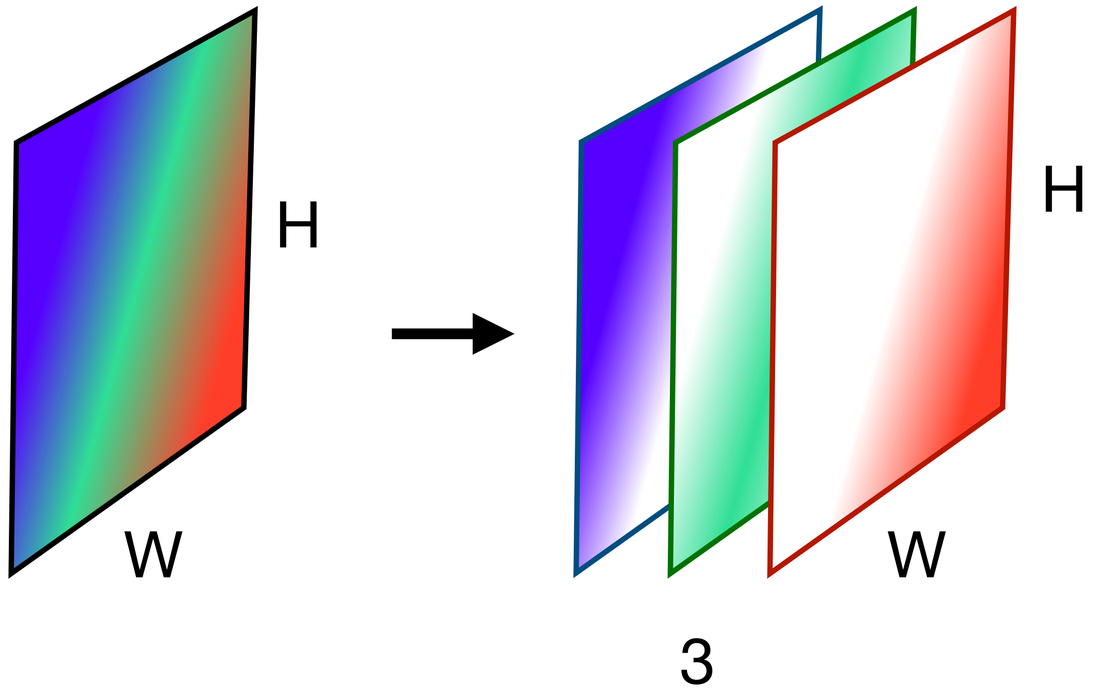
How many units in the output? 5 x 5

How many trainable weights? 9 + bias

What are the values of the bottom output row?

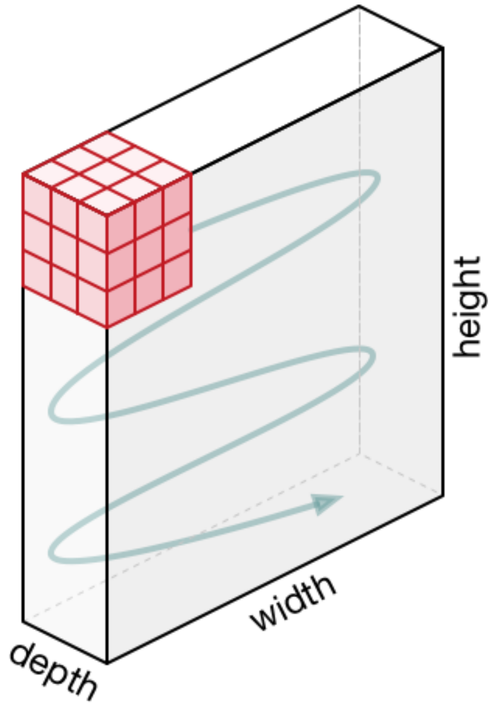
2, 3, 4

Convolutions for colour images



2D image with 3 channels (RGB)

A tensor!: PyTorch uses (N_B,C,H,W) shape

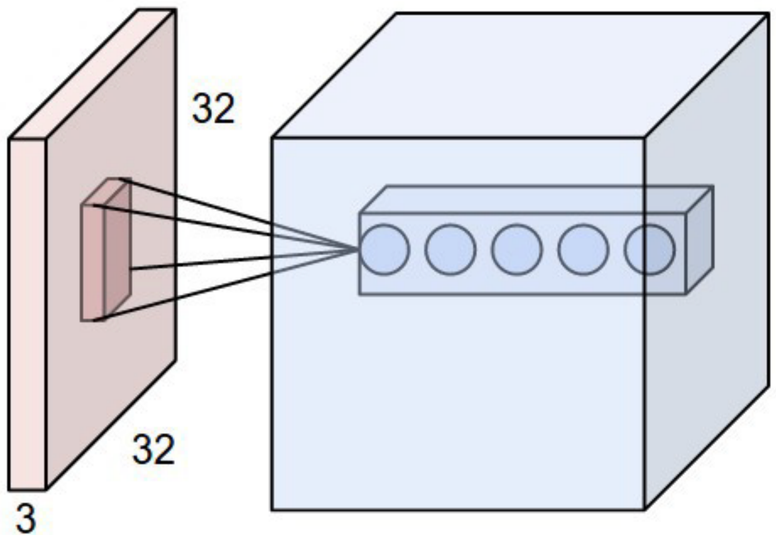


Kernel is also a 3D tensor. This example is **3** x 3 x 3

Number of **input channels** or **feature maps**

Detecting multiple features

Given a single filter map, how many features are being detected?



Have multiple filter maps to detect different features.

Example:

Input image size: 3 x 32 x 32

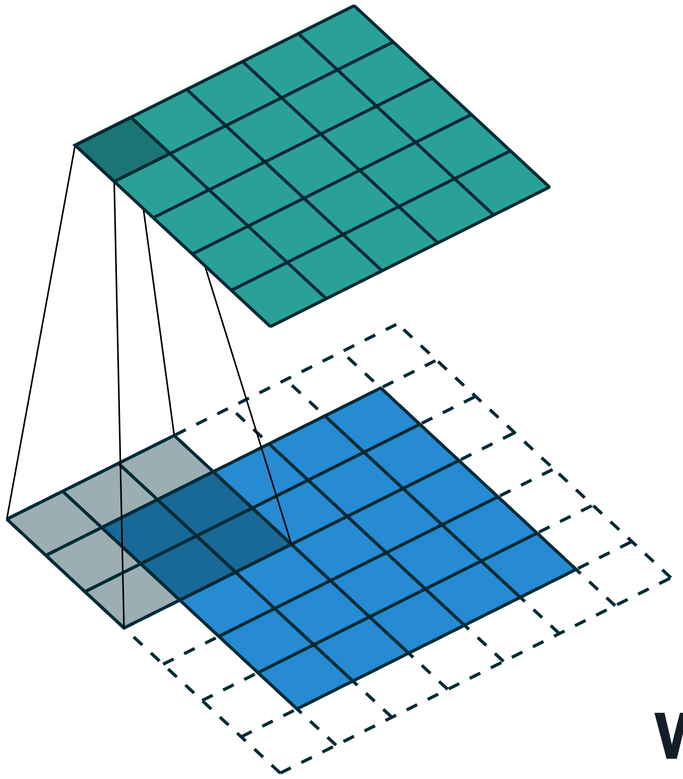
Convolutional kernel (4D): **3** x 3 x 3 x **5**

3 : number of **input channels** or **input feature maps**

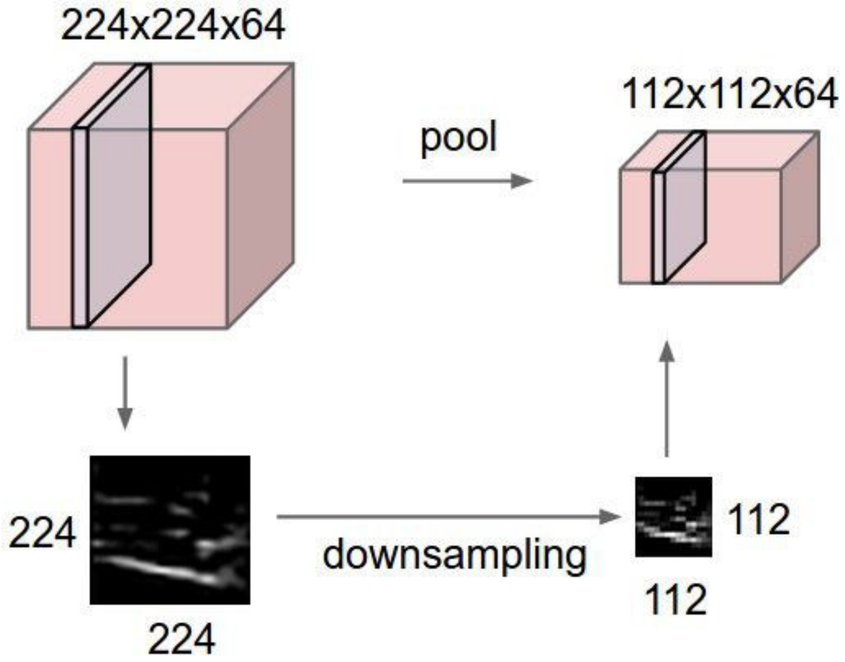
5 : number of **output channels** or **output feature maps**

Zero padding and pooling

Add zeros around the edge of the input image. Common padding size is (F-1)/2.



Downsample the feature maps by using either a max or average operation.



What are the benefits of these operations?

Rationale for zero padding and pooling

Zero padding:

To preserve the shape of the image.

To keep information that is around the edges.

Pooling:

Dimension reduction - make the representations smaller and more manageable

Operate over each feature map independently.

Common types are Max or Average.

Max pool example

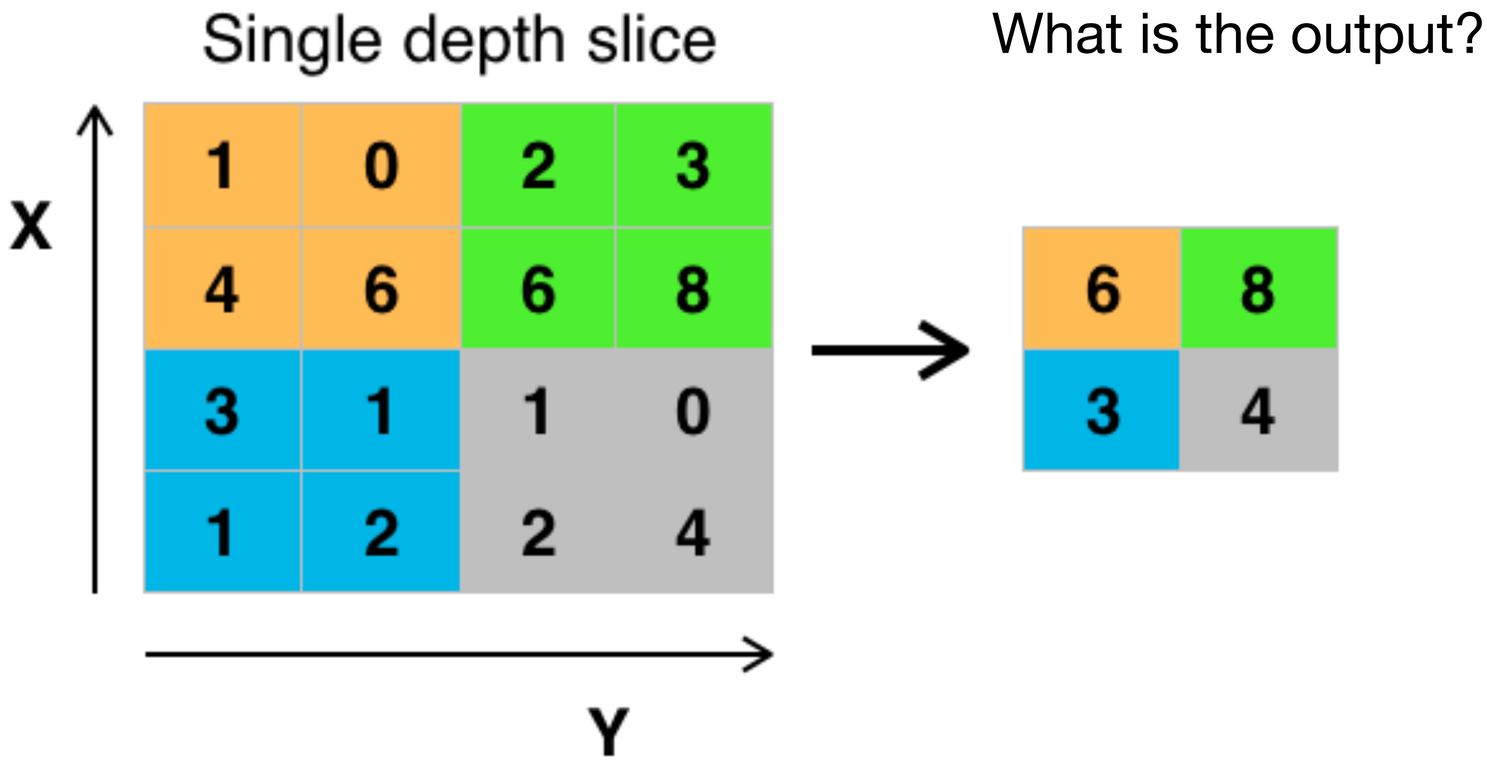
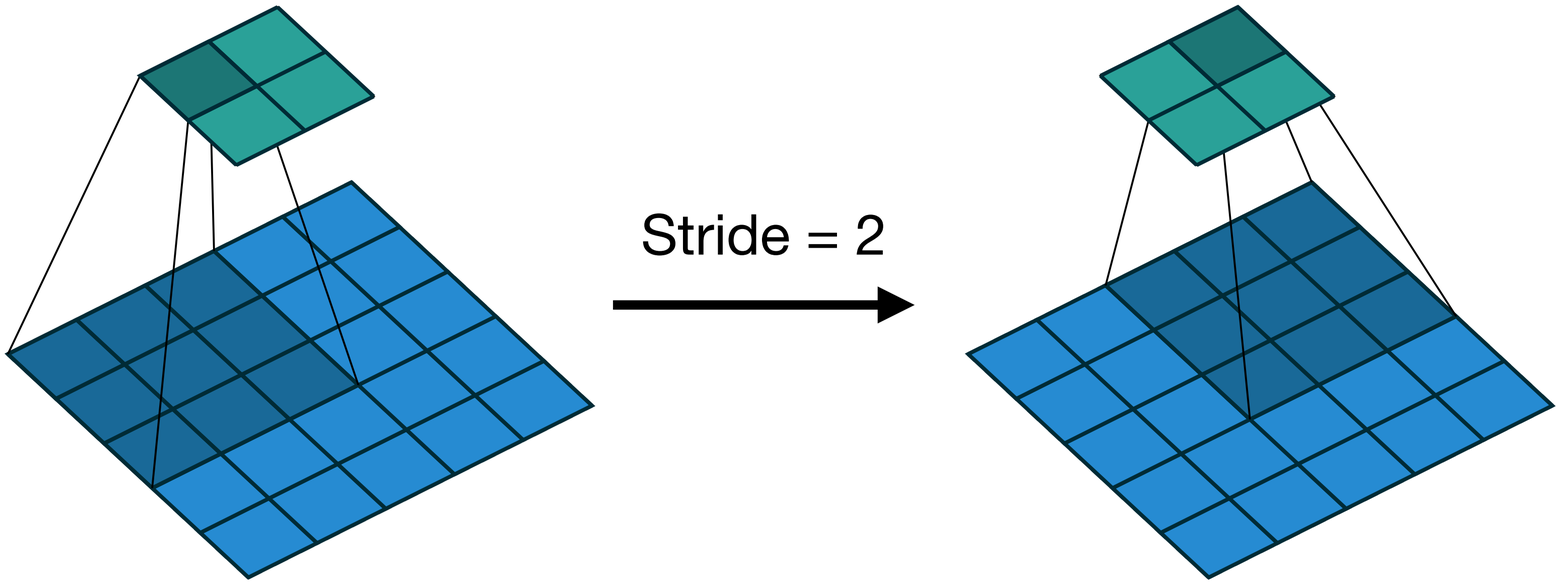


Image credit: Wikimedia (https://commons.wikimedia.org/wiki/File:Max_pooling.png)

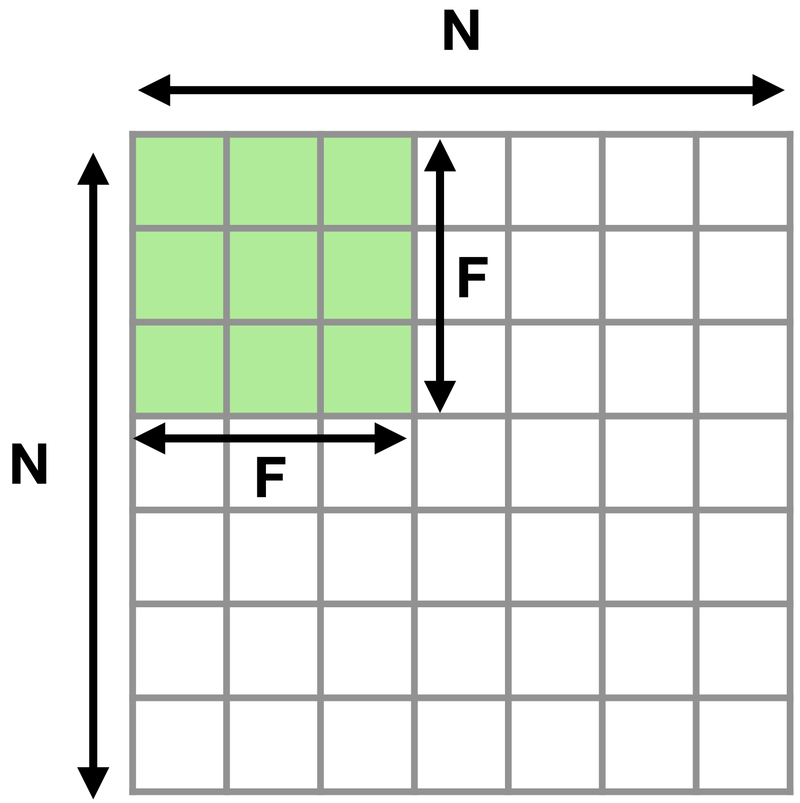
Strided convolutions

Shift the kernel by multiple pixels when computing the output feature:



Objective: to consolidate (summarise) information.

Size of output



Output size = (N-F)/stride + 1

Example

N = 7, F = 3

Stride = 1 → O = (7-3)/1 + 1 = 5

Stride = 2 or 3 ?

Exercise

Input volume = **3 x 32 x 32;** **10** filters, **5 x 5** shape with stride = **1**, pad = **2**

How many input and output channels? 3 input channels, 10 output channels

What is the output volume size? 10 x 32 x 32

How many parameters are in this layer? 10 3 x 5 x 5 filters = 750 weights

Take home messages

- Neurons perform a weighted sum of inputs followed by a non-linear function
- Neural networks connect many neurons together, the non-linearity allows for complex decision boundaries or functions to be learned.
- Deeper layers in a network detect smaller features of the input.
- Convolutional networks (generally) better for image data than fully connected layers.
- Convolutional layers share weights (filters) which detect whether features are within a local region.
- Zero padding, pooling and strides can be used to consolidate information.

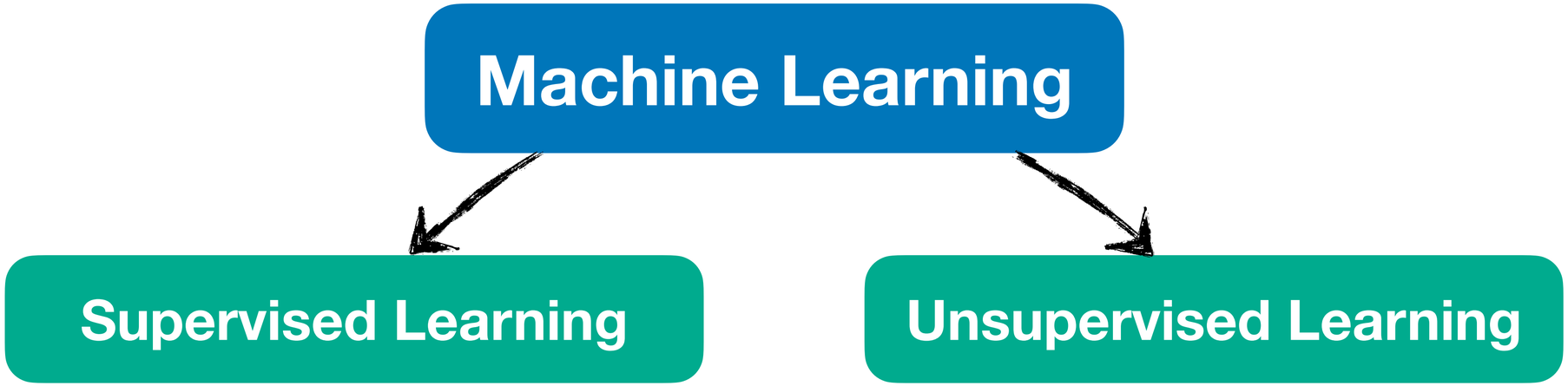
Reading

Chapter 9 of Deep Learning by Goodfellow.

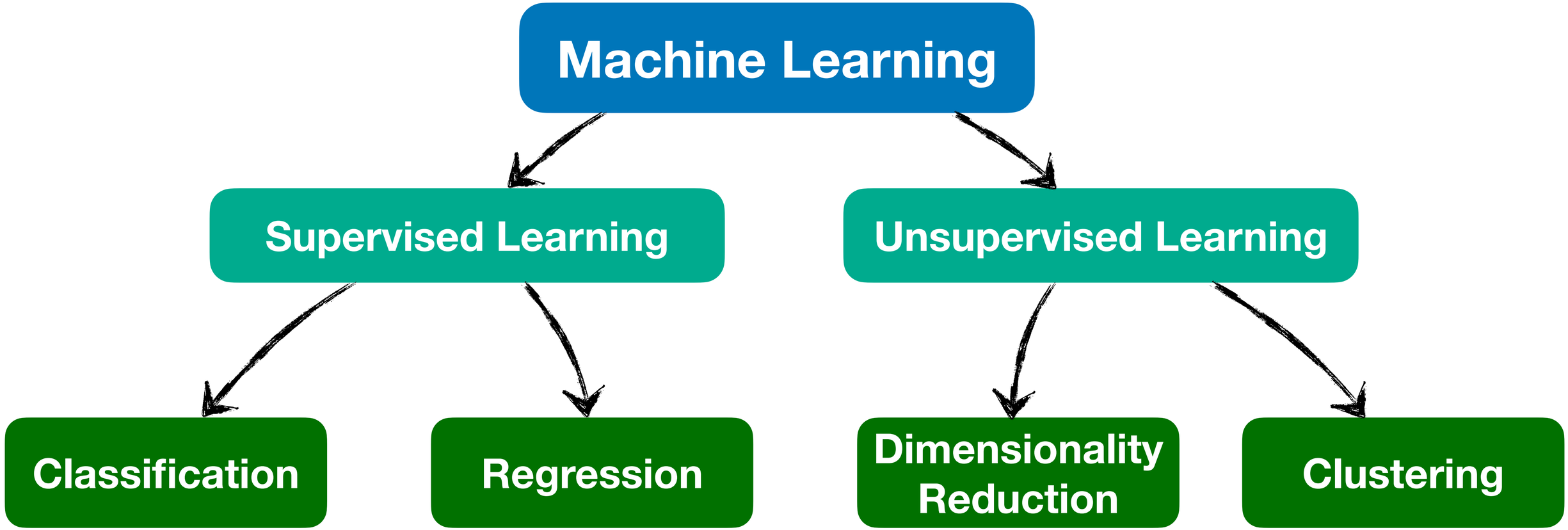
Available at https://www.deeplearningbook.org/

Unsupervised Learning

Supervised vs Unsupervised



Supervised vs Unsupervised



- Click-through-rate
- Image classification
- Diagnostics
- Fraud detection

- Market forecasting
- Ad popularity

- Big data visualisation
- Feature discovery
- Structure discovery

- Customer segmentation
- Targeted marketing
- Recommendation

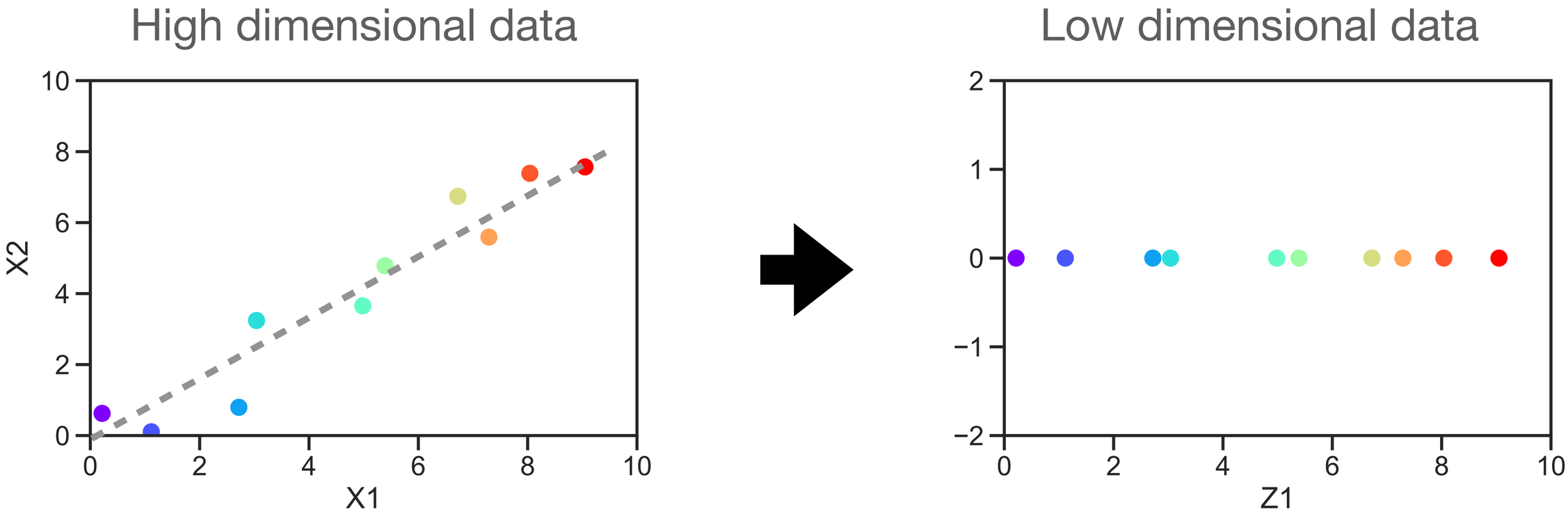
Unsupervised Learning

Supervised: each data point has a label (desired output)

Unsupervised: No labels for the data

	Supervised	Unsupervised
Discrete output	Classification	Clustering
Continuous output	Regression	Dimensionality Reduction

Dimensionality Reduction



Why apply dimensionality reduction?

Question

USPS handwritten digit dataset:

Image size: 64 by 57

Black or white pixels (1 bit)

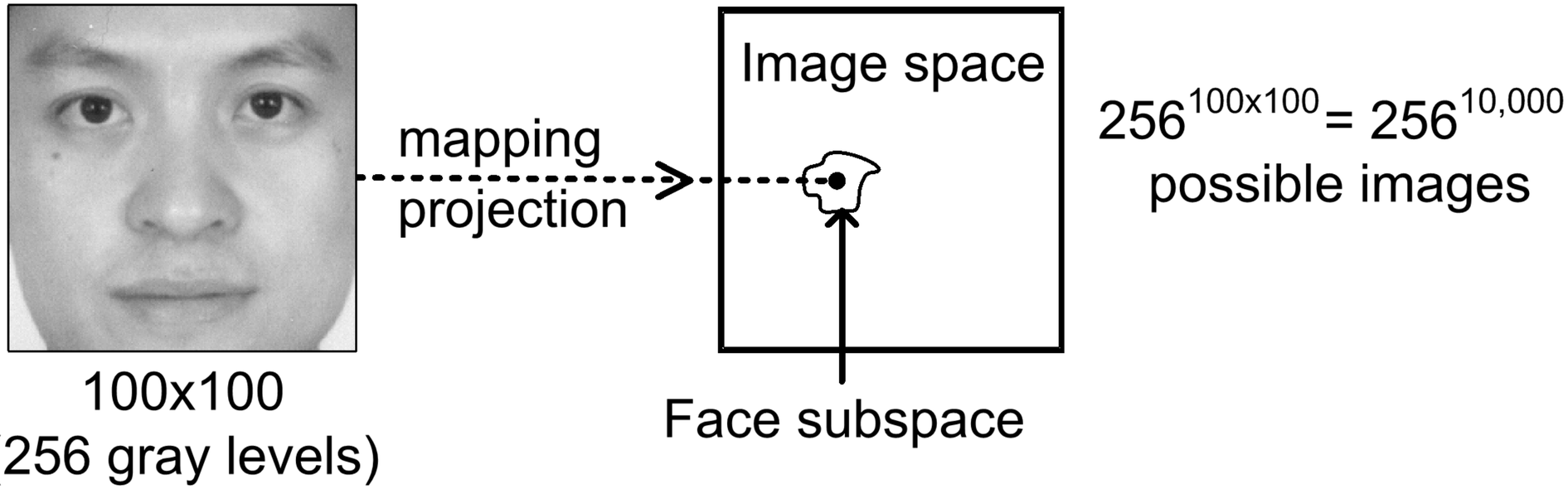
How many dimensions is a single image?

$$64 \times 57 = 3648$$

How many possible images?

$$2^{64 \times 57} = 2^{3648} = ?$$

Projection mapping



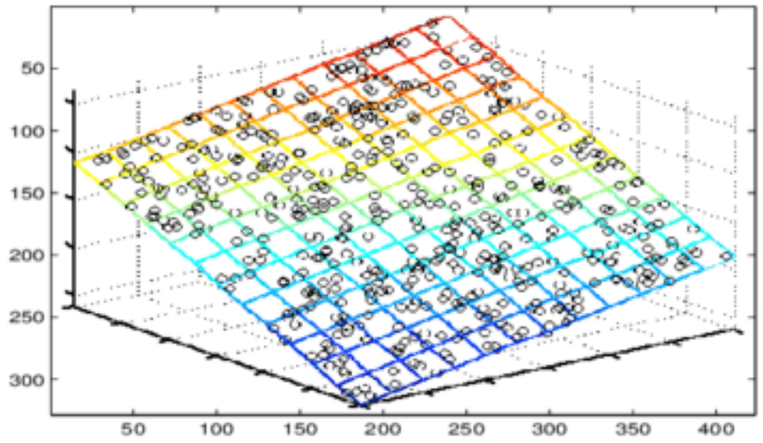
Low-D Subspace or Manifolds

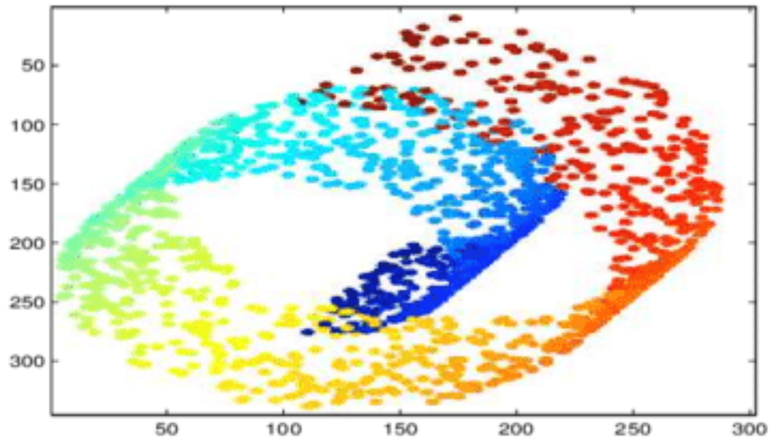
For high dimensional data with **structure:**

Fewer variations than dimensions

Data lives on a lower dimensional manifold

→ Deal with them by looking for a lower dimensional embedding (or projection)

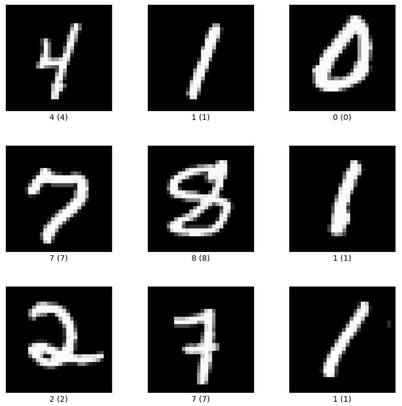


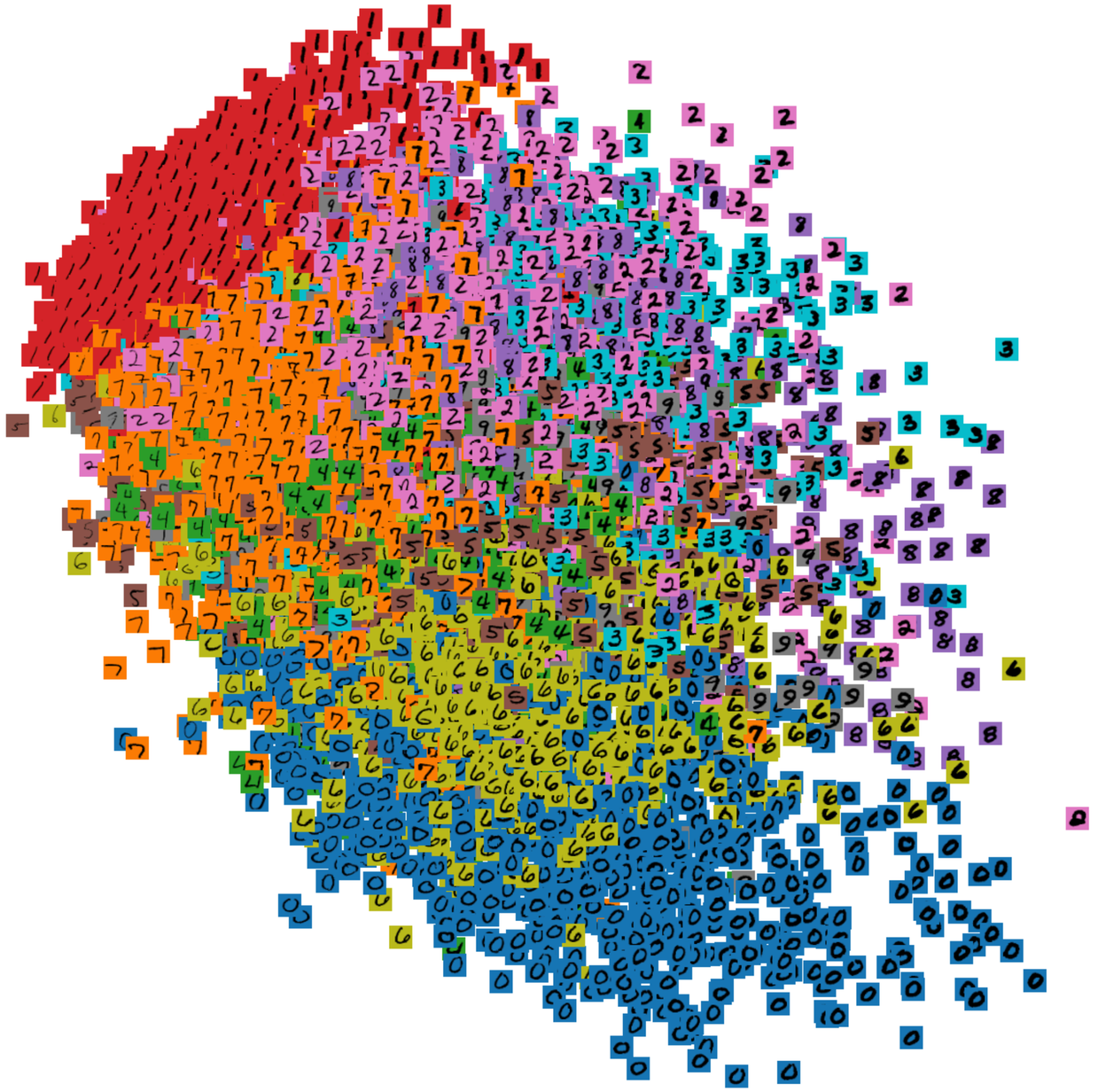


Principal Component Analysis

PCA Demo

Demonstration:
https://projector.tensorflow.org/



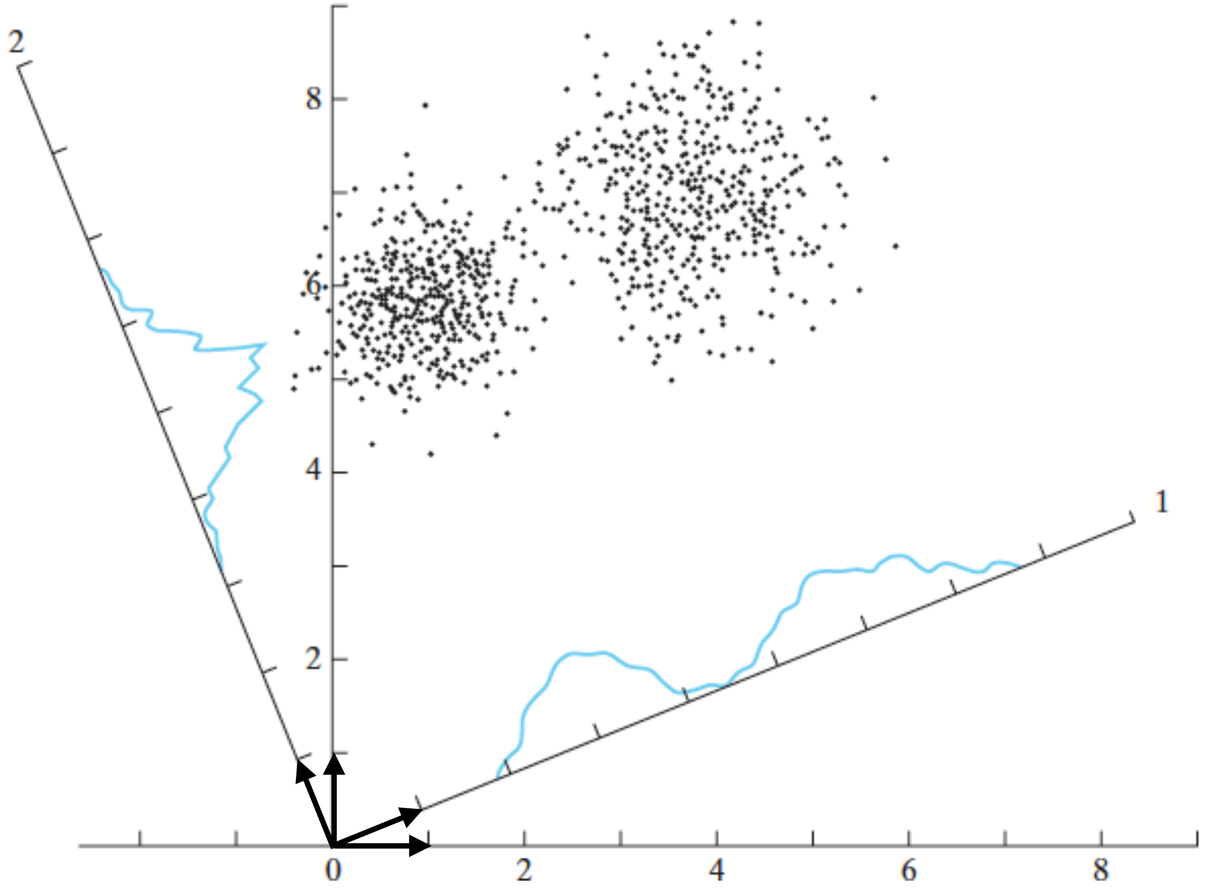


Finding a basis to best represent data

PCA

1. Rotate the data with some **rotation matrix** (linear transformation) so that features are uncorrelated.
2. Keep the dimensions with **highest variance** for DR.

Adapted from Neural Networks and Learning Machines by Simon Haykin (Pearson 2009)



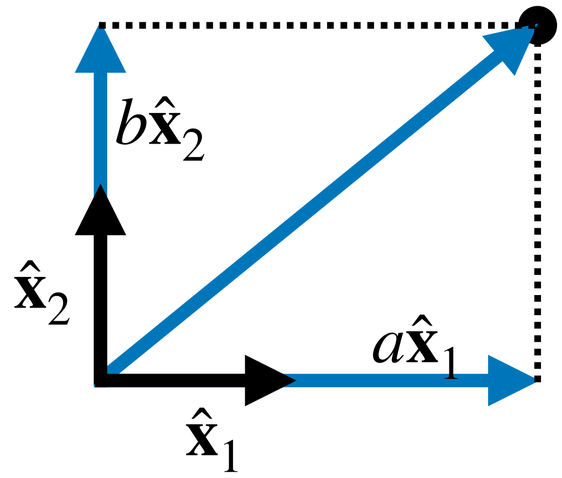
Unit vectors that form the basis

Basis vectors for datapoints

A point in space is described by its projections on a set of basis vectors.

Datapoint

$\mathbf{x} = a\hat{\mathbf{x}}_1 + b\hat{\mathbf{x}}_2$



Projections on basis vectors

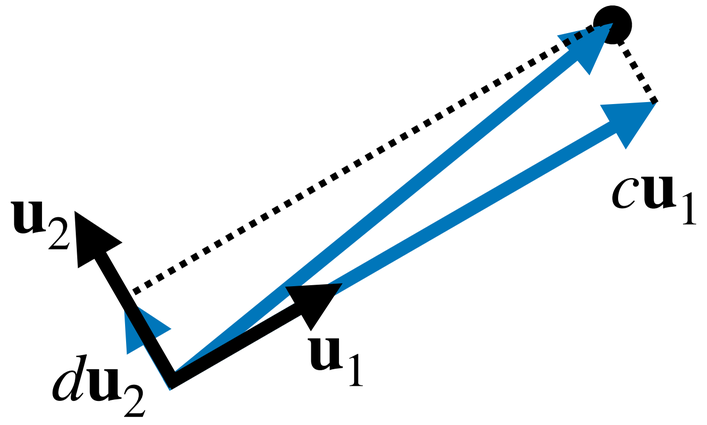
$a = \hat{\mathbf{x}}_1^T \mathbf{x}$

$b = \hat{\mathbf{x}}_2^T \mathbf{x}$

Datapoint vector = $[a, b]^T$

Datapoint in transformed basis

$\mathbf{x} = c\mathbf{u}_1 + d\mathbf{u}_2$



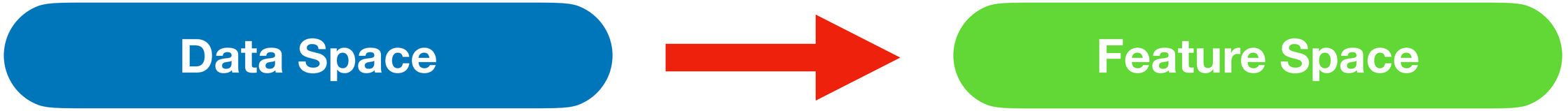
Projections on basis vectors

$c = \mathbf{u}_1^T \mathbf{x}$

$d = \mathbf{u}_2^T \mathbf{x}$

Datapoint vector in new basis = $[c, d]^T$

Rotating the basis as a transformation



$$\mathbf{x} = \begin{bmatrix} a \\ b \end{bmatrix}$$

$$\mathbf{y} = \begin{bmatrix} c \\ d \end{bmatrix} = \begin{bmatrix} \mathbf{u}_1^T \mathbf{x} \\ \mathbf{u}_2^T \mathbf{x} \end{bmatrix}$$

Transformation (rotation) matrix

$$\mathbf{U}^T = \begin{bmatrix} \mathbf{u}_1^T \\ \mathbf{u}_2^T \end{bmatrix}$$

$$\mathbf{y} = \mathbf{U}^T \mathbf{x}$$

Variances and Covariance

PCA: Find directions that **maximises the variance** of the transformed data.

Recall that:

Variance & Covariance - measure of the "spread" of a set of points around their centre (mean).

Variance (**scalar) -** measure of the spread **each dimension separately**.

Covariance - measure of how much each of the dimensions vary from the mean with **respect to each other**.

- Covariance is measured between two dimensions.
- Covariances sees if there is a relation between them.
- The covariance of a dimension with itself is the variance

PCA

Find directions that **maximises the variance** of the transformed data.

How do we find these directions?

Eigenvectors of the sample covariance (scatter) matrix

$$C_{ij} = \frac{1}{N} \sum_{n=1}^{N} (x_{ni} - \mu_i)(x_{nj} - \mu_j)$$

Question: what is the covariance matrix in the projected space?

Covariance in the projected space

The covariance is defined as

$$\mathrm{Cov}(y_i, y_j) = \frac{1}{N}\sum_{n=1}^{N}(y_{ni} - \mathbb{E}(y_i))(y_{nj} - \mathbb{E}(y_j))$$

$$y_{ni} = \mathbf{u}_i^T \mathbf{x}_n$$

$$\mathbb{E}(y_i) = \mathbf{u}_i^T \boldsymbol{\mu}$$

$$= \frac{1}{N}\sum_{n=1}^{N} \mathbf{u}_i^T(\mathbf{x}_n - \boldsymbol{\mu})(\mathbf{x}_n - \boldsymbol{\mu})^T\mathbf{u}_j$$

$$= \mathbf{u}_i^T \mathbf{C} \mathbf{u}_j$$

Eigenvectors are directions of maximum variance

Find the first direction $\mathbf{u}_1$ that maximises the variance in the projected space.

Optimisation criterion for 1st PCA including a unit-norm constraint via Lagrange multipliers:

$$L(\mathbf{u}_1, \lambda_1) = \mathbf{u}_1^T \mathbf{C} \mathbf{u}_1 + \lambda_1 \left(1 - \mathbf{u}_1^T \mathbf{u}_1\right)$$

Variance of the data in direction $\mathbf{u}_1$

Constraint to keep $|\mathbf{u}_1| = 1$

We want to maximise this, so what can we do?

Eigenvectors are directions of maximum variance

Find derivative w.r.t $\mathbf{u}_1$ and set to zero:

$$L(\mathbf{u}_1, \lambda_1) = \mathbf{u}_1^T \mathbf{C} \mathbf{u}_1 + \lambda_1 \left(1 - \mathbf{u}_1^T \mathbf{u}_1\right)$$

$$\frac{dL(\mathbf{u}_1, \lambda_1)}{d\mathbf{u}_1} = 2\mathbf{C}\mathbf{u}_1 - 2\lambda_1 \mathbf{u}_1 = \mathbf{0}$$

$$\mathbf{C}\mathbf{u}_1 = \lambda_1 \mathbf{u}_1$$

Eigenvalue problem!

Multiple solutions, which do we pick?

Selecting other transformation vectors

Further directions: **Orthogonal** (uncorrelated) to the first and each other

Use the remaining **eigenvectors** of **C:**

- **Eigenvectors** - basis vectors, principal components
- **Eigenvalues** - the variance of the data captured by that direction

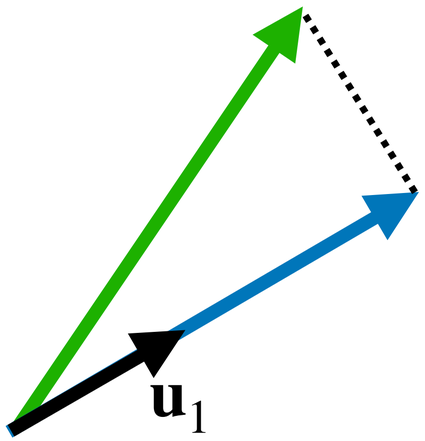
For a D dimensional input, we will have D eigenvectors.
Create transformation matrix by forming these as columns.

$$\mathbf{U} = [\mathbf{u}_1, \ldots, \mathbf{u}_D]$$

For DR, we select the *k* eigenvectors with the highest eigenvalues to keep, drop the rest.

Reconstruction error vs maximum variance

We could minimise the error when transforming back (reconstruction).



Transform $y_{n1} = \mathbf{u}_1^T \mathbf{x}_n$

Inverse Transform $\tilde{\mathbf{x}}_n = \mathbf{u}_1 y_{n1}$

$$E = \frac{1}{2N} \sum_n \left(\mathbf{x}_n - \tilde{\mathbf{x}}_n \right)^2$$

$$= \frac{1}{2N} \sum_n \left(\mathbf{x}_n - \mathbf{u}_1 (\mathbf{u}_1^T \mathbf{x}_n) \right)^2$$

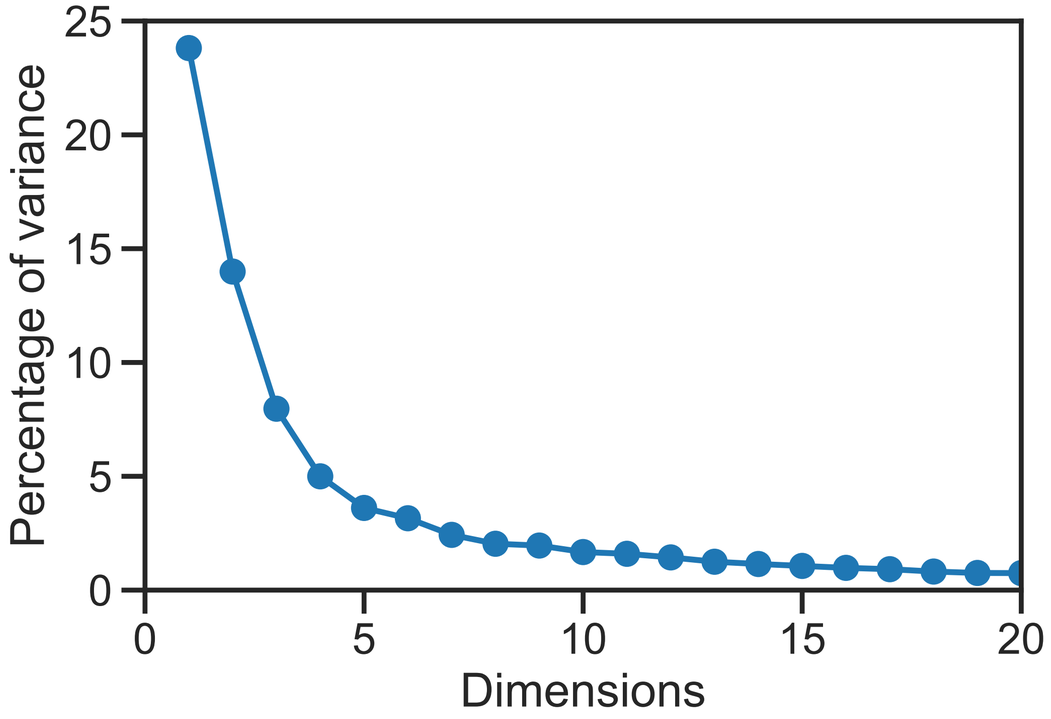
$$= - \mathbf{u}_1^T \mathbf{C} \mathbf{u}_1 + \frac{1}{N} \sum_n |\mathbf{x}_n|^2$$

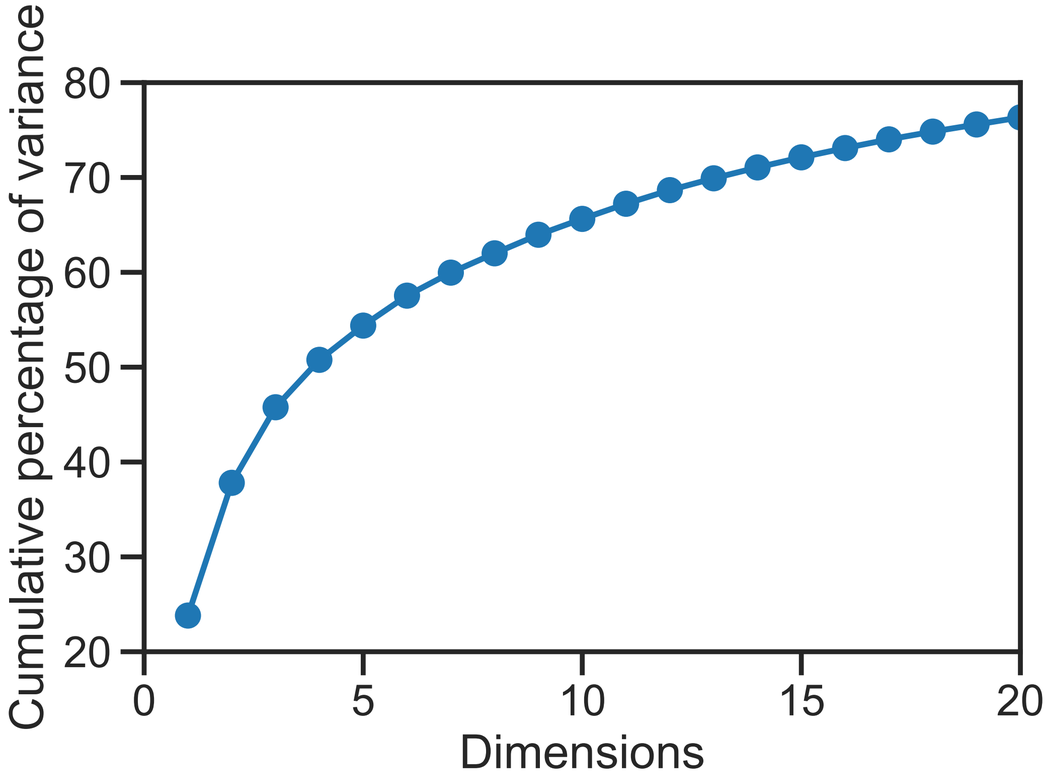
So min reconstruction error = max variance

How many dimensions to keep?

Pick based on percentage of variance kept/lost

Look for an ‘elbow’ in a **scree plot** (plot of explained variance or eigenvalues).





Representation and Reconstruction

Example: Olivetti faces dataset.

Representation:
Face $\mathbf{x}$ in 'face space' coordinates.

$$\mathbf{y} = \mathbf{U}^T(\mathbf{x} - \boldsymbol{\mu}) = \begin{bmatrix} y_1 \\ \vdots \\ y_k \end{bmatrix}$$

Reconstruction: use representation to rebuild using eigenvectors as basis.

$=$

$+$

$$\tilde{\mathbf{x}} = \boldsymbol{\mu} + y_1\mathbf{u}_1 + y_2\mathbf{u}_2 + y_3\mathbf{u}_3 + \ldots$$

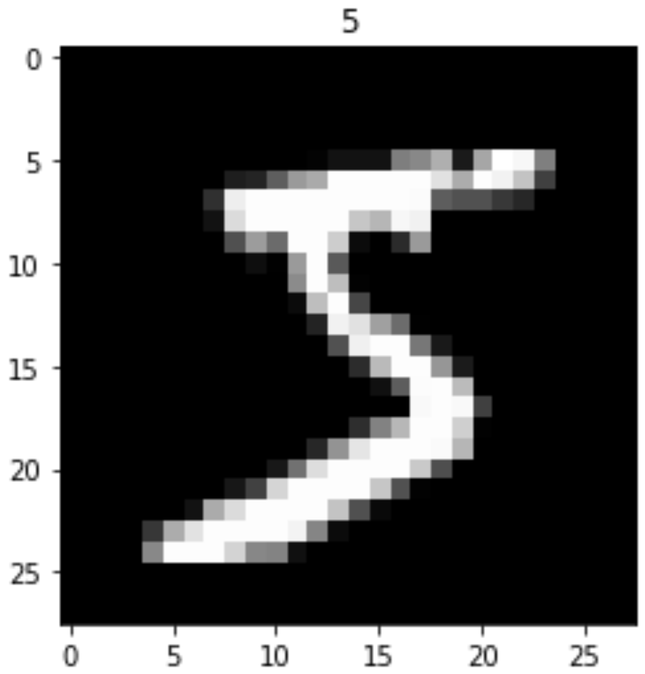
PCA ingredients

- **Data** + pre-processing
- **Model**:
  - **Structure/architecture**: Linear projection layer $\mathbf{y} = \mathbf{U}^T\mathbf{x}$
  - **Parameters**: The principal components (eigenvectors)
  - **Hyper-parameters**: Number of components to keep, *k*
- **Evaluation metric**: Variance explained by the PCs
- **Optimisation**: Eigen-decomposition

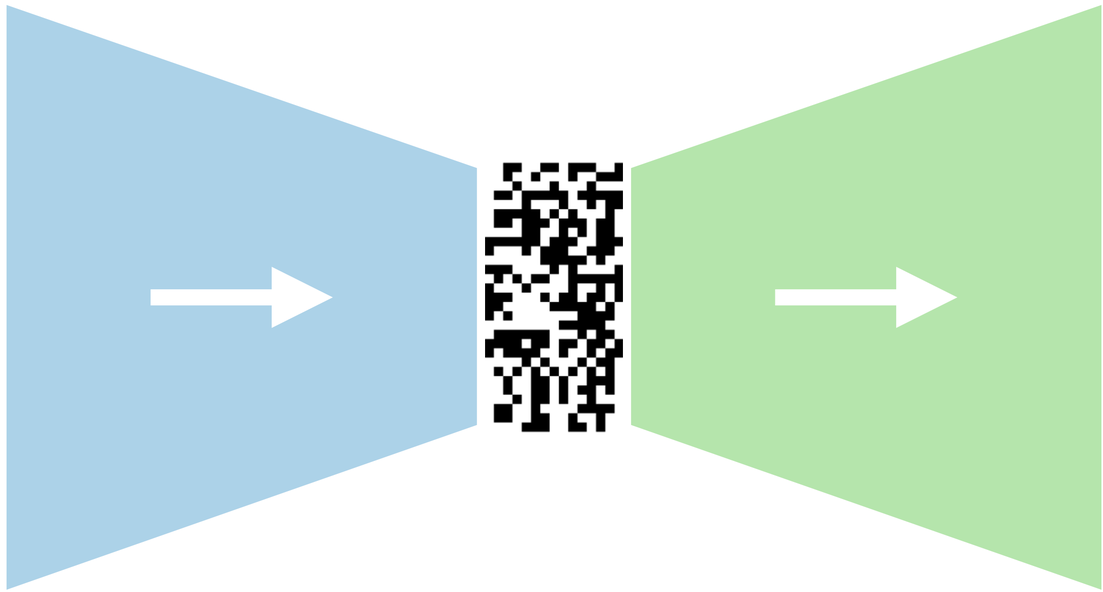
Auto-encoders

DR as encoding

Original data



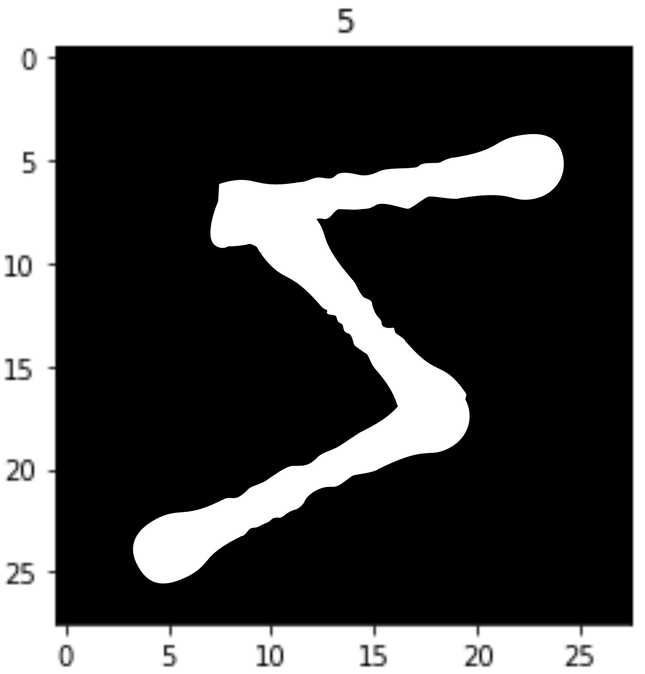
‘Code’
Useful representation



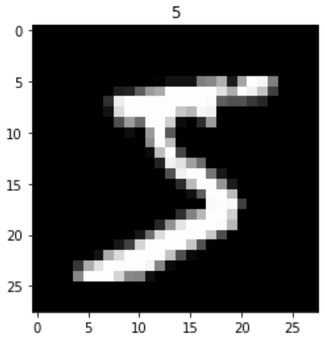
Encoder
Compress or extract features

Decoder
Recover data from code

Decoded data

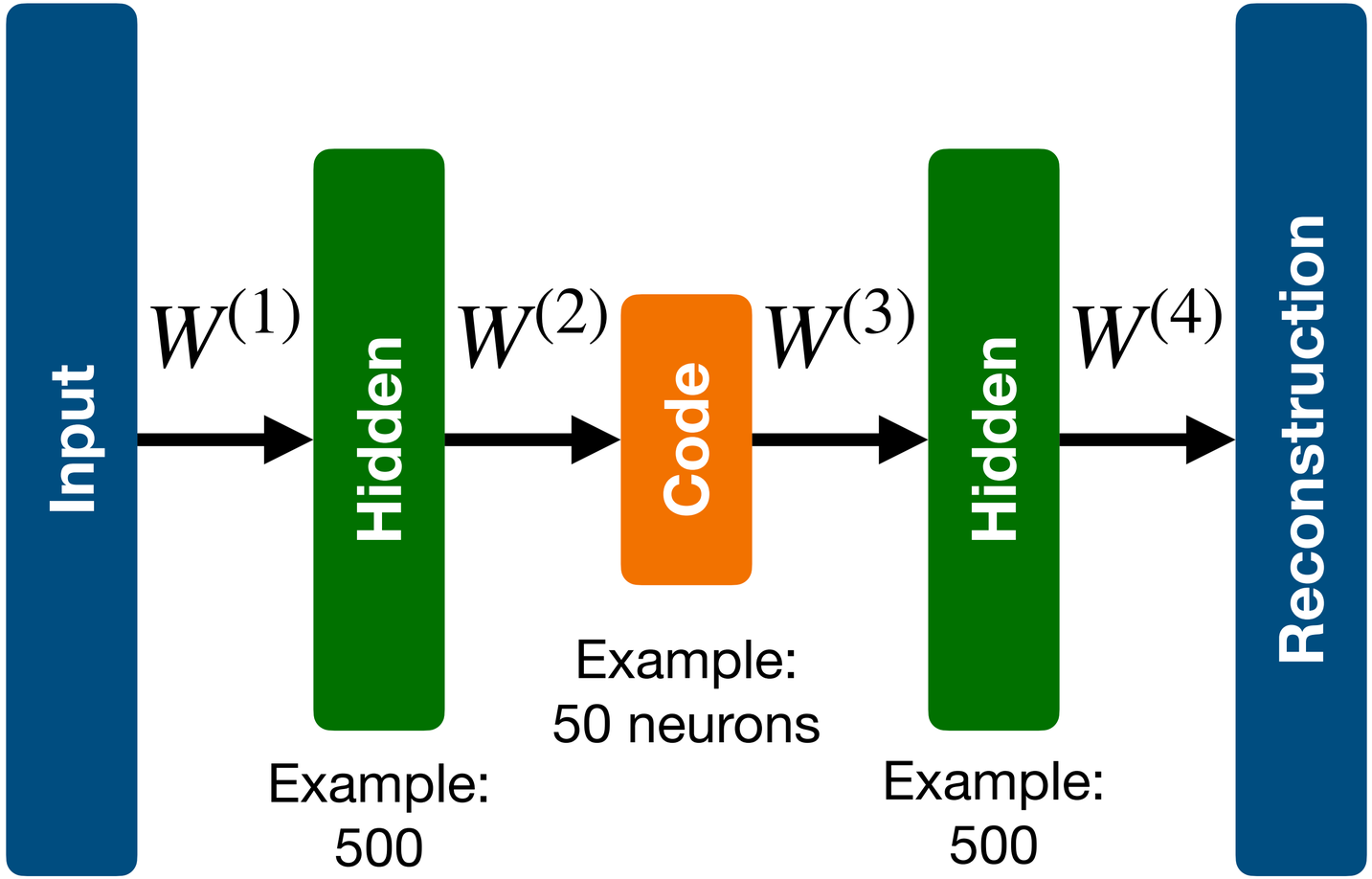


Auto-encoder structure



Example:
782 neurons

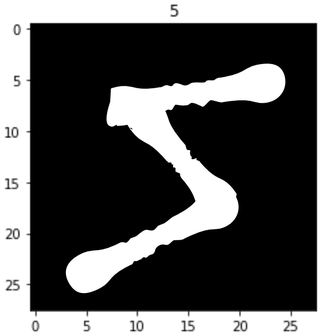
$\mathbf{x}_n$



Example:
500
neurons

Example:
50 neurons

Example:
500
neurons



Example:
782 neurons

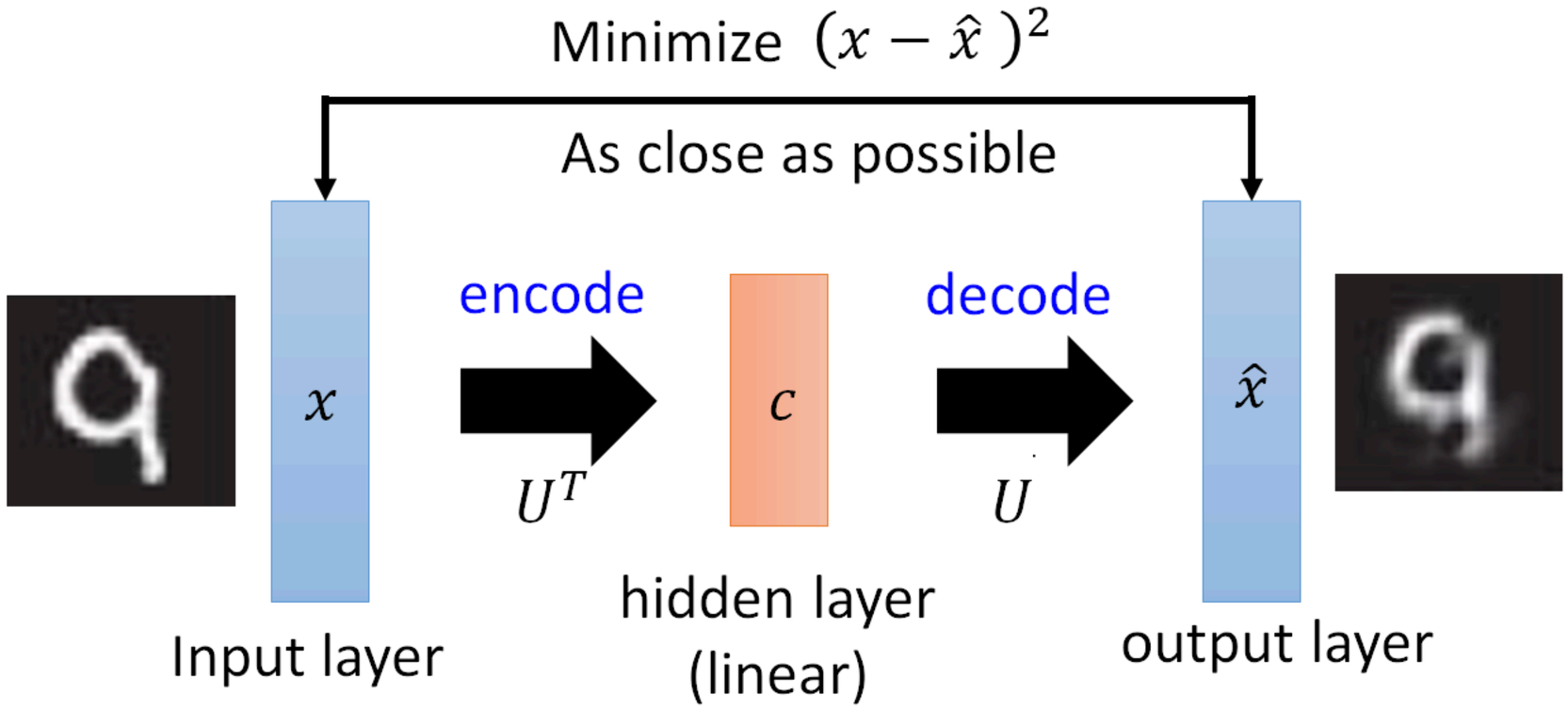
$\tilde{\mathbf{x}}_n$

What will our objective (error or loss) function be?

$$E = \frac{1}{2N} \sum_n \left(\mathbf{x}_n - \tilde{\mathbf{x}}_n \right)^2$$

PCA as a linear Auto-encoder

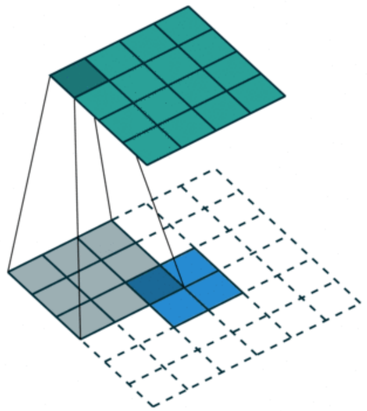
PCA: single weight layer shared as encoder and decoder.



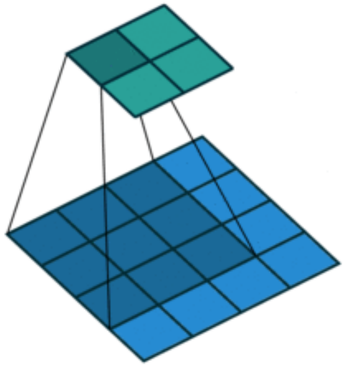
Transposed Convolutional Layer

Need to ‘decode’ the output of the convolutions.

No padding or strides

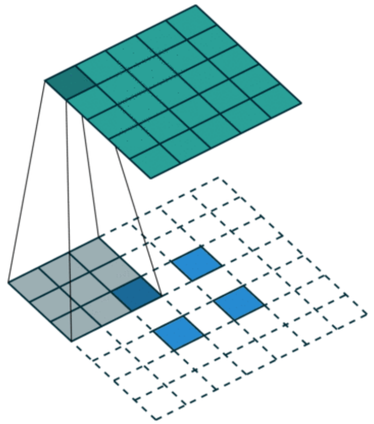


Reconstructed input
needs to be 4 by 4,
expand from 2 by 2

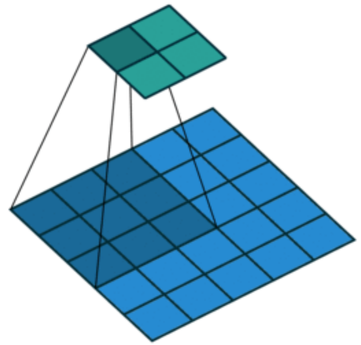


Original input is 4 by 4
Convolution result is 2 by 2

No padding, stride = 2



Reconstructed input
needs to be 5 by 5,
expand from 2 by 2



Original input is 5 by 6
Convolutional result is 2 by 2

More at https://github.com/vdumoulin/conv_arithmetic

Auto-encoder ingredients

- **Data** + pre-processing
- **Model**:
  - **Structure/architecture**: encoder + decoder, multiple neural network layers
  - **Parameters**: layer weights and biases
  - **Hyper-parameters**: number of layers, layer parameters, learning rate
- **Evaluation metric**: Mean-Squared Error (reconstruction error)
- **Optimisation**: Gradient descent (back-prop) or similar

PyTorch Auto-encoder

```
class Autoencoder(nn.Module):
    def __init__(self):
        super(Autoencoder, self).__init__()
        self.encoder = nn.Sequential(
            # 1 input image channel, 16 output channel, 3x3 square convolution
            nn.Conv2d(1, 16, 3, stride=2, padding=1),
            nn.ReLU(),
            nn.Conv2d(16, 32, 3, stride=2, padding=1),
            nn.ReLU(),
            nn.Conv2d(32, 64, 7)
        )
        self.decoder = nn.Sequential(
            nn.ConvTranspose2d(64, 32, 7),
            nn.ReLU(),
            nn.ConvTranspose2d(32, 16, 3, stride=2, padding=1, output_padding=1),
            nn.ReLU(),
            nn.ConvTranspose2d(16, 1, 3, stride=2, padding=1, output_padding=1),
            nn.Sigmoid()  #to range [0, 1]
        )

    def forward(self, x):
        x = self.encoder(x)
        x = self.decoder(x)
        return x
```

Take Home Messages

- Unsupervised learning: **no labels**, learn from **data only**.
- Dimensionality reduction: useful for visualisation, feature discovery.
- Principal component analysis (PCA): use **eigenvectors** of the **covariance matrix** to find directions of **maximum variance.**
- Auto-encoder: use **neural network** to **encode data into a 'code'**, use a mirrored network to **decode** it to **reconstruct original data**.
- **Reconstruction error** - mean squared difference of our recovered data (after transforming back) against the original data.

Further Reading

Chapter 7 (sections 7.1 and 7.2) in A First Course in Machine Learning by Rogers and Girolami.

Chapter 14 of Deep Learning by Goodfellow.

Available at https://www.deeplearningbook.org/