CS 410/510: Neural Networks intro

All good content from Brandon Rohrer, 3Blue1Brown
Anything not good, I added
Take

- Melanie Mitchell's CS 445/545: Machine Learning
This is your machine learning system?

Yup! You pour the data into this big pile of linear algebra, then collect the answers on the other side.

What if the answers are wrong?

Just stir the pile until they start looking right.

https://xkcd.com/1838/
Problem

• How can a computer resolve these three images as the number 3?

• What program would you write?
How do human brains solve the problem?

- Via a neural network
Once trained...

- Stimuli fire neurons
- GaTech Neurolab
Biological process

• Cells in eye light up optical neurons

• Create a chain of activations throughout brain (a massive biological neural network)

• Eventually activate regions of the brain used for object identification
  • Occipital lobe - color, orientation, and motion information
  • Perirhinal cortex - object recognition and memories
  • Temporal lobe – perception, face/object recognition, memory
  • etc.
Biological process

- Activations reinforced via continuous feedback to change structure of network (neuroplasticity)

- Feedback optimizes network to ingrain learning, habituate behavior (to conserve energy)
Simple example

- Recognizing handwritten characters
  - MNIST dataset of 60k labeled handwritten digit images, 10k testing images
  - 28x28 pixel grid of 784 pixels
- Goal: Emulate biological process for digit recognition
Simplest neural network

- Multi-layer perceptron
- Neurons with activation levels between 0 and 1
- Activations in one layer determine activations in next based on weights
- Weights trained via labeled input
- Neurons perceiving pixels on left activate appropriate output neuron on right
Animation

• 3Blue1Brown
What do the intermediate layers do?

- Also called "hidden" layers
- Consider parts of each digit
  - As one goes from left to right, higher-order features captured
- Example
  - Shared parts of digits
• Higher-level features towards end
Details

- Network structure
- Learning mechanism
- Layer functions
Network structure

- Designed based on application and device running model
- Depth of neural network
  - Complexity of pattern matching grows as the number of layers goes up.
  - Some 12 or more (but, training takes longer)
Network structure

- Neurons per layer
  - Each node represents a pattern (a combination of the neurons on the previous layer)
  - Number of neurons determines patterns being detected in layer
• Depth and number of neurons determine complexity and number of patterns
Examples: Letters to phrases
Examples: Audio to sentences

- sentence
- phrase
- word
- syllable
- phoneme
Examples: Face and car recognition

- Convolutional Deep Belief Networks for Scalable Unsupervised Learning of Hierarchical Representations, Honglak Lee, Roger Grosse, Rajesh Ranganath, Andrew Y. Ng
Example: Object recognition

- Understanding Neural Networks Through Deep Visualization.
  Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs, Hod Lipson
Details

• Network structure
• Learning
• Layer functions
Learning (updating layers)

• Randomly assign function weights
• Send a labeled example into neural network
• Examine activation of output neurons
• Update internal weights to make activation more accurate
  • Maximize activation of output neuron for label
  • Minimize activation of all other neurons
Example

- Food cart operator schedule
- Four input neurons
  - Open in am
  - Closed in am
  - Open in pm
  - Closed in pm
- Two output neurons (shown)
  - Schedule A and B
- Train on labeled observations
Initialize

• Start with random weights
Gather data

- Observation
  - Open in am, closed in pm
  - Labeled with B
Activate input neurons

• Check output neurons as an average of activations on input

\[
\frac{.3 + .1}{2} = .2
\]
Activate input neurons

\[ \frac{.8 + .4}{2} = .6 \]
Calculate errors

- $E =$ right answer – actual answer
  - Want B to be close to 1
  - Want all other neurons close to 0
- If fit is poor, adjust weights
  - How?

B am pm

Diagram:
- B
- am
- pm
- A: .3 .8 .9 .2
- B: .9 .6 .1 .4
- Error = 1 -.6 = .4
Backpropagation via Gradient Descent

- For each weight, nudge value up and down and see how the error changes.
  - Boost edges from open am and closed pm to B
  - Decrease edges from closed am and open pm to B

Error=.4
Gradient descent

- General problem solved via mathematical algorithm
  - For each weight, adjust up and down a bit and see how the error changes
  - Move towards lower error
- Multi-variable calculus on a gazillion variables (each link in network)
Example

• Adjustment based on size of gradient, error
  • e.g. adjustment = error * gradient * delta
  • Assume nudges of 0.1
Example
Example

A \quad B

\frac{.9 + .5}{2} = .7

B

am

pm

.3

.9

.9

.8

.6

.1

.1

.5

am

pm
Example

- Error lower

\[
\text{error} = 1 - 0.7 = 0.3
\]
Iterate

• Repeat this process with other examples until weights stop changing.
Iterate
Iterate
Iterate

A
am
pm

A
B

.2 .9 1 .8 .7 .1 0 .5

am am pm pm
Iterate

A am pm

A B

am pm

.2 .9↑1 .8↑7 .1↓0 .5
Iterate
Iterate

B am pm

A am 0.8
B am 0.9
A pm
B pm 0.5
Iterate

B am pm
A .1 .9 1 .8 0 .5
B am pm
Iterate

B am pm

A am

B pm

Weights: 0.1, 1.0, 1.7, 0.8, 0.0, 0.6
Iterate
Iterate

B
am
pm

A
am
.1
1↑
1
.7↓
.8
0↓
0
.6↑

B
pm
Iterate

B

am

pm

A

B

am

pm

0.1 1 1 0.6 0 0 0.7


Iterate
Iterate

• Eventually...

• But, what happens if food cart owner takes a holiday break?

• Did it learn or did it memorize?
Learning visualization

- Green = increase weights
- Orange = decrease weights
Learning visualization
Details

- Network structure
- Learning
- Layer functions
Up until now...

- Layers with perceptrons from biological model
- In practice, other layers often employed
  - Convolutional layers
  - Pooling layers
  - Normalization layers
  - Fully-connected layers
Convolutional layers

- Basis for Convolutional Neural Networks (CNNs)
  - Applications with spatial structure and locality (e.g. images, audio)
  - Many current ML APIs derived from CNNs
Motivating example

- Determine whether a picture is of an X or an O

A two-dimensional array of pixels

CNN

X or O
Cases to handle

translation  scaling  rotation  weight

CNN  X

CNN  O
Deciding is hard
What computers see

- Pixel values
What computers see

• Do a diff?
Computers are literal
Idea: Match pieces of the image
Match parts of image via "features"

- 3 features of interest for X
Filtering: Mathematical way of matching

1. Line up the feature and the image patch.
2. Multiply each image pixel by the corresponding feature pixel.
3. Add them up.
4. Divide by the total number of pixels in the feature.
Filtering:

\[
\begin{bmatrix}
1 & -1 & -1 \\
-1 & 1 & -1 \\
-1 & -1 & 1 \\
\end{bmatrix}
\]

\[
1 \times 1 = 1
\]
Filtering:

\[
\begin{array}{cccccccc}
-1 & 1 & -1 & -1 & -1 & -1 & -1 & -1 \\
-1 & 1 & -1 & -1 & -1 & -1 & -1 & 1 \\
-1 & 1 & 1 & -1 & -1 & 1 & -1 & -1 \\
-1 & -1 & -1 & -1 & 1 & -1 & -1 & -1 \\
-1 & -1 & 1 & -1 & -1 & -1 & -1 & -1 \\
-1 & 1 & -1 & -1 & 1 & -1 & -1 & -1 \\
-1 & -1 & -1 & -1 & -1 & 1 & -1 & -1 \\
-1 & 1 & -1 & -1 & -1 & -1 & 1 & -1 \\
-1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\
\end{array}
\]

\[
1 \times 1 = 1
\]
Filtering:

\[
\begin{array}{ccc}
-1 & 1 & -1 \\
-1 & 1 & -1 \\
-1 & -1 & 1 \\
\end{array}
\]

\[-1 \times -1 = 1\]

\[
\begin{array}{ccccccccc}
-1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\
-1 & -1 & -1 & -1 & -1 & -1 & -1 & 1 & -1 \\
-1 & -1 & -1 & -1 & -1 & 1 & -1 & -1 & -1 \\
-1 & -1 & -1 & -1 & 1 & -1 & -1 & -1 & -1 \\
-1 & -1 & -1 & 1 & -1 & 1 & -1 & -1 & -1 \\
-1 & -1 & 1 & -1 & -1 & -1 & 1 & -1 & -1 \\
-1 & 1 & -1 & -1 & -1 & -1 & -1 & 1 & -1 \\
-1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\
\end{array}
\]
Filtering:

\[
\begin{bmatrix}
1 & -1 & -1 \\
-1 & 1 & -1 \\
-1 & -1 & 1 \\
\end{bmatrix}
\]

\[-1 \times -1 = 1\]

\[
\begin{bmatrix}
1 & 1 & 1 \\
\end{bmatrix}
\]
Filtering:

\[
\begin{array}{ccc}
1 & -1 & -1 \\
-1 & 1 & -1 \\
-1 & -1 & 1 \\
\end{array}
\]

\[
-1 \times -1 = 1
\]
Filtering:

\[ \begin{array}{ccc}
1 & -1 & -1 \\
-1 & 1 & -1 \\
-1 & -1 & 1 \\
\end{array} \times \begin{array}{ccc}
1 & x & 1 \\
\end{array} = \begin{array}{ccc}
1 & 1 & 1 \\
1 & 1 & \text{Gray} \\
\end{array} \]
Filtering:

\[-1 \times -1 = 1\]
Filtering:

\[-1 \times -1 = 1\]
Filtering:

\[-1 \times -1 = 1\]
Filtering:

\[
\begin{bmatrix}
1 & -1 & -1 \\
-1 & 1 & -1 \\
-1 & -1 & 1
\end{bmatrix}
\times
\begin{bmatrix}
1 & 1 & 1 \\
1 & 1 & 1 \\
1 & 1 & 1
\end{bmatrix}
= 1
\]
Filtering:

\[
\begin{array}{ccc}
1 & 1 & 1 \\
1 & 1 & 1 \\
1 & 1 & 1 \\
\end{array}
\]

\[
\frac{1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1}{9} = 1
\]
Filtering:

\[
\begin{array}{cccccccc}
-1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\
-1 & 1 & -1 & -1 & 1 & -1 & 1 & -1 \\
-1 & 1 & -1 & 1 & 1 & 1 & -1 & -1 \\
-1 & -1 & 1 & 1 & -1 & -1 & -1 & -1 \\
-1 & -1 & -1 & 1 & -1 & -1 & -1 & -1 \\
-1 & -1 & -1 & -1 & 1 & -1 & -1 & -1 \\
-1 & -1 & 1 & -1 & 1 & 1 & -1 & -1 \\
-1 & 1 & -1 & -1 & -1 & -1 & 1 & -1 \\
-1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\
\end{array}
\]

\[1 \times 1 = 1\]
Filtering:

\[
\begin{bmatrix}
1 & -1 & -1 \\
-1 & 1 & -1 \\
-1 & -1 & 1
\end{bmatrix}
\times
\begin{bmatrix}
1 & 1 & -1
\end{bmatrix}
= -1
\]
Filtering:
Filtering:

\[
\begin{array}{ccc}
  1 & -1 & -1 \\
-1 & 1 & -1 \\
-1 & -1 & 1
\end{array}
\]

\[
\begin{array}{ccc}
  1 & 1 & -1 \\
1 & 1 & 1 \\
-1 & 1 & 1
\end{array}
\]

\[
\frac{1 + 1 - 1 + 1 + 1 + 1 - 1 + 1 + 1}{9} = .55
\]
Convolution: Try feature at every location
Convolution: Try feature at every location

- Convolution operator
Convolution layer

- One image becomes a stack of filtered images
- Filtered images used within network for learning
- Stack represents feature prevalence throughout image
Convolution layer

• But, now much more data to deal with!
Pool layer

Reduce size of filtered images

1. Pick a window size (usually 2 or 3).
2. Pick a stride (usually 2).
3. Walk your window across your filtered images.
4. From each window, take the maximum value.
Pooling

maximum

1.00
# Pooling

The diagram illustrates the pooling operation, specifically using the maximum function. The input is a 2x2 grid of numbers, and the output is a 1x1 grid where each element is the maximum value from the corresponding 2x2 block of the input.

<table>
<thead>
<tr>
<th>0.77</th>
<th>-0.11</th>
<th>0.11</th>
<th>0.33</th>
<th>0.55</th>
<th>0.11</th>
<th>0.33</th>
</tr>
</thead>
<tbody>
<tr>
<td>-0.11</td>
<td>1.00</td>
<td>-0.11</td>
<td>0.33</td>
<td>-0.11</td>
<td>0.11</td>
<td>-0.11</td>
</tr>
<tr>
<td>0.11</td>
<td>-0.11</td>
<td>1.00</td>
<td>-0.33</td>
<td>0.11</td>
<td>-0.11</td>
<td>0.55</td>
</tr>
<tr>
<td>0.33</td>
<td>0.33</td>
<td>-0.33</td>
<td>0.55</td>
<td>-0.33</td>
<td>0.33</td>
<td>0.33</td>
</tr>
<tr>
<td>0.55</td>
<td>-0.11</td>
<td>0.11</td>
<td>-0.33</td>
<td>1.00</td>
<td>-0.11</td>
<td>0.11</td>
</tr>
<tr>
<td>-0.11</td>
<td>0.11</td>
<td>-0.11</td>
<td>0.33</td>
<td>-0.11</td>
<td>1.00</td>
<td>-0.11</td>
</tr>
<tr>
<td>0.33</td>
<td>-0.11</td>
<td>0.55</td>
<td>0.33</td>
<td>0.11</td>
<td>-0.11</td>
<td>0.77</td>
</tr>
</tbody>
</table>
## Pooling

The diagram illustrates the process of pooling in a neural network. Pooling is a technique used in convolutional neural networks to downsample the spatial dimensions of the input feature maps. The example shows a 2x2 pooling operation with a stride of 2.

The input feature map is a 3x3 grid with values ranging from -0.11 to 0.77. The pooling operation is performed by taking the maximum value in each 2x2 sub-region of the input grid.

In the example, the maximum values are highlighted in white. The resulting output feature map after pooling is shown on the right side of the diagram.
Pooling

```
0.77 -0.11 0.11 0.33 0.55 -0.11 0.33
-0.11 1.00 -0.11 0.33 -0.11 0.11 -0.11
0.11 -0.11 1.00 -0.33 0.11 -0.11 0.55
0.33 0.33 -0.33 0.55 -0.33 0.33 0.33
0.55 -0.11 0.11 -0.33 1.00 -0.11 0.11
-0.11 0.11 -0.11 0.33 -0.11 1.00 -0.11
0.33 -0.11 0.55 0.33 0.11 -0.11 0.77
1.00 0.33 0.55 0.33
```

Maximum
### Pooling

The image illustrates the concept of pooling in deep learning, specifically using the maximum pooling operation. The input tensor is shown on the left, and the output tensor on the right. The maximum pooling operation involves selecting the maximum value from a small neighborhood of the input tensor, which is highlighted in the diagram. The input tensor is a 3x3 grid, and the output tensor is a 2x2 grid, indicating a 2x2 pooling stride. The values in the input tensor are varied, and the maximum values are highlighted in the output tensor.
Pooling

max pooling

0.77  -0.11  0.11  0.33  0.55  -0.11  0.33
-0.11  1.00  -0.11  0.33  -0.11  0.11  -0.11
0.11  -0.11  1.00  -0.33  0.11  -0.11  0.55
0.33  0.33  -0.33  0.55  -0.33  0.33  0.33
0.55  -0.11  0.11  -0.33  1.00  -0.11  0.11
-0.11  0.11  -0.11  0.33  -0.11  1.00  -0.11
0.33  -0.11  0.55  0.33  0.11  -0.11  0.77
**Pooling layer**

- A stack of images becomes a stack of smaller images.
Normalization layer (ReLU)

- No "negative" activations in biological neural networks
- Change everything negative to zero via
  - sigmoid function (old)
  - ReLU function
- Keeps the math from breaking
Rectified Linear Units (ReLUs)

<table>
<thead>
<tr>
<th></th>
<th>0.77</th>
<th>-0.11</th>
<th>0.11</th>
<th>0.33</th>
<th>0.55</th>
<th>-0.11</th>
<th>0.33</th>
</tr>
</thead>
<tbody>
<tr>
<td>-0.11</td>
<td>1.00</td>
<td>-0.11</td>
<td>0.33</td>
<td>-0.11</td>
<td>0.11</td>
<td>-0.11</td>
<td></td>
</tr>
<tr>
<td>0.11</td>
<td>-0.11</td>
<td>1.00</td>
<td>-0.33</td>
<td>0.11</td>
<td>-0.11</td>
<td>0.55</td>
<td></td>
</tr>
<tr>
<td>0.33</td>
<td>0.33</td>
<td>-0.33</td>
<td>0.55</td>
<td>-0.33</td>
<td>0.33</td>
<td>0.33</td>
<td></td>
</tr>
<tr>
<td>0.55</td>
<td>-0.11</td>
<td>0.11</td>
<td>-0.33</td>
<td>1.00</td>
<td>-0.11</td>
<td>0.11</td>
<td></td>
</tr>
<tr>
<td>-0.11</td>
<td>0.11</td>
<td>-0.11</td>
<td>0.33</td>
<td>-0.11</td>
<td>1.00</td>
<td>-0.11</td>
<td></td>
</tr>
<tr>
<td>0.33</td>
<td>-0.11</td>
<td>0.55</td>
<td>0.33</td>
<td>0.11</td>
<td>-0.11</td>
<td>0.77</td>
<td></td>
</tr>
<tr>
<td>0.77</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Rectified Linear Units (ReLUs)

![ReLU activation function](image)
## Rectified Linear Units (ReLUs)

$$
\begin{array}{cccccc}
0.77 & -0.11 & 0.11 & 0.33 & 0.55 & -0.11 & 0.33 \\
-0.11 & 1.00 & -0.11 & 0.33 & -0.11 & 0.11 & -0.11 \\
0.11 & -0.11 & 1.00 & -0.33 & 0.11 & -0.11 & 0.55 \\
0.33 & 0.33 & -0.33 & 0.55 & -0.33 & 0.33 & 0.33 \\
0.55 & -0.11 & 0.11 & -0.33 & 1.00 & -0.11 & 0.11 \\
-0.11 & 0.11 & -0.11 & 0.33 & -0.11 & 1.00 & -0.11 \\
0.33 & -0.11 & 0.55 & 0.33 & 0.11 & -0.11 & 0.77 \\
\end{array}
$$

$$
\begin{array}{cccccc}
0.77 & 0 & 0.11 & 0.33 & 0.55 & 0 & 0.33 \\
-0.11 & 0.11 & -0.33 & 0.11 & -0.11 & 0.55 & -0.11 \\
0.11 & -0.11 & 1.00 & -0.33 & 0.11 & -0.11 & 0.55 \\
0.33 & 0.33 & -0.33 & 0.55 & -0.33 & 0.33 & 0.33 \\
0.55 & -0.11 & 0.11 & -0.33 & 1.00 & -0.11 & 0.11 \\
-0.11 & 0.11 & -0.11 & 0.33 & -0.11 & 1.00 & -0.11 \\
0.33 & -0.11 & 0.55 & 0.33 & 0.11 & -0.11 & 0.77 \\
\end{array}
$$

Diagram of two neural network layers with ReLU activation function.
## Rectified Linear Units (ReLUs)

<table>
<thead>
<tr>
<th></th>
<th>0.77</th>
<th>-0.11</th>
<th>0.11</th>
<th>0.33</th>
<th>0.55</th>
<th>-0.11</th>
<th>0.33</th>
</tr>
</thead>
<tbody>
<tr>
<td>-0.11</td>
<td>1.00</td>
<td>-0.11</td>
<td>0.33</td>
<td>-0.11</td>
<td>0.11</td>
<td>-0.11</td>
<td></td>
</tr>
<tr>
<td>0.11</td>
<td>-0.11</td>
<td>1.00</td>
<td>-0.33</td>
<td>0.11</td>
<td>-0.11</td>
<td>0.55</td>
<td></td>
</tr>
<tr>
<td>0.33</td>
<td>0.33</td>
<td>-0.33</td>
<td>0.55</td>
<td>-0.33</td>
<td>0.33</td>
<td>0.33</td>
<td></td>
</tr>
<tr>
<td>0.55</td>
<td>-0.11</td>
<td>0.11</td>
<td>-0.33</td>
<td>1.00</td>
<td>-0.11</td>
<td>0.11</td>
<td></td>
</tr>
<tr>
<td>-0.11</td>
<td>0.11</td>
<td>-0.11</td>
<td>0.33</td>
<td>-0.11</td>
<td>1.00</td>
<td>-0.11</td>
<td></td>
</tr>
<tr>
<td>0.33</td>
<td>-0.11</td>
<td>0.55</td>
<td>0.33</td>
<td>0.11</td>
<td>-0.11</td>
<td>0.77</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>0.77</th>
<th>0.11</th>
<th>0.33</th>
<th>0.55</th>
<th>0</th>
<th>0.33</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.00</td>
<td>0</td>
<td>0.33</td>
<td>0</td>
<td>0.11</td>
<td>0</td>
</tr>
<tr>
<td>0.11</td>
<td>0</td>
<td>1.00</td>
<td>0</td>
<td>0.11</td>
<td>0</td>
<td>0.55</td>
</tr>
<tr>
<td>0.33</td>
<td>0.33</td>
<td>0</td>
<td>0.55</td>
<td>0</td>
<td>0.33</td>
<td>0.33</td>
</tr>
<tr>
<td>0.55</td>
<td>0</td>
<td>0.11</td>
<td>0</td>
<td>1.00</td>
<td>0</td>
<td>0.11</td>
</tr>
<tr>
<td>0</td>
<td>0.11</td>
<td>0</td>
<td>0.33</td>
<td>0</td>
<td>1.00</td>
<td>0</td>
</tr>
<tr>
<td>0.33</td>
<td>0</td>
<td>0.55</td>
<td>0.33</td>
<td>0.11</td>
<td>0</td>
<td>0.77</td>
</tr>
</tbody>
</table>
ReLU layer

• A stack of images becomes a stack of images with no negative values.
Layers get stacked

• The output of one becomes the input of the next.
Deep stacking

- Layers can be repeated several (or many) times.
Fully connected layer

- Typically at the end
- Every feature value from prior layer gets to vote on output activations
Fully connected layer

- Features vote on X or O based on weights
  - A list of feature values becomes a list of votes.
  - Vote depends on how strongly a value predicts X or O
Fully connected layer example

- Assume network trained
- Send image through to perform inference
Fully connected layer example

- Calculate votes for each in last layer
Fully connected layer example
Fully connected layer example
Fully connected layer example
Fully connected layer example

- Decide based on highest vote tally
Fully connected layer

• Notation
Fully connected layer

- These can also be stacked.
Putting it all together

- A set of pixels eventually becomes a set of votes.
Hyper-parameters (knobs)

- Set by you, the ML engineer
  - Number of layers, their types, their order, number of neurons in each
  - Number, structure, and size of features in convolutional layers
  - Window size/stride of pooling layers
  - Number of neurons in fully connected layer
  - Normalization function used (ReLU vs sigmoid)
Other applications for CNNs

• Not just images
  • Typically any 2D (or 3D) data.
  • Things closer together are more closely related than things far away
  • Convolution captures local “spatial” patterns in data.
  • Not as useful when minimal spatial patterns exist in data
Images

Columns of pixels

Rows of pixels
Text

- Too sparse?
Customer data

- Unrelated columns?
- If your data is just as useful after swapping any of your columns with each other, then convolution not appropriate

<table>
<thead>
<tr>
<th>A</th>
<th>22</th>
<th>1A</th>
<th>a@a</th>
<th>1</th>
<th>aa</th>
<th>a1.a</th>
<th>123</th>
<th>aa1</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>33</td>
<td>2B</td>
<td>b@b</td>
<td>2</td>
<td>bb</td>
<td>b2.b</td>
<td>234</td>
<td>bb2</td>
</tr>
<tr>
<td>C</td>
<td>44</td>
<td>3C</td>
<td>c@c</td>
<td>3</td>
<td>cc</td>
<td>c3.c</td>
<td>345</td>
<td>cc3</td>
</tr>
<tr>
<td>D</td>
<td>55</td>
<td>4D</td>
<td>d@d</td>
<td>4</td>
<td>dd</td>
<td>d4.d</td>
<td>456</td>
<td>dd4</td>
</tr>
<tr>
<td>E</td>
<td>66</td>
<td>5E</td>
<td>e@e</td>
<td>5</td>
<td>ee</td>
<td>e5.e</td>
<td>567</td>
<td>ee5</td>
</tr>
<tr>
<td>F</td>
<td>77</td>
<td>6F</td>
<td>f@f</td>
<td>6</td>
<td>ff</td>
<td>f6.f</td>
<td>678</td>
<td>ff6</td>
</tr>
<tr>
<td>G</td>
<td>88</td>
<td>7G</td>
<td>g@g</td>
<td>7</td>
<td>gg</td>
<td>g7.g</td>
<td>789</td>
<td>gg7</td>
</tr>
<tr>
<td>H</td>
<td>99</td>
<td>8H</td>
<td>h@h</td>
<td>8</td>
<td>hh</td>
<td>h8.h</td>
<td>890</td>
<td>hh8</td>
</tr>
<tr>
<td>I</td>
<td>111</td>
<td>9I</td>
<td>i@i</td>
<td>9</td>
<td>ii</td>
<td>i9.i</td>
<td>901</td>
<td>i9</td>
</tr>
</tbody>
</table>

Name, age, address, email, purchases, browsing activity,...
In practice...