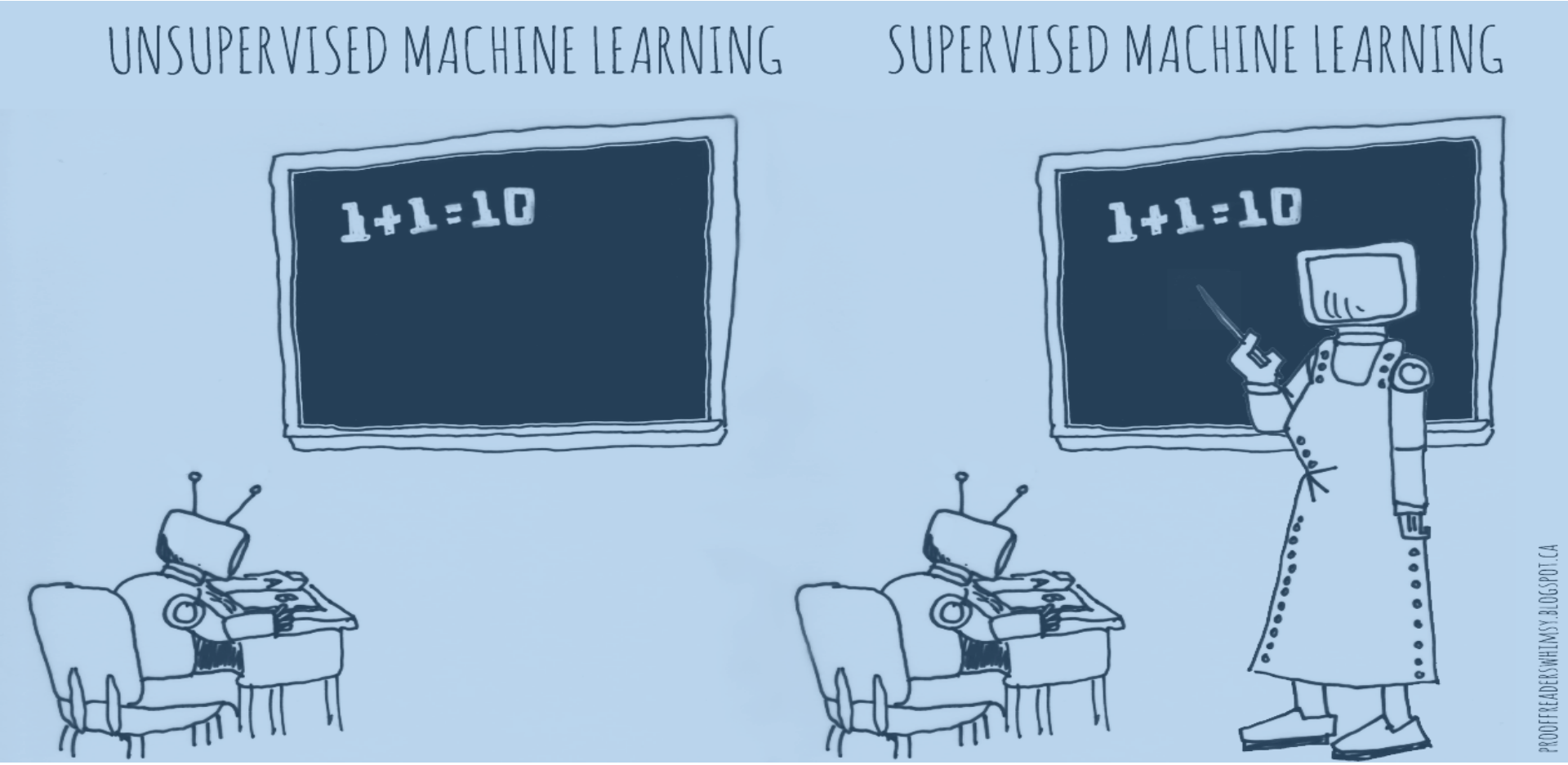


Lecture 7
Unsupervised Learning
PCA, k-means & Autoencoder

Haiping Lu - MLAI19

DL Coverage Poll: From 112

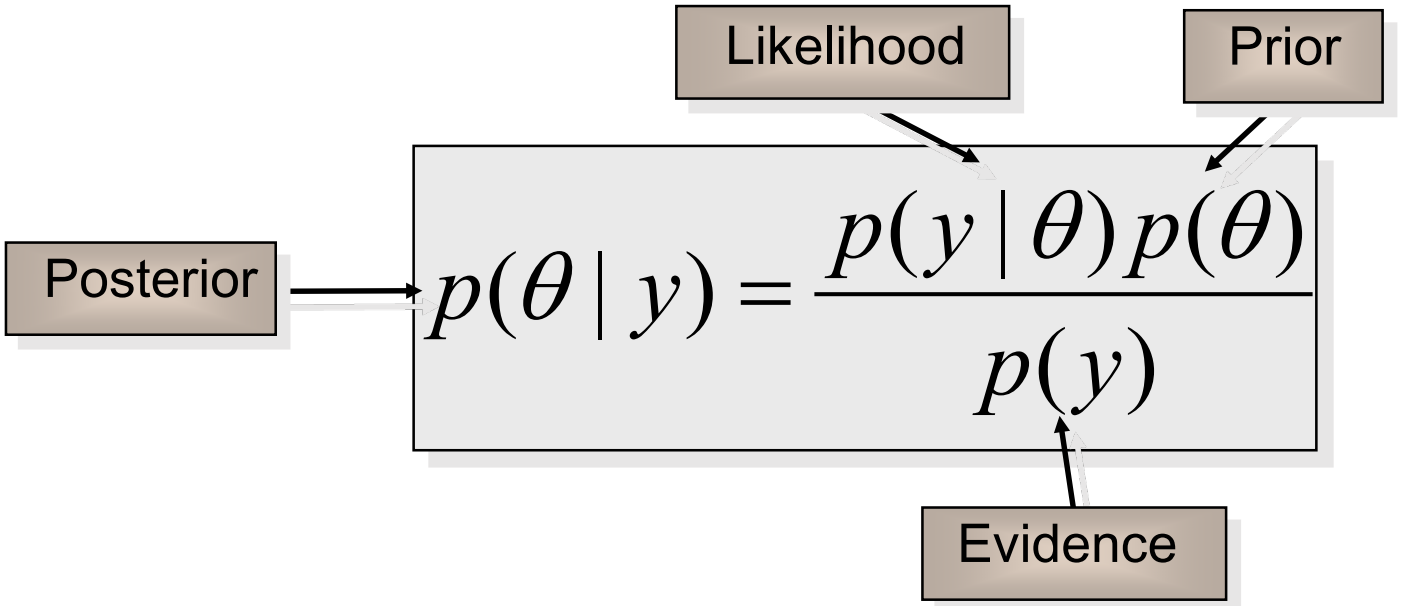
Question 1: Calculated Numeric **Average Score 0.00009 points**

Do you want more or less deep learning coverage? Enter **1** if you want less, **2** if you want more, **3** if you do not care

Correct	Answers	Percent Answered
✓	1	23.214%
✓	2	59.821%
✓	3	13.392%
	more	0.892%
	less	1.785%
	Unanswered	0.892%

Review of previous lecture

- Bayesian inference: placing a probability distribution (prior density) over the model parameters



- Process:

likelihood $p(y|\theta)$

prior distribution $p(\theta)$

y

$$p(\theta \mid y) \propto p(y \mid \theta)\, p(\theta)$$

Question from Discussion Board

Normal densities

$$p(\beta) = N(\beta; \mu_p, \alpha_p^{-1})$$

$$p(y \mid \beta) = N(y; \beta x, \alpha_e^{-1})$$

$$p(\beta \mid y) = N(\beta; \mu, \alpha^{-1})$$

$$\alpha = \alpha_e x^2 + \alpha_p$$

$$\mu = \alpha^{-1}\left(\alpha_e xy + \alpha_p \mu_p\right)$$

From Slide 18 – Lecture 6

Marginal and Conditional Gaussians

Given a marginal Gaussian distribution for x and a conditional Gaussian distribution for y given x in the form

$$p(\mathbf{x}) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Lambda}^{-1}) \quad (2.113)$$

$$p(\mathbf{y}|\mathbf{x}) = \mathcal{N}(\mathbf{y}|\mathbf{A}\mathbf{x} + \mathbf{b}, \mathbf{L}^{-1}) \quad (2.114)$$

the marginal distribution of y and the conditional distribution of x given y are given by

$$p(\mathbf{y}) = \mathcal{N}(\mathbf{y}|\mathbf{A}\boldsymbol{\mu} + \mathbf{b}, \mathbf{L}^{-1} + \mathbf{A}\boldsymbol{\Lambda}^{-1}\mathbf{A}^{\mathrm{T}}) \quad (2.115)$$

$$p(\mathbf{x}|\mathbf{y}) = \mathcal{N}(\mathbf{x}|\boldsymbol{\Sigma}\{\mathbf{A}^{\mathrm{T}}\mathbf{L}(\mathbf{y} - \mathbf{b}) + \boldsymbol{\Lambda}\boldsymbol{\mu}\}, \boldsymbol{\Sigma}) \quad (2.116)$$

where

$$\boldsymbol{\Sigma} = (\boldsymbol{\Lambda} + \mathbf{A}^{\mathrm{T}}\mathbf{L}\mathbf{A})^{-1}. \quad (2.117)$$

PRML book - Section 2.3.3

Check out other Qs there

- Why evidence is considered a constant (*univariate posterior in Lab 6 A3 – Main Trick*)?

- Why do we need to scale the input in Lab 6 A2?
  - VIP preprocessing: feature scaling

- Covariance (matrix) vs variance (scalar)

Week 7 Contents / Objectives

Part A

- Dimensionality Reduction
- Principal Component Analysis
- *k*-means Clustering

Part B

- Autoencoder
- Convolutional Neural Networks

Week 7 Contents / Objectives

Part A

- **Dimensionality Reduction**
- Principal Component Analysis
- *k*-means Clustering

Part B

- Autoencoder
- Convolutional Neural Networks

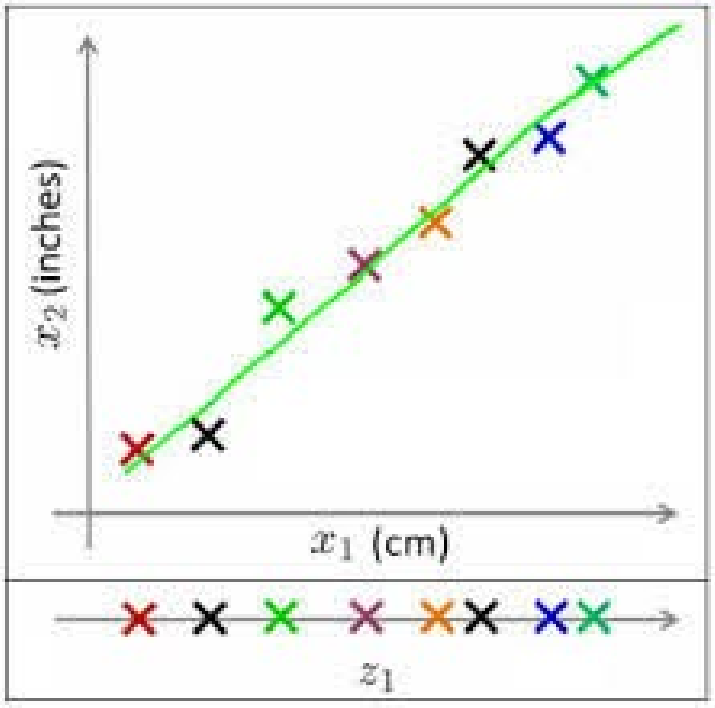
Unsupervised Learning

- Supervised learning: each data point has a label
- Unsupervised learning: no labels for the data

- Structure discovery
  - **Dimensionality reduction**
  - Learning features in the data / representations of data
  - Exploratory data analysis
  - Visualisation

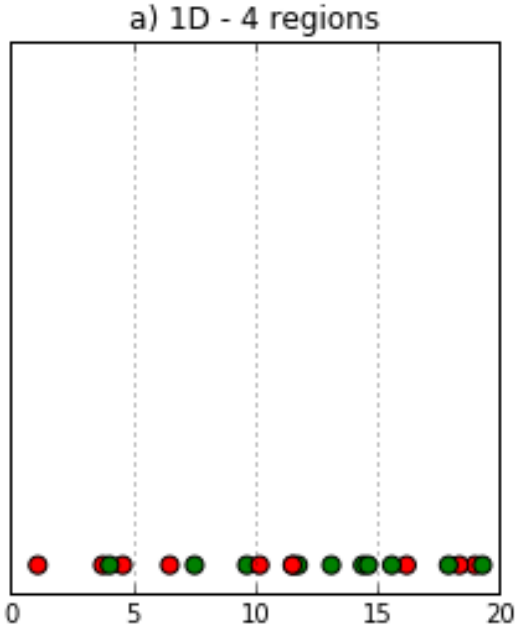
Dimensionality Reduction (DR)

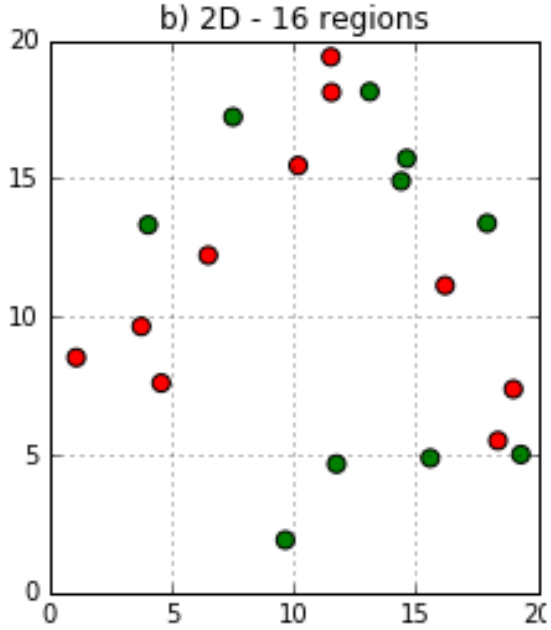
- 2 → 1

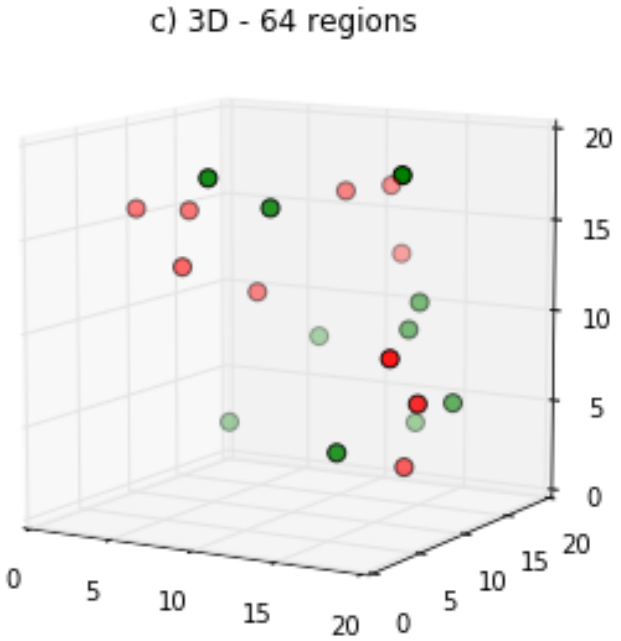


High Dimensional → Low Dimensional

Original data	Transformed
(1, 1.2)	1.15
(2, 2)	2
(3, 3.3)	3.1
…	…







Why DR?
High D → Low D

- **Curse of dimensionality**
  - **Nearest neighbours**
- Reduce redundancy (correlation)
- Visualisation

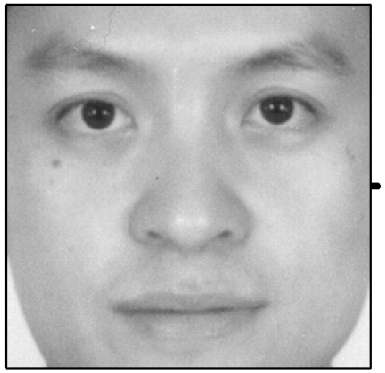
Question

- USPS Data Set Handwritten Digit
- Size: 64 x 57; binary (1-bit, BW)
- The binary image space contains much more than just this digit.

How many possible images of this size and bit depth?

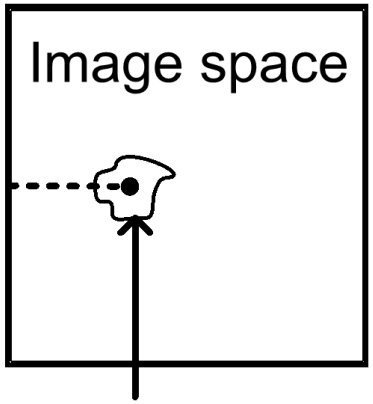
$$2^{64\times 57} = 2^{3648} = ?$$

How About a Face?



100x100
(256 gray levels)

mapping
projection

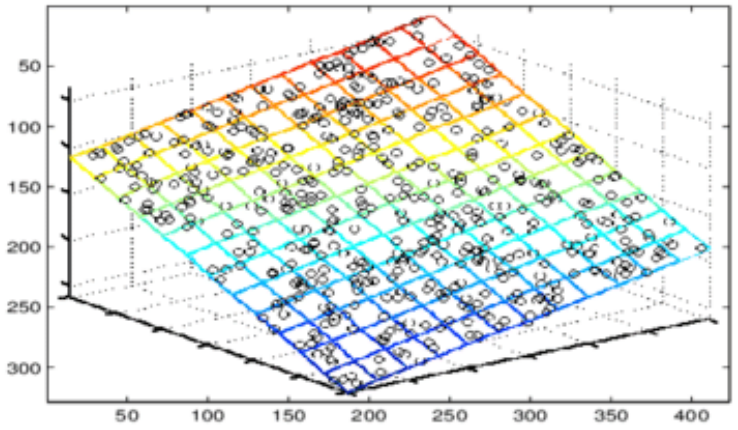


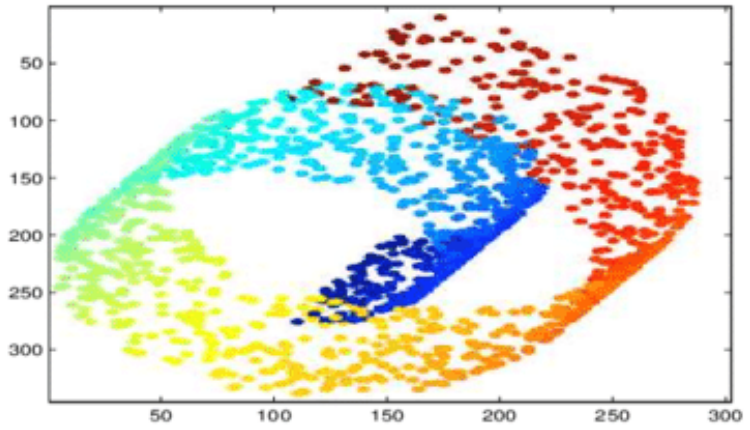
Face subspace

$256^{100x100} = 256^{10,000}$
possible images

Low-D Subspace/Manifolds

- For high dimensional data with **structure**:
  - Fewer variations than dimensions
  - Data to live on a lower dimensional manifold
  - → Deal with them by looking for a lower dimensional embedding (or projection, transformation)





Week 7 Contents / Objectives

Part A

- Dimensionality Reduction
- **Principal Component Analysis**
- *k*-means Clustering

Part B

- Autoencoder
- Convolutional Neural Networks

PCA?

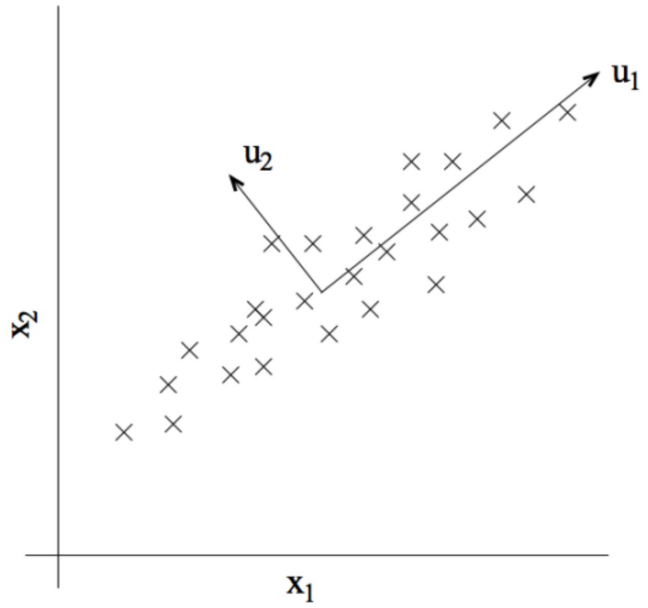
- Visualisation demo

https://projector.tensorflow.org/

Principal Component Analysis

Key Ideas

1. **Rotate** the data with some rotation matrix **R** (change of basis) so that the new features are **uncorrelated**
2. **Keep** the dimension with **the highest variance** for DR



Principal Component Analysis

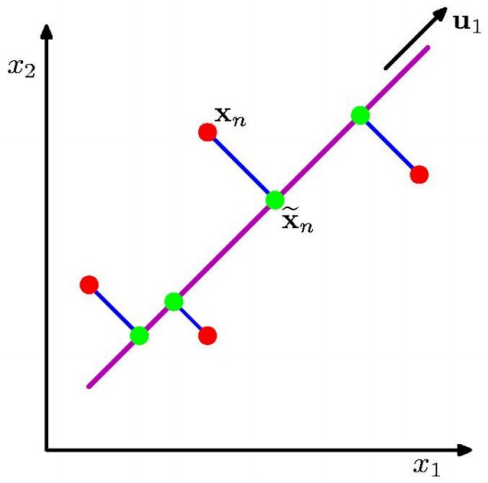
- PCA (@Hotelling:analysis33): a linear embedding
- Rotate to find *directions* in data with max-variance
- How do we find these directions?
  - Diagonalising the **sample covariance matrix** (a.k.a. **scatter matrix**)

$$\mathbf{S} = \frac{1}{n} \sum_{i=1}^{n} \left(\mathbf{x}^{(i)} - \boldsymbol{\mu} \right) \left(\mathbf{x}^{(i)} - \boldsymbol{\mu} \right)^{\top}$$

- **Interesting observation**: Compare this definition with the *k*-means objective function (Lab 7.A3)

Principal Component Analysis

- Given data **x**, PCA finds **orthogonal** directions defined by a projection (rotation) **U** capturing the **maximum variance** in the data
- PCA representation $\mathbf{y} = \mathbf{U}^T\mathbf{x}$
- Max variance = min reconstruction error



- **Question**: given the PCA representation **y**, how to obtain an approximation/reconstruction of **x**?

$$\hat{\mathbf{x}} = \mathbf{U}\mathbf{y}$$

Lagrangian for PCA Solution

- Find the first direction via (unit-norm) constrained optimisation, using Lagrange multipliers:

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

- Gradient w.r.t. $\mathbf{u}_1$: $\dfrac{\mathrm{d}L(\mathbf{u}_1, \lambda_1)}{\mathrm{d}\mathbf{u}_1} = 2\mathbf{S}\mathbf{u}_1 - 2\lambda_1\mathbf{u}_1$

分散を最大化させたいので，u を最大化
uの内積は1という条件が付いている

- Set to 0 and rearrange: $\mathbf{S}\mathbf{u}_1 = \lambda_1\mathbf{u}_1$
  - Eigenvalue problem
- Further directions: **orthogonal** (uncorrelated) to the first and each others → top *k* eigenvectors of **S**
  - Eigenvectors: basis functions, principal components
  - Eigenvalue: the **variance** captured respectively

Questions

- For $\mathbf{u}_1$, what if we do have the unit-norm constraint?

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

  - Trivial solution: infinity

- For further directions: what if we do not require them to be orthogonal?

  - We will have the same $\mathbf{u}_1$, useless solution

Representation & Reconstruction

- Face **x** in *k* “face space” coordinates:

$$\mathbf{x} \rightarrow [\mathbf{u}_1^{\mathrm{T}}(\mathbf{x}-\mu), \ldots, \mathbf{u}_k^{\mathrm{T}}(\mathbf{x}-\mu)]$$
$$= w_1, \ldots, w_k$$

- Reconstruction:

$$\hat{x} = \mu + w_1u_1 + w_2u_2 + w_3u_3 + w_4u_4 + \ldots$$

slide by Derek Hoiem

Reconstruction

k = 4

k = 200

k = 400

After computing eigenfaces using 400 face images from ORL face database

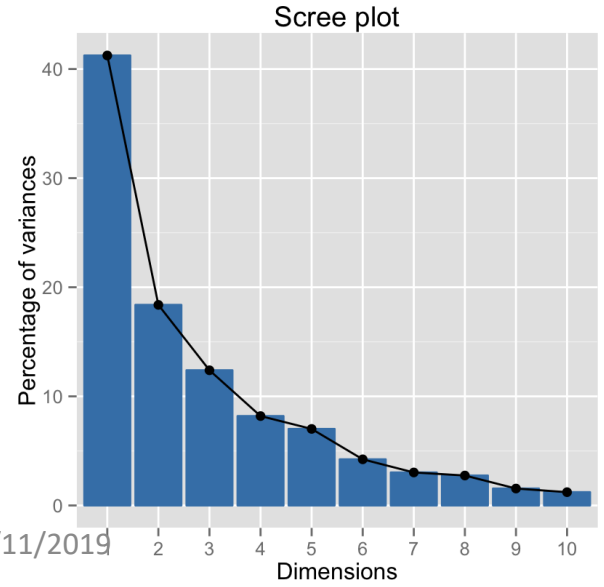
slide by Derek Hoiem

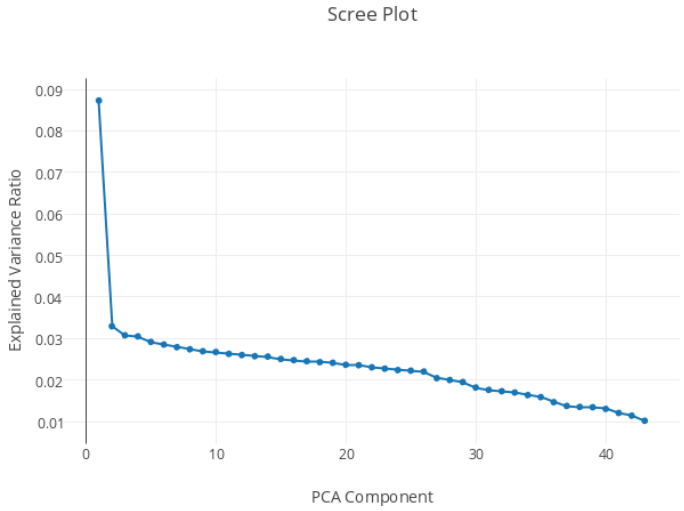
How many (*k*) PCs to keep?

- Pick based on percentage of variance captured / lost
  - Variance captured: the variance of the projected data
  - Pick smallest *k* that explains some % of variance

$$(\text{Sum of first } k \text{ EVs})/(\text{sum of all EVs})$$

- Look for an "elbow" in Scree plot (plot of explained variance or eigenvalues)





Week 7 Contents / Objectives

Part A

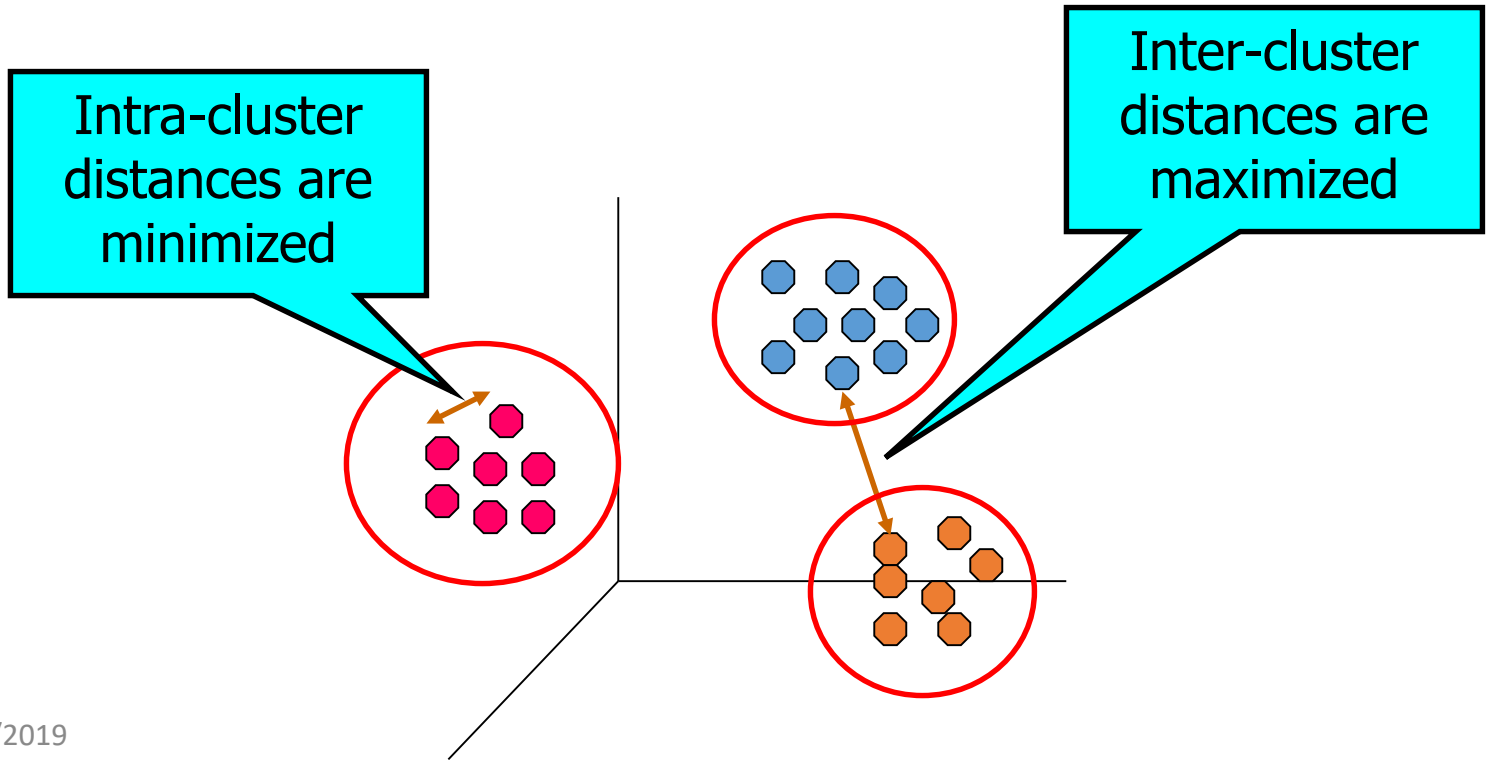
- Dimensionality Reduction
- Principal Component Analysis
- ***k*-means Clustering**

Part B

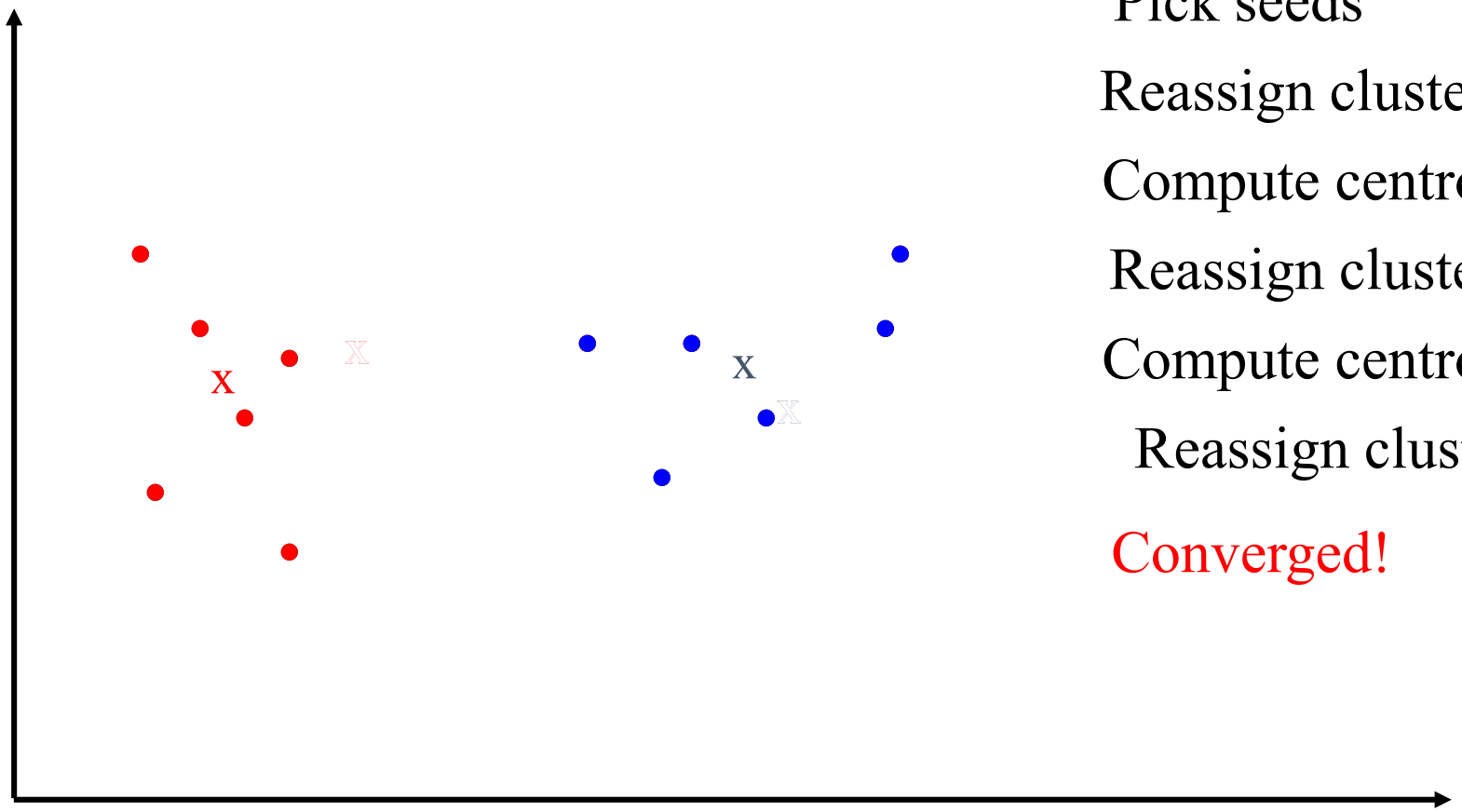
- Autoencoder
- Convolutional Neural Networks

What is Clustering?

- Finding groups of objects such that the objects in a group will be similar (or related) to one another and different from (or unrelated to) the objects in other groups



k-means Example (*k*=2)



Pick seeds

Reassign clusters

Compute centroids

Reassign clusters

Compute centroids

Reassign clusters

Converged!

Week 7 Contents / Objectives

Part A

- Dimensionality Reduction
- Principal Component Analysis
- *k*-means Clustering

Part B

- **Autoencoder**
- Convolutional Neural Networks

Conda Virtual Environment Setup

Open a command line terminal.

Create a new conda environment with Python 3.6

```
conda create -n mlai19 python=3.6 anaconda
```

Activate the conda environment `mlai19`

`activate mlai19` (Windows)

`source activate mlai19` (Mac/Linux)

You will see `(mlai19)` on the left indciating your environment

Install Pytorch and Torchvision (non-CUDA/GPU version for simplicity)

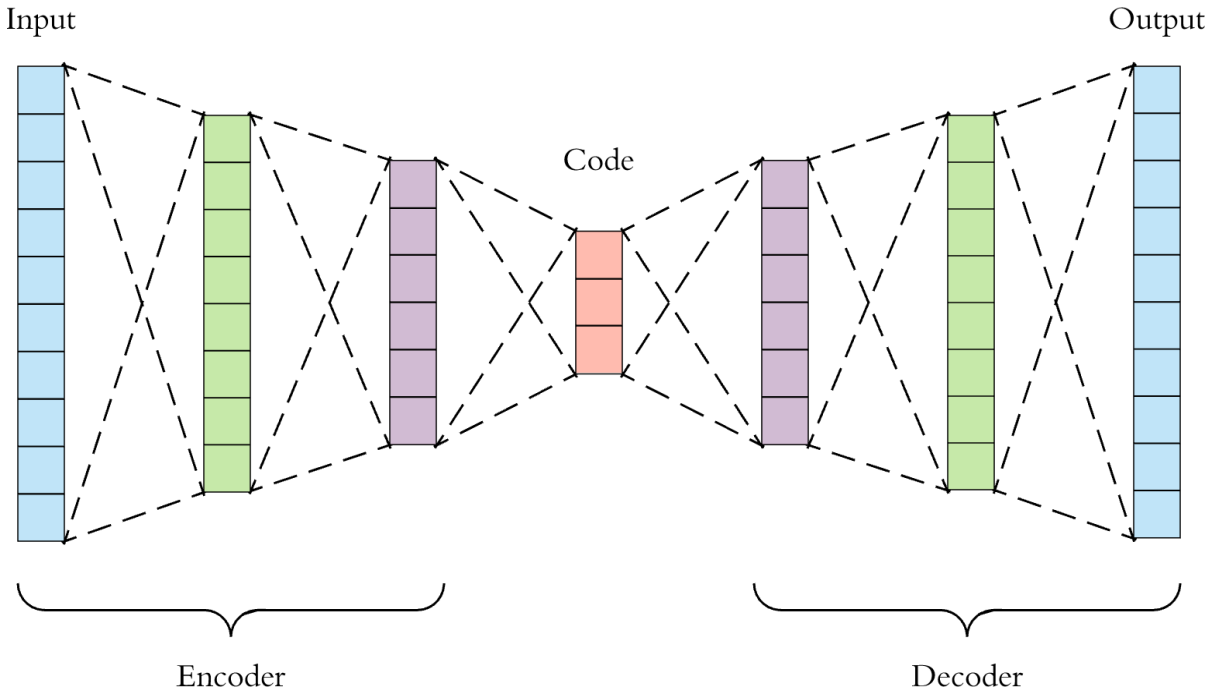
```
conda install pytorch torchvision cpuonly -c pytorch
```

If you have GPU, install the GPU version with command at here

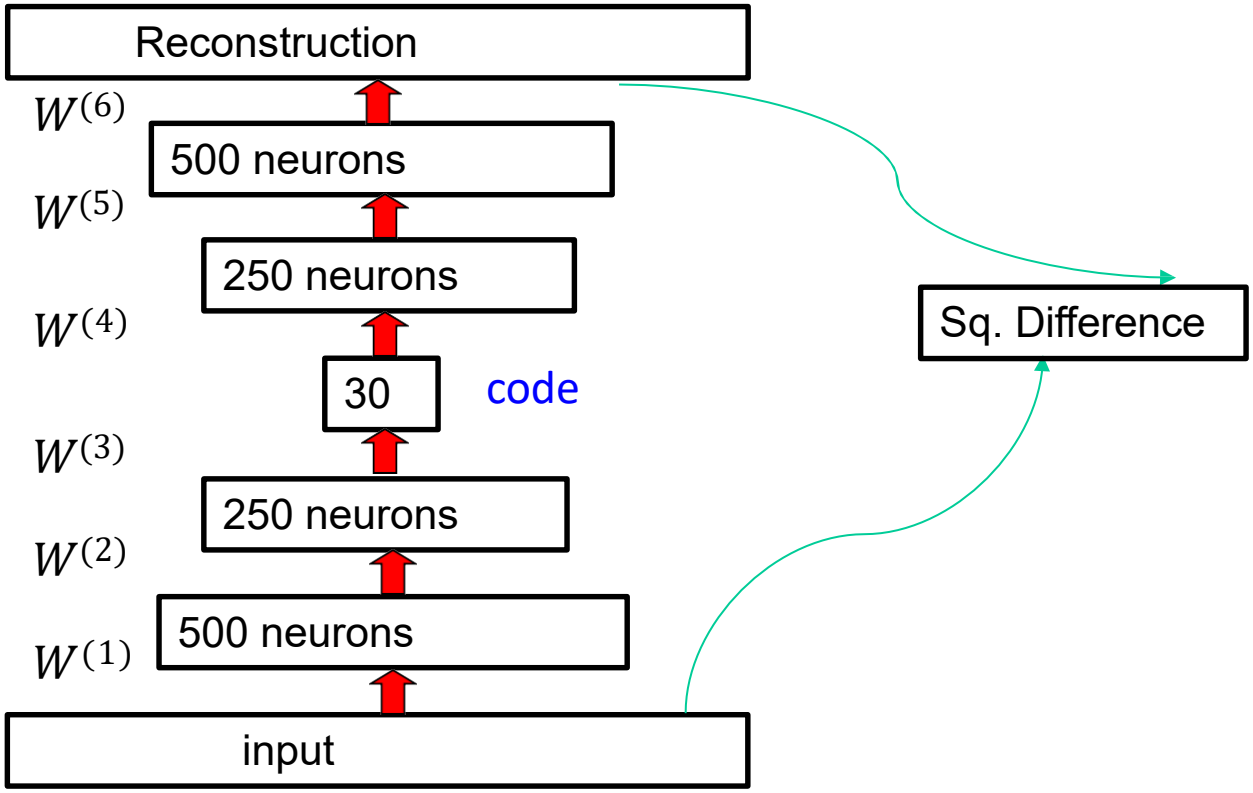
Start Jupyter notebook server: `jupyter notebook`

Autoencoders

- **Encoder:** compress data or extract features
- **Decoder:** generate images given a new code
- **Bottleneck** (**code**): to make it non-trivial, much smaller dimension as the latent representation



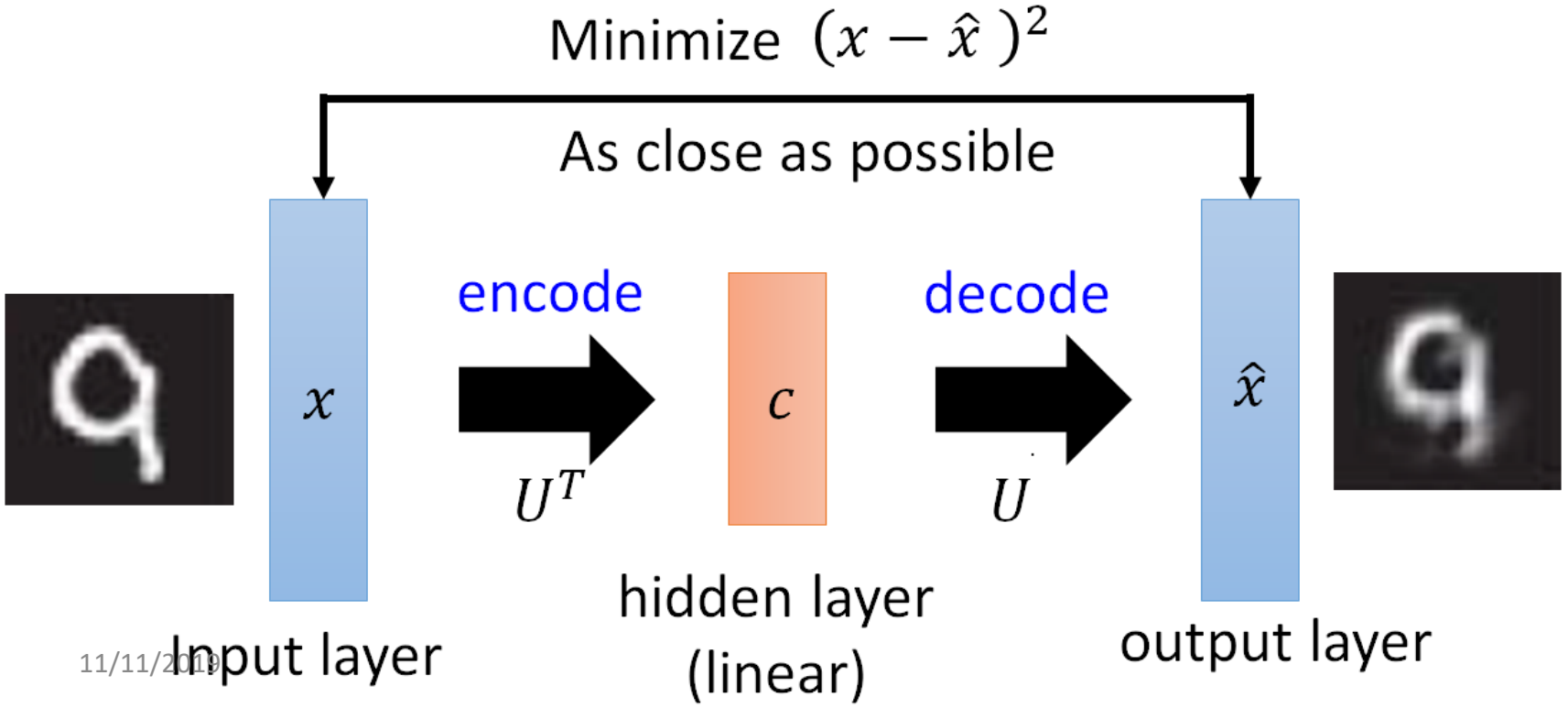
Autoencoders



- Find the weights that produce as small a difference as possible between the input and the reconstruction
- Train using Backprop
- The code layer is a low--dimensional summary of the input

PCA as Linear Autoencoder

- PCA: autoencoder w/t single-layer encoder, single-layer-decoder
- Weight sharing between the encoder and decoder



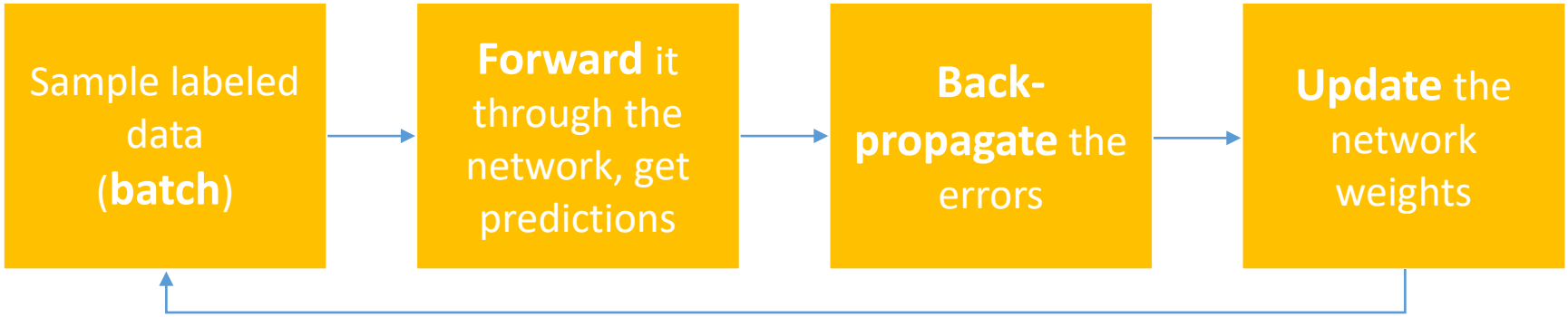
Key Modules in PyTorch

- **torch.autograd**
  - Automatic differentiation. A recorder records what operations have performed, and then it replays it backward to compute the gradients.
- **torch.optim**
  - Implementation of various optimization algorithms used for building neural networks (and other ML algorithms).
- **torch.nn**
  - High-level definition of the **computational graphs** of complex neural networks (and other ML algorithms)

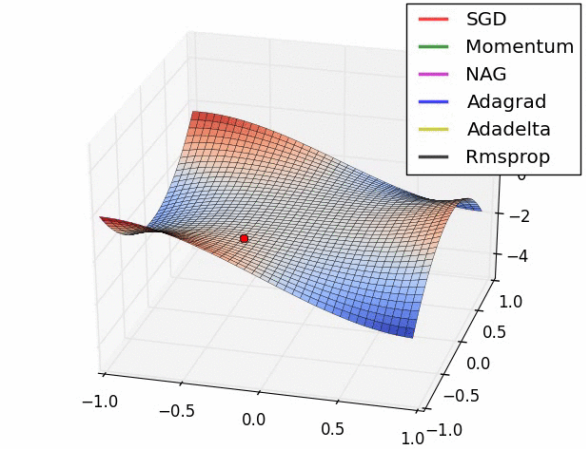
Convolutional Autoencoder

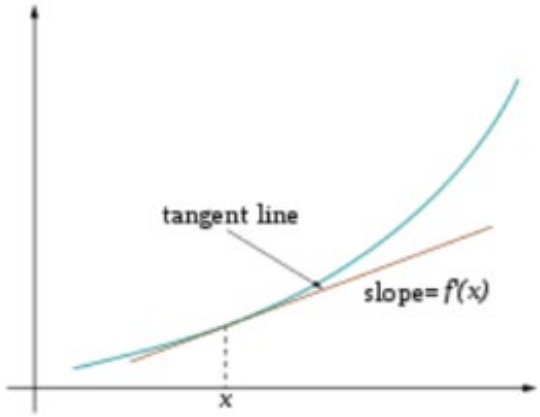
```
class Autoencoder(nn.Module):
    def __init__(self):
        super(Autoencoder, self).__init__()
        self.encoder = nn.Sequential(
            # 1 input image channel, 16 output channel, 3x3 square convolution
            nn.Conv2d(1, 16, 3, stride=2, padding=1),
            nn.ReLU(),
            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
```

Training



Optimize (min. or max.) **objective/cost function $J(\theta)$**
Generate **error signal** that measures difference between predictions and target values





Use error signal to change the **weights** and get more accurate predictions
Subtracting a fraction of the **gradient** moves you towards the **(local) minimum of the cost function**

https://medium.com/@ramrajchandradevan/the-evolution-of-gradient-descend-optimization-algorithm-4106a6702d39

Autoencoder Training

```
#Hyperparameters for training
batch_size=64
learning_rate=1e-3
max_epochs = 20

#Set the random seed for reproducibility
torch.manual_seed(509)
#Choose mean square error loss
criterion = nn.MSELoss()
#Choose the Adam optimiser
optimizer = torch.optim.Adam(myAE.parameters(), lr=learning_rate, weight_decay=1e-5)
#Specify how the data will be loaded in batches (with random shffling)
train_loader = torch.utils.data.DataLoader(mnist_data, batch_size=batch_size, shuffle=True)
#Storage
outputs = []

#Start training
for epoch in range(max_epochs):
    for data in train_loader:
        img, label = data
        optimizer.zero_grad()
        recon = myAE(img)
        loss = criterion(recon, img)
        loss.backward()
        optimizer.step()
    if (epoch % 3) == 0:
        print('Epoch:{}, Loss:{:.4f}'.format(epoch+1, float(loss)))
    outputs.append((epoch, img, recon),)
```

Week 7 Contents / Objectives

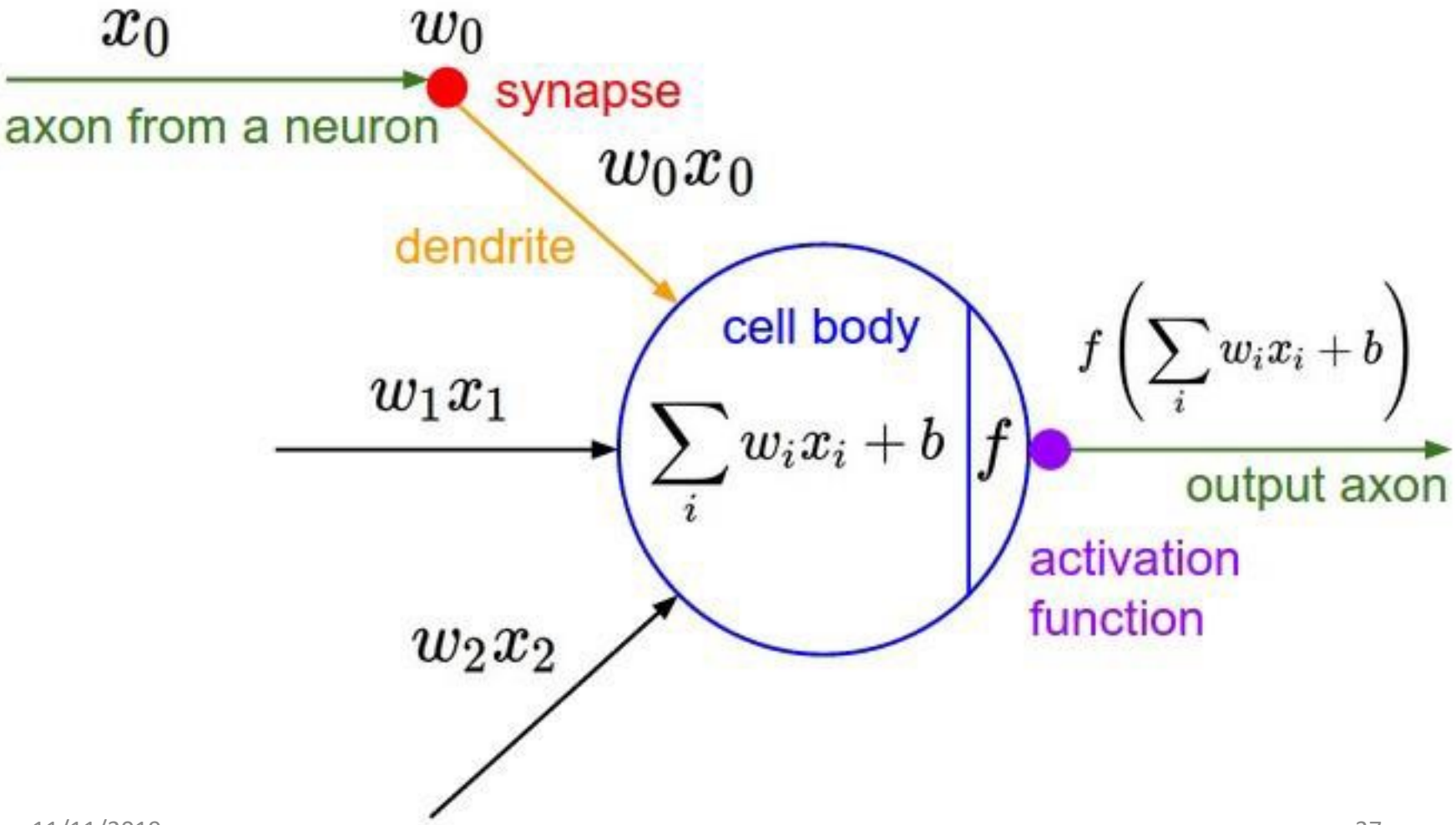
Part A

- Dimensionality Reduction
- Principal Component Analysis
- *k*-means Clustering

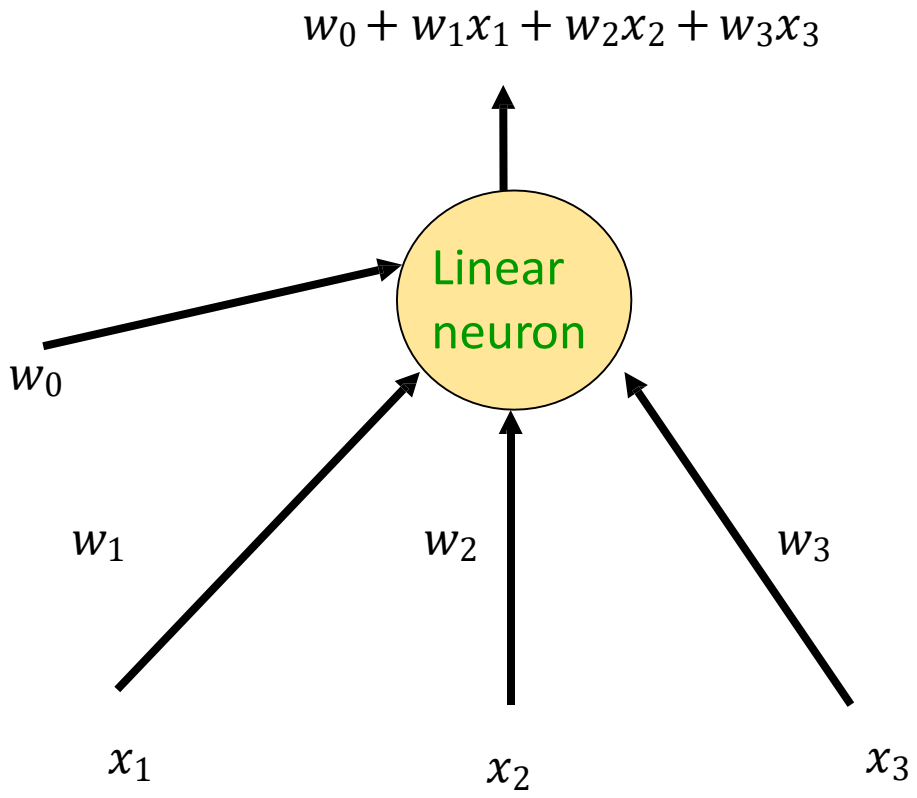
Part B

- Autoencoder
- **Convolutional Neural Networks**

A Neuron Analogous to the Brain



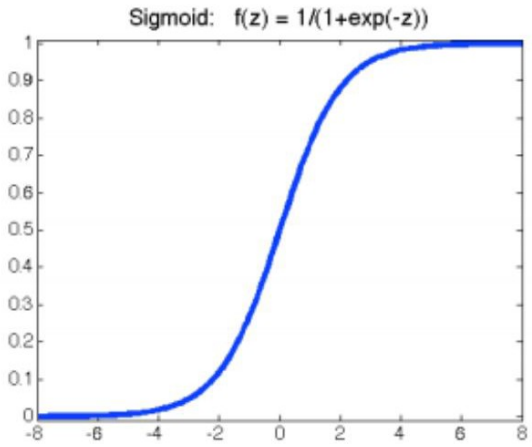
Linear Neuron

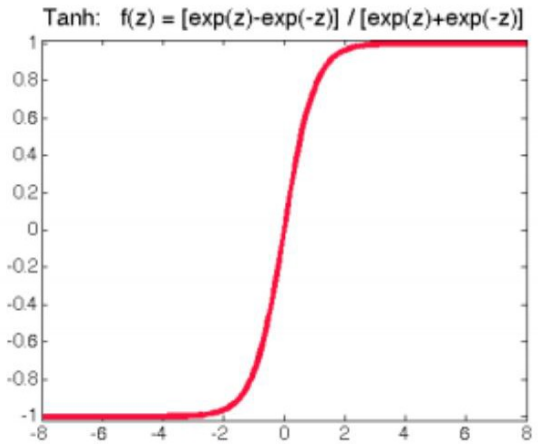


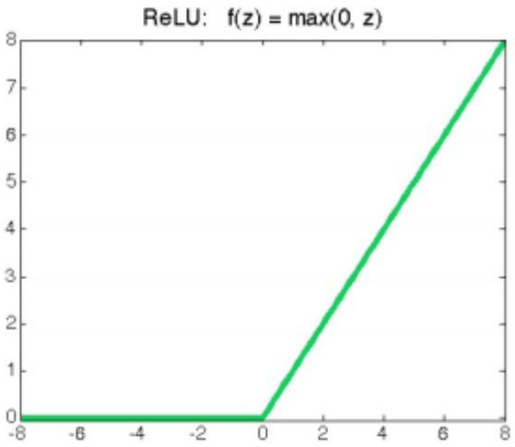
Activation functions

Most commonly used activation functions:

- Sigmoid: $\sigma(z) = \frac{1}{1+\exp(-z)}$
- Tanh: $\tanh(z) = \frac{\exp(z)-\exp(-z)}{\exp(z)+\exp(-z)}$
- ReLU (Rectified Linear Unit): $\mathrm{ReLU}(z) = \max(0, z)$





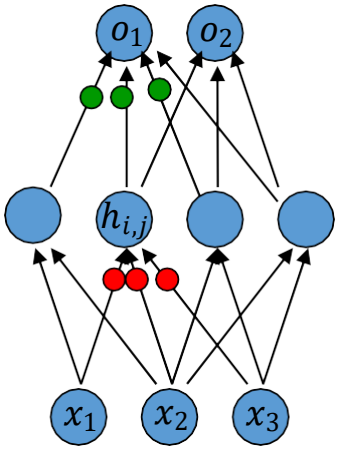


Computation in Neural Networks

- Forward pass
  - Making predictions
  - Plug in the input x, get the output y

$$\mathbf{o} = g\left(\left(W^{(2)}\right)^T \boldsymbol{h} + b^{(2)}\right)$$

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



- Backward pass (backpropagation)
  - Compute the gradient of the cost function with respect to the weights to find a good values for weights

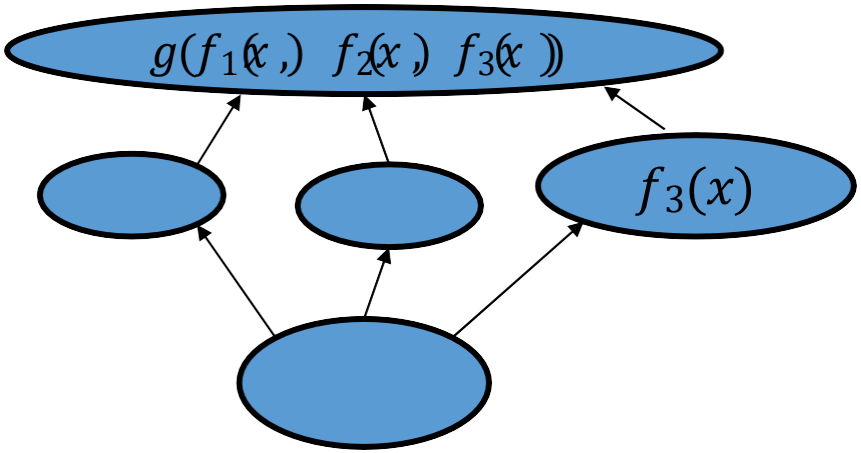
Autograd: Chain Rule

- Univariate Chain Rule

$$\frac{d}{dt}g\big(f(t)\big) = \frac{dg}{df}\cdot\frac{df}{dt}$$

- Multivariate Chain Rule

$$\frac{\partial g}{\partial x} = \Sigma\,\frac{\partial g}{\partial f_i}\frac{\partial f_i}{\partial x}$$



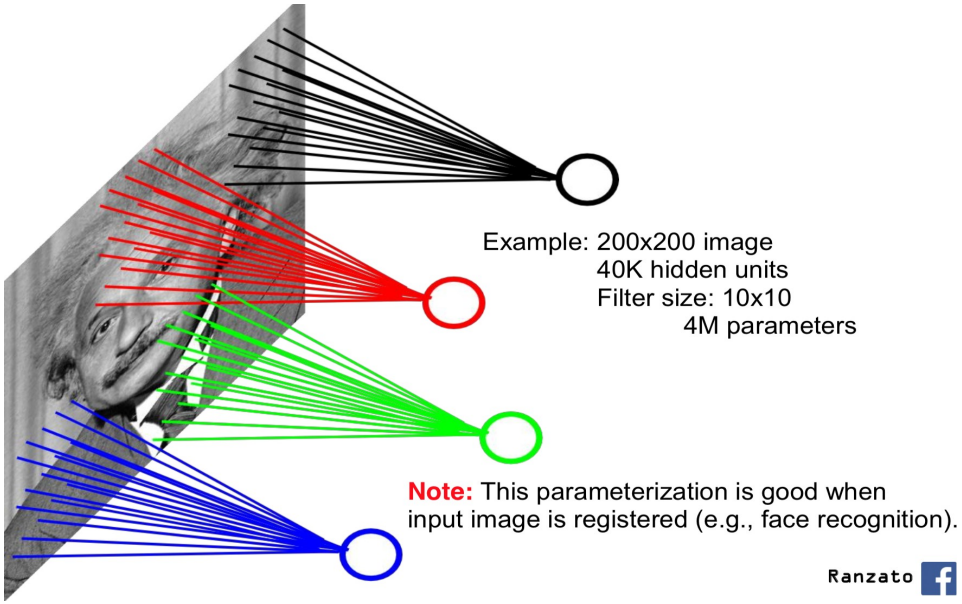
Fully Connected (FC) Layer

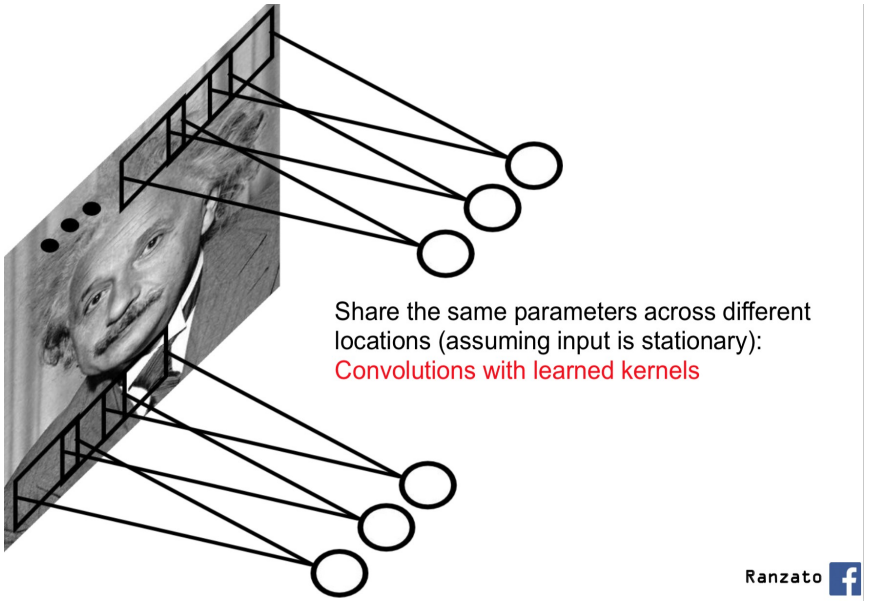
- Linear layers such as linear regression
- What if our network is bigger?
  - Input image: 200 × 200 pixels, first hidden layer: 500 units
  - **Question**: How many weights are there?

    20 million
  - **Q**: Why would using an FC layer be problematic?
    - computing predictions (forward pass) will take a long time
    - a large number of weights requires a lot of training data to avoid overfitting
    - small shift in image can result in large change in prediction
    - does not make use of the geometry of the image

Convolutional Neural Network

- Key ideas:
  - Locally-connected layers: look for local features in small regions of the image
  - Weight-sharing: detect the same local features across the entire image

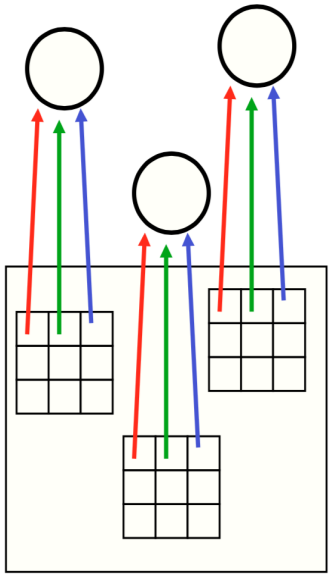




Weight Sharing

- Each neuron on the higher layer detects the same feature, but in different locations on the lower layer

The red connections all have the same weight.

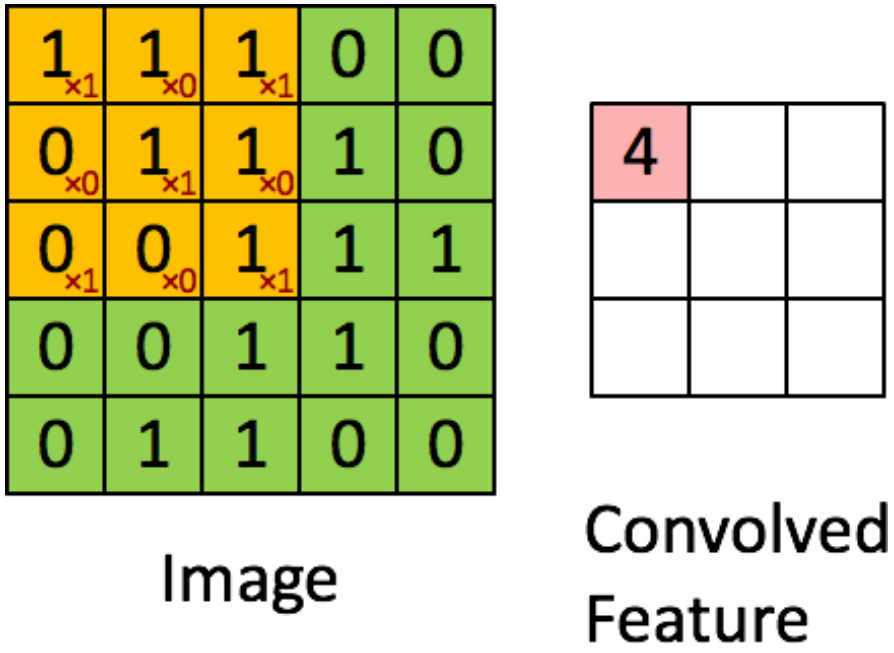


"Detecting" = the output (activation) is high if the feature is present

"Feature" = something in the image, like an edge, blob or shape

Forward Pass Example (Greyscale)

https://cdn-images-1.medium.com/max/1200/1*GcI7G-JLAQiEoCO



- The *kernel* or *filter* (red) contains the trainable weights. In this picture, the *kernel size* is 3 ×3
- The "*convolved features*" is another term for "convolution output"

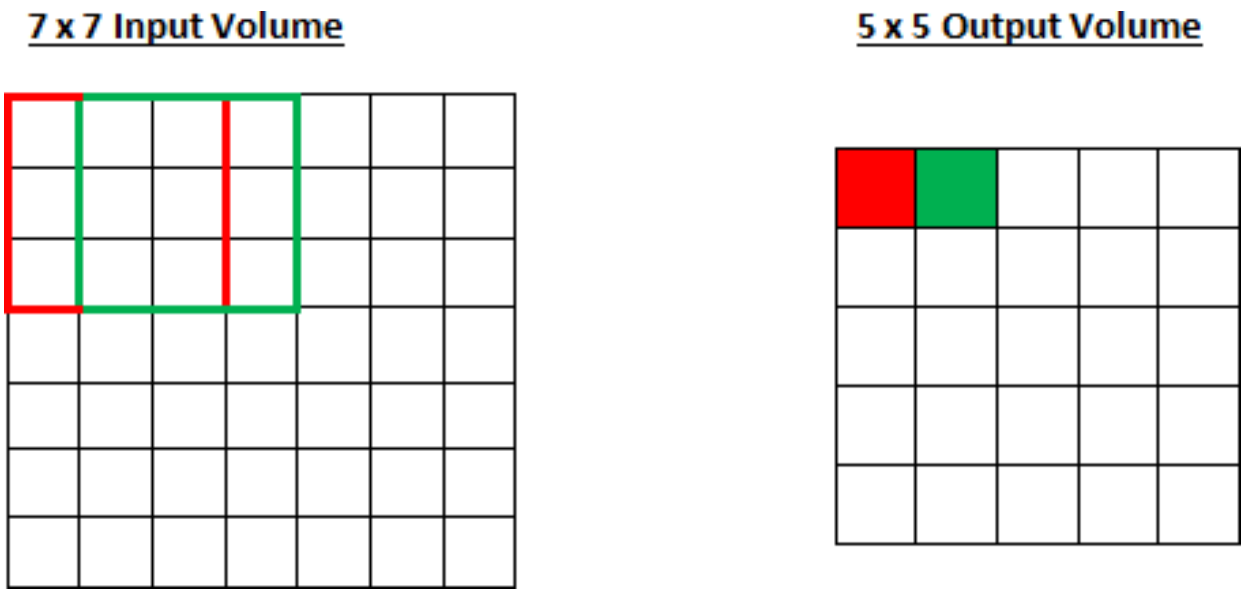
Example of convolution

Greyscale input image: 7 ×7

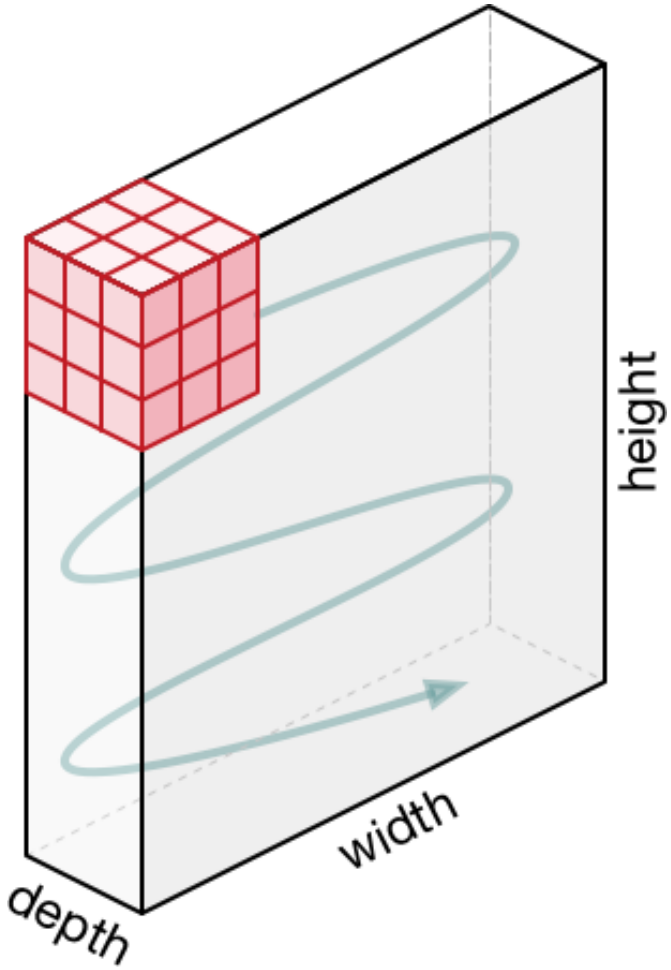
Convolution **kernel**: 3 × 3

Questions:

- How many units are in the output?
- How many trainable weights are there?



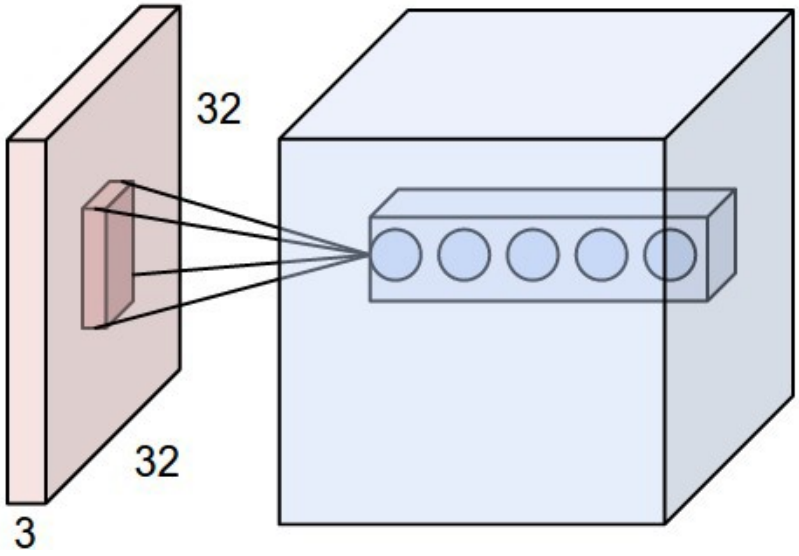
Convolution in RGB for colour images



The kernel: a 3-dimensional tensor! In this example, the kernel has size 3 ×3 × 3. The number 3: the number of **input channels** or **input feature maps**

Detecting Multiple Features

- **Q**: What if we want to detect many features of the input? (i.e. **both** horizontal edges and vertical edges, and maybe even other features?)
- **A**: Have many convolutional filters!

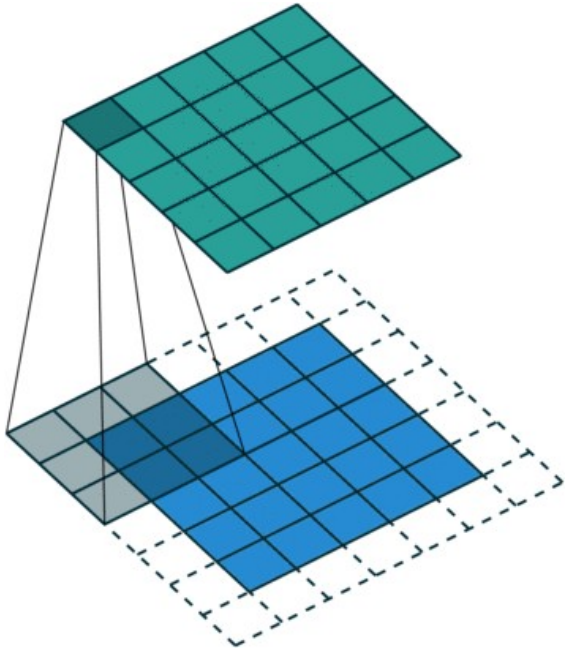


Input image of size 3 × 32 × 32

Convolution kernel of **3** ×3 × 3× **5**

- The number **3** is the number of **input channels** or **input feature maps**
- The number **5** is the number of **output channels** or **output feature maps**

Zero Padding

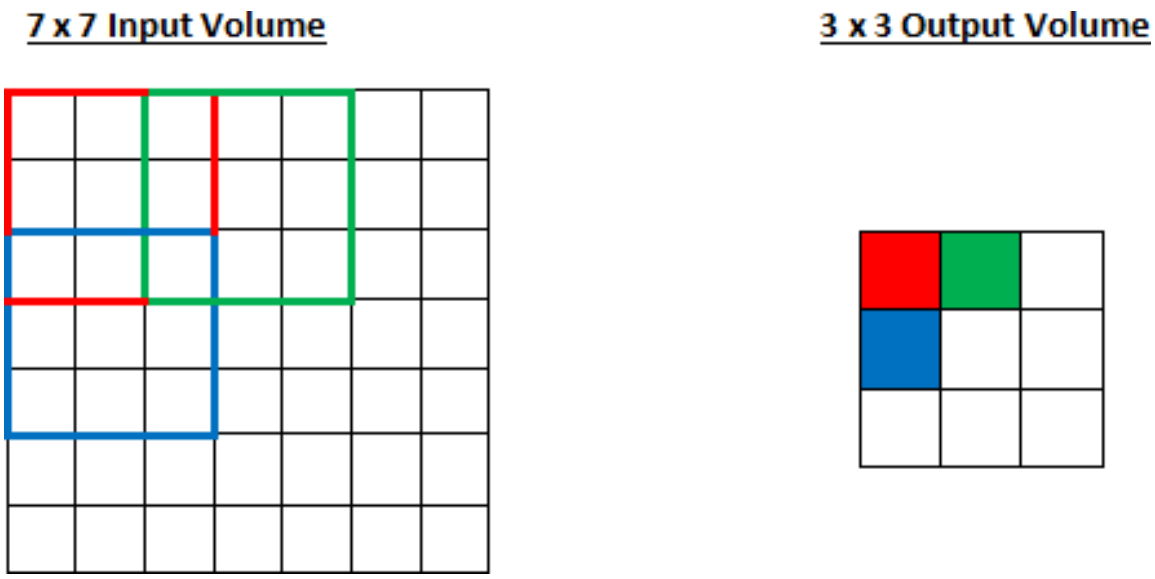


- Add zeros around the border of the image
- (Can add more than one pixel of zeros)

Q: Why might we want to add zero padding?

- Keep the next layer's width and height consistent with the previous
- Keep the information around the border of the image

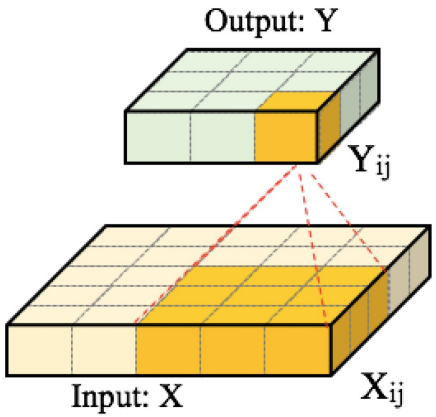
Strided Convolution



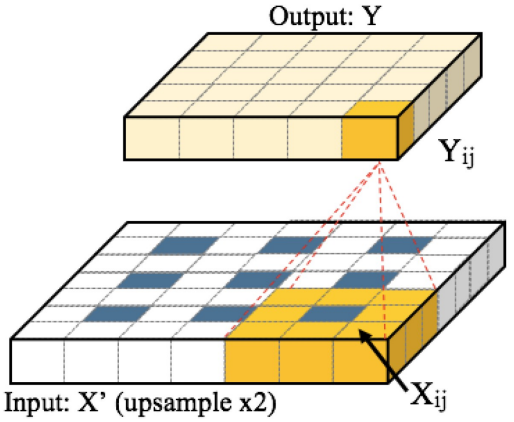
Shift the kernel by **2** (stride=2) when computing the next output feature.

Objective: to consolidate (summarise) information

Transpose Convolution Layer



(**a**) Convolutional layer: the input size is $W_1 = H_1 = 5$; the receptive field $F = 3$; the convolution is performed with stride $S = 1$ and no padding ($P = 0$). The output Yis of size $W_2 = H_2 = 3$.



(**b**) Transposed convolutional layer: input size $W_1 = H_1 = 3$; transposed convolution with stride $S = 2$; padding with $P = 1$; and a receptive field of $F = 3$. The output Yis of size $W_2 = H_2 = 5$.

Figure 2: https://www.mdpi.com/2072-4292/9/6/522/htm
More at https://github.com/vdumoulin/conv_arithmetic

Convolutional Autoencoder

```
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
```

Acknowledgement

- Part A used materials from: Lisa Zhang, Michael Guerzhoy, Neil Lawrence, Derek Hoiem, Pandu Nayak and Prabhakar Raghavan

- Part B used materials from: Lisa Zhang, Michael Guerzhoy, Fei-Fei Li & Justin Johnson & Serena Yeung

Recommended Reading

- **Dimension Reduction: A Guided Tour**
- Visualising high-dimensional datasets using PCA and t-SNE in Python
- CS231n: Convolutional Neural Networks for Visual Recognition from Stanford
- The lab notebook and references there