UCK358E – INTR. TO ARTIFICIAL INTELLIGENCE

SPRING ‘23

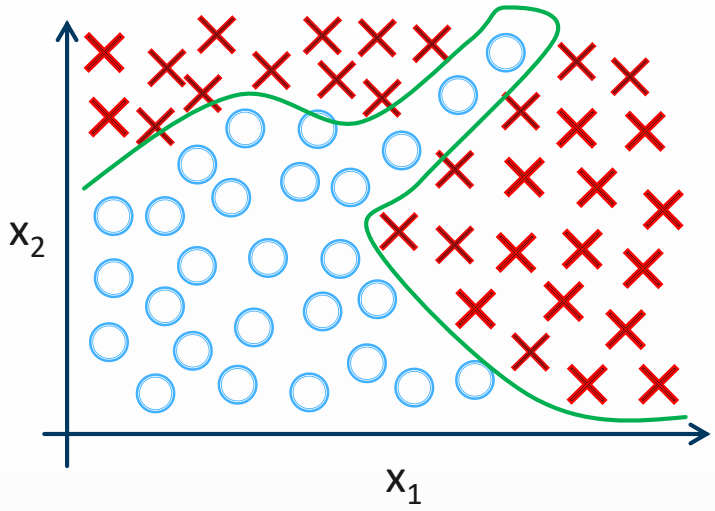
LECTURE 8

ARTIFICIAL NEURAL NETWORKS: REPRESENTATION

Instructor: Asst. Prof. Barış Başpınar

Non-linear Hypotheses

Non-linear Classification



$$g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 + \theta_4 x_1^2 x_2 + \theta_5 x_1^3 x_2 + \theta_6 x_1 x_2^2 + \dots)$$

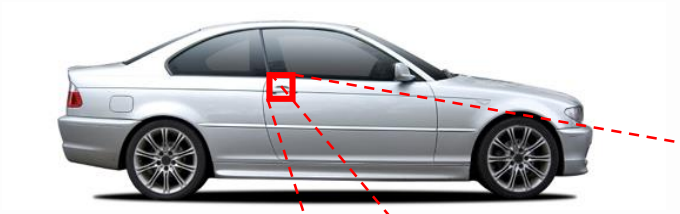
x_1 = size
x_2 = # bedrooms
x_3 = # floors
x_4 = age
$\cdots$
x_{100}

- The number of features will increase to create high-order polynomials
- This impact will be intensified when there are several features (e.g. 100 features in house pricing task -> 170000 features for 3rd degree polynomials)

Non-linear Hypotheses

What is this?

You see this:

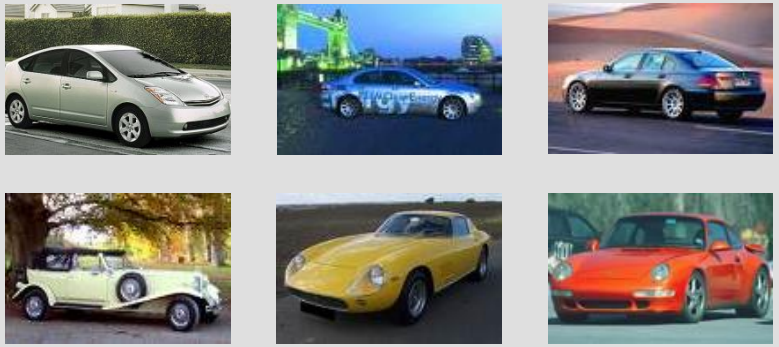


But the camera sees this:

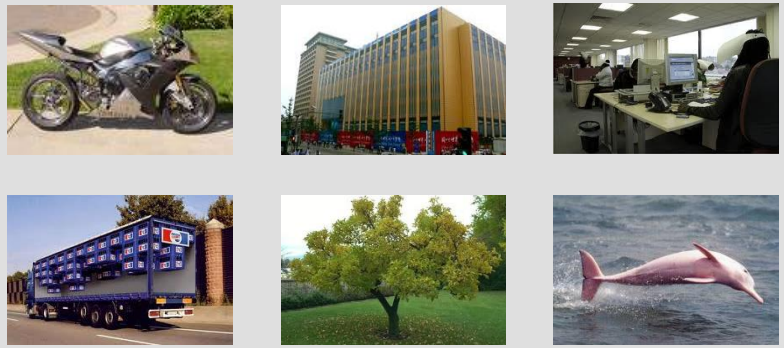
194	210	201	212	199	213	215	195	178	158	182	209
180	189	190	221	209	205	191	167	147	115	129	163
114	126	140	188	176	165	152	140	170	106	78	88
87	103	115	154	143	142	149	153	173	101	57	57
102	112	106	131	122	138	152	147	128	84	58	66
94	95	79	104	105	124	129	113	107	87	69	67
68	71	69	98	89	92	98	95	89	88	76	67
41	56	68	99	63	45	60	82	58	76	75	65
20	43	69	75	56	41	51	73	55	70	63	44
50	50	57	69	75	75	73	74	53	68	59	37
72	59	53	66	84	92	84	74	57	72	63	42
67	61	58	65	75	78	76	73	59	75	69	50

Non-linear Hypotheses

Computer Vision: Car detection



Cars

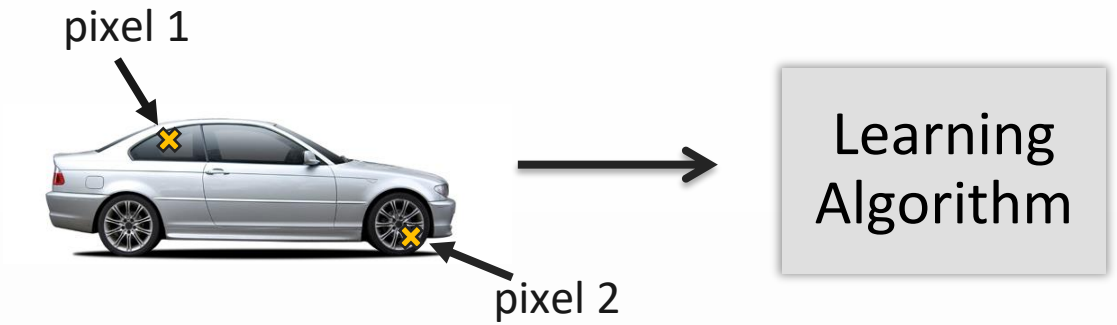


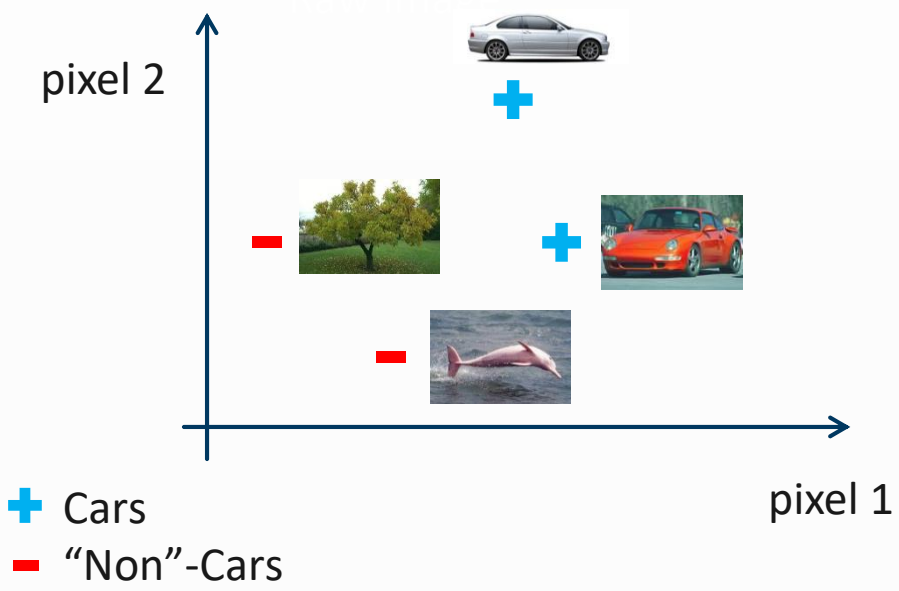
Not a car

Testing:

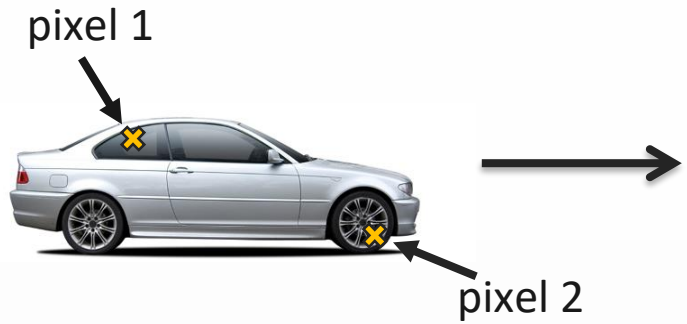
What is this?

Non-linear Hypotheses

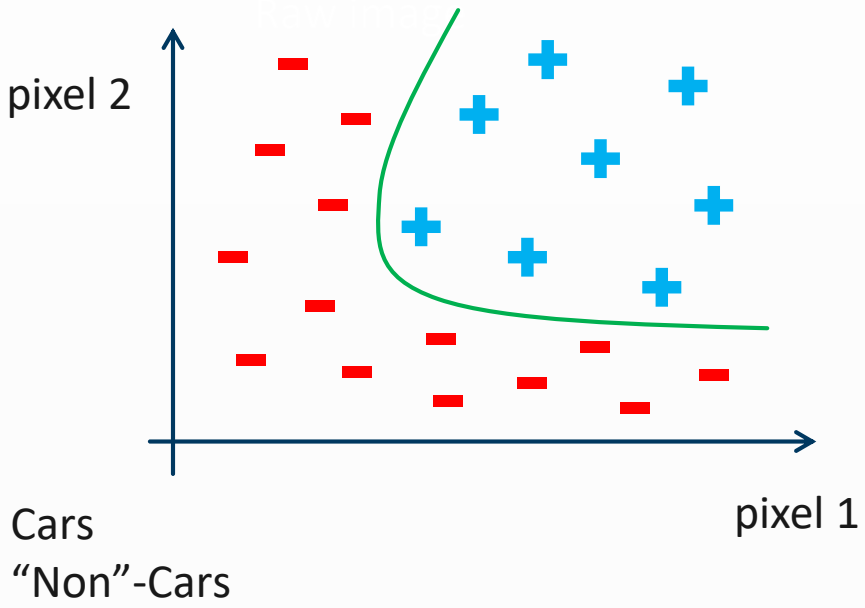




Non-linear Hypotheses



Learning Algorithm



50 x 50 pixel images→ 2500 pixels

$n = 2500$ (7500 if RGB)

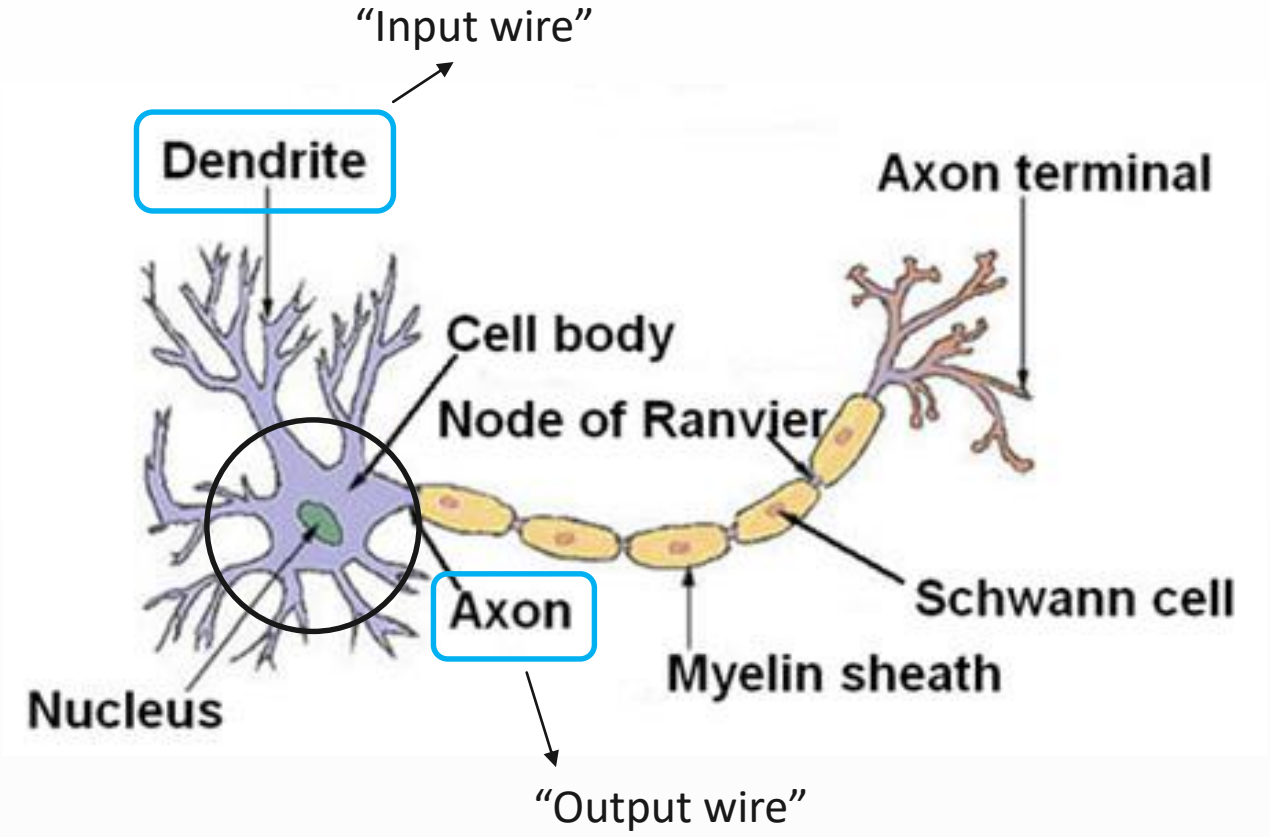
$$x = \begin{bmatrix} \text{pixel 1 intensity} \\ \text{pixel 2 intensity} \\ \vdots \\ \text{pixel 2500 intensity} \end{bmatrix}$$

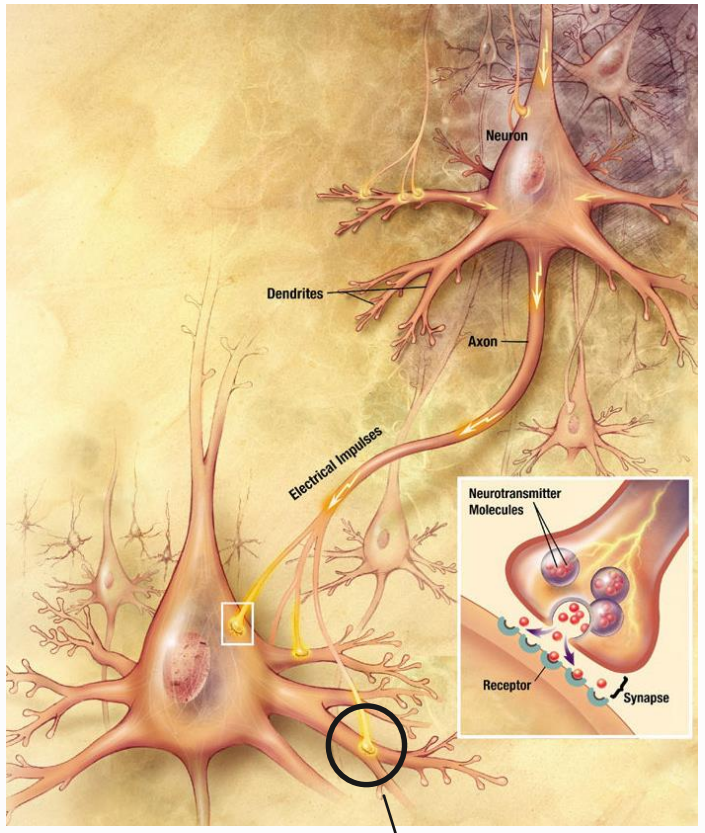
Quadratic features ($x_i \times x_j$): ≈3 million features

Neural Networks

- Origins: Algorithms that try to mimic the brain.
- Was very widely used in 80s and early 90s; popularity diminished in late 90s.
- Recent resurgence: State-of-the-art technique for many applications

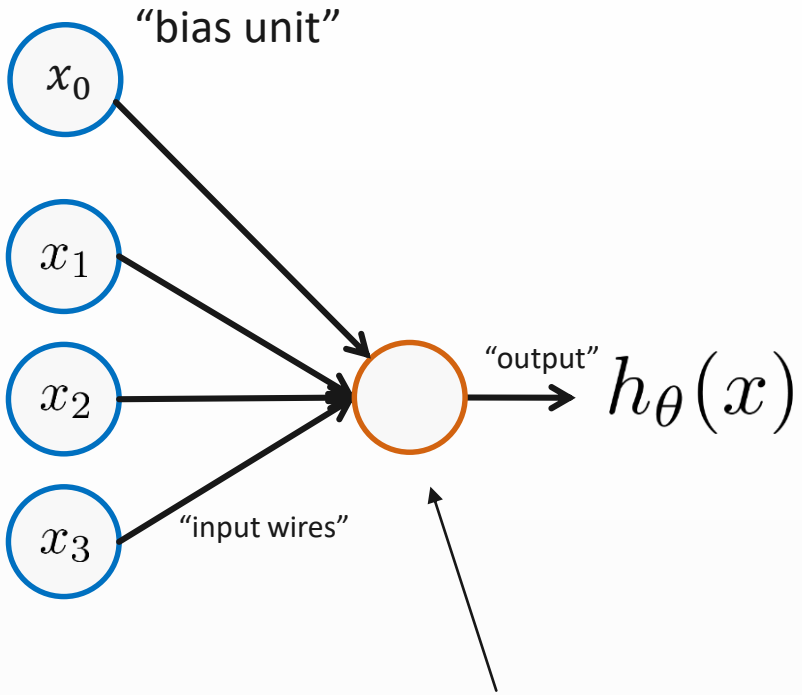
Neuron in the brain





"Output of neuron 1"
"Input of neuron 2"

Artificial Neuron Model: Logistic Unit

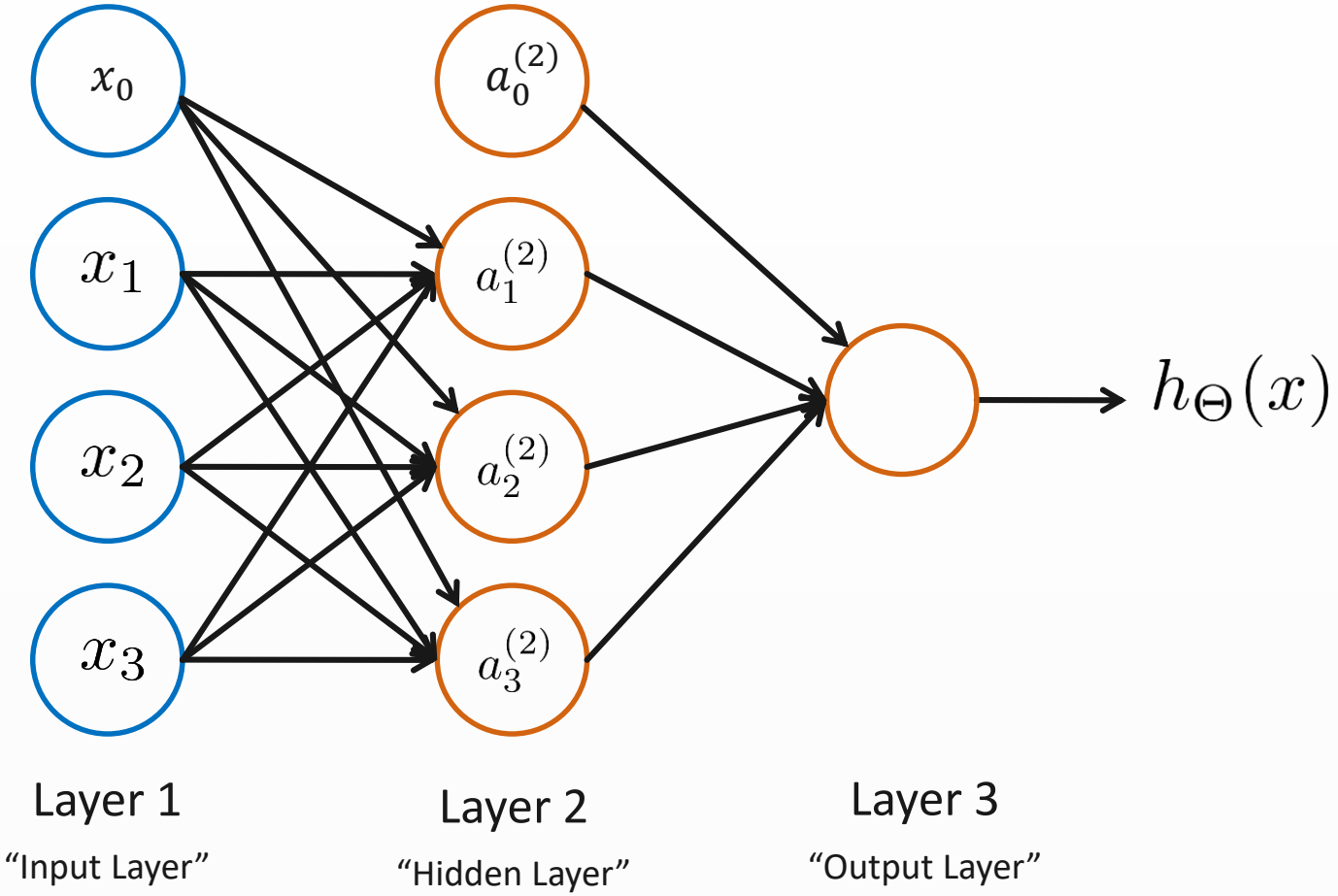


$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

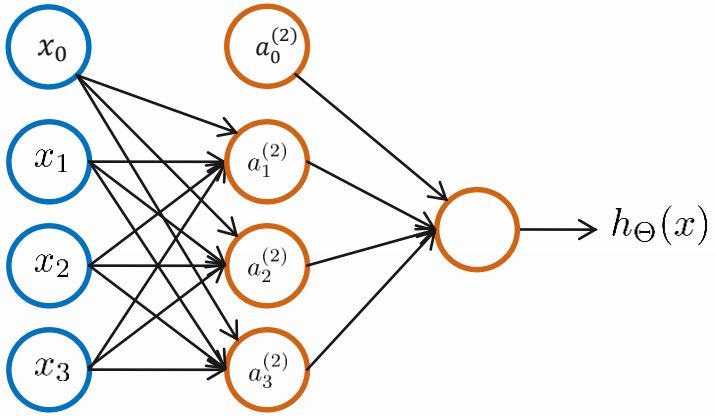
"weights"

An activation function (e.g. Sigmoid (logistic) activation function)

Artificial Neural Network (ANN)



Artificial Neural Network (ANN)



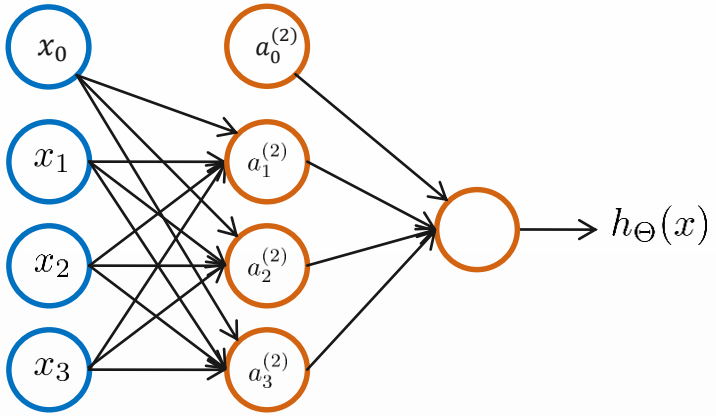
$a_i^{(j)}$ = "activation" of unit i in layer j

$\Theta^{(j)}$ = matrix of weights controlling function mapping from layer j to layer $j+1$

$$a_1^{(2)} = g(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3)$$
$$a_2^{(2)} = g(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3)$$
$$a_3^{(2)} = g(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3)$$
$$h_\Theta(x) = a_1^{(3)} = g(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)})$$

If network has s_j units in layer j, s_{j+1} units in layer $j+1$, then $\Theta^{(j)}$ will be of dimension $s_{j+1} \times (s_j + 1)$.

Forward propagation: Vectorized implementation



$$a_1^{(2)} = g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3) \quad z_1^{(2)}$$
$$a_2^{(2)} = g(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3) \quad z_2^{(2)}$$
$$a_3^{(2)} = g(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3) \quad z_3^{(2)}$$

$$h_\Theta(x) = g(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)}) \quad z^{(3)}$$

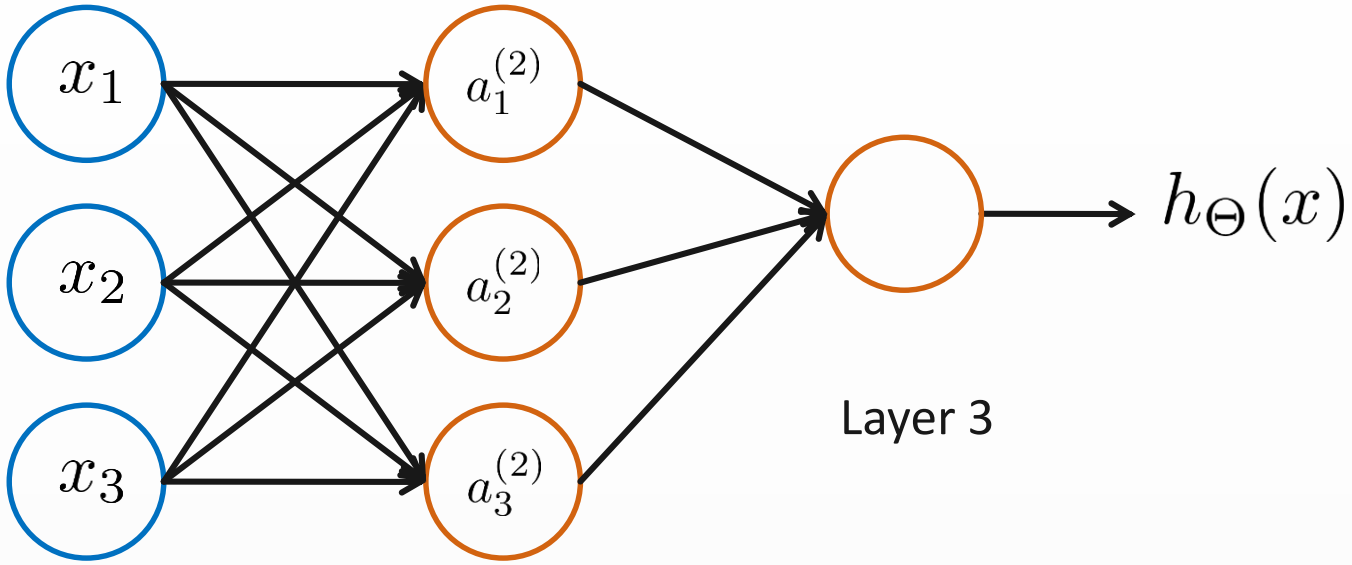
$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad z^{(2)} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix}$$

$$z^{(2)} = \Theta^{(1)} x$$
$$a^{(2)} = g(z^{(2)})$$

Add $a_0^{(2)} = 1$

$$z^{(3)} = \Theta^{(2)} a^{(2)}$$
$$h_\Theta(x) = a^{(3)} = g(z^{(3)})$$

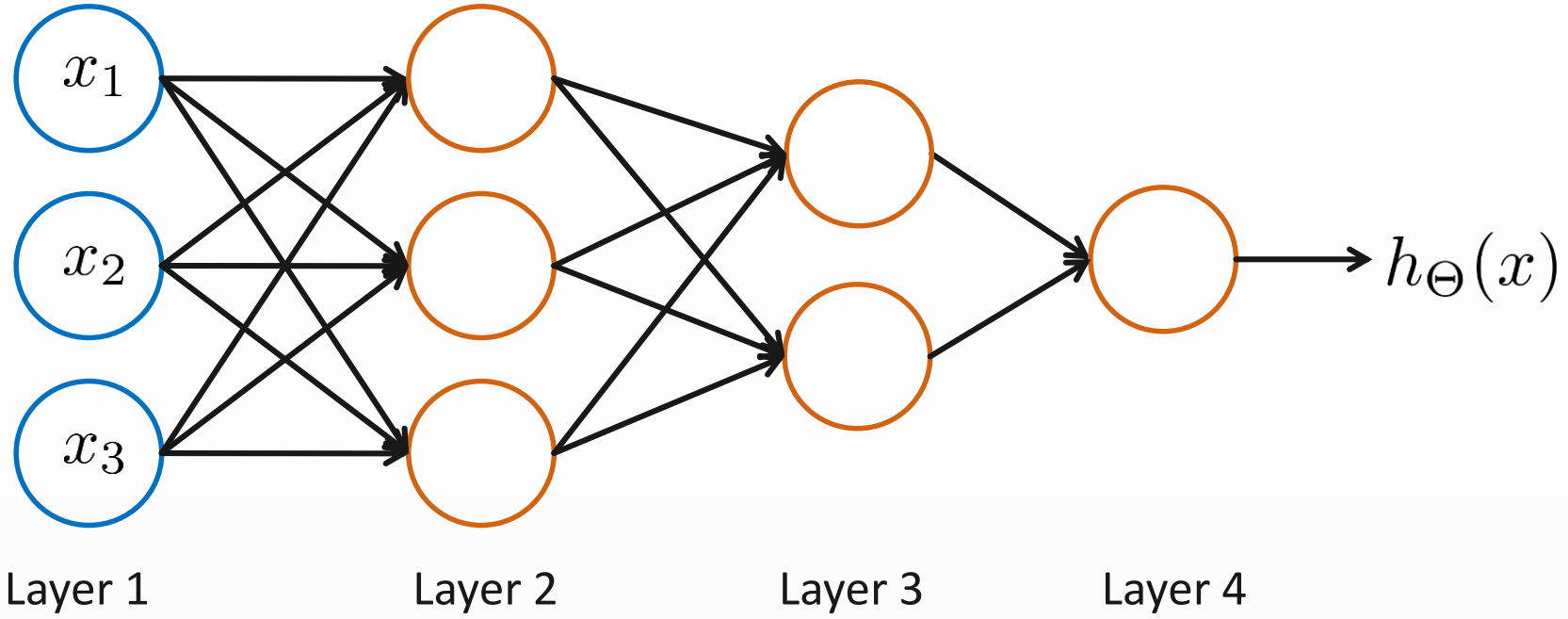
Neural Network learning its own features



$$h_\Theta(x) = g(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)})$$

- $a_1^{(2)}, a_2^{(2)}, a_3^{(2)}$ are new features that are derived by ANN

Other network architectures

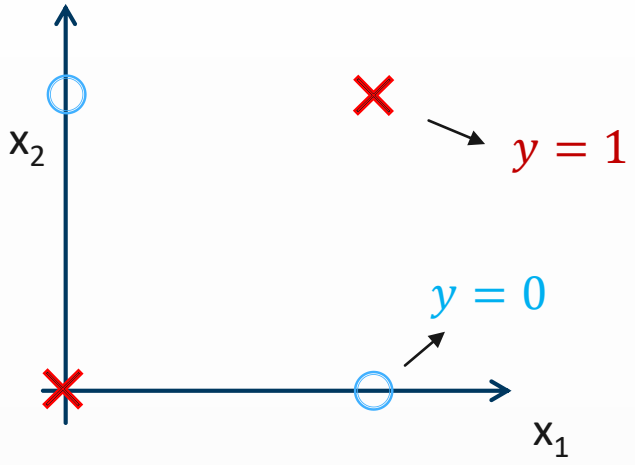


- Different architectures can be used
- You are not limited to single hidden layer
  - There could be several hidden layers with different number of neurons

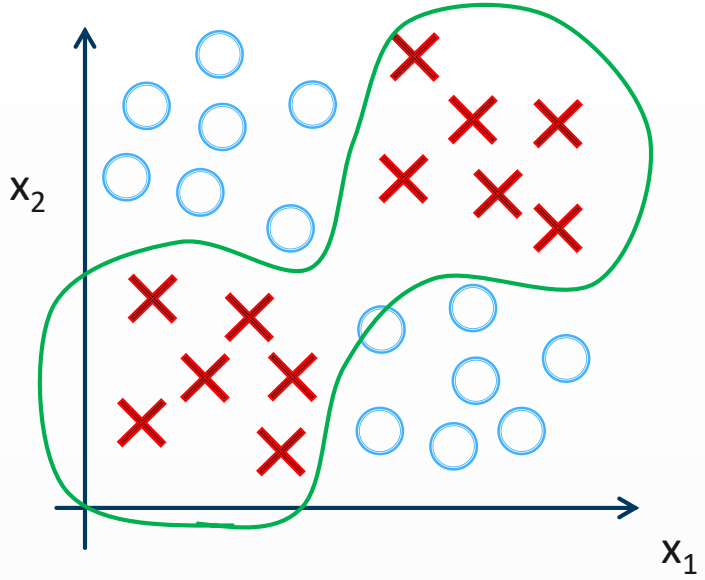
Examples and Intuitions

Non-linear classification example: XOR/XNOR

x_1, x_2 are binary (0 or 1).



$$y = x_1 \text{ XOR } x_2$$
$$x_1 \text{ XNOR } x_2$$
$$\text{NOT } (x_1 \text{ XOR } x_2)$$



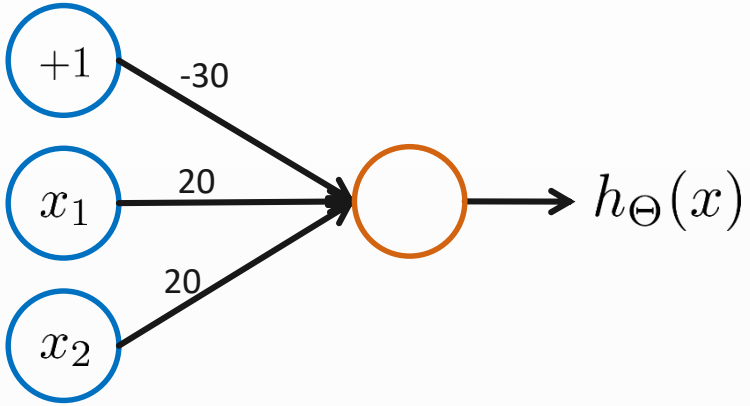
- It is not possible to present this operator using a linear decision boundary

Examples and Intuitions

Simple example: AND

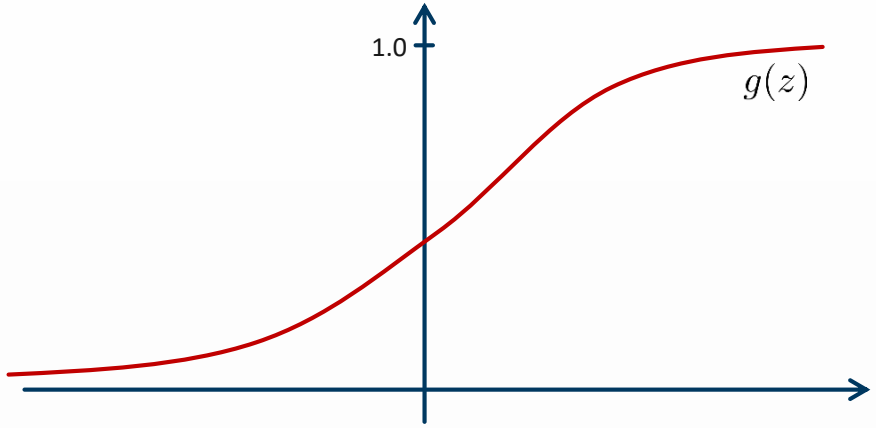
$x_1, x_2 \in \{0, 1\}$

$y = x_1 \text{ AND } x_2$



$h_\Theta(x) = g(-30 + 20x_1 + 20x_2)$

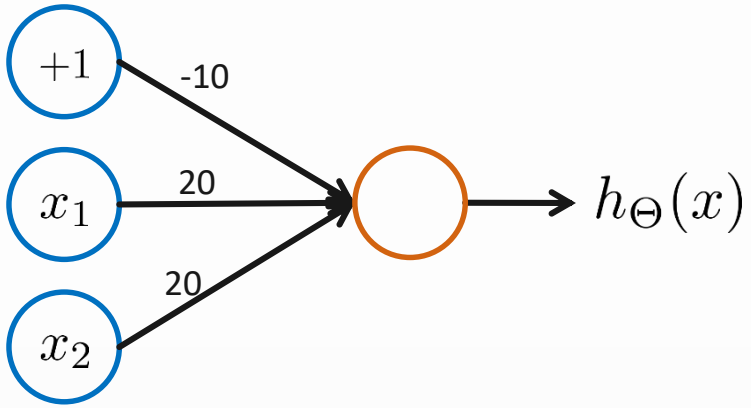
$g(z)$ → sigmoid activation function



x_1	x_2	$h_\Theta(x)$
0	0	$g(-30) \approx 0$
0	1	$g(-10) \approx 0$
1	0	$g(-10) \approx 0$
1	1	$g(10) \approx 1$

Examples and Intuitions

Example: OR function



x_1	x_2	$h_\Theta(x)$
0	0	$g(-10) \approx 0$
0	1	$g(10) \approx 1$
1	0	$g(10) \approx 1$
1	1	$g(30) \approx 1$

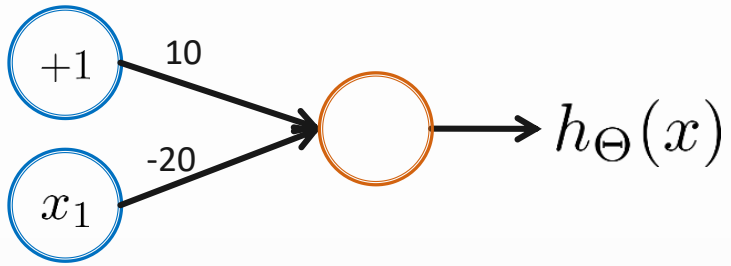
$$h_\Theta(x) = g(-10 + 20x_1 + 20x_2)$$

$g(z) \rightarrow$ sigmoid activation function

Examples and Intuitions

x_1 AND x_2 | x_1 OR x_2

Negation:

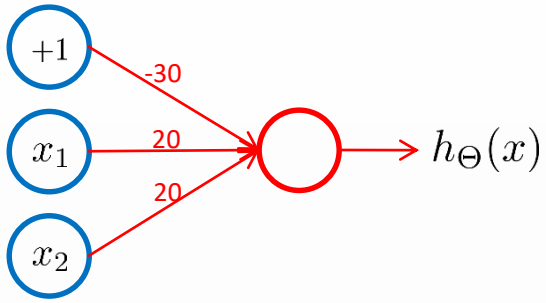


x_1	$h_\Theta(x)$
0	$g(10) \approx 1$
1	$g(-10) \approx 0$

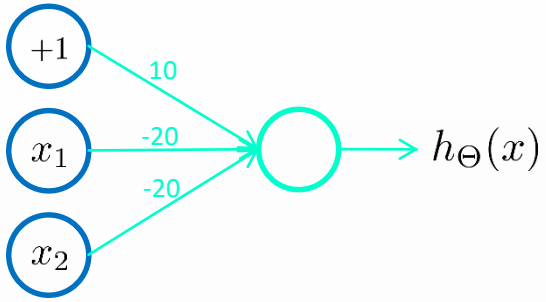
$h_\Theta(x) = g(10 - 20x_1)$ | (NOT x_1) AND (NOT x_2)

Examples and Intuitions

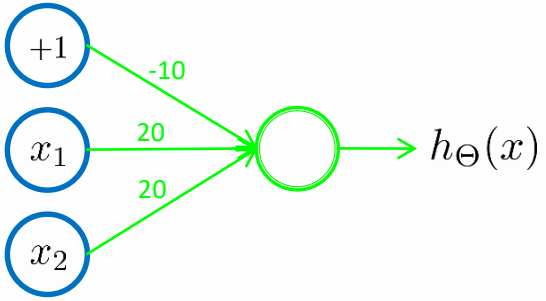
Putting it together: x_1 XNOR x_2



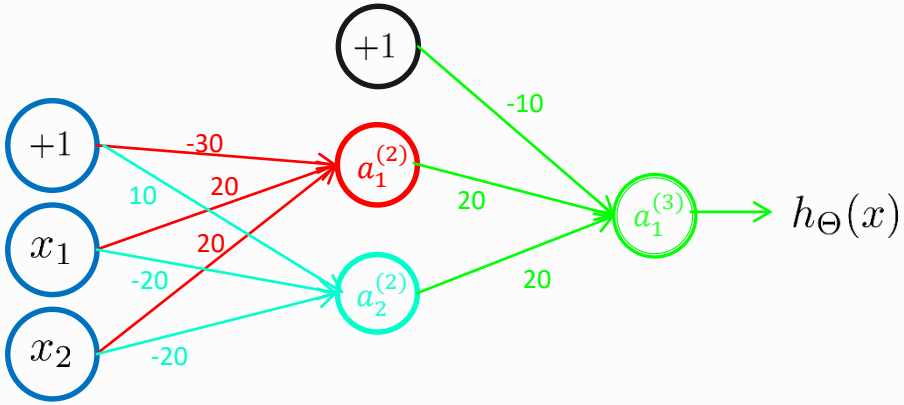
x_1 AND x_2



(NOT x_1) AND (NOT x_2)



x_1 OR x_2



x_1	x_2	$a_1^{(2)}$	$a_2^{(2)}$	$h_\Theta(x)$
0	0	0	1	1
0	1	0	0	0
1	0	0	0	0
1	1	1	0	1

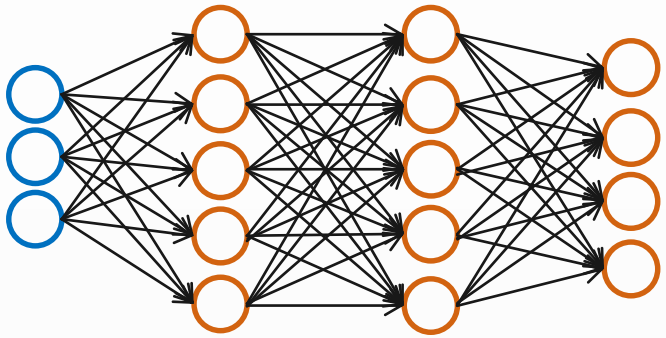
Multiple output units: One-vs-all

Pedestrian

Car

Motorcycle

Truck



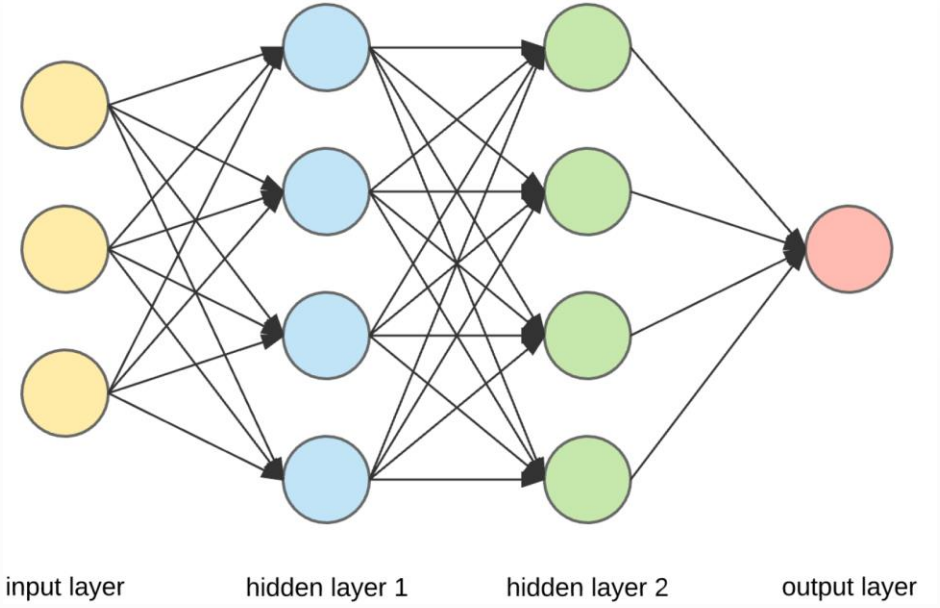
$h_\Theta(x) \in \mathbb{R}^4$

Want $h_\Theta(x) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$, $h_\Theta(x) \approx \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$, $h_\Theta(x) \approx \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$, etc.

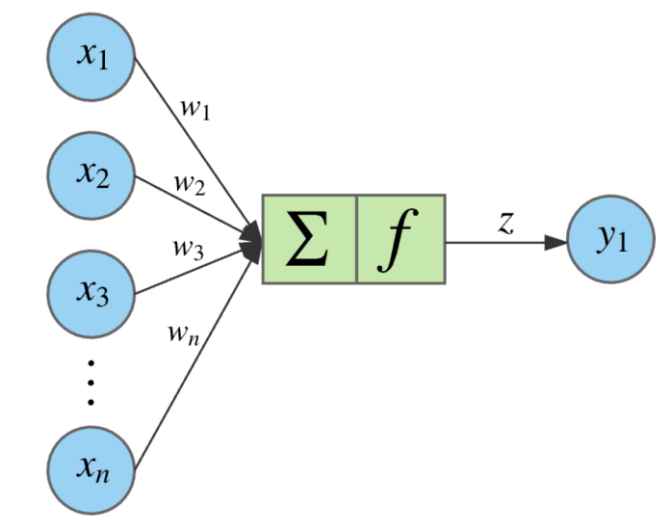
when pedestrian when car when motorcycle

ANN Summary

- An artificial neural network consists of an input and output layers and multiple hidden layers in between



- If we zoom in to one of hidden nodes, we see a linear combination followed by a nonlinear transformation



- Hence the network output is:

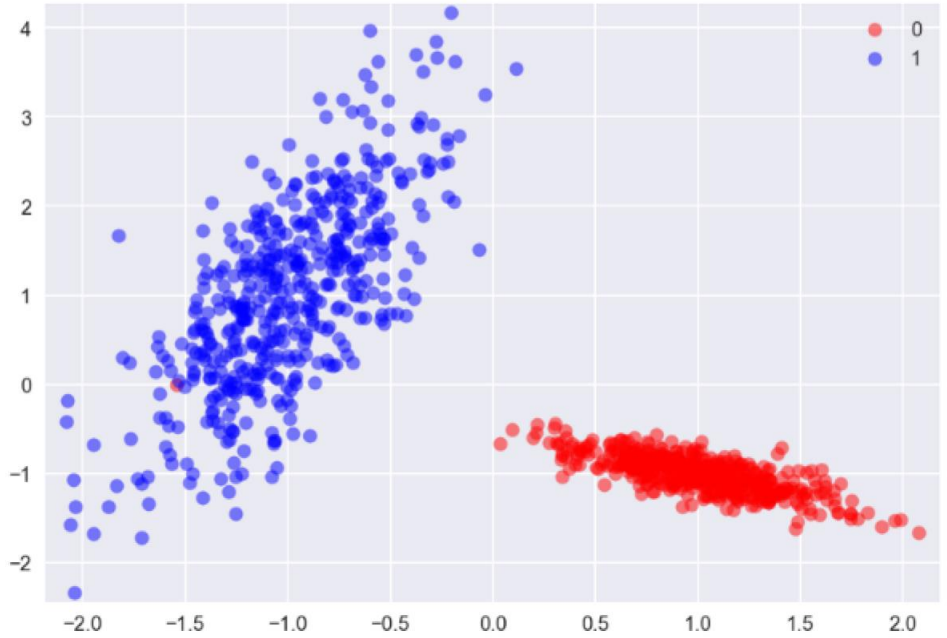
$$y = f(f(f(x.W_1).W_2).W_3)$$

Example: Linearly separable data

Let's implement logistic regression as a special case of ANN.
First, we generate some toy data

```
X, y = make_classification(n_samples=1000, n_features=2, n_redundant=0,
                           n_informative=2, random_state=7, n_clusters_per_class=1)
plot_data(X, y)
```

linearly_separable_data_1.py hosted with ❤ by GitHub view raw

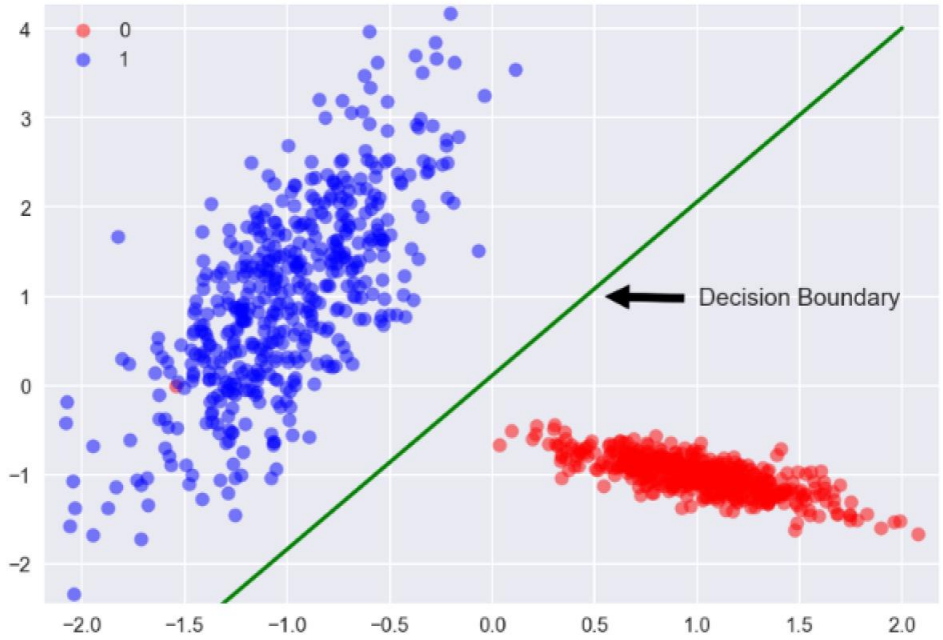


Example: Linearly separable data – Logistic Regression

Now call the logistic regression from scikit-learn

```
1  lr = LogisticRegression()
2  lr.fit(X, y)
```

linearly_separable_data_2.py hosted with ❤ by GitHub view raw



Example: Linearly separable data – Logistic Regression (with ANN)

We will use Keras with TensorFlow backend to solve the same example with ANNs

```
1  model = Sequential()
2  model.add(Dense(units=1, input_shape=(2,), activation='sigmoid'))
3
4  model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
5
6  history = model.fit(x=X, y=y, verbose=0, epochs=50)
7  plot_loss_accuracy(history)
```

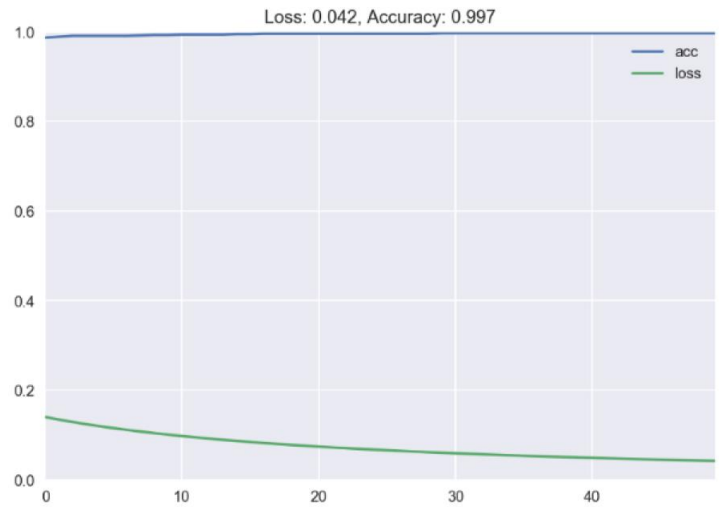
linearly_separable_data_3.py hosted with ❤ by GitHub view raw

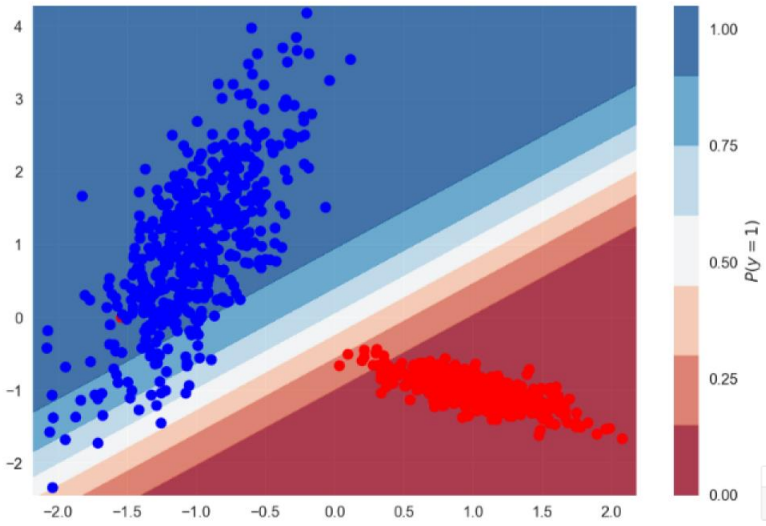
Defining ANN architectures with Keras is very straightforward, as you can see from the code snippet

- Here you can see how to define a simple feedforward hidden layer (referred to as *Dense* in Keras)
- Defining the solver parameters (loss function, optimizer, number of epochs etc) is also very straightforward

Example: Linearly separable data – Logistic Regression (with ANN)

Results look good, since this is a very easy problem:



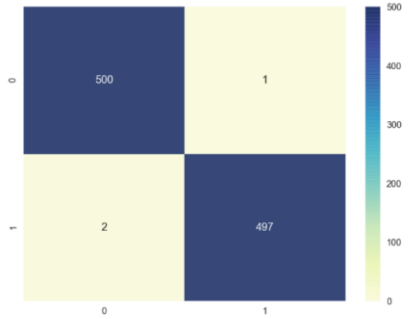



```
plot_confusion_matrix(model, X, y)
```

linearly_separable_data_6.py hosted with ♥ by GitHub view raw

```
plot_decision_boundary(lambda x: model.predict(x), X, y)
```

linearly_separable_data_4.py hosted with ♥ by GitHub view raw

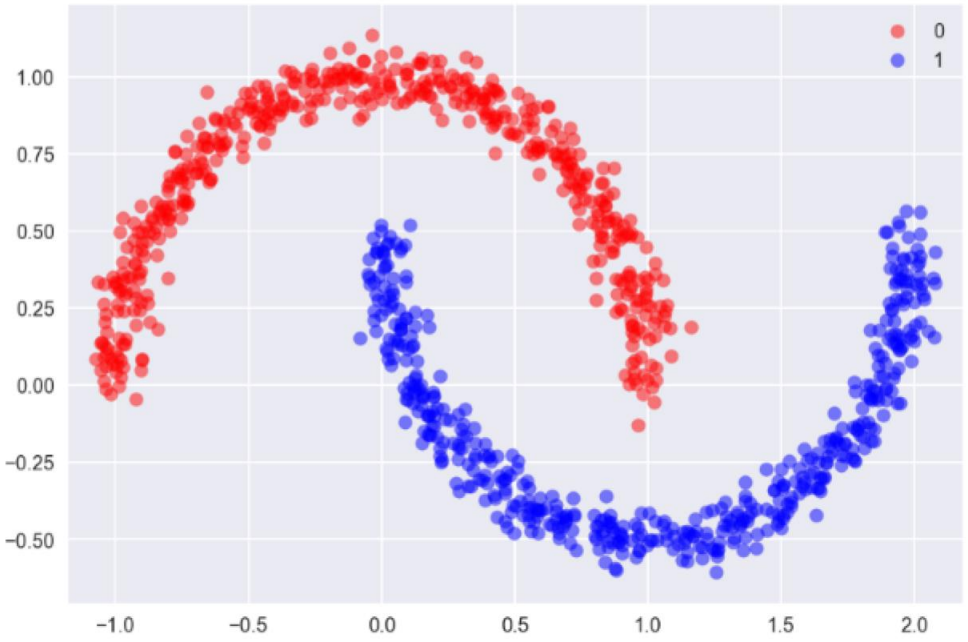


Example: Two Moons Data

So far so good, but what about data that are not linearly separable:

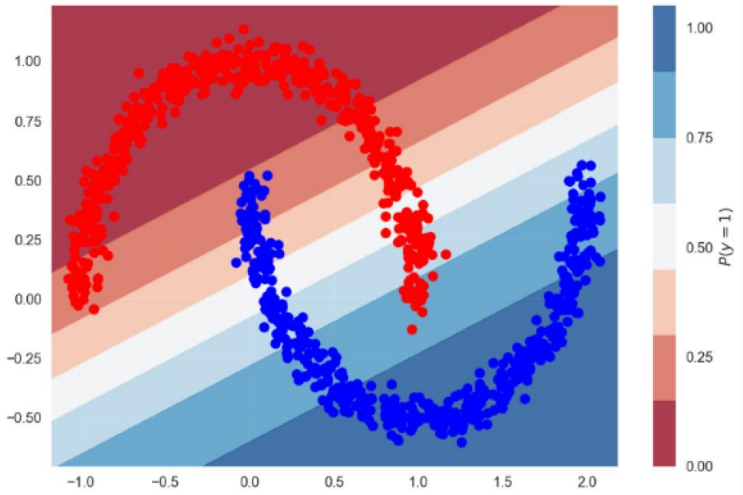
```python
X, y = make_moons(n_samples=1000, noise=0.05, random_state=0)
plot_data(X, y)
```

moons_1.py hosted with ❤ by GitHub view raw

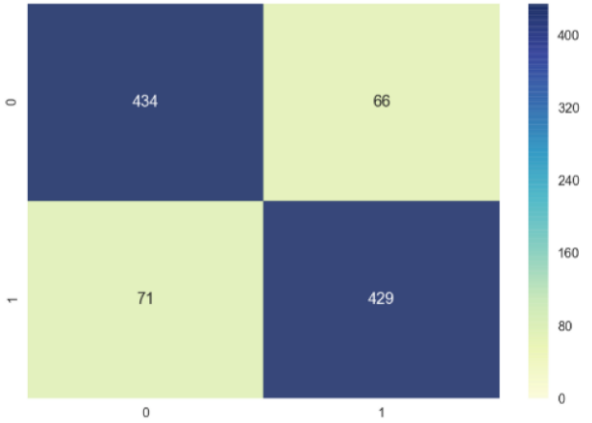


Example: Two Moons Data – Logistic Regression

Results are not that good anymore:



	precision	recall	f1-score	support
0	0.86	0.87	0.86	500
1	0.87	0.86	0.86	500
avg / total	0.86	0.86	0.86	1000

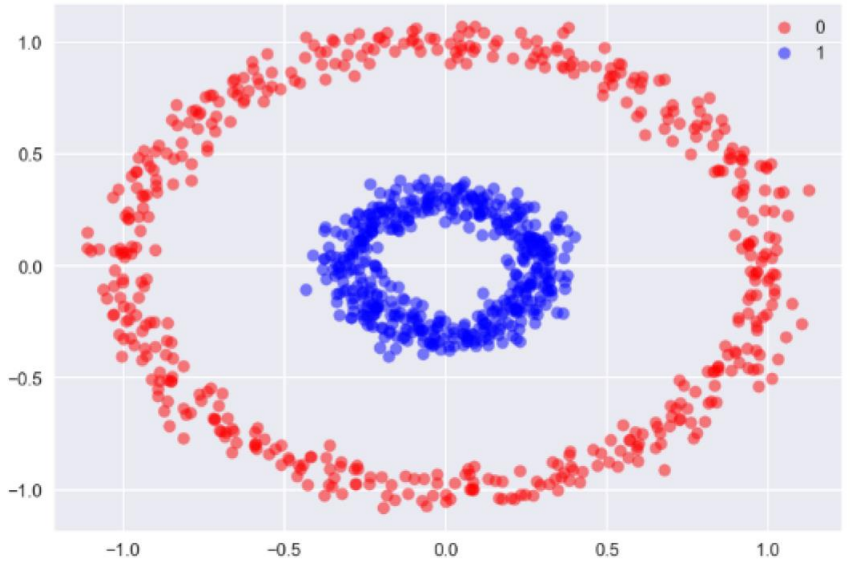


Example: Circles Data

We can also find examples where things go even worse:

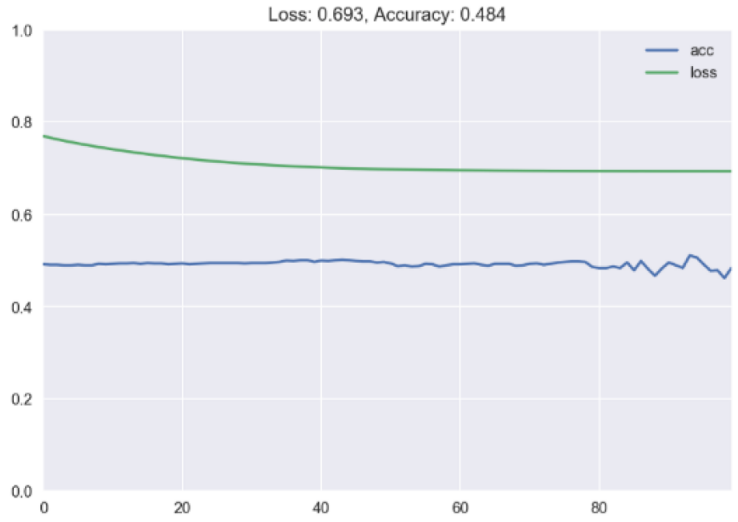
```
1  X, y = make_circles(n_samples=1000, noise=0.05, factor=0.3, random_state=0)
2  plot_data(X, y)
```

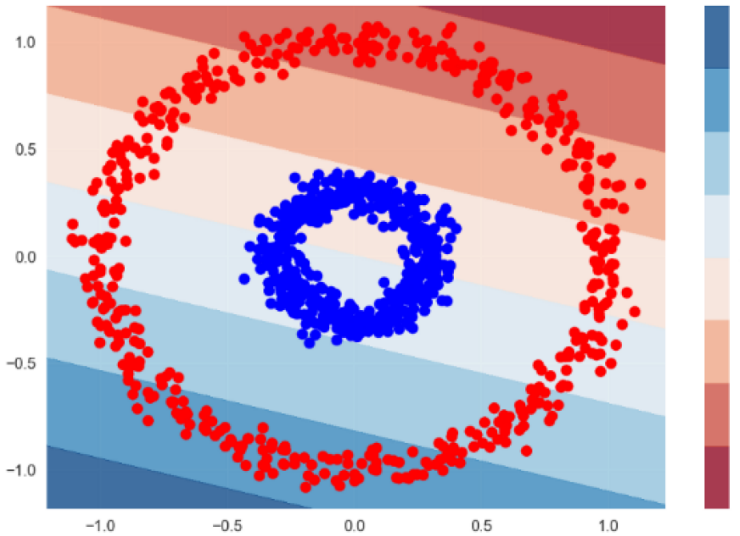
circles_1.py hosted with ❤ by GitHub view raw



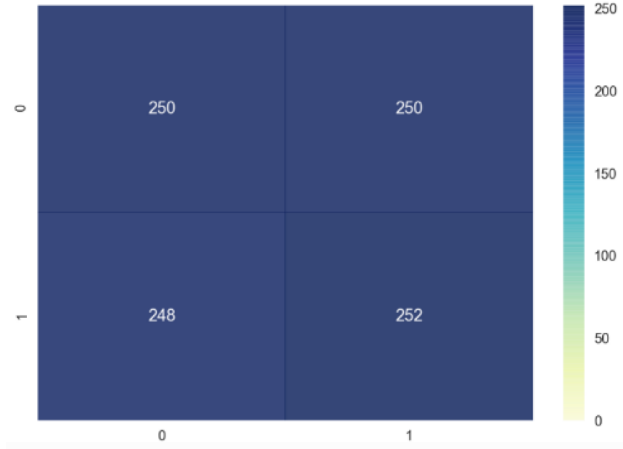
Example: Circles Data – Logistic Regression

We can also find examples where things go even worse:





	precision	recall	f1-score	support
0	0.50	0.50	0.50	500
1	0.50	0.50	0.50	500
avg / total	0.50	0.50	0.50	1000



Example: Two Moons Data – Deep ANN

Now let's redo this problematic examples using multiple layer ANNs:

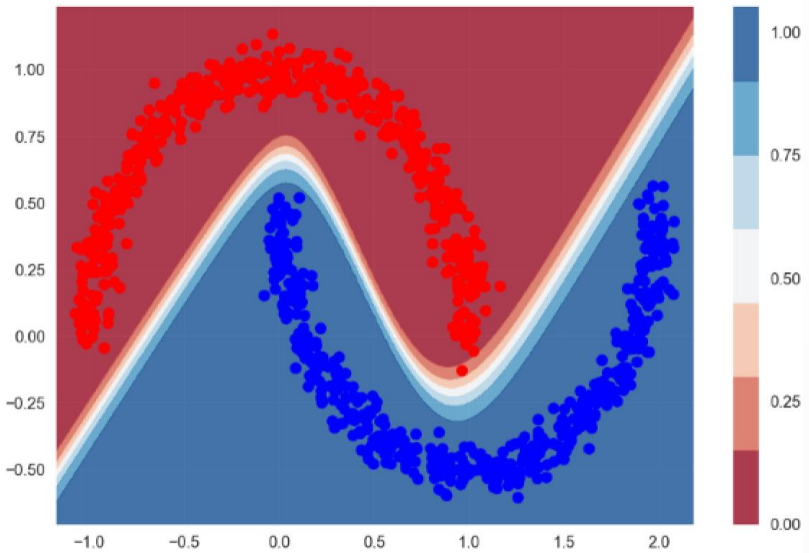
```python
X, y = make_moons(n_samples=1000, noise=0.05, random_state=0)

model = Sequential()
model.add(Dense(4, input_shape=(2,), activation='tanh'))
model.add(Dense(2, activation='tanh'))
model.add(Dense(1, activation='sigmoid'))

model.compile(Adam(lr=0.01), 'binary_crossentropy', metrics=['accuracy'])

history = model.fit(X, y, verbose=0, epochs=100)

plot_loss_accuracy(history)
```

moons_5.py hosted with ❤ by GitHub view raw

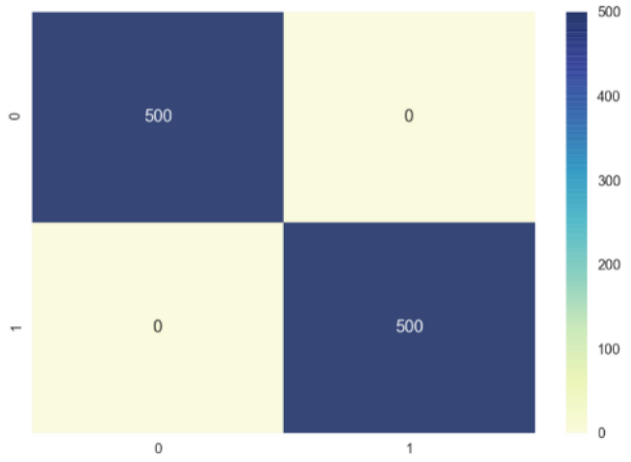
- Now we have 2 hidden layers with $\tanh$ activation function
- For the output layer we still use sigmoid since we are working on binary classification problems.

Example: Two Moons Data – Deep ANN

Although the model is not very deep, it still achieves full accuracy:

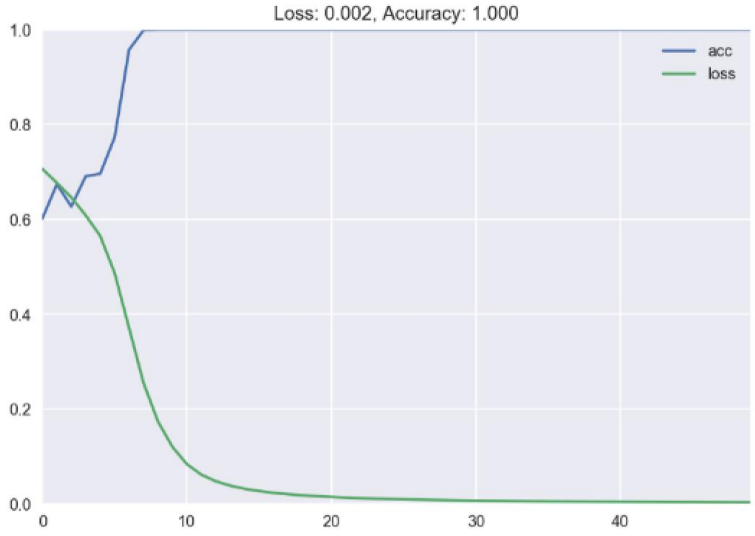


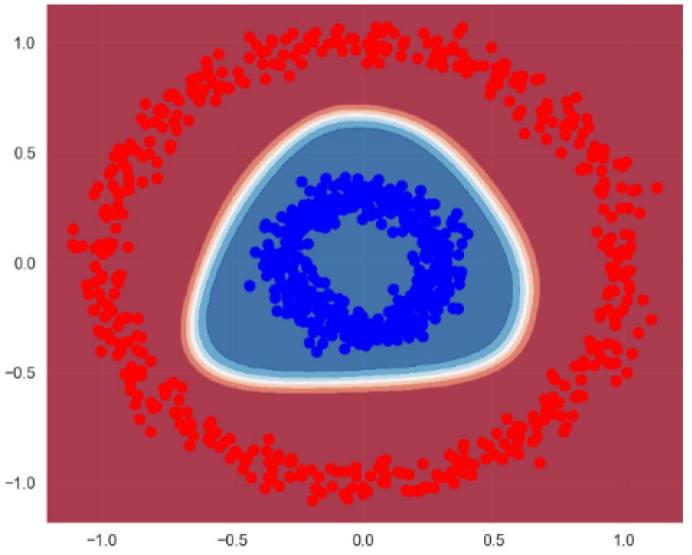
	precision	recall	f1-score	support
0	1.00	1.00	1.00	500
1	1.00	1.00	1.00	500
avg / total	1.00	1.00	1.00	1000



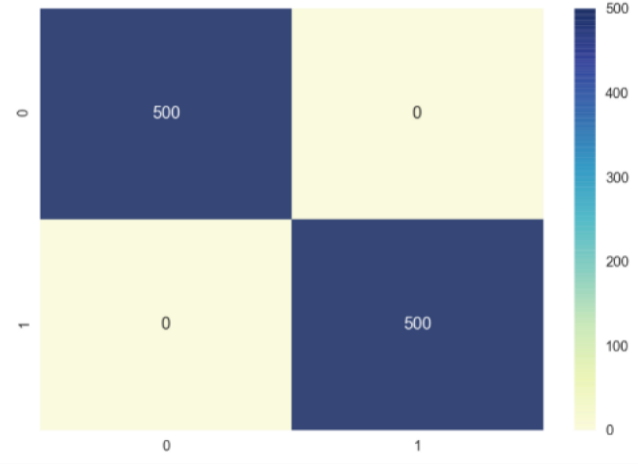
Example: Circles Data – Deep ANN

The same network also solves the circle problem:





	precision	recall	f1-score	support
0	1.00	1.00	1.00	500
1	1.00	1.00	1.00	500
avg / total	1.00	1.00	1.00	1000

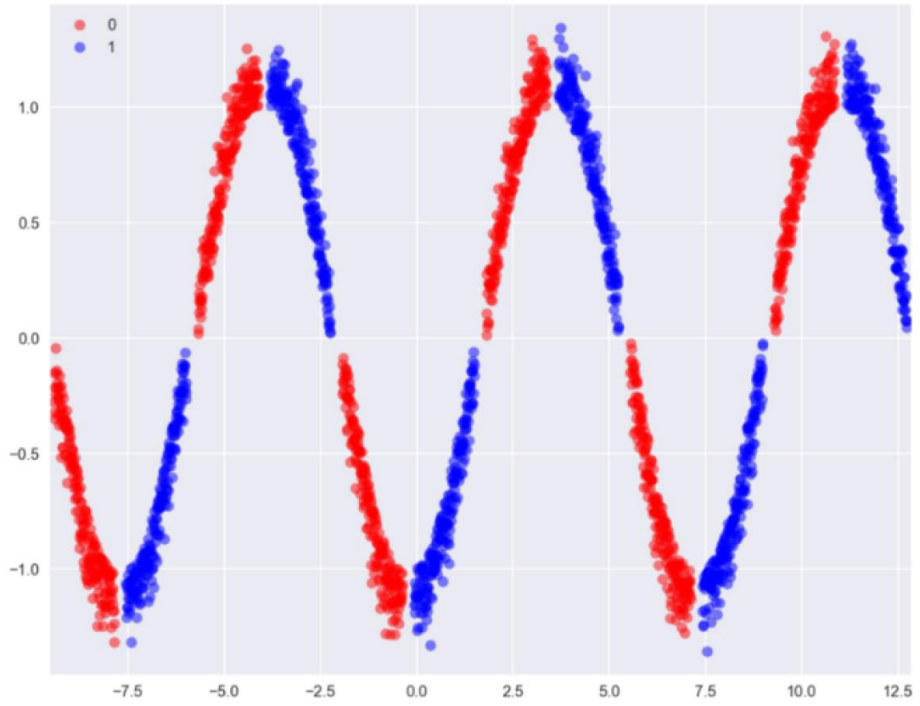


Example: Sin Wave Data – Deep ANN

Let's do something more interesting, a problem where a single connected decision boundary is not possible:

```
1  X, y = make_sine_wave()
2
3  plot_data(X, y, figsize=(10, 8))
```

sine_1.py hosted with ❤ by GitHub view raw



Example: Sin Wave Data – Deep ANN

We will need a deeper model for this problem

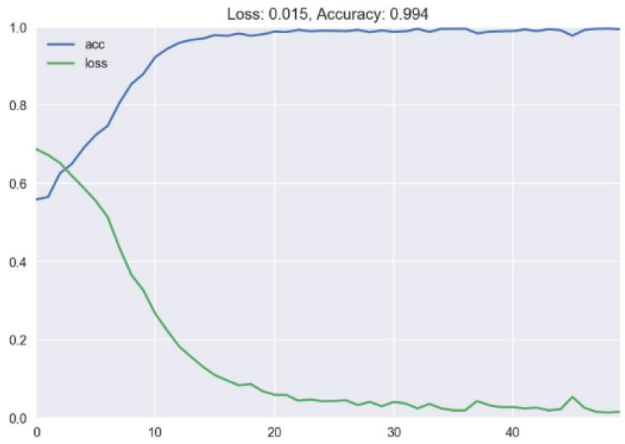
```
1  model = Sequential()
2  model.add(Dense(64, input_shape=(2,), activation='tanh'))
3  model.add(Dense(64, activation='tanh'))
4  model.add(Dense(64, activation='tanh'))
5  model.add(Dense(1, activation='sigmoid'))
6
7  model.compile('adam', 'binary_crossentropy', metrics=['accuracy'])
8
9  history = model.fit(X, y, verbose=0, epochs=50)
10
11 plot_loss_accuracy(history)
```

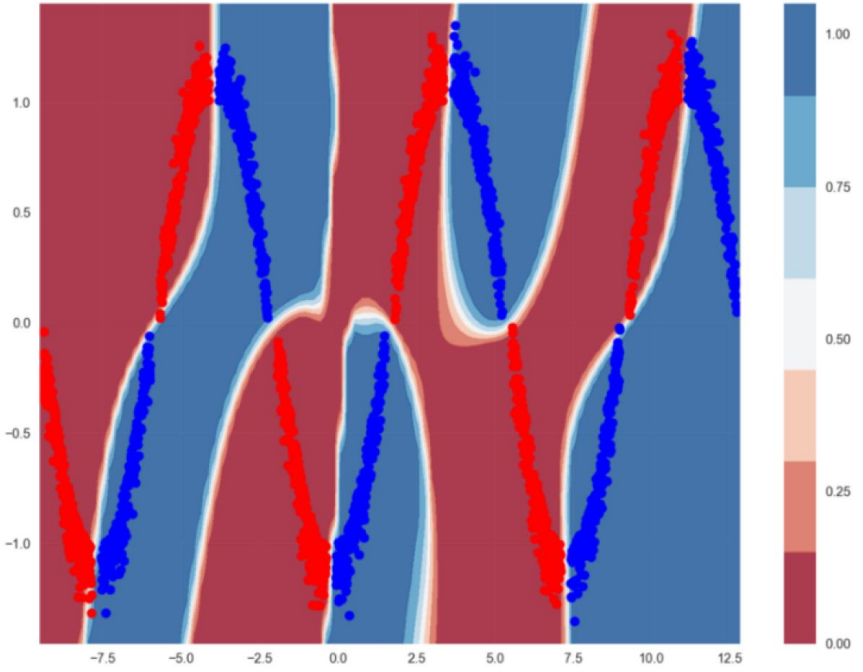
sine_2.py hosted with ❤ by GitHub view raw

- Setting number of layers and units for a problem is more of an art than technology, it is still an active area of research

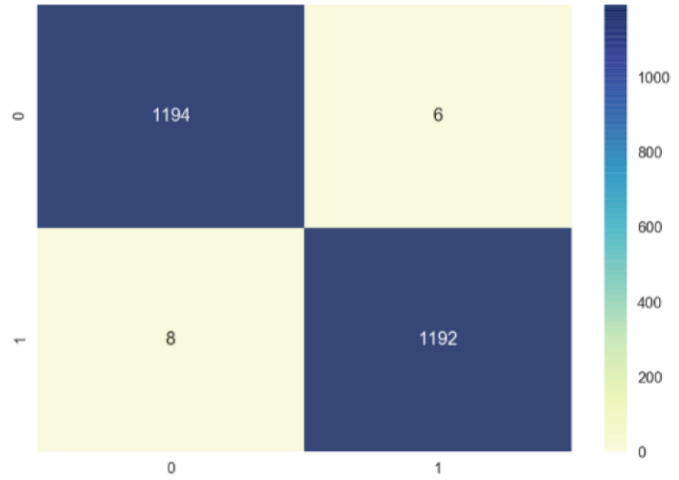
Example: Sin Wave Data – Deep ANN

Results for sine wave classification





	precision	recall	f1-score	support
0	0.99	0.99	0.99	1200
1	0.99	0.99	0.99	1200
avg / total	0.99	0.99	0.99	2400

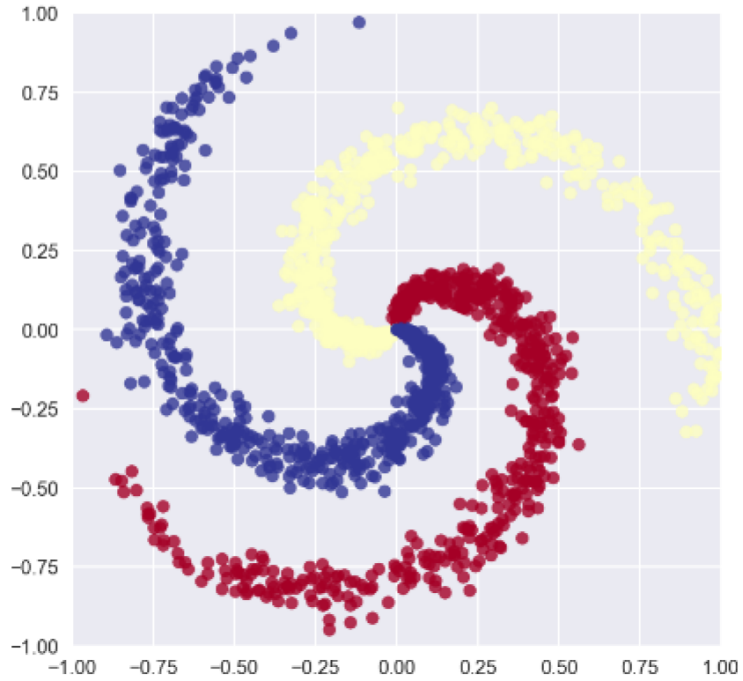


Example: Multi-class Classification

As a final toy example, let's attempt multi label classification

```
1  X, y = make_multiclass(K=3)
```

multiclass_1.py hosted with ❤ by GitHub view raw



Example: Multi-class Classification – Logistic Regression (Softmax)

First we generalize our linear classifier, the logistic regression algorithm, to handle multiple classes, this is known as the softmax classifier

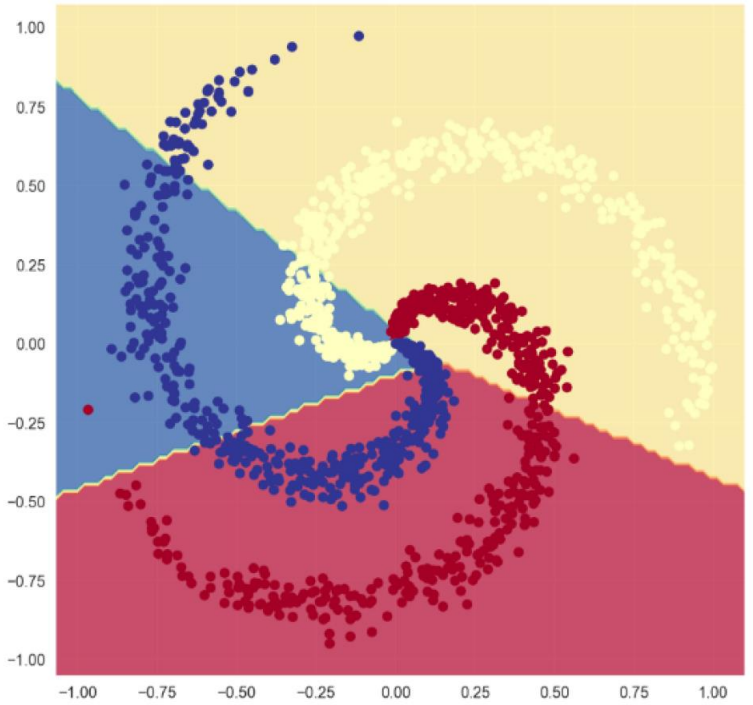
```
1  model = Sequential()
2  model.add(Dense(3, input_shape=(2,), activation='softmax'))
3
4  model.compile('adam', 'categorical_crossentropy', metrics=['accuracy'])
5
6  y_cat = to_categorical(y)
7  history = model.fit(X, y_cat, verbose=0, epochs=20)
8
9  plot_loss_accuracy(history)
```

multiclass_3.py hosted with ❤ by GitHub view raw

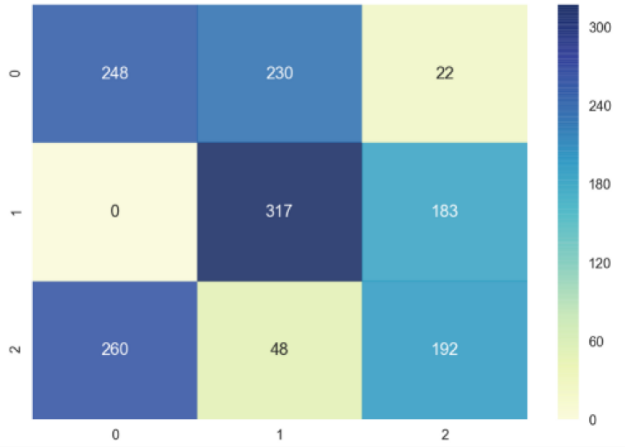
- Note that we need to convert our labels to categorical for this to work
- Output is a 3–vector, which can be interpreted as the probabilities of belonging to each class

Example: Multi-class Classification – Logistic Regression (Softmax)

Results for the linear classifier:



	precision	recall	f1-score	support
0.0	0.49	0.50	0.49	500
1.0	0.53	0.63	0.58	500
2.0	0.48	0.38	0.43	500
avg / total	0.50	0.50	0.50	1500



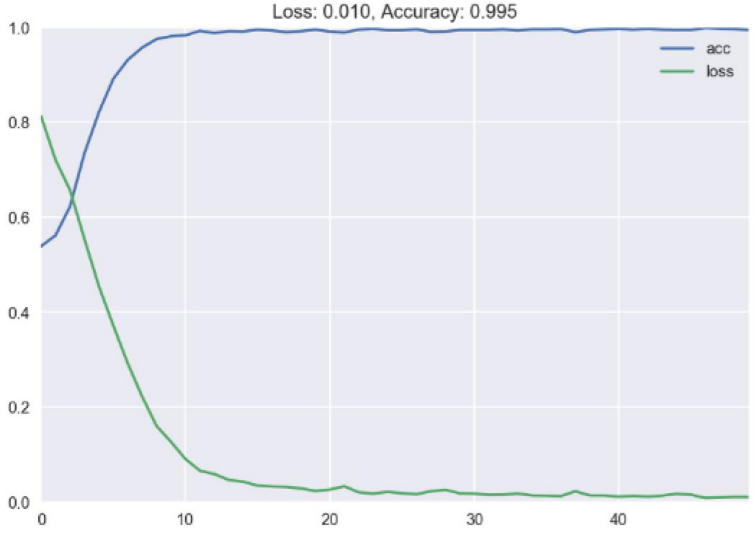
Example: Multi-class Classification – Deep ANN

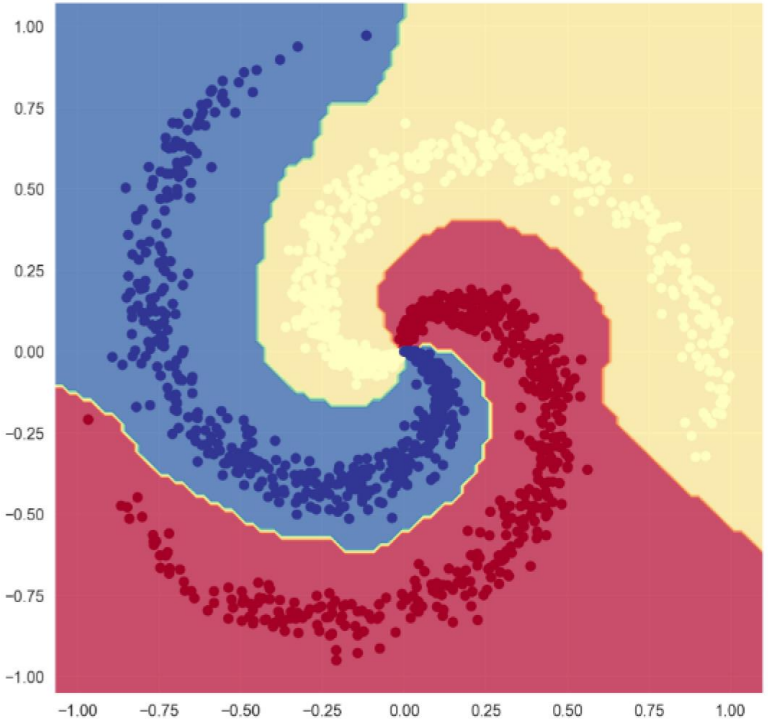
Now let's repeat the solution with a deep ANN:

```
model = Sequential()
model.add(Dense(64, input_shape=(2,), activation='tanh'))
model.add(Dense(32, activation='tanh'))
model.add(Dense(16, activation='tanh'))
model.add(Dense(3, activation='softmax'))

model.compile('adam', 'categorical_crossentropy', metrics=['accuracy'])

y_cat = to_categorical(y)
history = model.fit(X, y_cat, verbose=0, epochs=50)

plot_loss_accuracy(history)
```

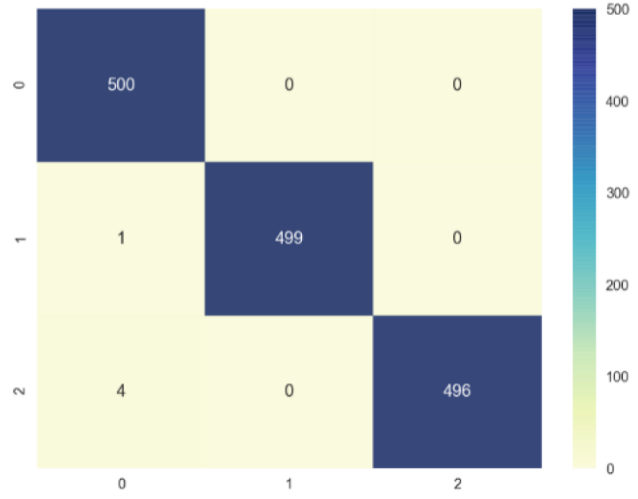
multiclass_4.py hosted with ❤ by GitHub view raw

Example: Multi-class Classification – Deep ANN





	precision	recall	f1-score	support
0.0	0.99	1.00	1.00	500
1.0	1.00	1.00	1.00	500
2.0	1.00	0.99	1.00	500
avg / total	1.00	1.00	1.00	1500



Example - Human Resources problem

Enough with toy datasets! Let's try attacking a real world problem. We will investigate the Human Resources problem (dataset taken form Kaggle).

- The objective is predicting if a given employee will leave the work based on his/her salary, number of years spent at the company etc.

	satisfaction_level	last_evaluation	number_project	average_monthly_hours	time_spend_company	Work_accident	promotion_last_5years	sales	salary	left
0	0.38	0.53	2	157	3	0	0	sales	low	1
1	0.80	0.86	5	262	6	0	0	sales	medium	1
2	0.11	0.88	7	272	4	0	0	sales	medium	1
3	0.72	0.87	5	223	5	0	0	sales	low	1
4	0.37	0.52	2	159	3	0	0	sales	low	1

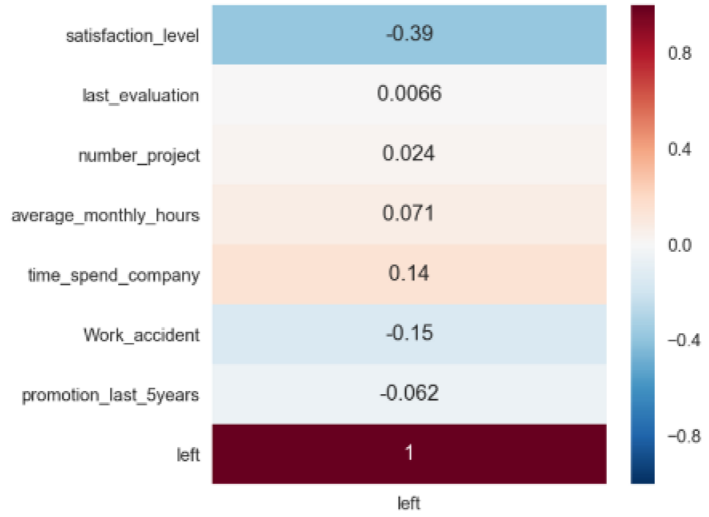
- This is a binary classification problem, since we only have two labels (left or stay)

Example - Human Resources problem

Let's do some data processing/cleaning. Here are the correlations between the label and other features

```
1  seaborn.heatmap(rawdf.corr()[['left']], annot=True, vmin=-1, vmax=1)
```

case_classification_1.py hosted with ❤ by GitHub view raw



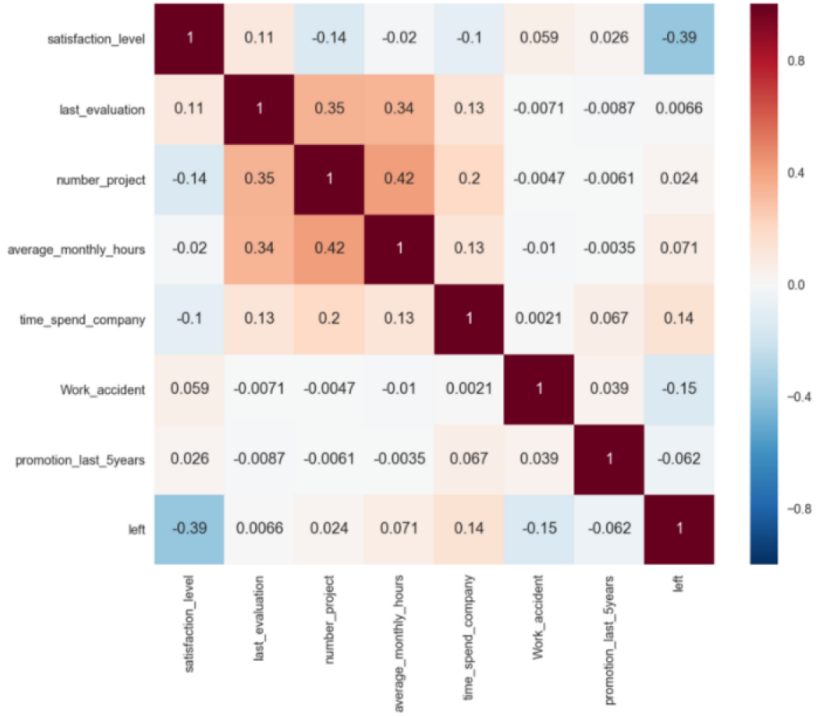
- As expected, there is a strong negative correlation between satisfaction level and leaving the company.

Example - Human Resources problem

What about all pairwise correlations?

```
1  seaborn.heatmap(rawdf.corr(), annot=True, square=True, vmin=-1, vmax=1)
```

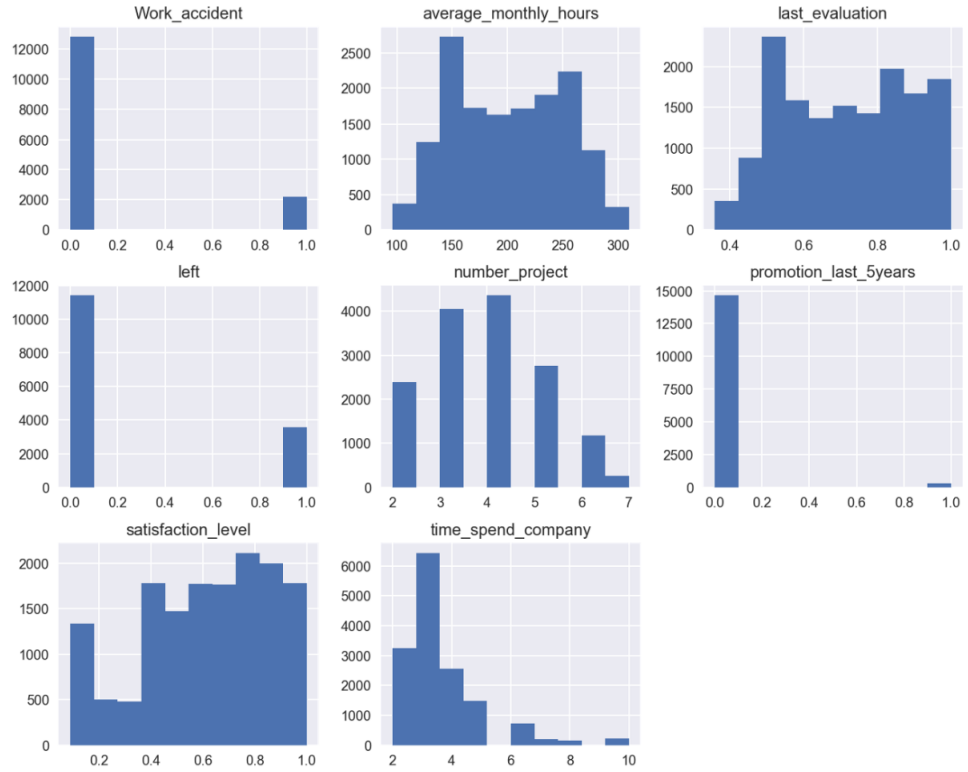
case_classification_2.py hosted with ♥ by GitHub view rav



- As expected, the monthly hours spent is positively correlated with number of projects

Example - Human Resources problem

Now lets look at the histograms to see which features might need normalization/scaling



- It seems like we need to normalize monthly hours, number of projects and time spend in company

Example - Human Resources problem

We can use standard scaler for normalization.

- We also need to convert our labels to categorical variables
- Then we separate our data as 30% for testing and 70% for training

```
ss = StandardScaler()
scale_features = ['average_monthly_hours', 'number_project', 'time_spend_company']
df[scale_features] = ss.fit_transform(df[scale_features])
```

case_classification_3.py hosted with ❤ by GitHub view raw

```
categorical_features = ['sales', 'salary']
df_cat = pd.get_dummies(df[categorical_features])
df = df.drop(categorical_features, axis=1)
df = pd.concat([df, df_cat], axis=1)
```

case_classification_4.py hosted with ❤ by GitHub view raw

```
X = df.drop('left', axis=1).values
y = df['left'].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
```

case_classification_5.py hosted with ❤ by GitHub view raw

Example - Human Resources problem

Now we are ready for training! Let's compare the classical logistic regression with a deep ANN:

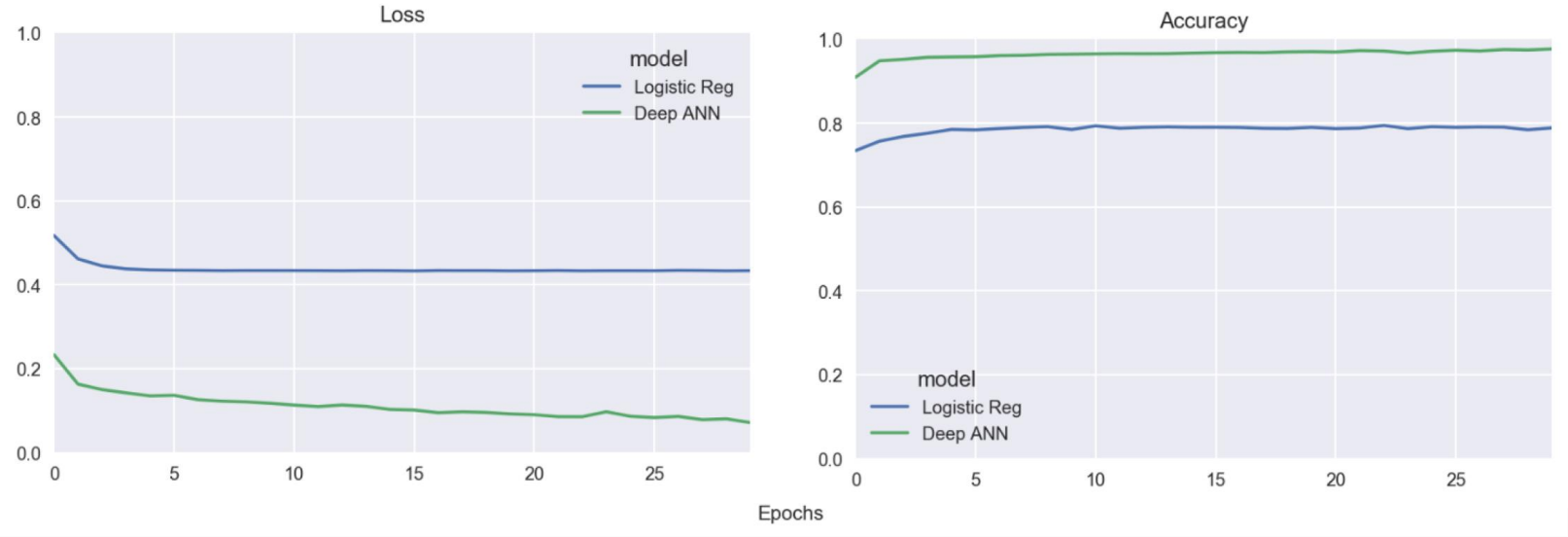
```python
deep_model = Sequential()
deep_model.add(Dense(64, input_shape=(X_train.shape[1],), activation='tanh'))
deep_model.add(Dense(16, activation='tanh'))
deep_model.add(Dense(1, activation='sigmoid'))

deep_model.compile(Adam(lr=0.01), 'binary_crossentropy', metrics=['accuracy'])

deep_history = deep_model.fit(X_train, y_train, verbose=0, epochs=30)
plot_loss_accuracy(deep_history)
```

case_classification_7.py hosted with ❤ by GitHub view raw

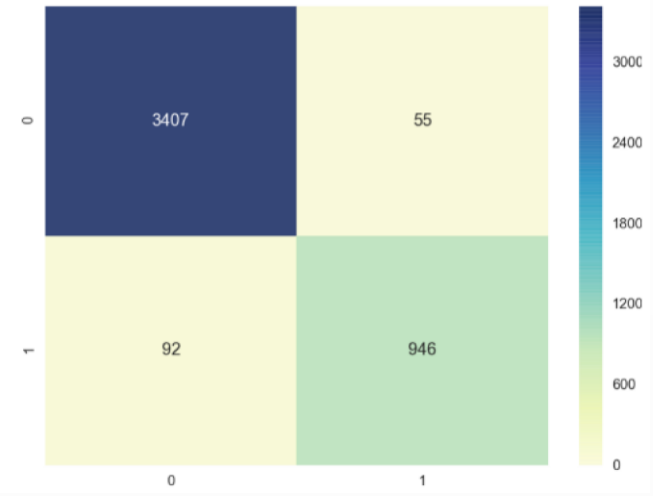
- This model is not that deep, but technically any model with more than 1 hidden layer can be considered as *deep*

Example - Human Resources problem

Training results look good! ANN smashes logistic regression



	precision	recall	f1-score	support
0	0.97	0.98	0.98	3462
1	0.95	0.91	0.93	1038
avg / total	0.97	0.97	0.97	4500



Confusion matrix computed with **test data**

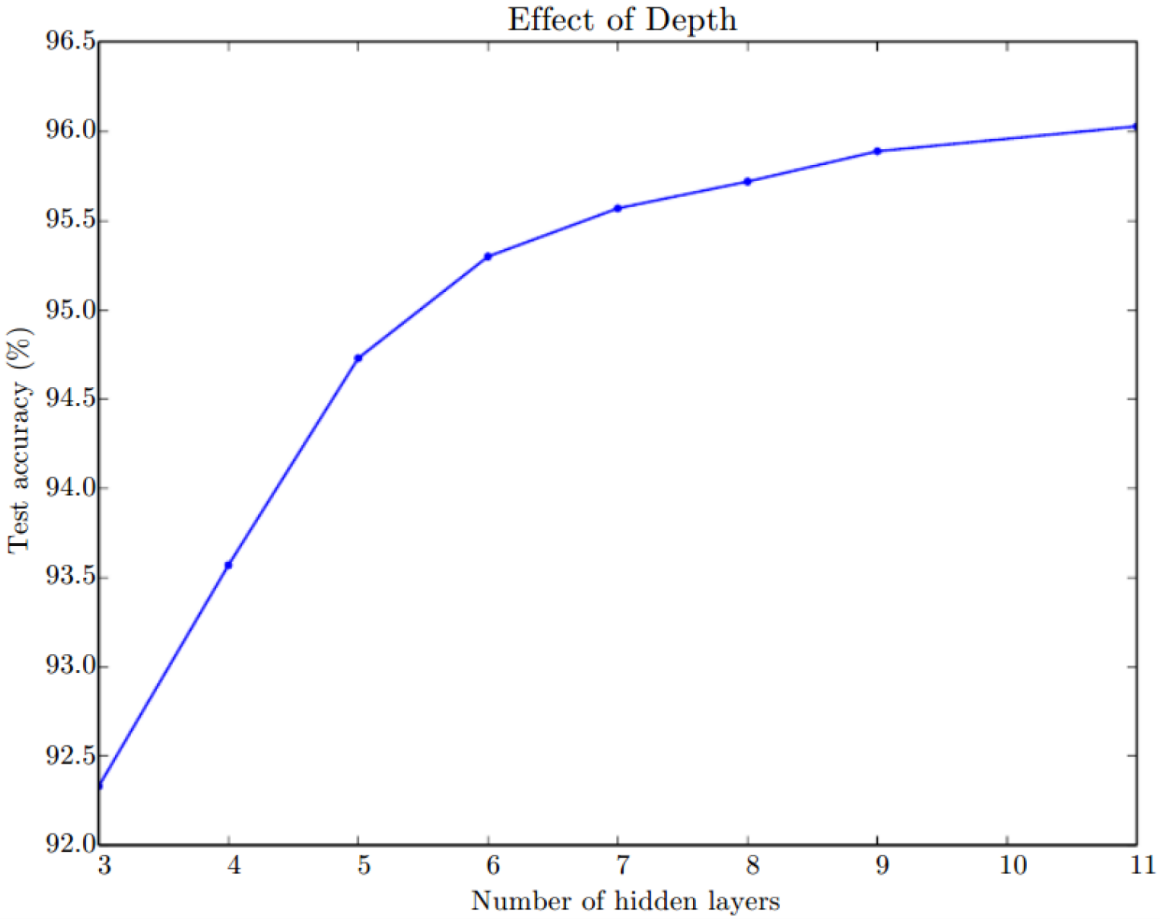
Improving the Performance

We can further improve the performance by doing the following:

- Train model for more epochs
- Use a different optimizer and/or loss function
- Increase the capacity by increasing the number of nodes per layer, such as instead of $64 - 16 - 1$, use $128 - 64 - 1$.
- Increase the capacity by increasing the number of layers, such as using $128 - 64 - 32 - 16 - 1$.

Architectural Considerations

Here are some plots that helps for gaining insight to model architecture design (taken from Goodfellow's book)



References

- A. Ng. Machine Learning, Lecture Notes.
- Arden Dertat's blog posts: https://medium.com/@ardendertat.
- I. Goodfellow, Y. Bengio and A. Courville, "Deep Learning", 2016.