Class is 170.

Announcements

Matlab Grader homework,
1 and 2 (of less than 9) homeworks Due 22 April tonight, Binary graded.
167, 165, 164 has done the homework. (If you have not done HW talk to me/TA!)

Homework 3 due 5 May
Homework 4 (SVM +DL) due ~24 May

Jupiter “GPU” home work released Wednesday. Due 10 May

Guidelines is on Piazza
May 5 proposal due. TAs and Peter can approve.
Email or use dropbox
https://www.dropbox.com/request/XGqCV0qXm9LBYz7J1msS
Format “Proposal”+groupNumber

May 20 presentation

Today:
• Stanford CNN 11, SVM, (Bishop 7)
• Play with Tensorflow playground before class http://playground.tensorflow.org
  Solve the spiral problem

Monday
• Stanford CNN 12, K-means, EM (Bishop 9),
Projects

• 3-4 person groups preferred
• Deliverables: Poster, Report & main code (plus proposal, midterm slide)
• **Topics** your own or chose form suggested topics. Some physics inspired.
• April 26 groups due to TA.
• **41 Groups formed. Look at Piazza for help.**
• **Guidelines is on Piazza**
• **May 5** proposal due. TAs and Peter can approve.
  Email or use dropbox  Format “Proposal”+groupName
  https://www.dropbox.com/request/XGqCV0qXm9LBYz7J1msS
• **May 20** Midterm slide presentation. Presented to a subgroup of class.
• **June 5** final poster. Upload June ~3
• Report and code due **Saturday 15 June.**
If a classification system has been trained to distinguish between cats, dogs and rabbits, a confusion matrix will summarize the test results. Assuming a sample of 27 animals — 8 cats, 6 dogs, and 13 rabbits, the confusion matrix could look like the table below:

<table>
<thead>
<tr>
<th>Predicted class</th>
<th>Actual class</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Cat</td>
<td>Dog</td>
</tr>
<tr>
<td>Cat</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>Dog</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Rabbit</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Actual class</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Cat</td>
<td>Non-cat</td>
</tr>
<tr>
<td>Predicted class</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cat</td>
<td>5 True Positives</td>
<td>2 False Positives</td>
</tr>
<tr>
<td>Non-cat</td>
<td>3 False Negatives</td>
<td>17 True Negatives</td>
</tr>
</tbody>
</table>
Let us define an experiment from $P$ positive instances and $N$ negative instances for some condition. The four outcomes can be formulated in a $2 \times 2$ confusion matrix, as follows:

<table>
<thead>
<tr>
<th></th>
<th>True condition</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Total population</td>
<td>Condition positive</td>
<td>Condition negative</td>
<td>Prevalence</td>
<td>Accuracy (ACC)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$\frac{\sum \text{Condition positive}}{\sum \text{Total population}}$</td>
<td>$\frac{\sum \text{True positive} + \sum \text{True negative}}{\sum \text{Total population}}$</td>
</tr>
<tr>
<td>Predicted condition</td>
<td>Predicted condition positive</td>
<td>True positive, Power</td>
<td>False positive, Type I error</td>
<td>Positive predictive value (PPV), Precision = $\frac{\sum \text{True positive}}{\sum \text{Predicted condition positive}}$</td>
<td>False discovery rate (FDR) = $\frac{\sum \text{False positive}}{\sum \text{Predicted condition positive}}$</td>
</tr>
<tr>
<td></td>
<td>Predicted condition negative</td>
<td>False negative, Type II error</td>
<td>True negative</td>
<td>False omission rate (FOR) = $\frac{\sum \text{False negative}}{\sum \text{Predicted condition negative}}$</td>
<td>Negative predictive value (NPV) = $\frac{\sum \text{True negative}}{\sum \text{Predicted condition negative}}$</td>
</tr>
</tbody>
</table>

- **True positive rate (TPR), Recall, Sensitivity**, probability of detection $= \frac{\sum \text{True positive}}{\sum \text{Condition positive}}$
- **False positive rate (FPR), Fall-out**, probability of false alarm $= \frac{\sum \text{False positive}}{\sum \text{Condition negative}}$
- **False negative rate (FNR), Miss rate**, $= \frac{\sum \text{False negative}}{\sum \text{Condition positive}}$
- **Specificity (SPC), Selectivity, True negative rate (TNR)**, $= \frac{\sum \text{True negative}}{\sum \text{Condition negative}}$
- **Positive likelihood ratio (LR+)** $= \frac{\text{TPR}}{\text{FPR}}$
- **Negative likelihood ratio (LR-)** $= \frac{\text{FNR}}{\text{TNR}}$
- **Diagnostic odds ratio (DOR)** $= \frac{\text{LR+}}{\text{LR-}}$
- **$F_1$ score** $= \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$
The ROC space and plots of the four prediction examples.
Other Computer Vision Tasks

Semantic Segmentation

Classification + Localization

Object Detection

Instance Segmentation

**GRASS, CAT, TREE, SKY**

No objects, just pixels

**CAT**

Single Object

**DOG, DOG, CAT**

Multiple Object
Semantic Segmentation Idea: Fully Convolutional

**Downsampling:**
Pooling, strided convolution

Design network as a bunch of convolutional layers, with **downsampling** and **upsampling** inside the network!

- **Input:** 3 x H x W
- **High-res:** D₁ x H/2 x W/2
- **Med-res:** D₂ x H/4 x W/4
- **Low-res:** D₃ x H/4 x W/4
- **High-res:** D₁ x H/2 x W/2
- **Predictions:** H x W

**Upsampling:**
Unpooling or strided transpose convolution
In-Network upsampling: “Unpooling”

**Nearest Neighbor**

```
1 2
3 4
```

Input: 2 x 2

```
1 1 2 2
1 1 2 2
3 3 4 4
3 3 4 4
```

Output: 4 x 4

**“Bed of Nails”**

```
1 2
3 4
```

Input: 2 x 2

```
1 0 2 0
0 0 0 0
3 0 4 0
0 0 0 0
```

Output: 4 x 4

**In-Network upsampling: “Max Unpooling”**

**Max Pooling**

Remember which element was max!

```
1 2 6 3
3 5 2 1
1 2 2 1
7 3 4 8
```

Input: 4 x 4

```
5 6
7 8
```

Output: 2 x 2

```
0 0 2 0
0 1 0 0
0 0 0 0
3 0 0 4
```

Input: 2 x 2

Output: 4 x 4

**Corresponding pairs of downsampling and upsampling layers**
Learnable Upsampling: Transpose Convolution

**Recall:** Normal 3 x 3 convolution, **stride 2** pad 1

Other names:
- Deconvolution (bad)
- Upconvolution
- Fractionally strided convolution
- Backward strided convolution

3 x 3 **transpose** convolution, **stride 2** pad 1

Filter moves 2 pixels in the **output** for every one pixel in the **input**

Stride gives ratio between movement in output and input

Sum where output overlaps

Input gives weight for filter

Dot product between filter and input

Filter moves 2 pixels in the input for every one pixel in the output

Stride gives ratio between movement in input and output
Transpose Convolution: 1D Example

Output contains copies of the filter weighted by the input, summing at where at overlaps in the output.

Need to crop one pixel from output to make output exactly 2x input.
Convolution as Matrix Multiplication (1D Example)

We can express convolution in terms of a matrix multiplication

\[ \tilde{x} \ast \tilde{a} = X \tilde{a} \]

\[
\begin{bmatrix}
x & y & x & 0 & 0 & 0 \\
0 & x & y & x & 0 & 0 \\
0 & 0 & x & y & x & 0 \\
0 & 0 & 0 & x & y & x \\
0 & 0 & 0 & 0 & 0 & 0 \\
\end{bmatrix}
\begin{bmatrix}
0 \\
a \\
b \\
c \\
d \\
0 \\
\end{bmatrix}
= \begin{bmatrix}
ay + bz \\
ax + by + cz \\
 bx + cy + dz \\
 cx + dy \\
end{bmatrix}
\]

Example: 1D conv, kernel size=3, stride=1, padding=1

Convolution transpose multiplies by the transpose of the same matrix:

\[ \tilde{x} \ast^T \tilde{a} = X^T \tilde{a} \]

\[
\begin{bmatrix}
x & 0 & 0 & 0 \\
y & x & 0 & 0 \\
z & y & x & 0 \\
0 & z & y & x \\
0 & 0 & z & y \\
0 & 0 & 0 & z \\
\end{bmatrix}
\begin{bmatrix}
a \\
b \\
c \\
d \\
\end{bmatrix}
= \begin{bmatrix}
ax \\
 ay + bx \\
 az + by + cx \\
bz + cy + dx \\
 cz + dy \\
dz \\
\end{bmatrix}
\]

When stride=1, convolution transpose is just a regular convolution (with different padding rules)

Convolution

\( f \)

\( g \)

\( f \ast g \)

\( g \ast f \)
Convolution as Matrix Multiplication (1D Example)

We can express convolution in terms of a matrix multiplication

\[
\vec{x} \ast \vec{a} = X \vec{a}
\]

Example: 1D conv, kernel size=3, stride=2, padding=1

| \begin{bmatrix} x & y & z & 0 & 0 & 0 \\ 0 & 0 & x & y & z & 0 \end{bmatrix} | \begin{bmatrix} 0 \\ a \\ b \\ c \\ d \\ 0 \end{bmatrix} = \begin{bmatrix} ay + bz \\ bx + cy + dz \end{bmatrix}

Convolution transpose multiplies by the transpose of the same matrix:

\[
\vec{x} \ast^T \vec{a} = X^T \vec{a}
\]

| \begin{bmatrix} x & 0 \\ y & 0 \\ z & x \\ 0 & y \\ 0 & z \\ 0 & 0 \end{bmatrix} | \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} ax \\ ay \\ az + bx \\ by \\ bz \\ 0 \end{bmatrix}

When stride>1, convolution transpose is no longer a normal convolution!
Region Proposals

- Find “blobby” image regions that are likely to contain objects
- Relatively fast to run; e.g. Selective Search gives 1000 region proposals in a few seconds on CPU
Kernels

We might want to consider something more complicated than a linear model:

Example 1: \([x^{(1)}, x^{(2)}] \rightarrow \Phi ([x^{(1)}, x^{(2)}]) = [x^{(1)2}, x^{(2)2}, x^{(1)}x^{(2)}]\)

Information unchanged, but now we have a linear classifier on the transformed points.

With the kernel trick, we just need kernel \(k(a, b) = \Phi(a)^T \Phi(b)\).
Dual representation, Sec 6.2

Primal problem: \[ \min_w E(w) \]
\[
E = \frac{1}{2} \sum_n \{w^T x_n - t_n\}^2 + \frac{\lambda}{2} \|w\|^2 = \|Xw - t\|^2 + \frac{\lambda}{2} \|w\|^2
\]

Solution \[ w = X^+ t = (X^T X + \lambda I_M)^{-1} X^T t \]
\[ = X^T (XX^T + \lambda I_N)^{-1} t = X^T (K + \lambda I_N)^{-1} t = X^T a \]

The kernel is \( K = XX^T \)

Dual representation is: \[ \min_a E(a) \]
\[
E = \frac{1}{2} \sum_n \{w^T x_n - t_n\}^2 + \frac{\lambda}{2} \|w\|^2 = \|Ka - t\|^2 + \frac{\lambda}{2} a^T Ka
\]

Prediction
\[ y = w^T x = a^T Xx = \sum_n a_n x_n^T x = \sum_n a_n k(x_n, x) \]
Dual representation, Sec 6.2

Prediction

\[ y = w^T x = a^T X x = \sum_{n}^N a_n x_n^T x = \sum_{n}^N a_n k(x_n, x) \]

- Often **a is sparse** (… Support vector machines)
- We don’t need to know \( x \) or \( \varphi(x) \). **Just the Kernel**

\[ E(a) = ||Ka - t||^2_2 + \frac{\lambda}{2} a^T Ka \]
Lecture 10
Support Vector Machines

Non Bayesian!

Features:
• Kernel
• Sparse representations
• Large margins
Regularize for plausibility

- Which one is best?
- We maximize the margin

Figure 14.11 Illustration of the large margin principle. Left: a separating hyper-plane with large margin. Right: a separating hyper-plane with small margin.

R₁
R₀
y = 0
y > 0
y < 0
ξ > 1
ξ < 1
ξ = 0

Figure 14.12 (a) Illustration of the geometry of a linear decision boundary in 2d. A point x is classified as belonging in decision region R₁ if f(x) > 0, otherwise it belongs in decision region R₂; here f(x) is known as a discriminant function. The decision boundary is the set of points such that f(x) = 0. w is a vector which is perpendicular to the decision boundary. The term w₀ controls the distance of the decision boundary from the origin. The signed distance of x from its orthogonal projection onto the decision boundary, x⊥, is given by f(x)/||w||. Based on Figure 4.1 of (Bishop 2006a).

(b) Illustration of the soft margin principle. Points with circles around them are support vectors. We also indicate the value of the corresponding slack variables. Based on Figure 7.3 of (Bishop 2006a).
Regularize for plausibility
Support Vector Machines

• The line that maximizes the minimum margin is a good bet.
  – The model class of “hyper-planes with a margin $m$” has a low VC dimension if $m$ is big.

• This maximum-margin separator is determined by a subset of the datapoints.
  – Datapoints in this subset are called “support vectors”.
  – It is useful computationally if only few datapoints are support vectors, because the support vectors decide which side of the separator a test case is on.

The support vectors are indicated by the circles around them.
Lagrange multiplier (Bishop App E)

\[
\max(f(x)) \text{ subject to } g(x) = 0
\]

*Taylor expansion*

\[
g(x + \varepsilon) = g(x) + \varepsilon^T \nabla g(x)
\]

\[
L(x, \lambda) = f(x) + \lambda g(x)
\]
Lagrange multiplier (Bishop App E)

\[ \max(f(x)) \text{ subject to } g(x) > 0 \]

\[ L(x, \lambda) = f(x) + \lambda g(x) \]

Either \( \nabla f(x) = 0 \)

Then \( g(x) \) is inactive, \( \lambda = 0 \)

Or \( g(x) = 0 \) but \( \lambda > 0 \)

Thus optimizing \( L(x, \lambda) \) with the Karesh-Kuhn-Trucker (KKT) equations

\[ g(x) \geq 0 \]
\[ \lambda \geq 0 \]
\[ \lambda g(x) = 0 \]
Testing a linear SVM

- The separator is defined as the set of points for which:

\[ \mathbf{w} \cdot \mathbf{x} + b = 0 \]

so if \( \mathbf{w} \cdot \mathbf{x}^c + b > 0 \) say its a positive case

and if \( \mathbf{w} \cdot \mathbf{x}^c + b < 0 \) say its a negative case
**Discriminant functions**

The planar decision surface in data-space for the simple linear discriminant function:

\[ \mathbf{w}^T \mathbf{x} + w_0 \geq 0 \]

\[ y = \mathbf{w}^T \mathbf{x} + w_0 \]

\[ y_\perp = \mathbf{w}^T \mathbf{x}_\perp + w_0 = 0 \quad \sqrt{||\mathbf{w}||^2 = \mathbf{w}^T \mathbf{w}} \]

**Distance from plane**

\[ r = \frac{y}{||\mathbf{w}||_2} \]

\[ X = X_\perp + r \frac{\mathbf{w}}{||\mathbf{w}||_2} \]

\[ \mathbf{w}^T X = \mathbf{w}^T X_\perp + r \frac{\mathbf{w}^T \mathbf{w}}{||\mathbf{w}||^2} \]

\[ X = \frac{-w_0}{||\mathbf{w}||_2} + r \frac{\mathbf{w}}{||\mathbf{w}||_2} \]
Large margin

\[ y = w^T x + b \]

\[ x_n = x_\perp + r_n \frac{w}{\|w\|} \]

\[ \mathbf{x} \text{ on plane } \implies y=0 \implies b = -w^T x \]

\[ r_n = \frac{w^T x_n + b}{\|w\|} = \frac{y_n}{\|w\|} \]

\[ t_n y_n \geq 1 \]

\[ \max_{w} \frac{1}{\|w\|} \min_{n} t_n y_n \]
Maximum margin (Bishop 7.1)

\[
\arg\min_{w,b} \frac{1}{2} \|w\|^2 \\
\text{Subject to} \quad t_n \left( w^T \phi(x_n) + b \right) \geq 1, \quad n = 1, \ldots, N. \quad (7.5)
\]

Lagrange function

\[
L(w, b, a) = \frac{1}{2} \|w\|^2 - \sum_{n=1}^{N} a_n \{ t_n (w^T \phi(x_n) + b) - 1 \} \quad (7.7)
\]

Differentiation

\[
w = \sum_{n=1}^{N} a_n t_n \phi(x_n) \quad (7.8)
\]

\[
0 = \sum_{n=1}^{N} a_n t_n. \quad (7.9)
\]

Dual representation

\[
\tilde{L}(a) = \sum_{n=1}^{N} a_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} a_n a_m t_n t_m k(x_n, x_m) \quad (7.10)
\]

with respect to \( a \) subject to the constraints

\[
a_n \geq 0, \quad n = 1, \ldots, N, \quad (7.11)
\]

\[
\sum_{n=1}^{N} a_n t_n = 0. \quad (7.12)
\]

This can be solved with quadratic programming
Maximum margin (Bishop 7.1)

- KKT conditions

\[ a_n \geq 0 \]  \hspace{1cm} (7.14)
\[ t_n y(x_n) - 1 \geq 0 \]  \hspace{1cm} (7.15)
\[ a_n \{t_n y(x_n) - 1\} = 0. \]  \hspace{1cm} (7.16)

either \( a_n = 0 \) or \( t_n y(x_n) = 1 \).

- Solving for \( a_n \)

\[ w = \sum_{n=1}^{N} a_n t_n \phi(x_n) \]  \hspace{1cm} (7.8)

- Prediction

\[ y(x) = \sum_{n=1}^{N} a_n t_n k(x, x_n) + b. \]  \hspace{1cm} (7.13)
If there is no separating plane...

- Use a bigger set of features.
  - Makes the computation slow? “Kernel” trick makes the computation fast with many features.

- Extend definition of maximum margin to allow non-separating planes.
  - Use “slack” variables $\xi = |t_n - y(x_n)|$

$$t_n y(x_n) \geq 1 - \xi_n, \quad n = 1, \ldots, N$$  \hfill (7.20)

**Objective function**

$$C \sum_{n=1}^{N} \xi_n + \frac{1}{2} \|w\|^2$$  \hfill (7.21)
SVM classification summarized--- Only kernels

- Minimize with respect to $\mathbf{w}, w_0$
  \[ C \sum_n \zeta_n + \frac{1}{2} \| \mathbf{w} \|^2 \]  
  (Bishop 7.21)

- Solution found in dual domain with Lagrange multipliers
  - $a_n, n = 1 \cdots N$ and

- This gives the support vectors $S$
  \[ \hat{\mathbf{w}} = \sum_{n \in S} a_n t_n \varphi(x_n) \]  
  (Bishop 7.8)

- Used for predictions

  \[ \hat{y} = w_0 + \mathbf{w}^T \varphi(x) = w_0 + \sum_{n \in S} a_n t_n \varphi(x_n)^T \varphi(x) \]

  \[ = w_0 + \sum_{n \in S} a_n t_n k(x_n, x) \]  
  (Bishop 7.13)
How to make a plane curved

• Fitting hyperplanes as separators is mathematically easy.
  – The mathematics is **linear**.

• Replacing the raw input variables with a much larger set of features we get a nice property:
  – A planar separator in high-D feature space is a curved separator in the low-D input space.

![A planar separator in a 20-D feature space projected back to the original 2-D space](image)
SVMs are Perceptrons!

- SVM’s use each training case, \( x \), to define a feature \( K(x, .) \) where \( K \) is user chosen.
  - So the user designs the features.
- SVM do “feature selection” by picking support vectors, and learn feature weighting from a big optimization problem.
- \( \Rightarrow \) SVM is a clever way to train a standard perceptron.
  - What a perceptron cannot do, SVM cannot do.

- SVM DOES:
  - Margin maximization
  - Kernel trick
  - Sparse
SVM Code for classification (libsvm)

Part of ocean acoustic data set http://noiselab.ucsd.edu/ECE285/SIO209Final.zip

case 'Classify'

% train
model = svmtrain(Y, X,['-c 7.46 -g ' gamma ' -q ' kernel]);

% predict
[predict_label,~,~] = svmpredict(rand([length(Y),1]), X, model,'-q');

>> model = struct with fields:
   Parameters: [5x1 double]
   nr_class: 2
   totalSV: 36
   rho: 8.3220
   Label: [2x1 double]
   sv_indices: [36x1 double]
   ProbA: []  ProbB: []
   nSV: [2x1 double]
   sv_coef: [36x1 double]
   SVs: [36x2 double]
Finding the Decision Function

- **w**: maybe infinite variables
- The dual problem

\[
\min_{\alpha} \frac{1}{2} \alpha^T Q \alpha - e^T \alpha
\]

subject to \(0 \leq \alpha_i \leq C, i = 1, \ldots, l\)

\(y^T \alpha = 0,\)

where \(Q_{ij} = y_i y_j \phi(x_i)^T \phi(x_j)\) and \(e = [1, \ldots, 1]^T\)

- At optimum

\[
w = \sum_{i=1}^l \alpha_i y_i \phi(x_i)
\]

- A finite problem: \#variables = \#training data

Using these results to eliminate \(w, b,\) and \(\{\xi_n\}\) from the Lagrangian, we obtain the dual Lagrangian in the form

\[
\tilde{L}(\alpha) = \sum_{n=1}^N a_n - \frac{1}{2} \sum_{n=1}^N \sum_{m=1}^N a_n a_m t_n t_m k(x_n, x_m)
\]

Corresponds to (Bishop 7.32)

With y=t
Gaussian Kernels

- Gaussian Kernel

\[ k(x, x') = \exp \left( -\frac{1}{2} (x - x')^T \Sigma^{-1} (x - x') \right) \]

Diagonal \( \Sigma \): (this gives ARD)

\[ k(x, x') = \exp \left( -\frac{1}{2} \sum_{i=1}^{N} \left( \frac{x_i - x_i'}{\sigma_i^2} \right)^2 \right) \]

Isotropic \( \sigma_i^2 \) gives an RBF

\[ k(x, x') = \exp \left( -\frac{||x - x'||_2^2}{2\sigma^2} \right) \]
Can be inner product in infinite dimensional space

Assume $x \in R^1$ and $\gamma > 0$.

$$e^{-\gamma \|x_i - x_j\|^2} = e^{-\gamma (x_i - x_j)^2} = e^{-\gamma x_i^2 + 2\gamma x_i x_j - \gamma x_j^2}$$

$$= e^{-\gamma x_i^2 - \gamma x_j^2} \left( 1 + \frac{2\gamma x_i x_j}{1!} + \frac{(2\gamma x_i x_j)^2}{2!} + \frac{(2\gamma x_i x_j)^3}{3!} + \cdots \right)$$

$$= e^{-\gamma x_i^2 - \gamma x_j^2} \left( 1 \cdot 1 + \sqrt{\frac{2\gamma}{1!}} x_i \cdot \sqrt{\frac{2\gamma}{1!}} x_j + \sqrt{\frac{(2\gamma)^2}{2!}} x_i^2 \cdot \sqrt{\frac{(2\gamma)^2}{2!}} x_j^2 \\
+ \sqrt{\frac{(2\gamma)^3}{3!}} x_i^3 \cdot \sqrt{\frac{(2\gamma)^3}{3!}} x_j^3 + \cdots \right) = \phi(x_i)^T \phi(x_j),$$

where

$$\phi(x) = e^{-\gamma x^2} \left[ 1, \sqrt{\frac{2\gamma}{1!}} x, \sqrt{\frac{(2\gamma)^2}{2!}} x^2, \sqrt{\frac{(2\gamma)^3}{3!}} x^3, \cdots \right]^T.$$
1. Fitting the spiral with default settings fail due to the small training set. The NN will fit to the training data which is not representative of the true pattern and the network will **generalize** poorly. Increasing the ratio of training to test data to 90% the NN finds the correct shape (1st image).
Tensorflow Playground

You can fix the generalization problem by adding **noise** to the data. This allows the small training set to generalize better as it reduce **overfitting** of the training data (2nd image).
Adding an additional hidden layer the NN fails to classify the shape properly. **Overfitting** once again becomes a problem even after you've added noise. This can be fixed by adding appropriate **L2 regularization** (third image).
NOT USED