Deep learning

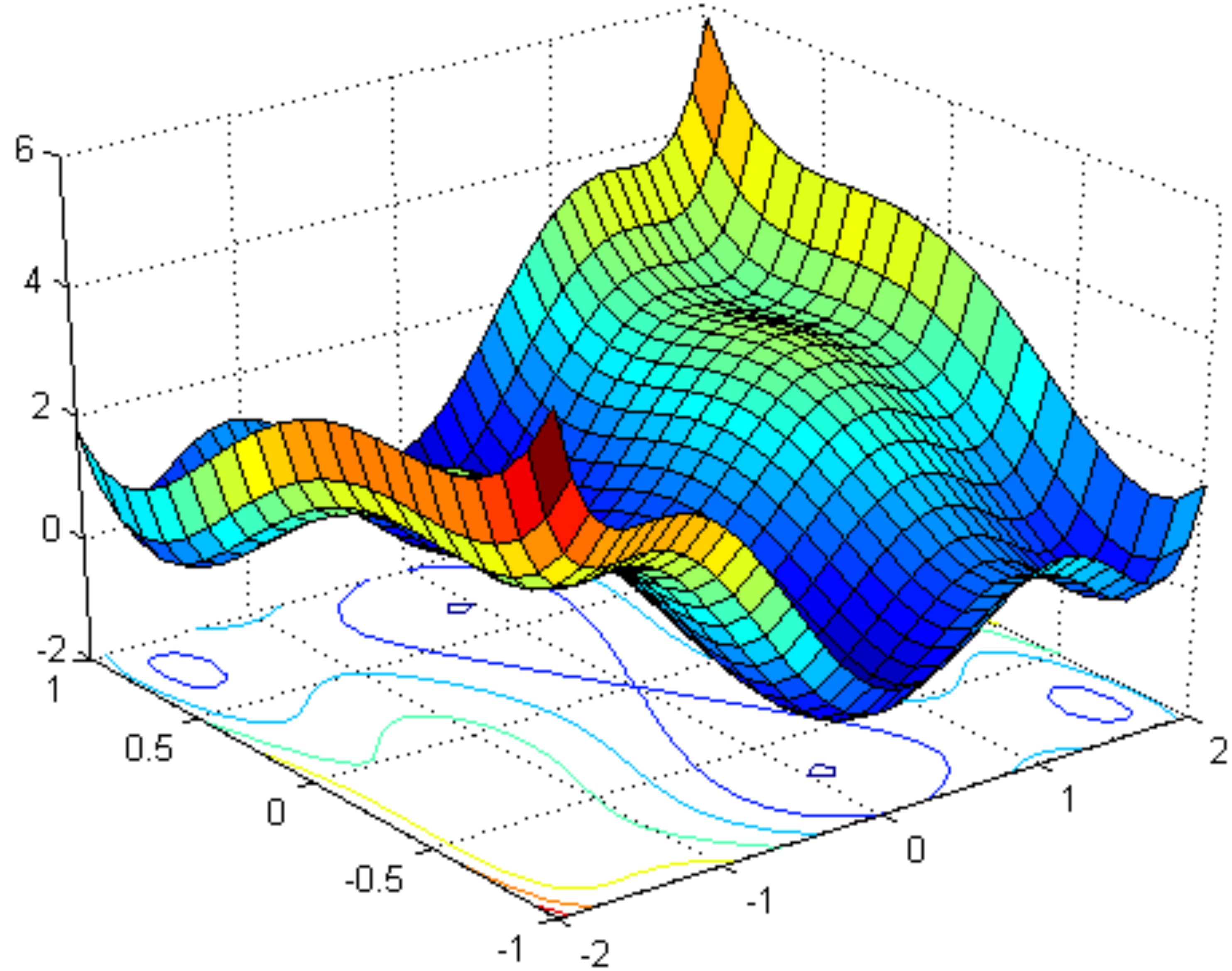
Classification & Regularisation

Announcements

- Matthijs had a daughter!
  - Haukur will give the lectures for the next weeks, in English.
  - Unexpected, so bare with me for the next few weeks.
- About the environment
  - The environment should be more stable and faster, and we have a quick fix to solve the issue when it arises.
  - You do not need to use it, and you can install the libraries by yourself or run the docker image. Talk to me in the break / after class for this.
  - We still believe it is the best option, especially for heavy processing.

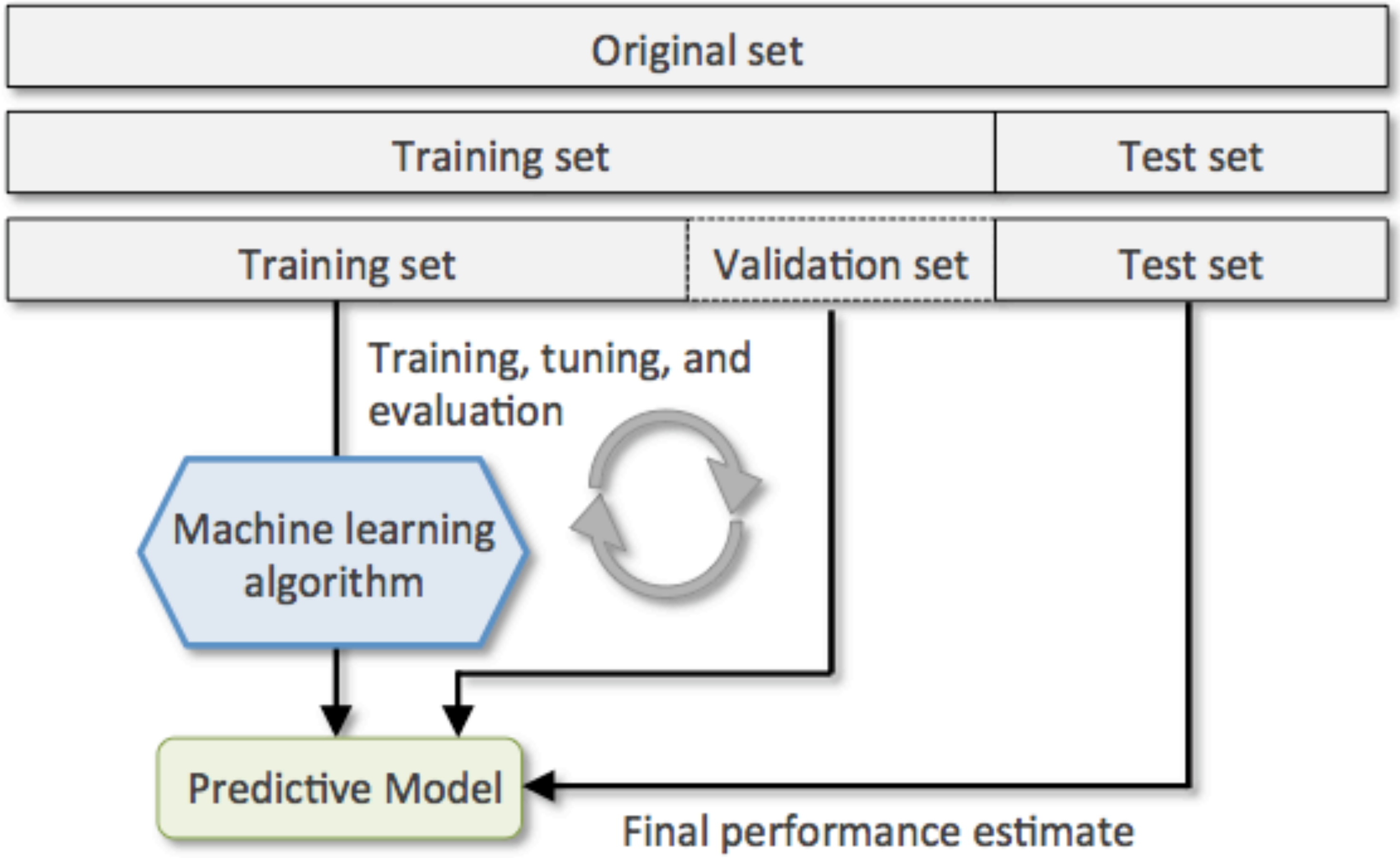
Recap / Questions?

- How to train a neural network.
  - Loss function
  - Gradient descent



Recap / Questions?

- How to evaluate the performance of a network
  - Over- and under-fitting
  - Hyperparameters



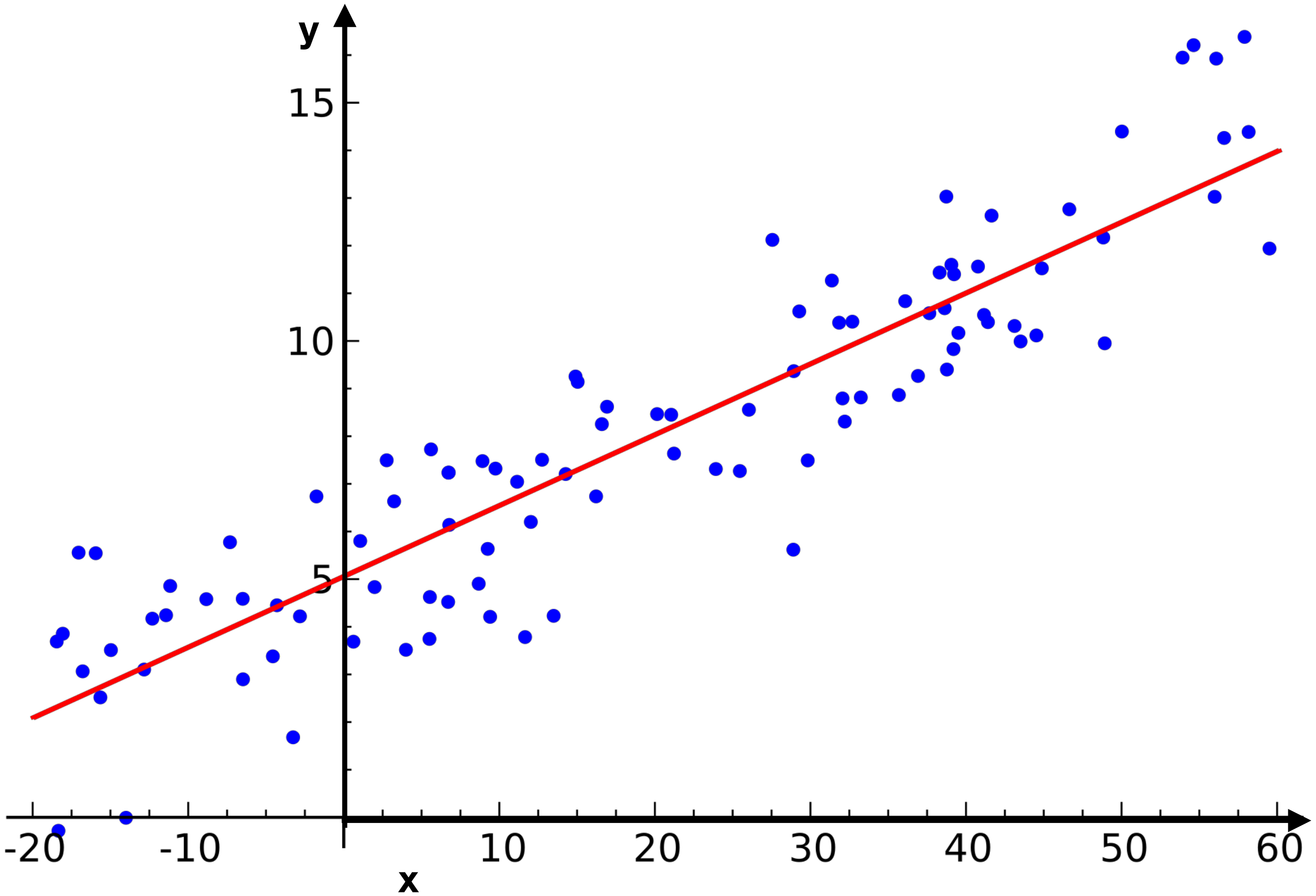
Overview

Today we will cover

- Machine learning tasks
  - Regression
  - Binary classification
  - Multi-class classification
- Improving networks
  - Preventing overfitting = Regularisation, dropout
  - Speeding up learning = Advanced optimisers, batch normalisation

Regression

- In **regression** the output is a single value.
  - Predicting house price $ given house size

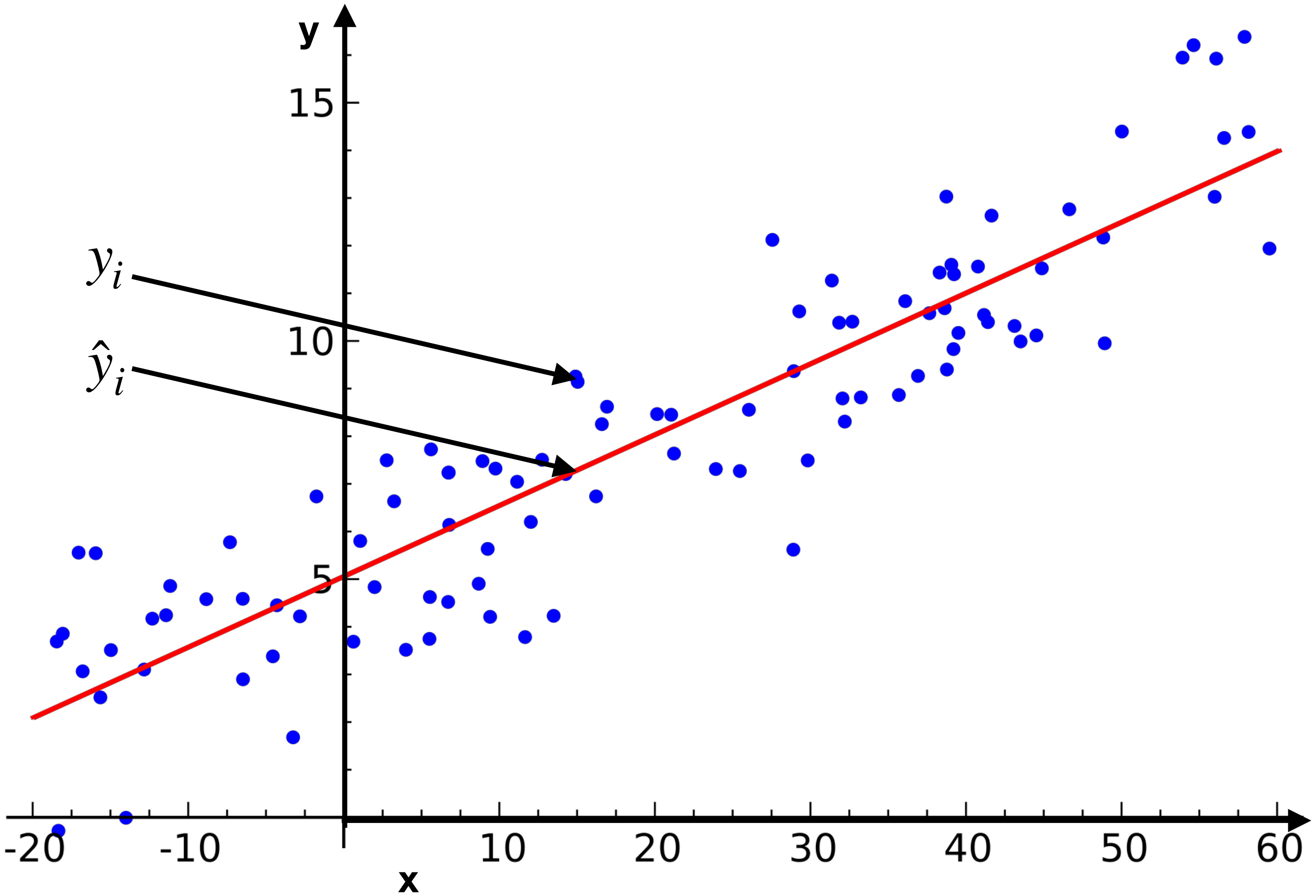


Source: https://en.wikipedia.org/wiki/Regression_analysis

Regression

- In **regression** the output is a single value.
  - Predicting house price $ given house size

$$\hat{y} = wx + b$$



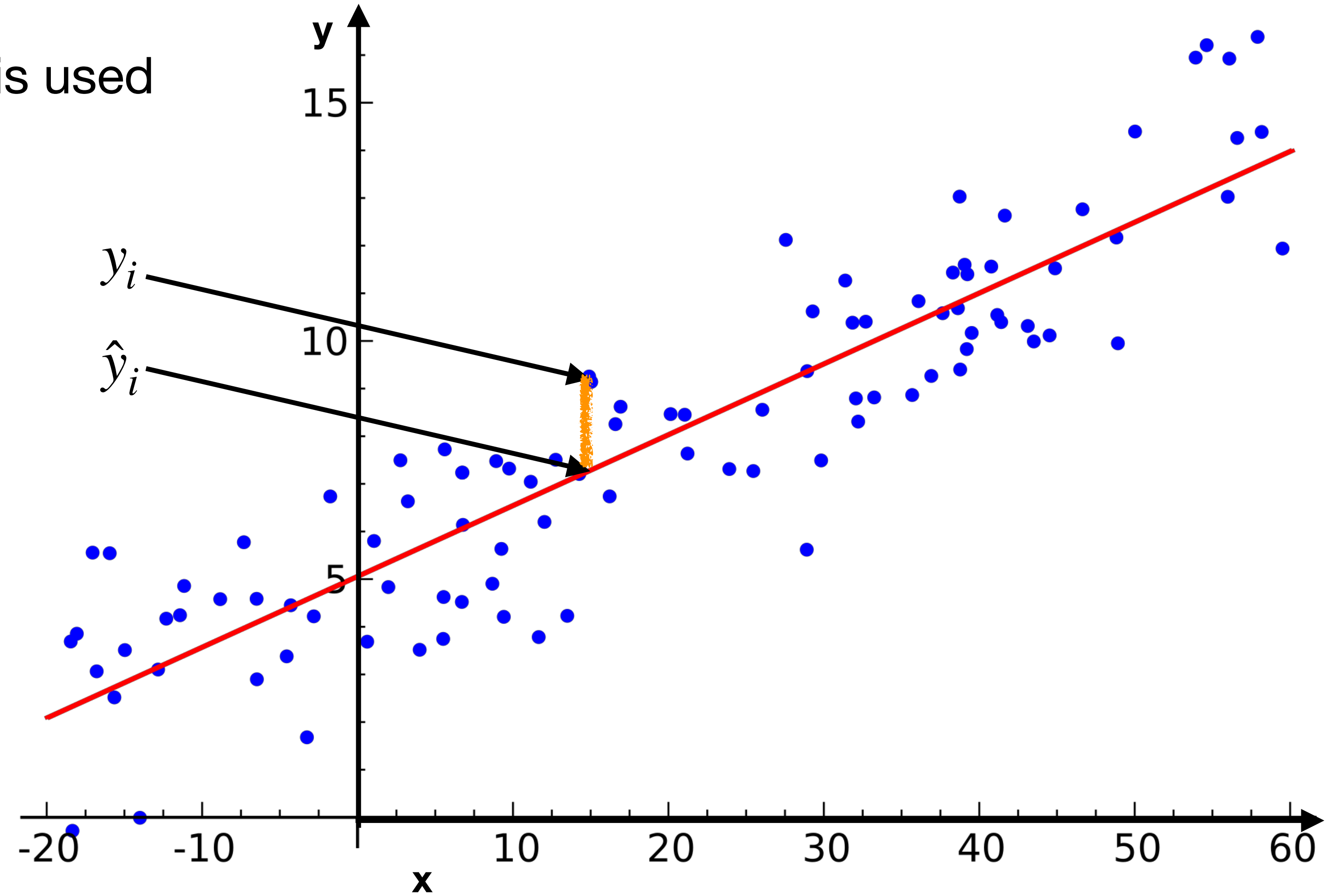
Source: https://en.wikipedia.org/wiki/Regression_analysis

Regression

- In **regression** the output is a single value.
  - Predicting house price $ given house size.
- Usually Mean Square Error (**MSE**) is used as the loss function.

$$\hat{y} = wx + b$$

$$l_{MSE}(\hat{y}, y) = (y - \hat{y})^2$$



Source:
https://en.wikipedia.org/wiki/Regression_analysis

Regression

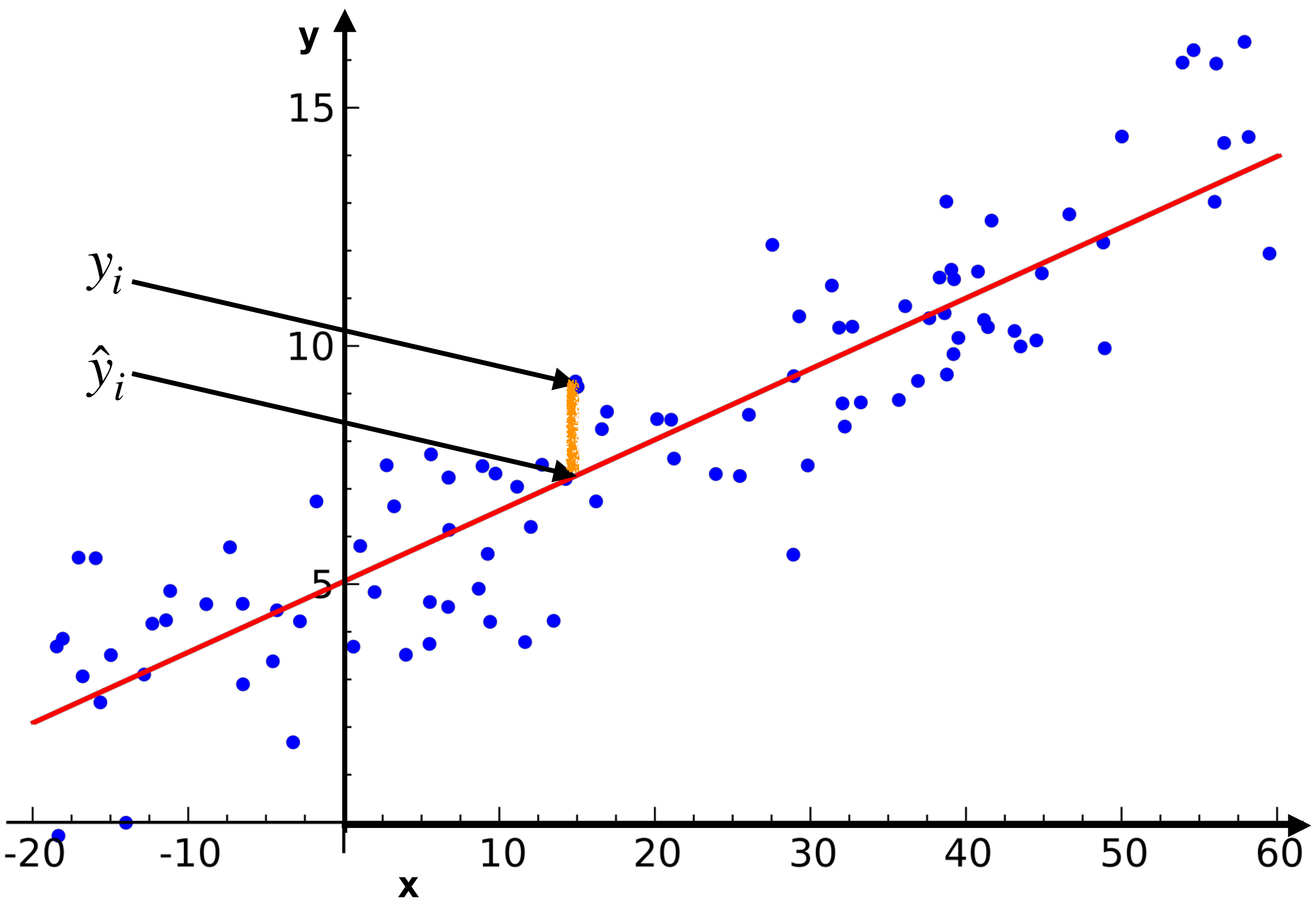
MSE

- In **regression** the output is a single scalar.
    - Predicting house price $ given house size.
- Usually Mean Square Error (**MSE**) is used as the **loss** function.

$$\hat{y} = wx + b$$

$$l_{MSE}(\hat{y}, y) = (y - \hat{y})^2$$

- We then seek to minimise this loss.
- We can represent our model as a line through our data.



Source: https://en.wikipedia.org/wiki/Regression_analysis

Regression

MSE

- When we have more dimensions for our input we will get a hyperplane.

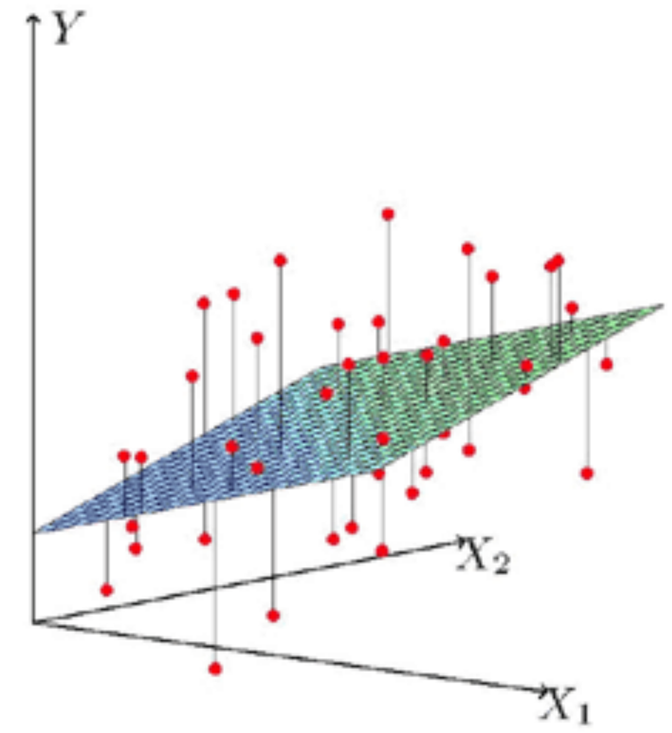


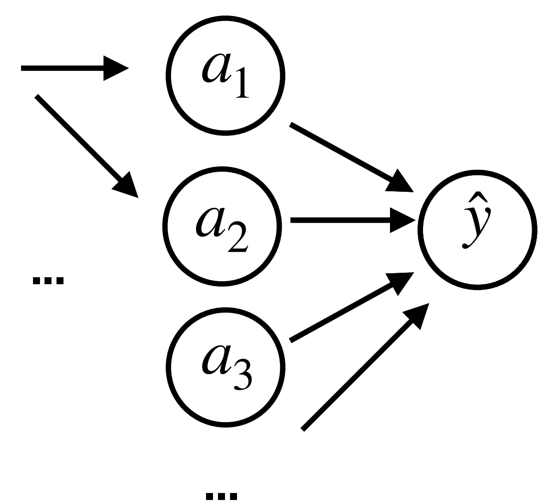
Figure 3.1: *Linear least squares fitting with $X \in \mathbb{R}^2$. We seek the linear function of X that minimizes the sum of squared residuals from Y.*

Source: https://onlinecourses.science.psu.edu/stat508/book/export/html/641

Regression

MSE in NN

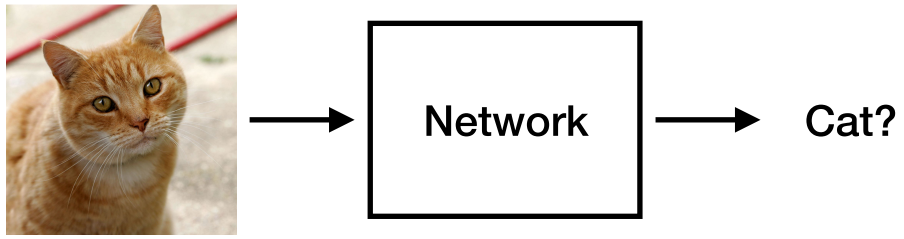
- To implement regression we simply add a linear regression layer as the last layer.
- "layer_dense(unit = 1)"
- No activation.
- This can output negative, 0 and positive numbers.



$\hat{y} = \boldsymbol{wa} + b$

Classification

- In **classification** we want to assign a **label/category** to each input.
- In **binary classification** there are **two** categories and each data belongs in either category.
  - Spam / No-spam
  - Cat / No-cat
  - Similar to what we have done in the notebooks.

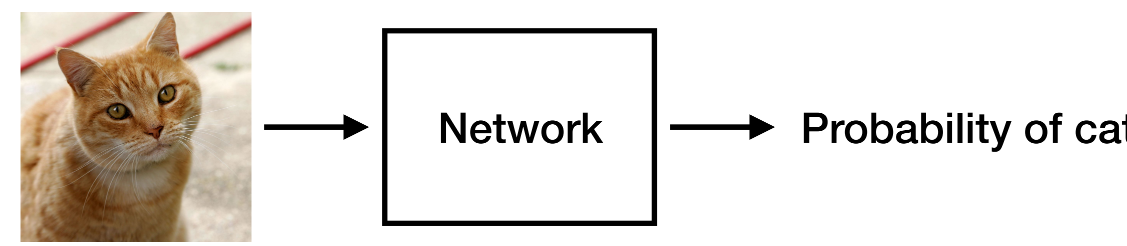


Source: The internet, really, all of it

Classification

Binary classification

- In classification we output a **probability** of belonging to a class.
- Lets say that our dataset contains images which are labelled as "cat" and "not cat".
- First we pre-process the labels so that **"cat" is 1**, and **"not cat" is 0**.
- We will output the probability of being a cat.



$y = 1$ It's a cat

$y = 0$ It's not a cat

$\hat{y} = 0.7$

Source: The internet, really, all of it

Classification

Binary classification - architecture

- How do we output a probability from a neuron?

- We can not simply have the output be a linear regression of last layer since it can output negative numbers and large positive numbers.

- We need $0 \leq \hat{y} \leq 1$

- To fix this we simply apply a sigmoid activation as an activation after the linear regression output, as we have already seen.

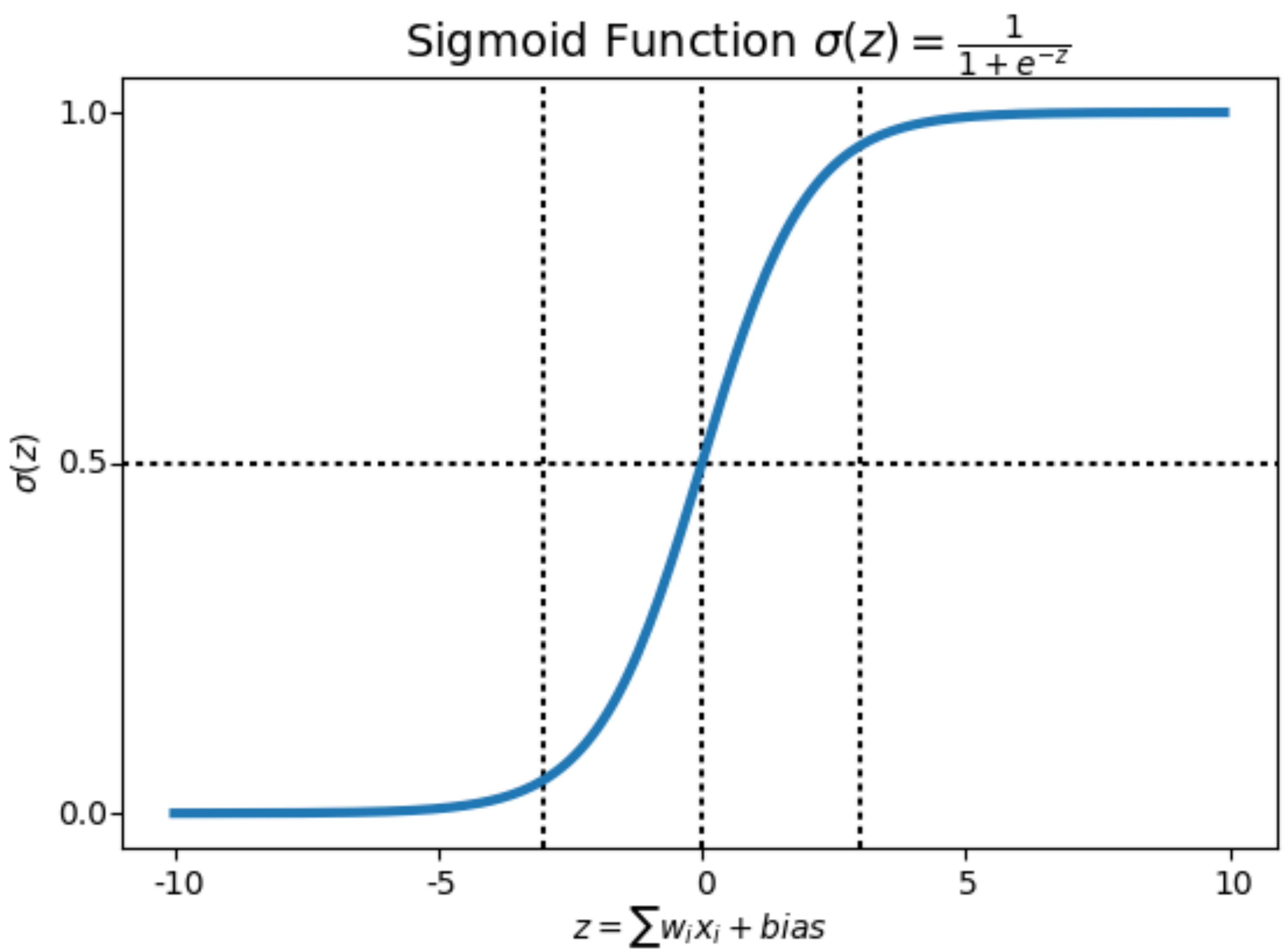
$$z = \boldsymbol{wa} + b$$ **Linear regression**

$$\hat{y} = \sigma(z)$$ **Apply sigmoid after linear regression**

"layer_dense(unit = 1, activation = "sigmoid")"

Classification

Binary classification - sigmoid



Classification

Binary classification - loss

- Then we need to provide a loss function, since MSE is a bit too primitive for this.
- The standard approach in binary classification using sigmoid is to use the following loss.

$$l(\hat{y}, y) = -y\log(\hat{y}) - (1-y)\log(1-\hat{y})$$

Classification

Binary classification - loss justification

$$l(\hat{y}, y) = -y\log(\hat{y}) - (1-y)\log(1-\hat{y})$$

- To see how it works go through the cases.

$$y := 1 \qquad l(\hat{y},1) = -\log(\hat{y})$$

$$y := 0 \qquad l(\hat{y},0) = -\log(1-\hat{y})$$

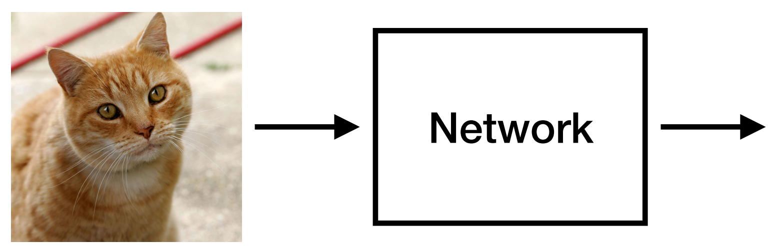
- This allows us to express 0 loss when making correct predictions, and infinitely large loss when making incorrect predictions.

$$y := 1 \; \hat{y} := 1 \qquad -\log(1) = 0$$

$$y := 1 \; \hat{y} := 0 \qquad -\log(0) = \text{inf}$$

Classification

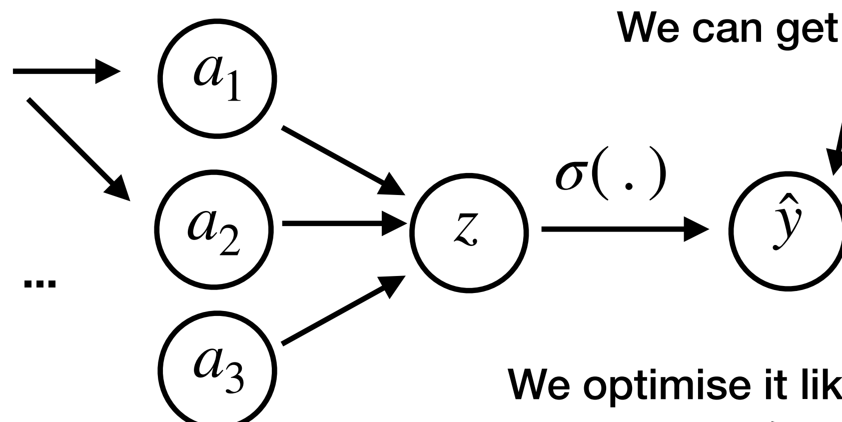
Binary classification summary



We want this

$$\hat{y} = p(y = 1 \mid x)$$

We can get it like this



$$\begin{aligned}\hat{y} &= \sigma(\boldsymbol{wa} + b)\\ &= \sigma(z)\end{aligned}$$

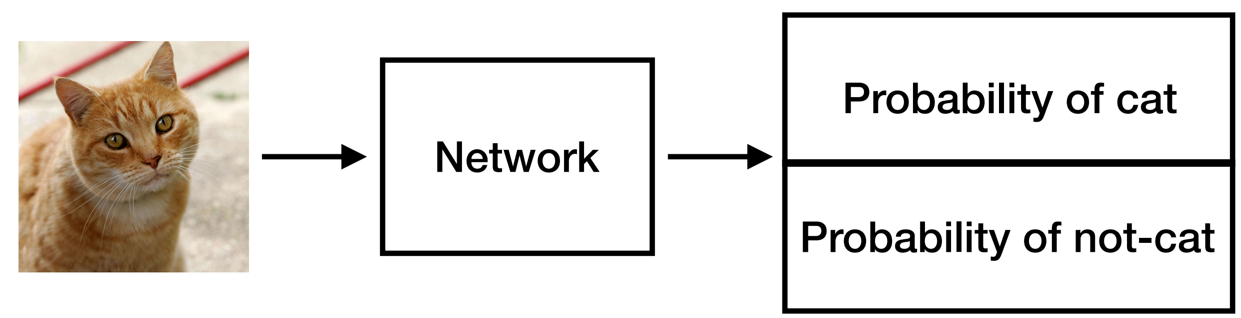
We optimise it like this

$$l(\hat{y}, y) = -\,y\log(\hat{y}) - (1-y)\log(1-\hat{y})$$

Classification

Binary classification as multi-class classification

- We can also go another approach and output two values. The probability of "cat" and the probability of "not cat".
- We output two things at once!
- Why? This approach generalises to more classes.
- In multi-class classification we want to label the input as **one of multiple classes**.



Classification

One-hot encoding

- If we have 5 classes, dog, cat, fish, bird and mouse.
- We could represent them as 0, 1, 2, 3, 4 but that does not work well.
  - We would need to use regression.
- We rather use **one-hot encoding**.

$$y = \begin{bmatrix}0\\1\\0\\0\\0\end{bmatrix} \begin{array}{l} \longleftarrow \text{Not a dog}\\ \longleftarrow \text{a cat}\\ \longleftarrow \text{Not a fish}\\ \longleftarrow \text{Not a bird}\\ \longleftarrow \text{Not a mouse}\end{array}$$

$$y = \begin{bmatrix}0\\0\\0\\0\\1\end{bmatrix}$$

Classification

Multiclass classification

- Lets start by representing our correct labels using a vector using **one-hot encoding.**
- This is then the **true probability distribution** of the example.

It's a cat!

$$y = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

It's not a cat!

$$y = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$\hat{y} = \begin{bmatrix} 0.23 \\ 0.77 \end{bmatrix}$$

Our predictions

Classification

Softmax layer

- The dimensions of y and $\hat{y}$ must match, so $\hat{y}$ must be a vector.
- For $\hat{y}$ to represent probabilities there are two conditions.
    1. The sum of all elements must be 1.
    2. Each element needs to be in the range [0;1].
- The **Softmax layer** ensures that these properties are present.

Classification

Softmax properties

$$\hat{\boldsymbol{y}} = \begin{bmatrix} 0.23 \\ 0.77 \end{bmatrix}$$

Our predictions

$$\hat{\boldsymbol{y}}_1 + \hat{\boldsymbol{y}}_2 = 0.23 + 0.77 = 1$$ 1. Check

$$0 \leq \hat{\boldsymbol{y}}_1 \leq 1$$ 2. Check

$$0 \leq \hat{\boldsymbol{y}}_2 \leq 1$$

Classification

Mathematics of softmax

- First output two real values,

$$z_1 = \boldsymbol{w_1 a} + b_1 \qquad z_2 = \boldsymbol{w_2 a} + b_2$$

- Then normalise these values and deal with negative values.

$$y_1 = a_1 = \frac{e^{z_1}}{\sum_{i=1}^{2} e^{z_i}} \qquad y_2 = a_2 = \frac{e^{z_2}}{\sum_{i=1}^{2} e^{z_i}}$$

"layer_dense(unit = 2, activation = "softmax")"

Classification

Multiclass classification

- We can now output $\hat{y}$ as a probability distribution and represent y using one-hot encoding, the true/correct distribution.

- Now we need a loss function to train our model.

- We borrow from information theory, there we have a function which compares two probabilities distributions, the **categorical cross-entropy function**.

It's a cat!

$$\boldsymbol{y} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \qquad \hat{\boldsymbol{y}} = \begin{bmatrix} 0.23 \\ 0.77 \end{bmatrix}$$

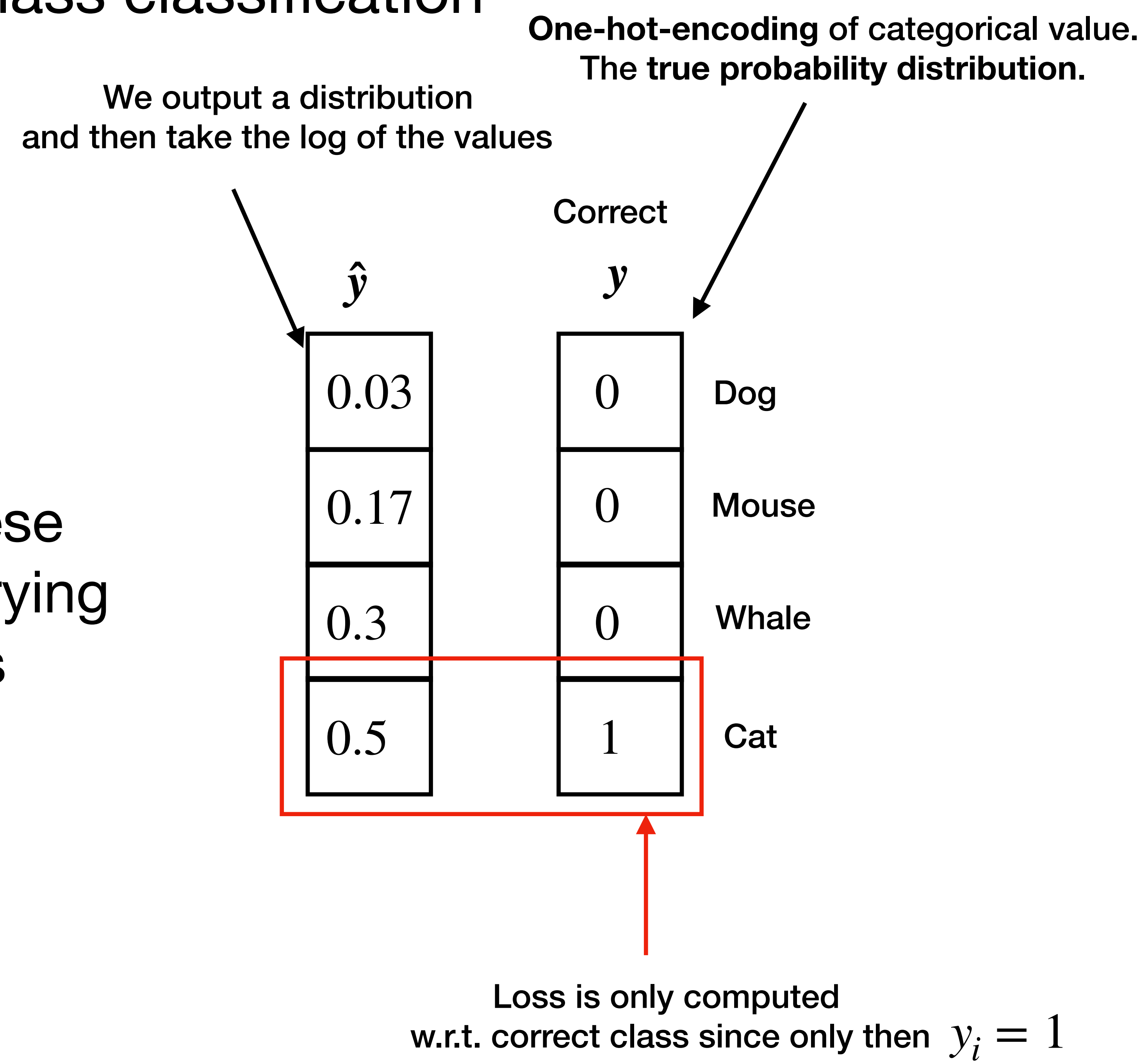
Our predictions

$$l(\hat{y}, y) = -\boldsymbol{y} \cdot \log(\hat{\boldsymbol{y}})$$

$$l(\hat{y}, y) = -\boldsymbol{y} \cdot \log(\hat{\boldsymbol{y}}) = -\sum_{i=1}^{2} y_i \log(\hat{y}_i) = -y_1 \log(\hat{y}_1) - y_2 \log(\hat{y}_2)$$

Classification

Multiclass classification



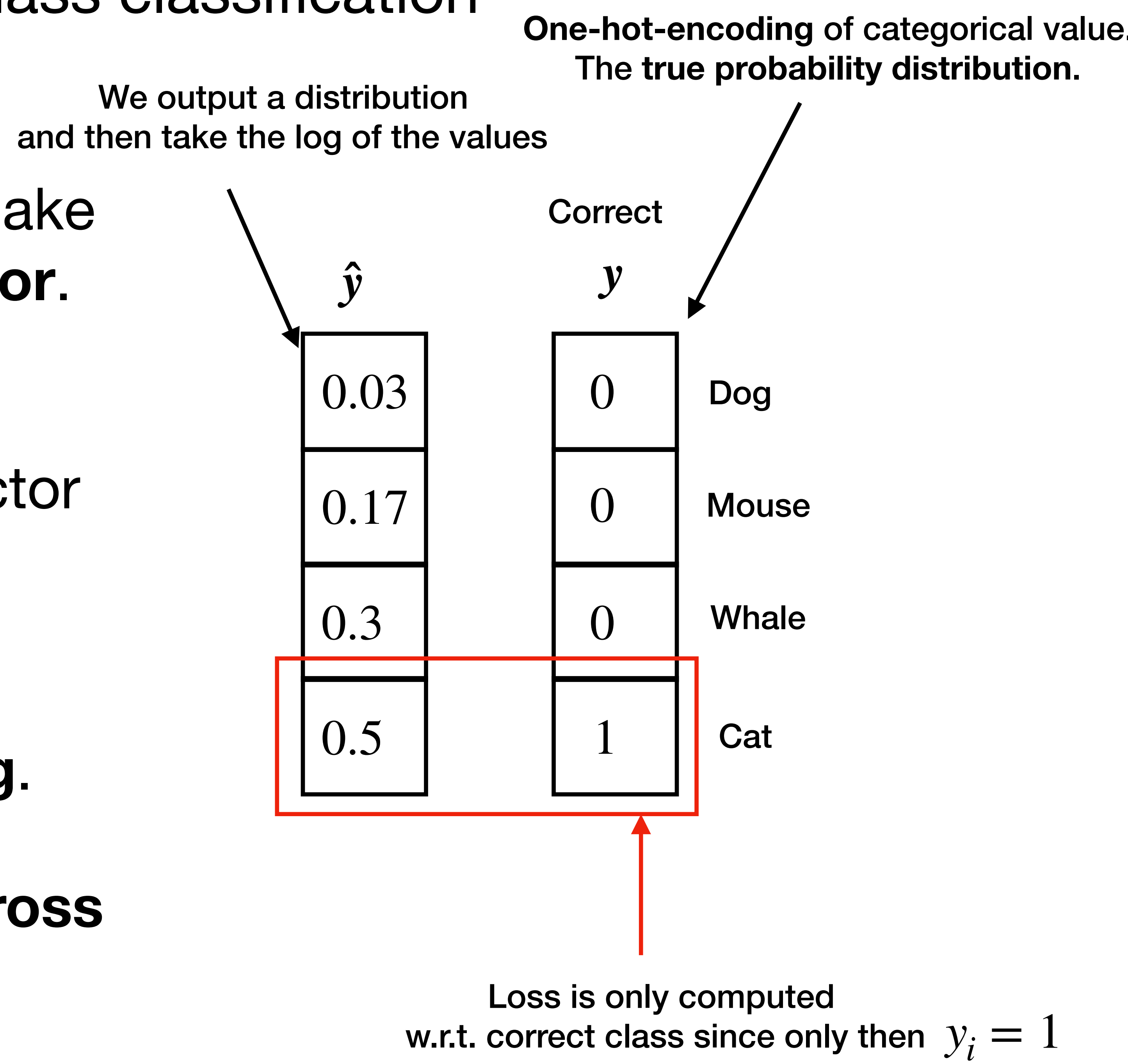
- By minimising categorical cross entropy between these two distributions, we are trying to make them as similar as possible.

$$l(\hat{y}, y) = -\boldsymbol{y} \cdot \log(\hat{\boldsymbol{y}}) = -\sum_{i=1}^{c} y_i \log(\hat{y}_i)$$

Summary

Multiclass classification

- We just saw that we can make our network **output a vector**. Powerful stuff!
- We even constrain that vector to have certain properties, **softmax**.
- We saw **one-hot encoding**.
- We saw the **categorical cross entropy** loss function.



$$l(\hat{y}, y) = -\boldsymbol{y} \cdot \log(\hat{\boldsymbol{y}}) = -\sum_{i=1}^{c} y_i \log(\hat{y}_i)$$

Hands-on

Go to https://dba.projects.sda.surfsara.nl/

Notebook: `03a-fashion-mnist-multiclass.ipynb`

Break at 11:00 / 15:00

Second part at 11:10 / 15:10

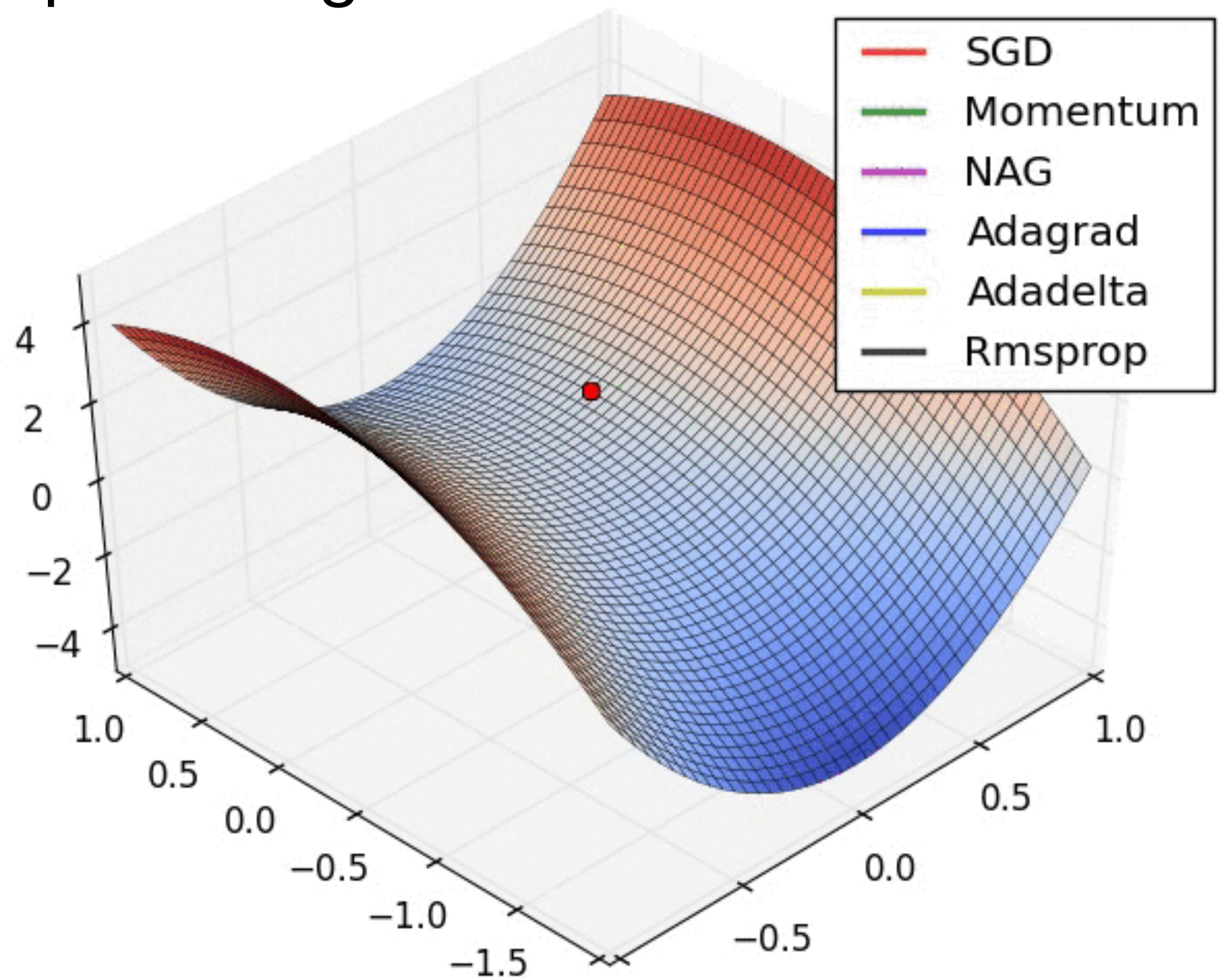
Improving networks

- We can split up the ways to improve networks to two categories (some belong in both categories).
- **Speed up learning** while training the network.
  - Advanced optimisers (using momentum and per parameter step size)
  - Input data normalisation
  - Batch normalisation
  - (weight initialisation)
- After we have fit the training data, we want to focus on **reducing overfitting.**
  - L1/L2 regularisation
  - Dropout

Optimisers

Speeding up learning

- In practice we don't just use mini-batch gradient descent but more dynamic implementations.
- Some optimisers have been shown to do well for certain architectures.



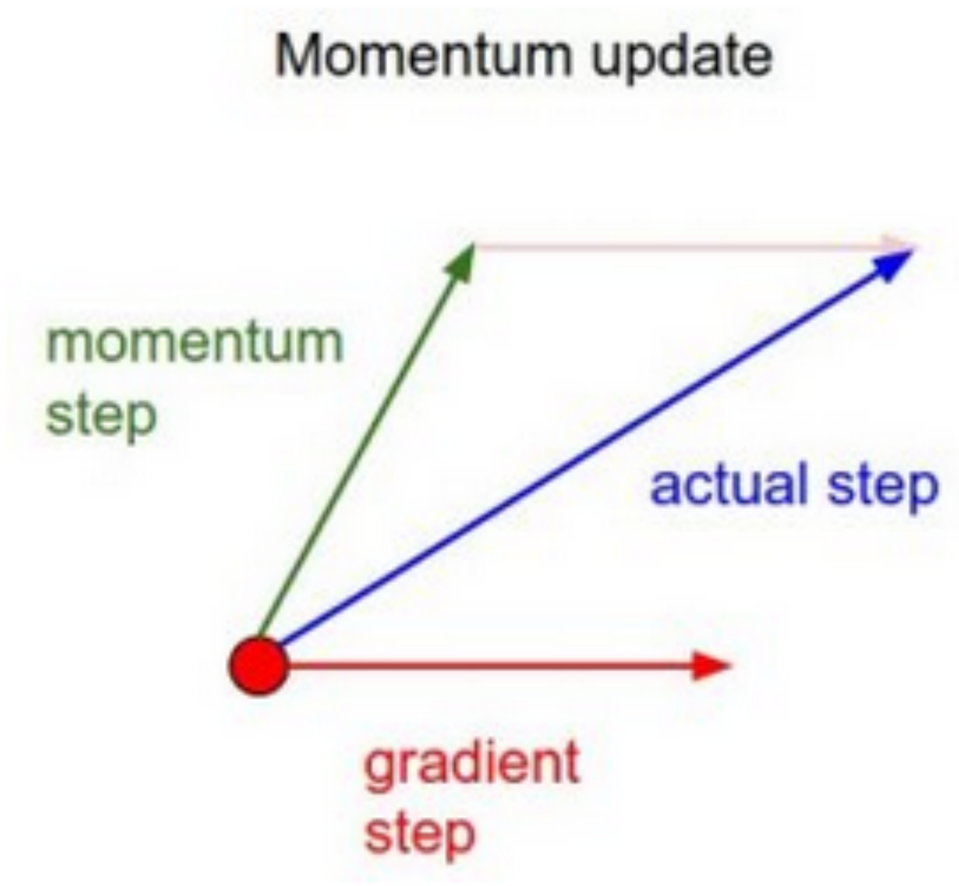
Source:
http://ruder.io/optimizing-gradient-descent/index.html

Optimisers

Speeding up learning

- Some feature **momentum** which takes the previous updated values into account (exponentially decaying averages).

  - Momentum < NAG

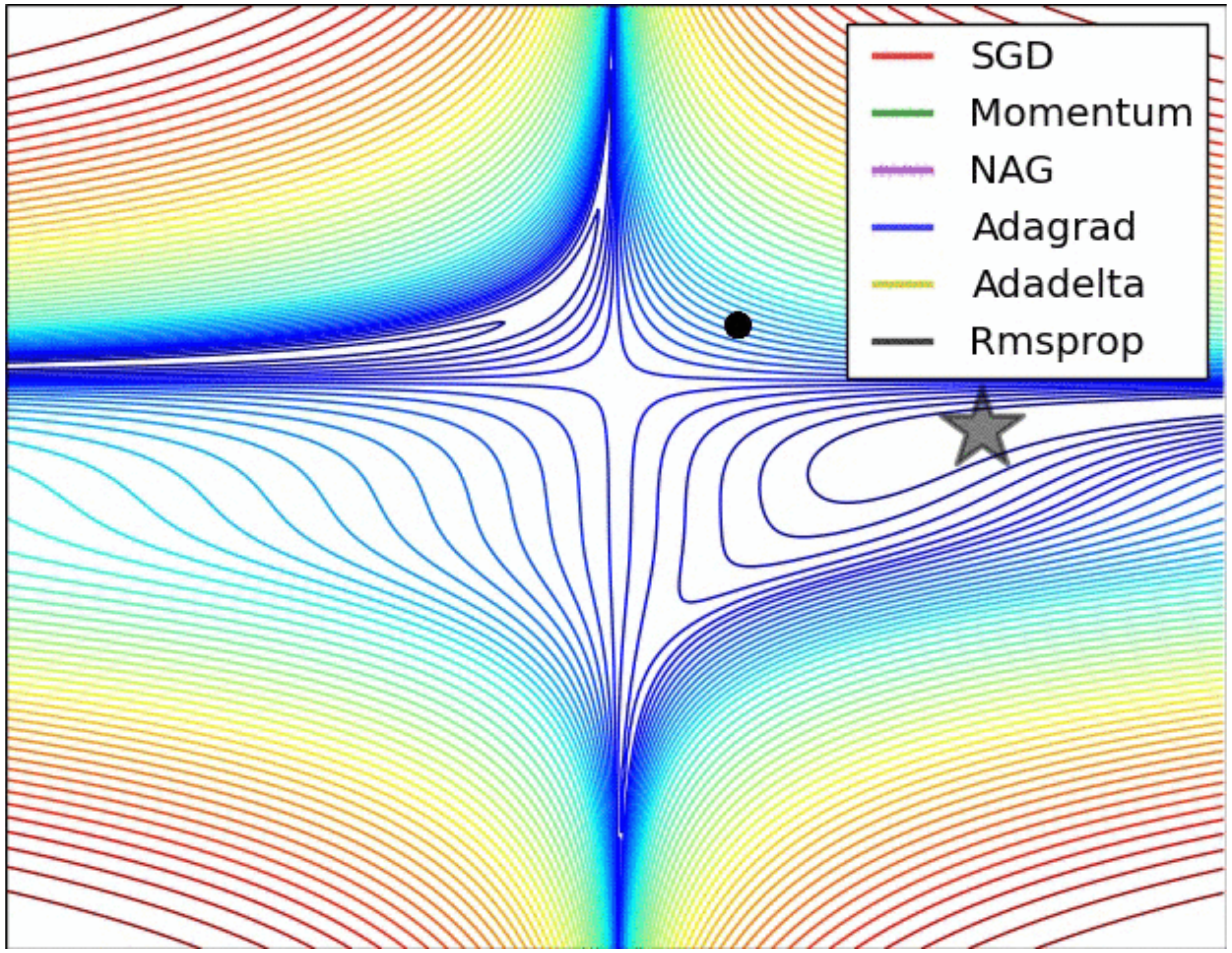


Source:
https://cs231n.github.io/neural-networks-3/

Optimisers

Speeding up learning

- And **feature sensitive step sizes**, which perform smaller updates (you can think of it as lower learning rate) for frequent features and larger for more unfrequent features.
    - Adagrad < Adadelta = RMSprop
- **Adam** has been shown to be a good general choice.
    - Adam = RMSprop + Momentum



Source:
http://ruder.io/optimizing-gradient-descent/index.html

Input normalisation

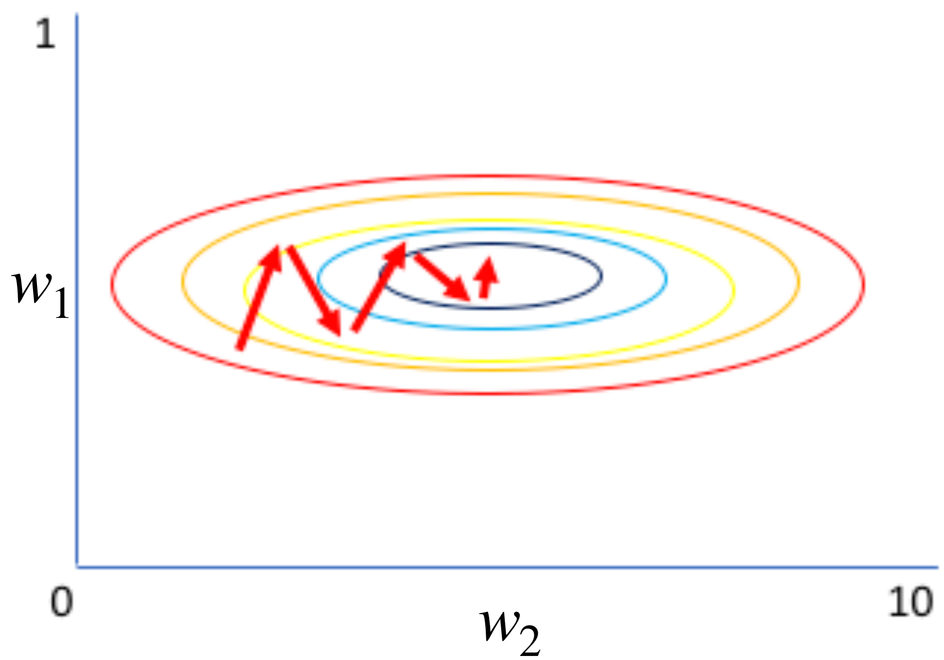
Speeding up learning

- We have already covered input normalisation.
- As a preprocessing stage for the input features.
- This has been shown to speed up training of neural networks.
- All features should have the same range.
  - Mean 0, variance 1.
  - We can use the "scale" function or in some cases (f.ex. images) divide by 255.

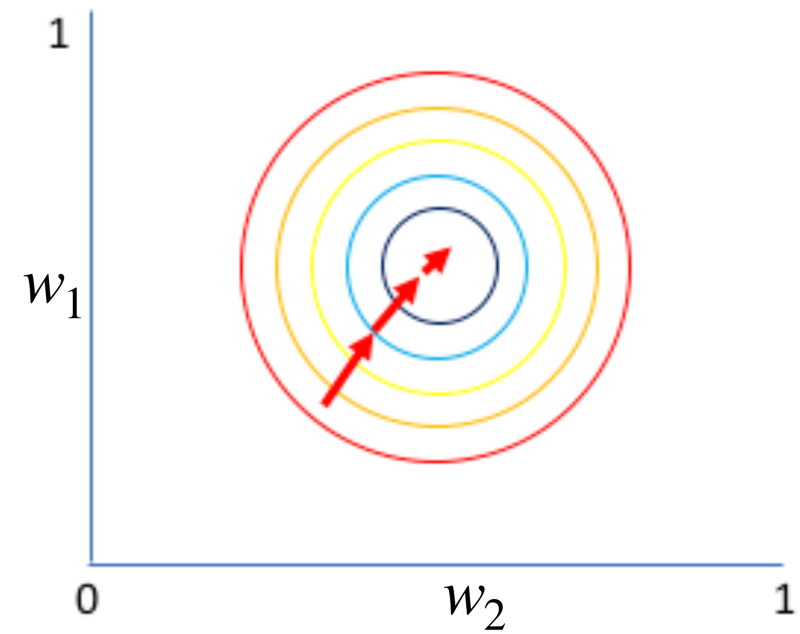
Input normalisation

Speeding up learning

Why normalize?



Gradient of larger parameter dominates the update



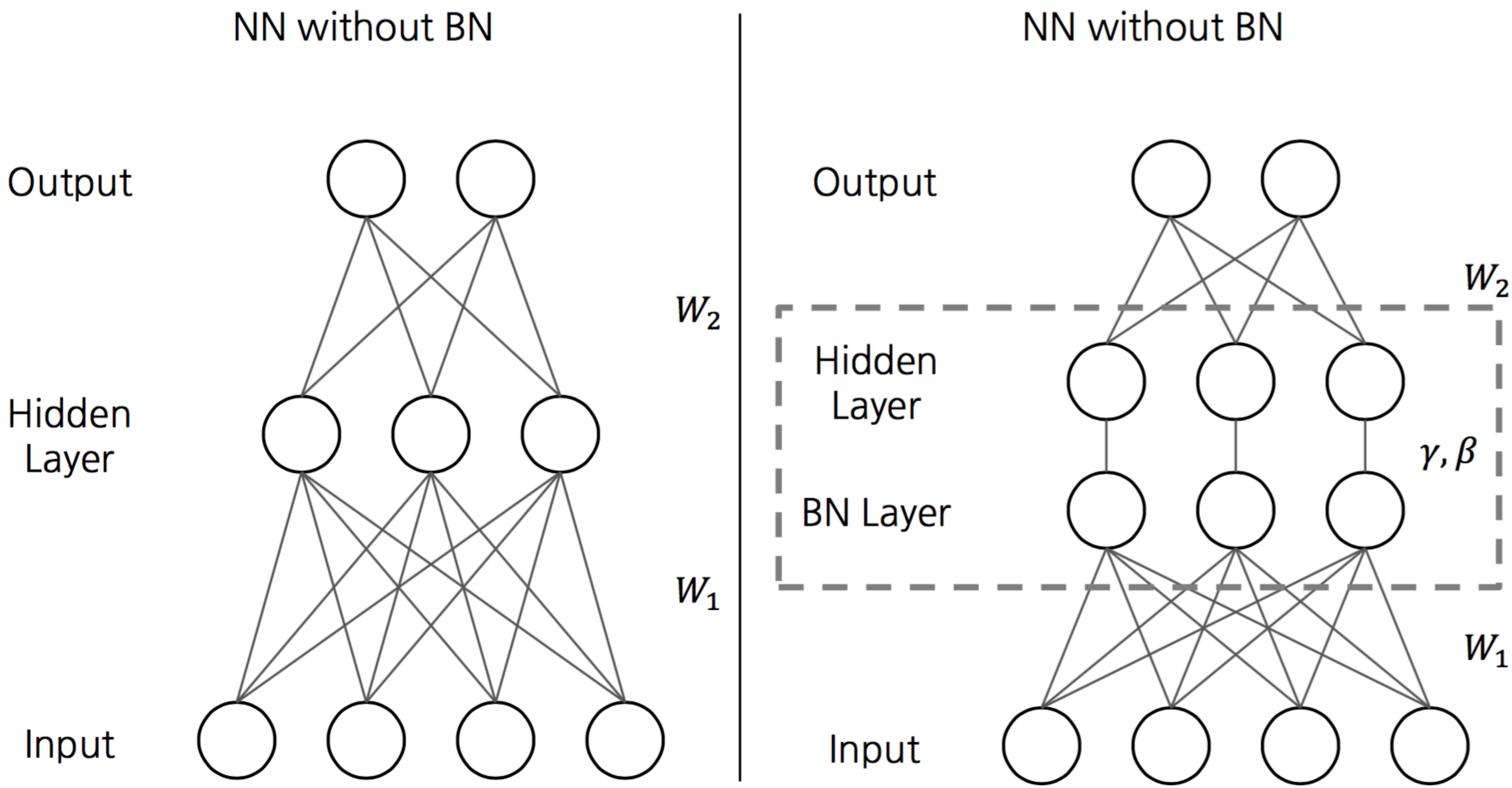
Both parameters can be updated in equal proportions

Source:
https://www.jeremyjordan.me/batch-normalization/

Batch normalisation (BN)

Speeding up learning

- Why only do this normalisation on the input?
- In 2015 it was shown that renormalising in an intermediary layer speeds up learning.
- We compute the mean and variance per batch and uses it to normalise to **0 mean** and **variance 1**.



Source:
https://wiki.tum.de/display/lfdv/Batch+Normalization

Batch normalisation (BN)

Speeding up learning

- We might not always want 0 mean and variance 1 so we add **two more parameters** to scale the values out again.
- γ and β are parameters learnt by the model, 2 per neuron.
- Worst case scenario, BN is not helpful at all and the model will just learn the mean and variance of the batches.

Batch normalisation (BN)

Speeding up learning

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

Source:
https://wiki.tum.de/display/lfdv/Batch+Normalization

Batch normalisation

Speeding up learning

- Adding BN will add more parameters to the model and extra computation.
- BN allow us to more easily train deeper networks.
- BN makes the network more robust to hyperparameter selections.
- BN allows us to train with a higher learning rate.
- We apply BN before the activations.

```
layer_dense(unit = 10)
layer_batch_normalization()
layer_activation_relu()
```

Regularisation

Reducing overfitting

- What is regularisation?
- Any kind of technique which helps you select one model over another using a structured approach.
- We will add extra terms to the loss function (L2)
- We will add intermediary layers to the network (Dropout)

L2 Regularisation

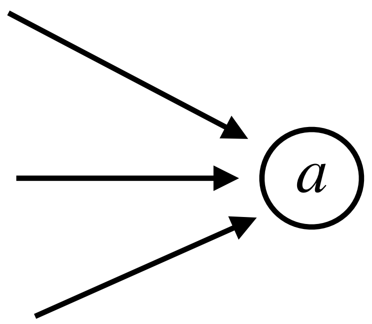
Reducing overfitting

- We add a new term to the total loss function.
- This term adds additional loss to the function which takes the value of the weights into account.
- We then optimise this new loss function instead.
- A new **hyperparameter**, λ is added. This is usually a small value and we will need trial and error to find an acceptable value. It can be considered as a discount factor.

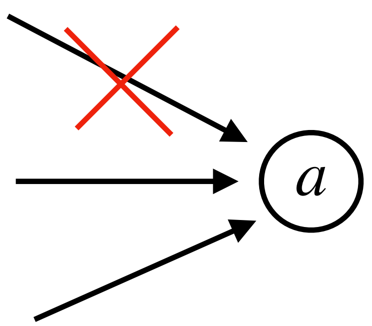
L2 Regularisation

Reducing overfitting

- Why does L2 regularisation work?
- We add a cost to the weights, thus making a "**more complex**" model **more expensive.**
- If some learnt weight is high (say, 10) it "costs" more than a weight with value 1.
- Thus, our model becomes "simpler" by **forcing the weights down**.
- When some weights are forced to 0, we are effectively "**removing**" **connections.**

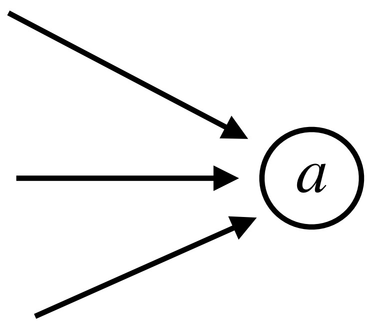


Set weight to 0

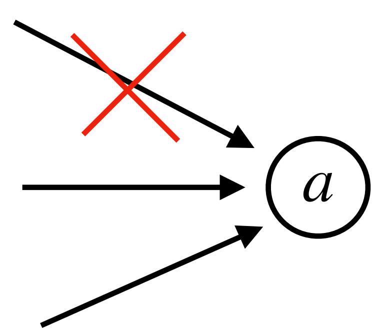


L2 Regularisation

Reducing overfitting



- We use L2 regularisation to fight overfitting, because it makes our model less expressive.
- We use it **after we have fitted the data**.
- It will **increase** the **training loss** during training and **hopefully reduce** the **test loss**.
- Also known as **weight decay**.



L2 Regularisation

Reducing overfitting

$$\begin{tikzcd}
 & \boldsymbol{w} \arrow[d] & & \\
 & f(x_i, \boldsymbol{w}) \arrow[r] & \hat{y}_i \arrow[r] & l(\hat{y}_i, y_i) \\
x_i \arrow[ur] & & y_i \arrow[ur] &
\end{tikzcd}$$

Total loss $= J(\boldsymbol{w}) = \frac{1}{n}\sum_i^n (l(f(x_i, \boldsymbol{w}), y_i)$

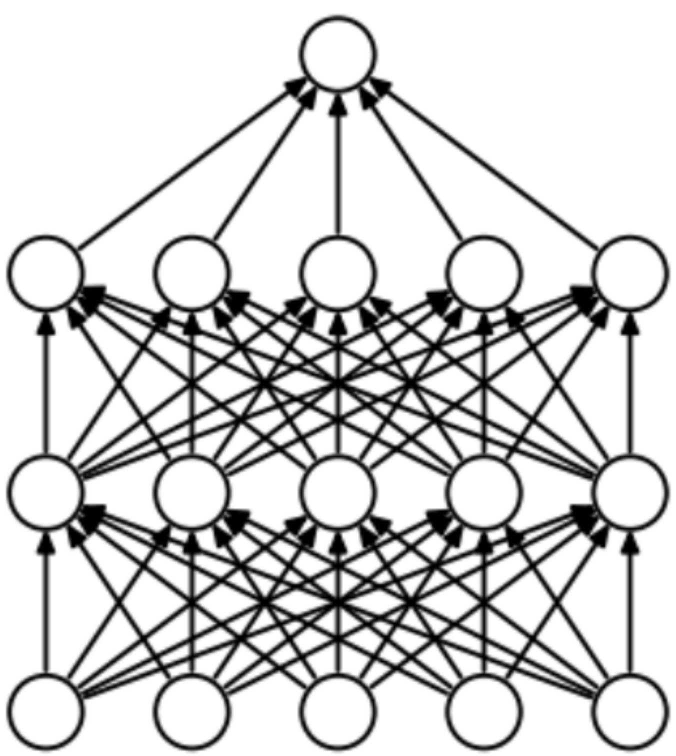
Now becomes

$$J(\boldsymbol{w}) = \frac{1}{n}\sum_i^n (l(f(x_i, \boldsymbol{w}), y_i) + \boxed{\frac{\lambda}{2n}\sum_j w_j^2}$$

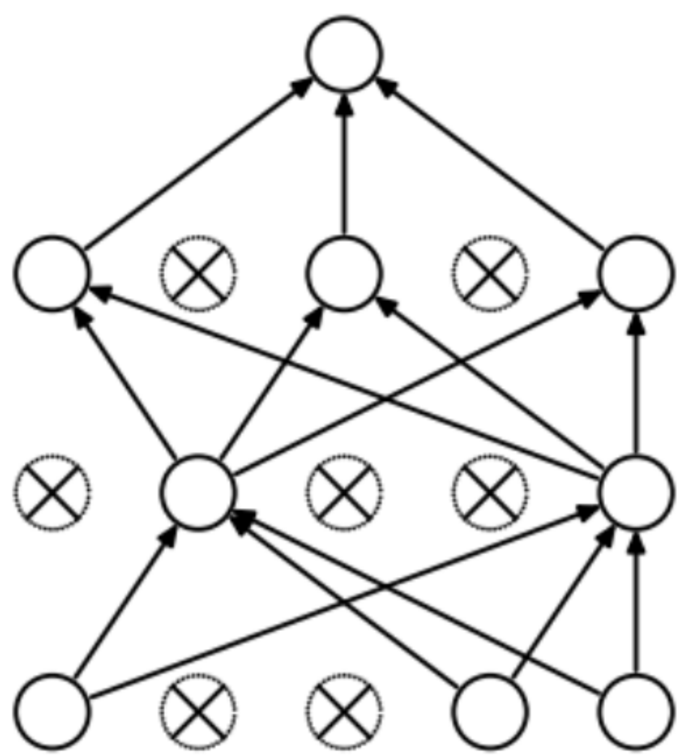
Dropout

Reducing overfitting

- Dropout is a similar form of regularisation. It will **randomly set** the **activations** of neurons **to 0**.



(a) Standard Neural Net



(b) After applying dropout.

Source:
Srivastava et al., 2014, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting"

Dropout

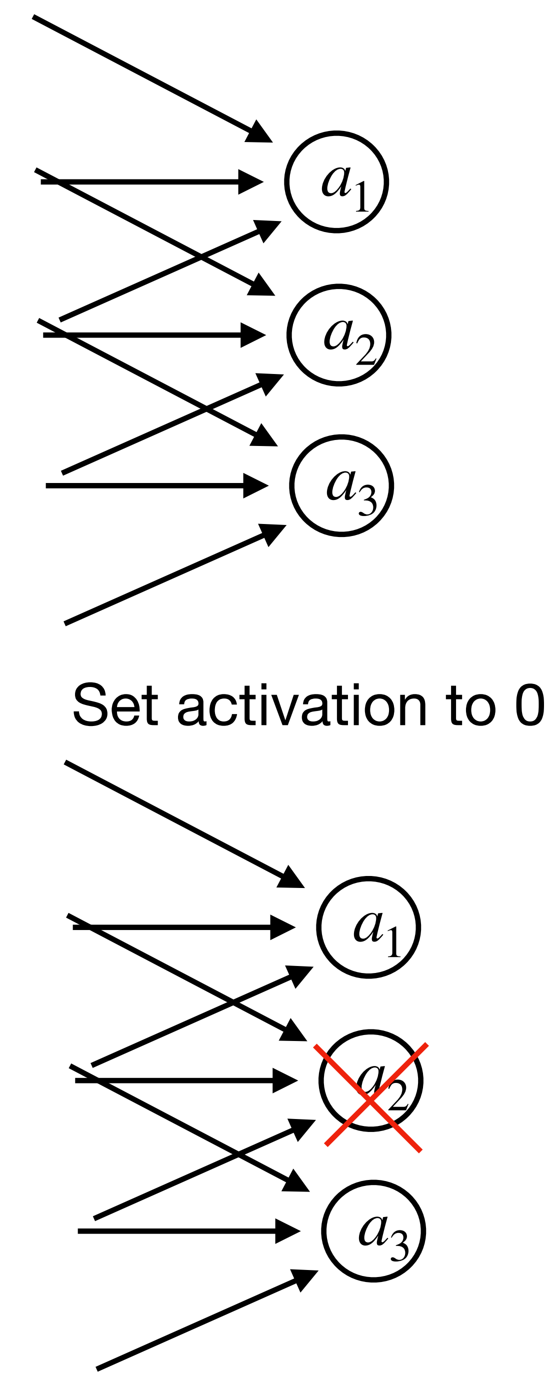
Reducing overfitting

- This **reduces dependance on specific features** thus making the network more robust.

- The fraction of neurons we set to 0 for each layer is a **new hyperparameter** which is also found by trial and error (usually between 0.5 and 0.2).

- We apply dropout after the activations.

- This has been shown to increase performance during test time.



Summary

- Use Adam.
- Normalise input.
- Apply BN before activations.
- Use L2 regularisation.
- Apply Dropout after activations.
- These techniques will make your loss function a lot more noisier (higher variance), but we will perform better during test time.

Hands-on

Go to https://dba.projects.sda.surfsara.nl/

Notebook: `03b-regression-regularisation.ipynb`

Wrap-up at 12:20 / 16:20

Notebook recap

- We were not really able to improve the baseline much, but made it converge faster.
- We saw that we really need to test if the regularisation technique is helping us.
  - L2 regularisation was not very stable. Dropout was better.
- It depends on the task, architecture, ..., trial and error.

Summary

- Machine learning tasks
  - Regression
  - Binary classification
  - Multi-class classification
- Improving networks
  - Preventing overfitting = Regularisation, dropout
  - Speeding up learning = Advanced optimisers, batch normalisation