Improving Neural Networks

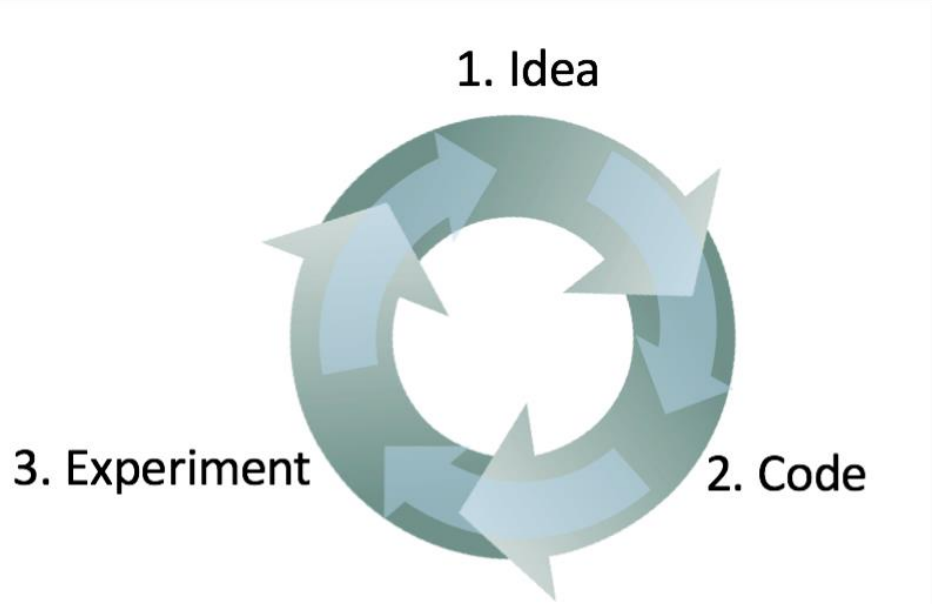
Prof. Giuseppe Serra

University of Udine

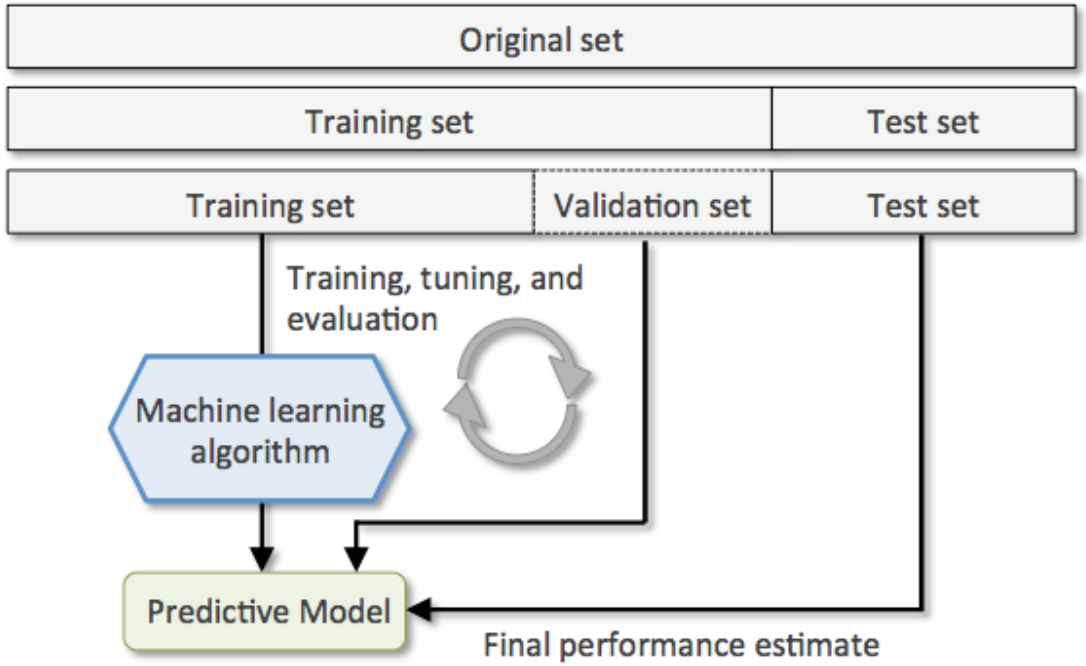
Applied Deep Networks is an iterative process

Hyperparameters:

- Learning rate (in our slide: α)
- Number of interaction (epochs)
- Number of hidden layers
- Number of hidden units in each hidden layers
- Activation functions



Train/Dev/Test Set

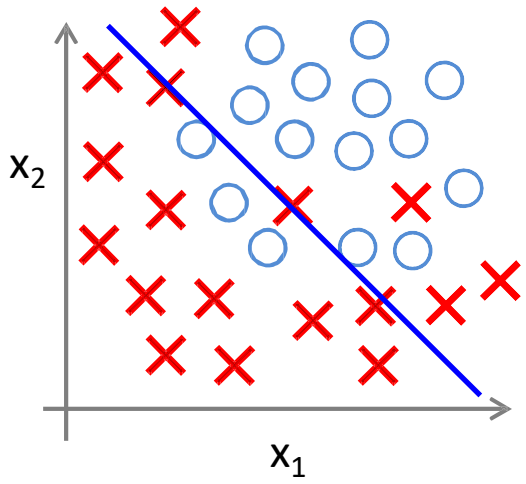


In the Big data era it's important the validation and the test sets should have the same distribution….

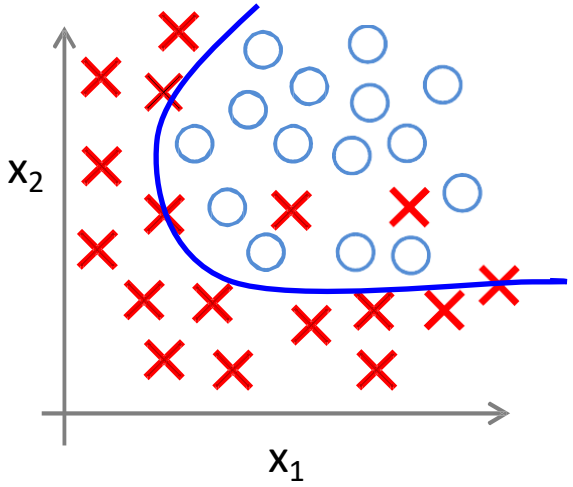
If you want to recognize cat photos taken using a smartphone, you can use in the training set (to increase the number of samples) "general" cat photos, but in the validation and test sets you should use images taken by a smartphone.

Underfit/Overfit

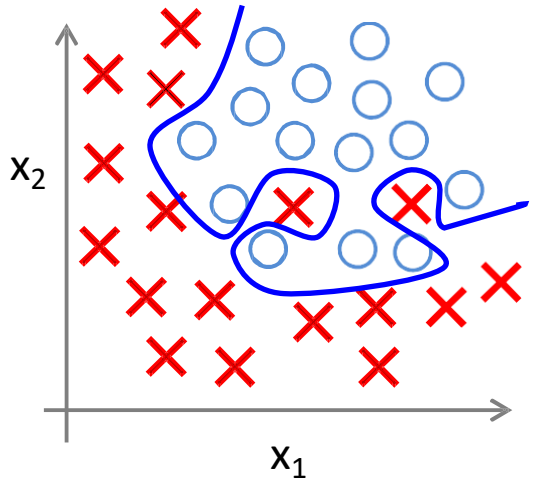
Let's start with 2D example



Underfit
(Model does not have capacity to fully learn the data)



Ideal fit



Overfit
(Too complex, extra parameters, does not generalize well)

Underfit/Overfit

Let's see a more complex, it is not possible to visualize it…..

Cat classification

Train set error:	1%	15%	0.5%
Dev set error:	11%	16%	1%
	Overfitting	Underfitting	Good*

*Suppose for human this problem is easy

Basic recipe for Machine Learning

To try to solve underfitting you can:

- Bigger Networks
- Training the neural networks for more epochs

To try to solve overfitting you can:

- More data;
- Regularization

Since the recipes are different for different scenarios, it is important to understand in which case you are.

Neural networks are quite flexible (with respect to “traditional” Machine Learning approaches) to reduce underfitting/overfitting problem.

Regularization

Regularization: Logistic regression

Minimization problem with regularization:

$$\min_{(w,b)} \; J(w,b)$$

$$J(w,b) = \frac{1}{m}\left[\sum_{i=1}^{m} \text{Cost}\left(h_{w,b}\left(x^{(i)}\right), y^{(i)}\right) + \frac{\lambda}{2}\sum_{j=1}^{n} w_j^2\right]$$

λ is called regularization parameter

Usually, b is ignored in the regularization process

Regularization: Neural Network

Minimization problem with regularization:

$$\min_{(W^{[1]},b^{[1]}W^{[2]},b^{[2]})} J(W^{[1]}, b^{[1]}W^{[2]}, b^{[2]})$$

$$\mathrm{J}\left(W^{[1]}, b^{[1]}W^{[2]}, b^{[2]}\right) = \frac{1}{m}\left[\sum_{i=1}^{m}\mathcal{L}\left(\hat{y}^{(i)}, y^{(i)}\right) + \frac{\lambda}{2}\sum_{l=1}^{L}\left\|W^{[l]}\right\|_F^2\right]$$

Where

λ is called regularization parameter;

l is the l-layer

$\left\|W^{[l]}\right\|_F^2 = \sum_{i=1}^{n^{[l-1]}}\sum_{j=1}^{n^{[l]}}\left(W_{ij}^{[l]}\right)^2$, where $W^{[l]} \in \mathbb{R}^{n^{[l]}\times n^{[l-1]}}$

Backpropagation with regularization term*:

$$\frac{\boldsymbol{d\mathcal{L}}}{\boldsymbol{d\,W^{[1]}}} = \frac{d\mathcal{L}}{d\,z^{[1]}}x^T + \frac{\lambda}{m}W^{[1]}$$

Red part is the "regularization effect"

Therefore, the gradient descent …

$$W^{[1]} := W^{[1]} - \alpha\left[\frac{d\mathcal{L}}{d\,z^{[1]}}x^T + \frac{\lambda}{m}W^{[1]}\right]$$

$$= W^{[1]} - \alpha\frac{\lambda}{m}W^{[1]} - \alpha\frac{d\mathcal{L}}{d\,z^{[1]}} = \left(1 - \alpha\frac{\lambda}{m}\right)\mathrm{W}^{[1]} - \alpha\frac{d\mathcal{L}}{d\,z^{[1]}}$$

Regularization in neural network is called "Weight Decay" too, because of this term: $W^{[1]} - \alpha\frac{\lambda}{m}W^{[1]}$

*considering the neural network we saw in previous classes.

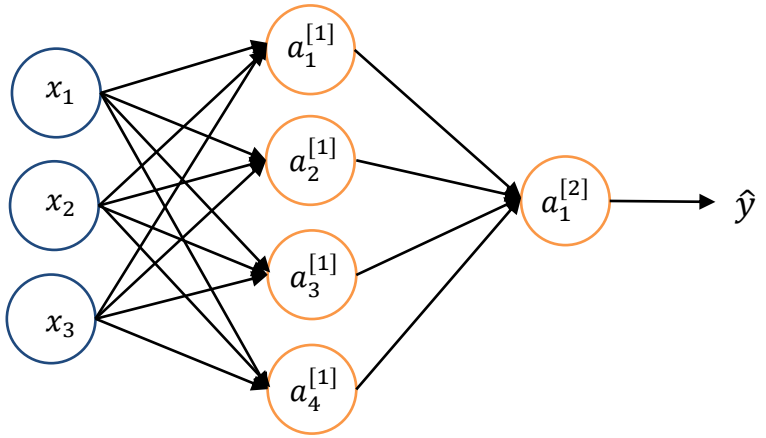
How does regularization prevent overfitting?

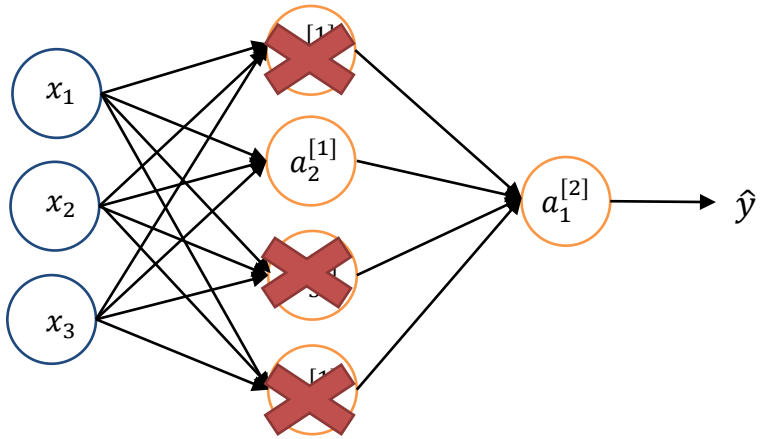
Minimization problem with regularization:

$$\min_{(W^{[1]},b^{[1]}W^{[2]},b^{[2]})} J(W^{[1]},b^{[1]}W^{[2]},b^{[2]})$$

$$J\left(W^{[1]},b^{[1]}W^{[2]},b^{[2]}\right) = \frac{1}{m}\left[\sum_{i=1}^{m}\mathcal{L}\left(\hat{y}^{(i)},y^{(i)}\right) + \frac{\lambda}{2}\sum_{l=1}^{L}\left\|W^{[l]}\right\|_F^2\right]$$

If λ is high -> $W^{[l]} \approx 0$ (neural networks becomes simpler)



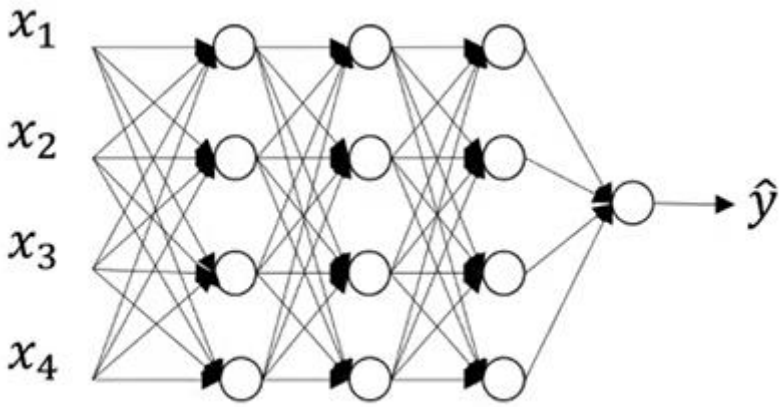


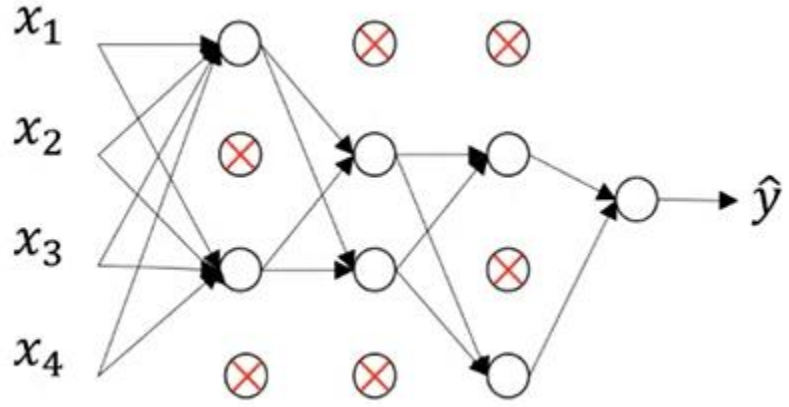
Dropout Regularization

During training, some number of layer outputs are randomly ignored or "*dropped out*."

Dropout has the effect of making the training process noisy, forcing nodes within a layer to probabilistically take on responsibility for the input. **The idea is to train your model using a smaller neural networks**.

Dropout is not used after training when making a prediction with the fit network.

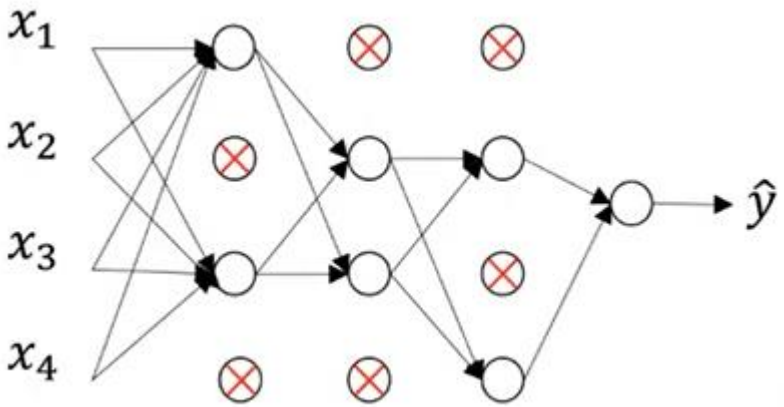




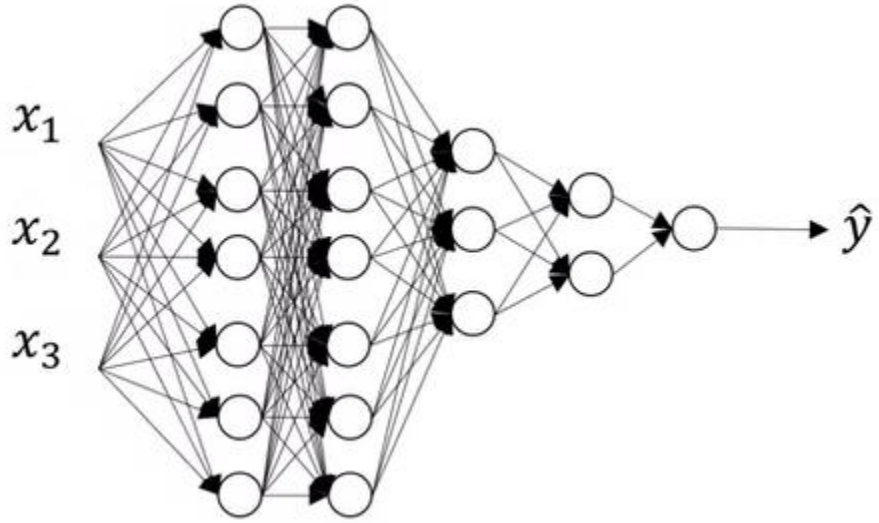
Dropout 0.5 0.5 0.5

Dropout Regularization

Intuition: A neuron/unit can't rely its prediction on any one node, so it must spread out weights



Some practical hints….



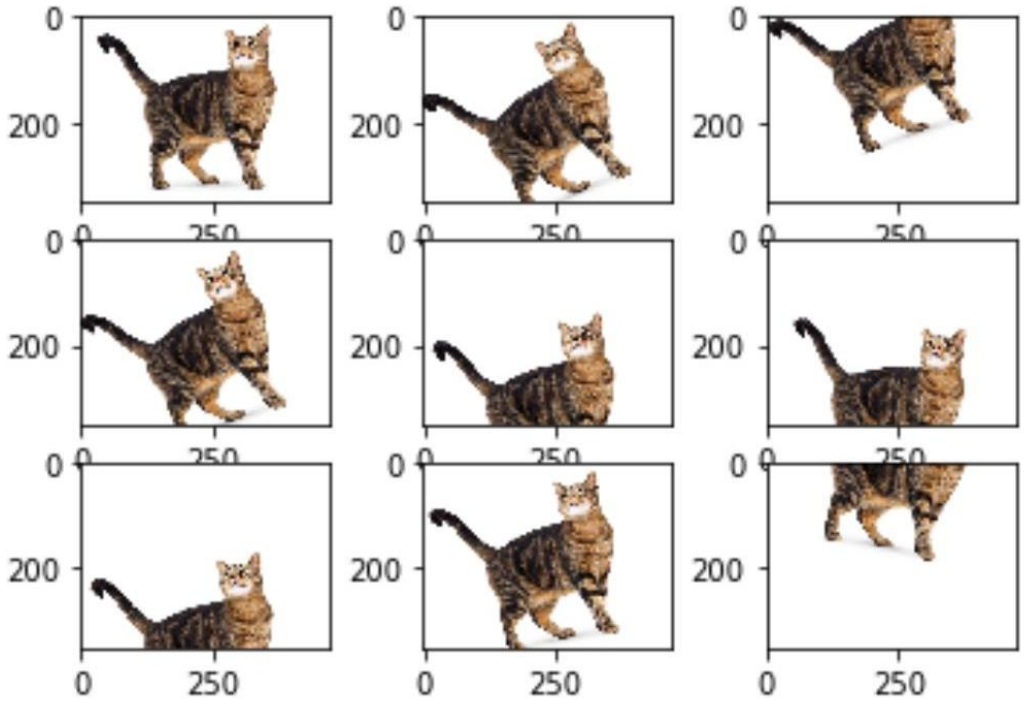
Dropout	**1.0**	**0.7**	**0.5**	**0.7**	**1.0**	**1.0**

Data Augmentation

Data augmentation in data analysis are techniques used to increase the amount of data by adding slightly modified copies of already existing data or newly created synthetic data from existing data.

Typical transformations: geometric transformations, flipping, color modification, cropping, rotation, noise injection and random erasing

Some researchers use videogames to try Deep Learning system (e.g. Self-driving car domain)

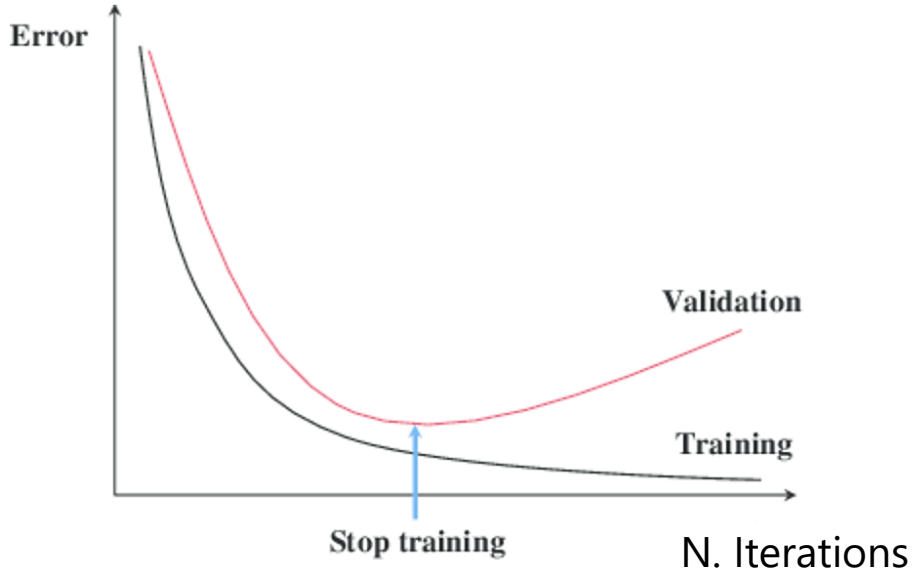


Early Stopping

When training a large network**, there will be a point during training when the model will stop generalizing and start learning the statistical noise in the training dataset**.

This overfitting of the training dataset will result in an increase in generalization error, making the model less useful at making predictions on new data.

One approach to solving this problem is to **treat the number of training epochs as a hyperparameter** and train the model multiple times with different values, then **select the number of epochs that result in the best performance** on the train or a holdout validation dataset.



Normalizzation

Mean Normalization

Mean Normalization transforms data to have a mean of zero and a standard deviation of 1.

It's the preferred normalization for Neural Networks.

Given a feature x_i the mapped feature is $x' = \frac{x_i - \mu_i}{\sigma_i}$

where μ_i, σ_i are the mean and standard deviation computed on the training set.

Mean

$$\mu_i = \frac{1}{m}\sum_{i=1}^{m} x_i$$

Standard Deviation

$\sigma_i = \sqrt{\frac{1}{m-1}\sum_{i=1}^{m}(x_i - \mu_i)^2}$ where m is the number of samples of the training set

MinMax Normalization

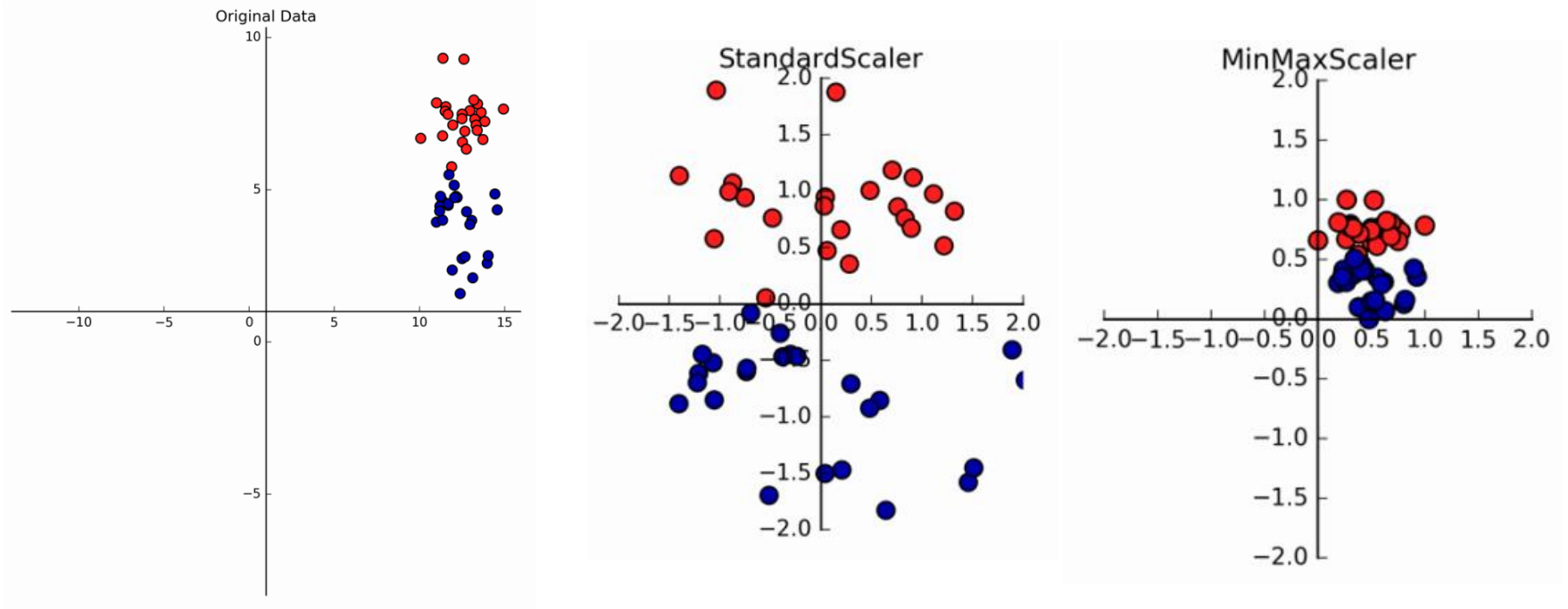
The MinMax normalization shifts the data such that all features are exactly between 0 and 1.

$$x'_i = \frac{x_i - \min(x)}{\max(x) - \min(x)}$$

Where x_i is the original value, x'_i is the normalized value.

Mean and MinMax Normalization Comparison

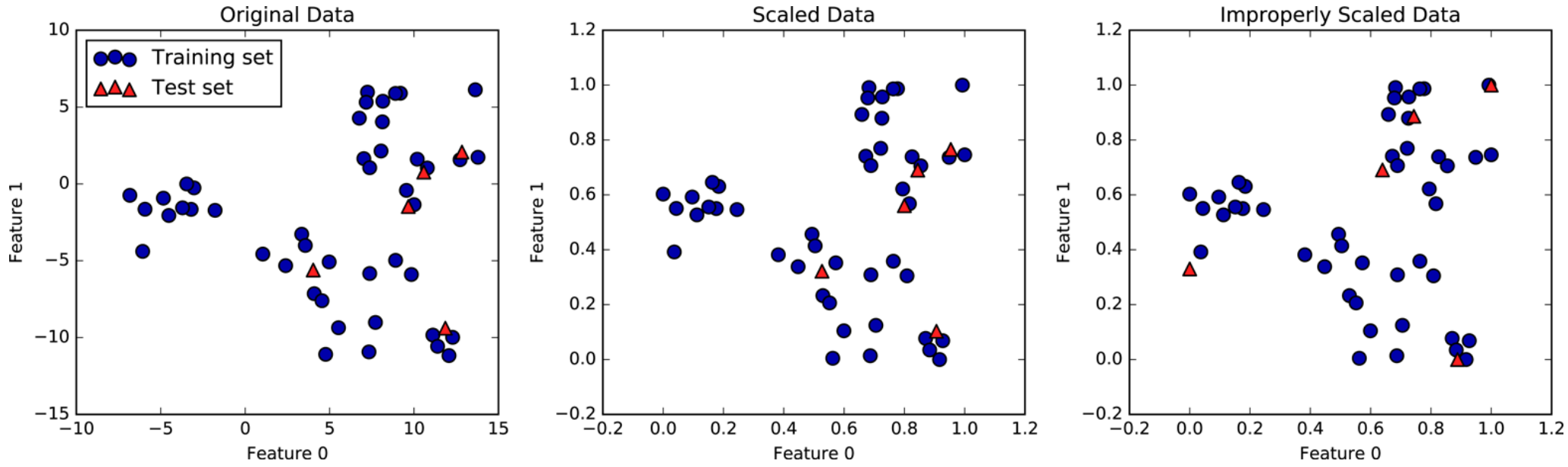
Figure shows a two-class classification dataset with two features and a comparison between Mean Normalization (StandardScaler) and MinMax Normalization (MinMaxScaler)



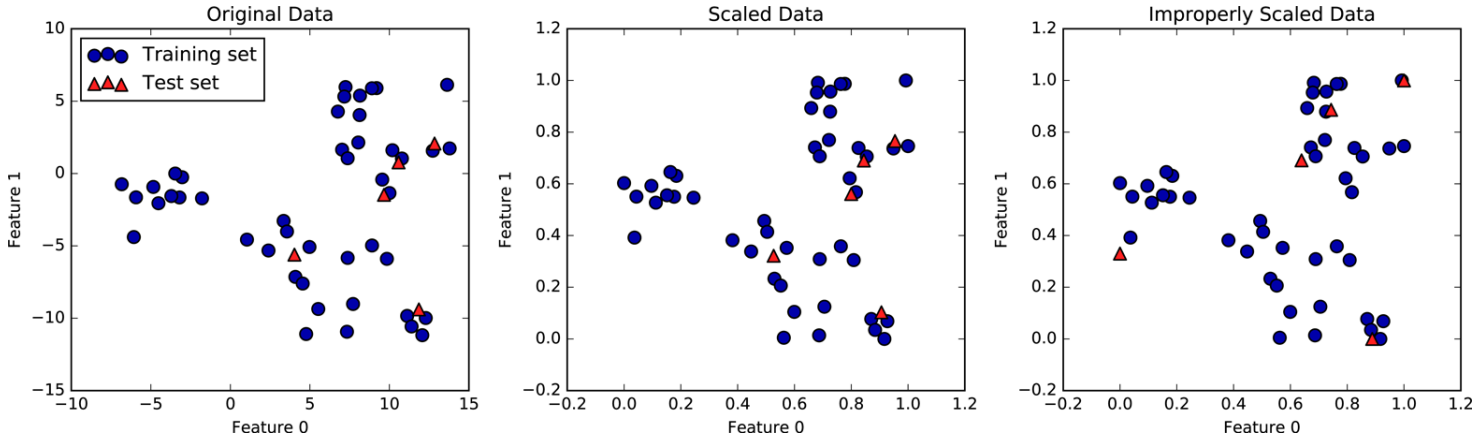
Scaling Training and Test Data

It is important to **apply the same transformation to the training set and the test set** for the supervised model to work on the test set.

The figure illustrates what would happen if we were to use the minimum and range of the test set instead:



Scaling Training and Test Data … more details



The third panel shows what would happen if we scaled the training set and test set separately**. In this case, the minimum and maximum feature values for both the training and the test set are 0 and 1. But now the dataset looks different**. The test points moved incongruously to the training set, as they were scaled differently. We changed the arrangement of the data in an arbitrary way. Clearly this is not what we want to do.

Scaling Training and Test Data the Same Way

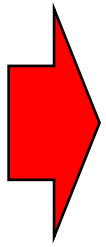
A synthetic example: we scaled the training set and test set with same transformation (parameter transformation inferred by the training set) - **Correct**

Training Set

X	Y
0	0
5	10
10	20

Test Set

X	Y
10	20
15	30
20	40



MinMax feature Normalization

Training Set

X-Norm	Y
0	0
0.5	10
1	20

Test Set

X-Norm	Y
1	20
1.5	30
2	40

Scaling Training and Test Data the Same Way

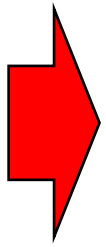
A synthetic example: we scaled the training set and test set with same transformation (parameter transformation inferred by the training set) - **Incorrect**

Training Set

X	Y
0	0
5	10
10	20

Test Set

X	Y
10	20
15	30
20	40



MinMax feature Normalization

Training Set

X-Norm	Y
0	0
0.5	10
1	20

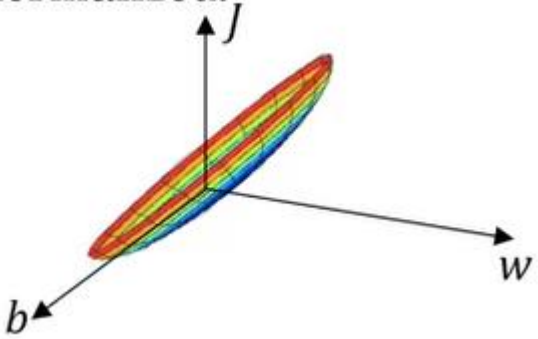
Test Set

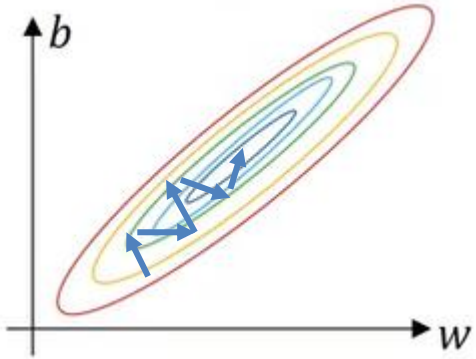
X-Norm	Y
0	20
0.5	30
1	40

Why normalize inputs?

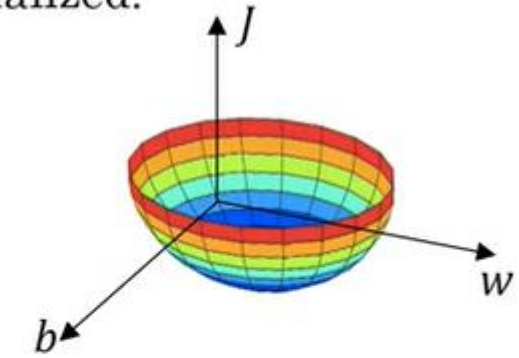
$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}(\hat{y}^{(i)}, y^{(i)})$$

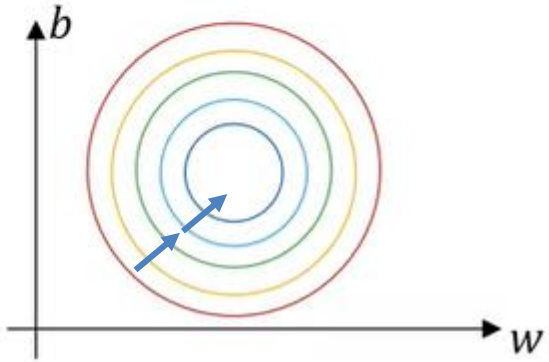
Unnormalized:





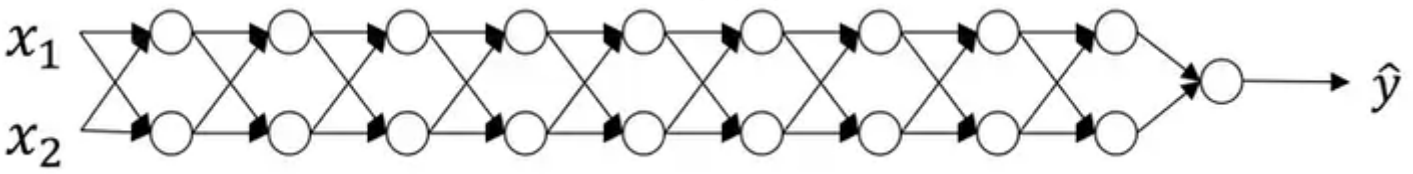
Normalized:





Vanishing/Exploding Gradients

Vanishing/Exploding Gradients



Let's suppose $g^{[i]}(z) = z$ and $b^{[i]} = 0$ for each layers

$\hat{y} = W^{[L]}W^{[L-1]}W^{[L-2]} \dots W^{[2]}W^{[1]}x$

Let's suppose $W^{[L]} = W^{[L-1]} = \cdots = W^{[1]} = \begin{bmatrix} 1.5 & 0 \\ 0 & 1.5 \end{bmatrix}$ then $\hat{y} = \begin{bmatrix} 1.5 & 0 \\ 0 & 1.5 \end{bmatrix}^L x$ → in this case values will be very large

In the other hand, if you have $W^{[L]} = W^{[L-1]} = \cdots = W^{[1]} = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}$ → values will be very small

The idea is to set randomly the weights around to 1.