# Deadlines

<table>
<thead>
<tr>
<th>Assignment</th>
<th>Date of assignment</th>
<th>Deadline (midnight 23:59)</th>
</tr>
</thead>
<tbody>
<tr>
<td>HW1</td>
<td>Sep 6</td>
<td>Sep 19</td>
</tr>
<tr>
<td>HW2</td>
<td>Sep 20</td>
<td>Oct 3</td>
</tr>
<tr>
<td>HW3</td>
<td>Oct 4</td>
<td>Oct 17</td>
</tr>
<tr>
<td>Paper summary</td>
<td>Oct 11</td>
<td>Oct 31</td>
</tr>
<tr>
<td>HW4</td>
<td>Oct 18</td>
<td>Oct 31</td>
</tr>
<tr>
<td>HW5</td>
<td>Nov 1</td>
<td>Nov 14</td>
</tr>
<tr>
<td>HW6</td>
<td>Nov 22</td>
<td>Dec 5</td>
</tr>
<tr>
<td>Project</td>
<td>Oct 12</td>
<td>Nov 15 - 17</td>
</tr>
</tbody>
</table>

*All deadlines are subject to change, check out CampusWire and website for updates*
There are two main ways of looking at this (A and B).
There are two main ways of looking at this (A and B)

**way A:** something wrong with a model
There are two main ways of looking at this (A and B).

way A: something wrong with a model

way B: something wrong with a data
There are two main ways of looking at this (A and B):

**way A**: something wrong with a model

**way B**: something wrong with a data

---

**Epoch #5**

![Graph showing loss over epochs](image)

- **Overfitting**
- **loss**
- **epoch**

---

**Dataset**

- batch #1
- batch #2
- batch #3

**Validation**
Regularisation methods

Way A methods

Way B methods
Regularisation methods

Way A methods

Way B methods
\[ \text{ML} + \text{L1 or L2 regularisation} = \]
\[ \text{LR} + \text{L1 regularisation} = \]
\[ \text{LR} + \text{L2 regularisation} = \]
\[ \text{DL} + \text{L1 or L2 regularisation} = \]
Machine Learning model with L1 or L2 regularisation

ML + \text{L1 or L2 regularisation} =

LR + \text{L1 regularisation} =

LR + \text{L2 regularisation} =

DL + \text{L1 or L2 regularisation} =
**ML** + **L1 or L2 regularisation** = Machine Learning model with **L1 or L2 regularisation**

**LR** + **L1 regularisation** = **Lasso (LASSO) regression**

**LR** + **L2 regularisation** =

**DL** + **L1 or L2 regularisation** =
Machine Learning model with L1 or L2 regularisation

Lasso (LASSO) regression

Ridge regression
Machine Learning model with L1 or L2 regularisation

Lasso (LASSO) regression

Ridge regression

Deep Learning model with weight decay
Machine Learning model with L1 or L2 regularisation

Lasso (LASSO) regression

Ridge regression

Deep Learning model with weight decay
Two regularisation methods are very similar

$$\text{Error} + \lambda \sum_{i}^{n} (w_i)^2 \quad \text{L2 regularisation}$$

$$\text{Error} + \lambda \sum_{i}^{n} |w_i| \quad \text{L1 regularisation}$$

Yet, this difference is very important
Polynomial model
trained with

No regularisation ($\lambda = 0$)

Conventional error function (RSS)

$L2$ regularisation ($\lambda = 1$)

$\hat{y} = w_0 + w_1 x + w_2 x^2 + w_3 x^3 + w_4 x^4$

$L2$ regularisation results in

$\text{Error} = \sum_{i=1}^{5} (y_i - \hat{y}_i)^2 + \lambda \sum_{i}^{5} (w_i)^2$

L2 regularisation ($\lambda = 1$)

No regularisation ($\lambda = 0$)

$L1$ regularisation ($\lambda = 1$)

L1 regularisation can drive some weights all the way down to 0

While L2 regularisation forces some weights to be close to 0

$5 - 8.8*x + 7.8*x^2 - 2.3*x^3 + 0.2*x^4$

$L1$ regularisation

Red and blue curves look similar

What is the difference?
$\text{Machine Learning model with L1 or L2 regularisation} = \text{ML} + \text{L1 or L2 regularisation}$

$\text{Lasso (LASSO) regression} = \text{LR} + \text{L1 regularisation}$

$\text{Ridge regression} = \text{LR} + \text{L2 regularisation}$

$\text{Deep Learning model with weight decay} = \text{DL} + \text{L1 or L2 regularisation}$
Neural Network from the last time

Point coordinates as input

(X, Y)

Y

X

(class 1 score)

(class 2 score)

Error

Neural Network from the last time

Point coordinates as input

(X, Y)

Y

X

(class 1 score)

(class 2 score)

Error

Neural Network from the last time

Point coordinates as input

(X, Y)

Y

X

Error

Neural Network from the last time

Point coordinates as input

(X, Y)

Y

X

(class 1 score)

(class 2 score)

Error

Neural Network from the last time

Point coordinates as input

(X, Y)

Y

X

Error

Neural Network from the last time

Point coordinates as input

(X, Y)

Y

X

Error

Neural Network from the last time

Point coordinates as input

(X, Y)

Y

X

(class 1 score)

(class 2 score)

Error

Neural Network from the last time

Point coordinates as input

(X, Y)

Y

X

Error

Neural Network from the last time

Point coordinates as input

(X, Y)

Y

X

Error

Neural Network from the last time

Point coordinates as input

(X, Y)

Y

X

Error

Neural Network from the last time

Point coordinates as input

(X, Y)

Y

X

Error

Neural Network from the last time

Point coordinates as input

(X, Y)

Y

X

Error

Neural Network from the last time

Point coordinates as input

(X, Y)

Y

X

Error

Neural Network from the last time

Point coordinates as input

(X, Y)

Y

X

Error

Neural Network from the last time

Point coordinates as input

(X, Y)

Y

X

Error

Neural Network from the last time

Point coordinates as input

(X, Y)

Y

X

Error

Neural Network from the last time

Point coordinates as input

(X, Y)

Y

X

Error

Neural Network from the last time

Point coordinates as input

(X, Y)

Y

X

Error

Neural Network from the last time

Point coordinates as input

(X, Y)

Y

X

Error

Neural Network from the last time

Point coordinates as input

(X, Y)

Y

X

Error

Neural Network from the last time

Point coordinates as input

(X, Y)

Y

X

Error

Neural Network from the last time

Point coordinates as input

(X, Y)

Y

X

Error

Neural Network from the last time

Point coordinates as input

(X, Y)

Y

X

Error

Neural Network from the last time

Point coordinates as input

(X, Y)

Y

X

Error

Neural Network from the last time

Point coordinates as input

(X, Y)

Y

X

Error

Neural Network from the last time

Point coordinates as input

(X, Y)

Y

X

Error

Neural Network from the last time

Point coordinates as input

(X, Y)

Y

X

Error

Neural Network from the last time

Point coordinates as input

(X, Y)

Y

X

Error

Neural Network from the last time

Point coordinates as input

(X, Y)

Y

X

Error

Neural Network from the last time

Point coordinates as input

(X, Y)

Y

X

Error

Neural Network from the last time

Point coordinates as input

(X, Y)

Y

X

Error

Neural Network from the last time

Point coordinates as input

(X, Y)

Y

X

Error

Neural Network from the last time

Point coordinates as input

(X, Y)

Y

X

Error

Neural Network from the last time

Point coordinates as input

(X, Y)

Y

X

Error

Neural Network from the last time

Point coordinates as input

(X, Y)

Y

X

Error

Neural Network from the last time

Point coordinates as input

(X, Y)

Y

X

Error

Neural Network from the last time

Point coordinates as input

(X, Y)

Y

X

Error

Neural Network from the last time

Point coordinates as input

(X, Y)

Y

X

Error

Neural Network from the last time

Point coordinates as input

(X, Y)

Y

X

Error

Neural Network from the last time

Point coordinates as input

(X, Y)

Y

X

Error

Neural Network from the last time

Point coordinates as input

(X, Y)

Y

X

Error

Neural Network from the last time

Point coordinates as input

(X, Y)

Y

X

Error

Neural Network from the last time

Point coordinates as input

(X, Y)

Y

X

Error

Neural Network from the last time

Point coordinates as input

(X, Y)

Y

X

Error

Neural Network from the last time

Point coordinates as input

(X, Y)

Y

X

Error

Neural Network from the last time

Point coordinates as input

(X, Y)

Y

X

Error

Neural Network from the last time

Point coordinates as input

(X, Y)

Y

X

Error

Neural Network from the last time

Point coordinates as input

(X, Y)

Y

X

Error

Neural Network from the last time

Point coordinates as input

(X, Y)

Y

X

Error

Neural Network from the last time

Point coordinates as input

(X, Y)

Y

X

Error

Neural Network from the last time

Point coordinates as input

(X, Y)

Y

X

Error

Neural Network from the last time

Point coordinates as input

(X, Y)

Y

X

Error

Neural Network from the last time

Point coordinates as input

(X, Y)

Y

X

Error
During **Feed-forward** path just add \( \text{L2/L1} \) penalty to the total error.
During **Feed-forward** path just add **L2/L1** penalty to the total error.

Adding **L2** regularisation (or weight decay)

\[ \text{Error} = \lambda \sum_{i}^{n} (w_i)^2 \]
During **Feed-forward** path just add \( \textbf{L2}/\textbf{L1} \) penalty to the total error

Adding \( \textbf{L2} \) regularisation (or weight decay)

\[
\text{Error} = \lambda \sum_{i}^{n} (w_i)^2
\]
During **Feed-forward** path just add \( \textbf{L2}/\textbf{L1} \) penalty to the total error

Adding **L2** regularisation (or weight decay)

\[
\lambda \sum_{i} (w_i)^2
\]
During **Feed-forward** path just add **L2/L1** penalty to the total error

Point coordinates as **input**

During **Feed-forward** path just add **L2/L1** penalty to the total error

Point coordinates as **input**

During **Feed-forward** path just add **L2/L1** penalty to the total error

Point coordinates as **input**

During **Feed-forward** path just add **L2/L1** penalty to the total error

Point coordinates as **input**

During **Feed-forward** path just add **L2/L1** penalty to the total error

Point coordinates as **input**

During **Feed-forward** path just add **L2/L1** penalty to the total error

Point coordinates as **input**

During **Feed-forward** path just add **L2/L1** penalty to the total error

Point coordinates as **input**

During **Feed-forward** path just add **L2/L1** penalty to the total error

Point coordinates as **input**
When doing **backpropagation** take $\mathbf{L2}$/$\mathbf{L1}$ term into account.
When doing **backpropagation** take \( \text{L2/L1} \) term into account.

Remember this horrible formula from last time?

\[
\frac{\partial E_{\text{total}}}{\partial w_5} = \frac{\partial E_{\text{total}}}{\partial E_{o_1}} \times \frac{\partial E_{o_1}}{\partial \text{out}_{o_1}} \times \frac{\partial \text{out}_{o_1}}{\partial \text{in}_{o_1}} \times \frac{\partial \text{in}_{o_1}}{\partial w_5}
\]

Point coordinates as input.
When doing **backpropagation** take $L_2/L_1$ term into account.

Point coordinates as input.

Remember this horrible formula from last time?

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial E_{o_1}} \ast \frac{\partial E_{o_1}}{\partial out_{o_1}} \ast \frac{\partial out_{o_1}}{\partial in_{o_1}} \ast \frac{\partial in_{o_1}}{\partial w_5}$$

Total Error

$\lambda \sum_i (w_i)^2$
When doing **backpropagation** take **L2**/**L1** term into account.

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{error}}{\partial w_5} + \ldots$$

$$\lambda \sum_{i} (w_i)^2$$
When doing **backpropagation** take \(\textbf{L2}/\textbf{L1}\) term into account.

\[
\frac{\partial E_{\text{total}}}{\partial w_5} = \frac{\partial E_{\text{error}}}{\partial w_5} + \frac{\partial L^2_{\text{penalty}}}{\partial w_5}
\]

\[
\lambda \sum_{i}^{8} (w_i)^2
\]
When doing **backpropagation** take \( \text{L2/L1} \) term into account.

We saw how to calculate this:

\[
\frac{\partial E_{\text{total}}}{\partial w_5} = \frac{\partial E_{\text{error}}}{\partial w_5} + \frac{\partial L^2_{\text{penalty}}}{\partial w_5}
\]

the hell is this?
When doing **backpropagation** take $\frac{\lambda}{L1}$ term into account

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{error}}{\partial w_5} + \frac{\partial L2_{penalty}}{\partial w_5}$$

Point coordinates as input
When doing **backpropagation** take $L2/L1$ term into account

$$
\frac{\partial E_{\text{total}}}{\partial w_5} = \frac{\partial E_{\text{error}}}{\partial w_5} + \frac{\partial L2_{\text{penalty}}}{\partial w_5}
$$

$$
\lambda \sum_{i} (w_i)^2
$$
When doing **backpropagation** take \( \text{L2/L1} \) term into account

\[
\frac{\partial E_{\text{total}}}{\partial w_5} = \frac{\partial E_{\text{error}}}{\partial w_5} + \frac{\partial (\lambda \sum_{i}^{8} (w_i)^2)}{\partial w_5}
\]

Point coordinates as input
When doing **backpropagation** take **L2/L1** term into account.

\[
\frac{\partial E_{\text{total}}}{\partial w_5} = \frac{\partial E_{\text{error}}}{\partial w_5} + \lambda \frac{\partial (\sum_i^8 (w_i)^2)}{\partial w_5}
\]
When doing **backpropagation** take **L2/L1** term into account

\[
\frac{\partial E_{\text{total}}}{\partial w_5} = \frac{\partial E_{\text{error}}}{\partial w_5} + \lambda \frac{\partial (\sum_{i}^{8} (w_i)^2)}{\partial w_5}
\]

\[
\Rightarrow \frac{\partial (w_1^2 + w_2^2 + \ldots + w_8^2)}{\partial w_5}
\]
When doing \textbf{backpropagation} take \textbf{L2/L1} term into account

\[ \frac{\partial E_{\text{total}}}{\partial w_5} = \frac{\partial E_{\text{error}}}{\partial w_5} + \lambda \frac{\partial \left( \sum_{i}^{8} (w_i)^2 \right)}{\partial w_5} \]

\[ \frac{\partial (w_1^2 + w_2^2 + w_3^2 + w_4^2 + w_5^2 + w_6^2 + w_7^2 + w_8^2)}{\partial w_5} \]
When doing **backpropagation** take **L2/L1** term into account

\[
\frac{\partial E_{\text{total}}}{\partial w_5} = \frac{\partial E_{\text{error}}}{\partial w_5} + \lambda \frac{\partial (\sum_{i}^{8} (w_i)^2)}{\partial w_5}
\]

\[
= \frac{\partial (w_1^2)}{\partial w_5} + \frac{\partial (w_2^2)}{\partial w_5} + \frac{\partial (w_3^2)}{\partial w_5} + \frac{\partial (w_4^2)}{\partial w_5} + \frac{\partial (w_5^2)}{\partial w_5} + \frac{\partial (w_6^2)}{\partial w_5} + \frac{\partial (w_7^2)}{\partial w_5} + \frac{\partial (w_8^2)}{\partial w_5}
\]

Point coordinates as input

\((X,Y)\)
When doing **backpropagation** take **L2/L1** term into account.

\[
\frac{\partial E_{\text{total}}}{\partial w_5} = \frac{\partial E_{\text{error}}}{\partial w_5} + \lambda \frac{\partial (\sum_i^8 (w_i)^2)}{\partial w_5} + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 0
\]

Point coordinates as input
When doing **backpropagation** take $L2/L1$ term into account.

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{error}}{\partial w_5} + \lambda * \frac{\partial (\sum_{i=1}^{8} (w_i)^2)}{\partial w_5}$$
When doing **backpropagation** take \( \frac{L2}{L1} \) term into account

\[
\frac{\partial E_{\text{total}}}{\partial w_5} = \frac{\partial E_{\text{error}}}{\partial w_5} + \lambda \frac{\partial \left( \sum_{i}^{8} (w_i)^2 \right)}{\partial w_5} \]

\[
= \frac{\partial \text{MSE Error}}{\partial w_5} + 2 \ast w_5
\]
When doing **backpropagation** take \( \text{L2/L1} \) term into account

\[
\frac{\partial E_{\text{total}}}{\partial w_5} = \frac{\partial E_{\text{error}}}{\partial w_5} + \lambda * 2 * w_5
\]

This part is usually multiplied by 0.5

Point coordinates as input

\((X,Y)\)
When doing backpropagation take L2/L1 term into account

\[
\frac{\partial E_{\text{total}}}{\partial w_5} = \frac{\partial E_{\text{error}}}{\partial w_5} + \lambda * w_5
\]

\[
\lambda \sum_{i} (w_i)^2
\]

Point coordinates as input
When doing **backpropagation** take \( \text{L2/ L1} \) term into account

\[
\frac{\partial E_{\text{total}}}{\partial w_5} = \frac{\partial E_{\text{error}}}{\partial w_5} + \lambda * w_5
\]

**L2** term for backpropagation

MSE Error

\[
\lambda \sum_{i} (w_i)^2
\]
When doing **backpropagation** take \( \frac{L_2}{L_1} \) term into account

What about \( L_1 \)?
When doing **backpropagation** take $\text{L2}/\text{L1}$ term into account

\[
\frac{\partial E_{\text{total}}}{\partial w_5} = \frac{\partial E_{\text{error}}}{\partial w_5} + \lambda * w_5
\]

\[
\frac{\partial E_{\text{total}}}{\partial w_5} = \frac{\partial E_{\text{error}}}{\partial w_5} + \ldots
\]

**L2** term for backpropagation

(MSE Error) $\sum_{i}^{8} (w_i)^2$
When doing **backpropagation** take \( \text{L2}/\text{L1} \) term into account

\[
\frac{\partial E_{\text{total}}}{\partial w_5} = \frac{\partial E_{\text{error}}}{\partial w_5} + \lambda \star w_5
\]

\[
\frac{\partial E_{\text{total}}}{\partial w_5} = \frac{\partial E_{\text{error}}}{\partial w_5} + \lambda
\]

**L2** term for backpropagation

**L1** term for backpropagation
When doing **backpropagation** take **L2/L1** term into account

\[ \frac{\partial E_{\text{total}}}{\partial w_5} = \frac{\partial E_{\text{error}}}{\partial w_5} + \lambda * w_5 \]

The truth being said, I have not heard people using **L1** in DL a lot

*I might want to update this slide if I find evidence to the contrary*
(X, Y)

Error + weight decay
Another very popular regularisation method used together with weight decay is called **Dropout**.
Another very popular regularisation method used together with weight decay is called **Dropout**

Some of you might have read the **Dropout** paper

Error + weight decay

\[(X, Y)\]

\[
\begin{align*}
  \text{class 1 score} & = w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4 + b_1 \\
  \text{class 2 score} & = w_5 x_1 + w_6 x_2 + w_7 x_3 + w_8 x_4 + b_2
\end{align*}
\]
For each new training point

\[(X, Y)\]

\[
\begin{align*}
    \text{class 1 score} & = w_1 x + b_1 \\
    \text{class 2 score} & = w_2 x + b_2 \\
    \text{Error} & = \text{class 1 score} - \text{class 2 score}
\end{align*}
\]

\[+ \text{weight decay}\]
For each new training point

\[(X, Y)\]
For each new training point

For each neuron in the network toss a coin

For each neuron in the network toss a coin

Error + weight decay
For each new training point

For each neuron in the network toss a coin

(\(X, Y\))

\[ 0.0 \]

\[ 0.0 \]

Error + weight decay
For each new training point

For each neuron in the network toss a coin

For each new training point

For each neuron in the network toss a coin
For each new training point

Silence a neuron with probability $p$
Leave it be with probability $1 - p$

For each neuron in the network toss a coin

Error + weight decay
For each new training point, for each neuron in the network, toss a coin. Silence a neuron with probability $p$. Leave it be with probability $1 - p$. *Unlike a real coin, $p$ does not have to be 0.5.*
For each new training point

For each neuron in the network toss a coin

Let our $p$ be 0.25

Error + weight decay

For each new training point

For each neuron in the network toss a coin

Let our $p$ be 0.25

Error + weight decay
For each neuron in the network toss a coin

Let our $p$ be 0.25
For each neuron in the network toss a coin
Let our $p$ be 0.25

Toss a coin for this neuron

Error + weight decay
For each new training point, toss a coin. Let our $p$ be 0.25. If the coin comes out heads, switch off the neuron.

Toss a coin for this neuron.
For each new training point

For each neuron in the network toss a coin

Let our $p$ be 0.25

Error + weight decay
For each neuron in the network toss a coin

Let our $p$ be 0.25

For each new training point

Toss a coin for this neuron
For each neuron in the network toss a coin
Let our \( p \) be 0.25
For each neuron in the network toss a coin
Let our $p$ be 0.25
For each new training point

Let our $p$ be 0.25

For each neuron in the network toss a coin

Error + weight decay
For each new training point

Let our $p$ be 0.25
For each new training point

For each neuron in the network toss a coin

Let our $p$ be 0.25
What happens to this “switched off” neuron?

Error + weight decay
What happens to this “switched off” neuron?

It takes in input as usual.
What happens to this “switched off” neuron?

It takes in input as usual but it outputs 0.
What happens to this “switched off” neuron?

It takes in input as usual but it outputs 0
For each new training point

For each neuron in the network toss a coin

Let our $p$ be 0.25

For each new training point, for each neuron in the network, toss a coin with probability $p = 0.25$. If the coin lands heads, update the weights accordingly; if tails, do not update. This process helps in avoiding overfitting by introducing randomness into the model.
The **main idea** is that for every point you get **slightly different network** to work.
The **main idea** is that for every point you get a **slightly different network** to work.
The main idea is that for every point you get slightly different network to work.
The **main idea** is that for every point you get slightly different network to work.

This forces network **not to rely** on a particular set of weights.
During the **backpropagation**, weights associated with 0 signal are not changed

\((w_5 \text{ and } w_6 \text{ in the example below})\)
L2/L1 regularisation VS Dropout
Forces all the weights be close to 0

L2/L1 regularisation vs Dropout
Forces **all** the **weights** be **close** to 0

**L2/L1** regularisation  VS  **Dropout**

Forces **all** the **weights** be **close** to 0 and **some exactly 0**
L2/L1 regularisation vs Dropout

Forces all the weights be close to 0

Forces random weights to be 0

Forces all the weights be close to 0 and some exactly 0
Despite some differences, these methods belong to so called **explicit regularisation methods**

- **L2/L1 regularisation**
  - Forces **all the weights** be **close** to 0
  - VS
  - Forces **random weights** to be 0

- **Dropout**
  - Forces **all the weights** be **close** to 0 and some exactly 0
Despite some differences, these methods belong to so called **explicit regularisation methods**

They **explicitly limit** predictive power of the model by introducing **additional constrains**

**L2/L1** regularisation VS **Dropout**
Despite some differences, these methods belong to so called explicit regularisation methods. They explicitly limit predictive power of the model by introducing additional constrains.
Explicit regularisation methods (Dropout, L1/L2 regularisation)
Explicit regularisation methods
(Dropout, L1/L2 regularisation)

Way B methods
Explicit regularisation methods
(Dropout, L1/L2 regularisation)

Implicit regularisation methods
Dataset
Dataset

Test
Dataset
Dataset

batch #1
batch #2
batch #3

Validation

Epoch #1

batch #1
batch #2
batch #3

Model

Training

loss

epoch
In the second Epoch you put in the **same** training images → **Epoch #2**

![Graph showing loss vs epoch with points at (1, 0.4) and (2, 0.6)]
Dataset

batch #1  batch #2  batch #3

Epoch #2

batch #1
batch #2
batch #3

Model

Training loss decreases

Validation

loss

1 2 3 4 5
epoch
Dataset

Validation batch #1

Validation batch #2

Validation batch #3

Epoch #2

Validation loss may not change

Model

loss

epoch
Epoch #2

Dataset

Model

Validation

batch #1  batch #2  batch #3

loss

epoch
Epoch #3

Dataset

batch #1  batch #2  batch #3

Validation

Epoch #3

Model

loss

epoch
Dataset

Epoch #3

Model

Validation

batch #1  batch #2  batch #3

batch #1
batch #2
batch #3

loss

epoch
Epoch #3

Training loss continues to improve
Epoch #3
Epoch #3

Validation loss may start growing
Epoch #3

This process may go on
This process may go on
Every epoch we fed the same training images.
Every epoch we fed the same training images

Model managed to **memorise training examples** (like a poem)
Every epoch we fed the same training images.

Since the model had no access to validation images it was getting worse at predicting them.

Epoch #5

Model managed to memorise training examples (like a poem).
We had too few training images.

Since the model had no access to validation images it was getting worse at predicting them.

Model managed to memorise training examples (like a poem).
Where can we find more training data?

Since the model had no access to validation images it was getting worse at predicting them.

Model managed to memorise training examples (like a poem)
Dataset
(100%)
Dataset (100%)

Test (20%)
Used only once
Dataset (100%)

Validation (20%)
- Used occasionally

Test (20%)
- Used only once
Dataset (100%)

<table>
<thead>
<tr>
<th>Training (60%)</th>
<th>Validation (20%)</th>
<th>Test (20%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Used all the time</td>
<td>Used occasionally</td>
<td>Used only once</td>
</tr>
</tbody>
</table>

- **Training (60%)**: Used all the time
- **Validation (20%)**: Used occasionally
- **Test (20%)**: Used only once
Increase size of the training data by reducing test/validation sets.
Increase size of the training data by reducing test/validation sets.
Increase size of the training data by reducing test/validation sets.
Increase size of the training data by reducing test/validation sets.
<table>
<thead>
<tr>
<th>Dataset (100%)</th>
<th>Training (60%)</th>
<th>Validation (20%)</th>
<th>Test (20%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Used all the time</td>
<td>Used occasionally</td>
<td>Used only once</td>
</tr>
</tbody>
</table>

Extremely small **test** and **validation** sets will result in **unstable** performance.
What else can we do?
What if we just **clone** training images?
Unfortunately, this will not do - there is no new information
Since the model had no access to validation images it was getting worse at predicting them.

Every epoch we fed the same training images. The model managed to memorise training examples (like a poem).
Unfortunately, this will not do - there is no new information
Instead of naïvely copying, transform images a bit.

Unfortunately, this will not do - there is no new information.
Instead of naively copying, **transform** images a bit.
Instead of naively copying, **transform** images a bit
How do you **transform** images?
How do you **transform** images?

By flipping:
- Original
- Vertically flipped
- Horizontally flipped
How do you **transform** images?

By **turning** (by any angle)

- Original
- 90 degrees
- 270 degrees
How do you **transform** images?

- **Original**
- **Low saturation**
- **High Saturation**

*By changing saturation levels*
How do you **transform** images?

By changing **contrast**

- Original
- Low contrast
- High contrast
How do you **transform** images?

By combining several strategies:

- Original
- Turned 90 degrees & low contrast
- High saturation & vertically flipped
These are the most basic transforms
More advanced things are available via github/aleju/imgaug
Another popular Python augmentation library:
https://github.com/albumentations-team/albumentations
New augmentation strategies are being constantly proposed

Fig. 1: Comparison of our proposed Attentive CutMix with Mixup [5], Cutout [1] and CutMix [3].
Instead of naively copying, perform augmentation.
When should I use augmentation?
When should I use augmentation?

 Everywhere you can…
Augmentation is very domain specific technique
Augmentation is very domain specific technique
Not all types of data are easy to augment

Augmentation is very domain specific technique
Only one instance of opposite class
Only one instance of **opposite** class.

The **number** of points per **class**.
Only one instance of opposite class

The number of points per class

3x or 4x times more data in one of the classes
Only one instance of \textbf{opposite} class

The \textbf{number} of points per \textbf{class}

3x or 4x times more data in one of the classes

\textbf{Majority class}

\textbf{Minority class}
What are the potential causes of the class imbalance?

The number of points per class

3x or 4x times more data in one of the classes

Only one instance of opposite class

Majority class

Minority class
What are the potential challenges?

The number of points per class

**Majority class**

**Minority class**

Only one instance of opposite class

3x or 4x times more data in one of the classes
Models tend to **ignore** the **minority** class.

![Diagram showing data points and class distribution.]

- Only one instance of the **opposite** class.
- The number of points per class: 3x or 4x times more data in one of the classes.

**Majority class** vs. **Minority class**
Models tend to **ignore** the **minority** class.

Only one instance of **opposite** class

Accuracy = **80%**

The number of points per **class**

3x or 4x times more data in one of the classes

**Majority** class

**Minority** class
Models tend to **ignore the minority class**.

Only one instance of **opposite** class

Accuracy = **80%**

What are the ways to remedy the situation?

The **number** of points per class

3x or 4x times more data in one of the classes

Minority class

Majority class
The number of points per class

3x or 4x times more data in one of the classes

Majority class

Minority class
The number of points per class

Majority class

Minority class

Blue

Red
We can artificially **increase the number of minority** class points.
We can artificially **increase the number** of minority class points.

Make 3 copies of this minority class point.

The **number** of points per class.
We can artificially **increase the number** of minority class points.

Make 3 copies of this **minority** class point.

The **number** of points per class:

- **Majority class**
- **Minority class**
We can artificially **increase the number of minority** class points [**Over**sampling]

- Make 3 copies of this *minority* class point
- The **number** of points per class

![Diagram](image-url)
In case there is just one point in the minority class, you simply make copies of it.
In case there is just one point in minority class, you simply make copies of it.

But what if you have more than one point in the minority class?
In case there is just one point in the minority class, you simply make copies of it.

But what if you have more than one point in the minority class?
In case there is just **one point** in **minority** class, you simply make copies of it.

But what if you have **more than one point** in the **minority** class?

We can copy each point **equally**.
In case there is just one point in the minority class, you simply make copies of it.

But what if you have more than one point in the minority class?

We can copy each point equally, or alternatively, we can copy random points.
Since we do not create new information, the practical utility of such approach is limited.

```
from imblearn.over_sampling import RandomOverSampler
ros = RandomOverSampler(random_state=0)
X_resampled, y_resampled = ros.fit_resample(X, y)
```
What are other approaches to oversampling?
What are other approaches to oversampling?
What are other approaches to oversampling?
What are other approaches to oversampling?

Instead of simply copying points of the minority class, we can try to create new.

The number of points per class:
- Majority class: Blue
- Minority class: Red
What are other approaches to oversampling?

Instead of simply copying points of the minority class, we can try to create new.
What are other approaches to oversampling?

Instead of simply copying points of the minority class, we can try to create new minority points.

The number of points per class:

- Majority class: Blue
- Minority class: Red
What are other approaches to **oversampling**?

Instead of simply copying points of the *minority* class, we can try to *create new* points. Draw lines between minority points and synthesise new points *randomly on these lines*. The number of points per *class* is shown in the bar chart.
What are other approaches to oversampling?

Instead of simply copying points of the minority class, we can try to create new ones. Draw lines between minority points. Synthesise new points randomly on these lines.

The number of points per class

Majority class

Minority class
Instead of simply copying points of the minority class, we can try to create new minority points randomly on these lines. 

**SMOTE: Synthetic Minority Oversampling Technique**

The number of points per class:

- Blue: Majority class
- Red: Minority class
SMOTE: Synthetic Minority Oversampling Technique

The number of points per class

Minority class

Majority class
SMOTE: Synthetic Minority Over-sampling Technique

The number of points per class

9 blue vs 9 red

Blue

Red

Majority class

Minority class
SMOTE: Synthetic Minority Oversampling Technique

The number of points per class

9 blue vs 9 red

from imblearn.over_sampling import SMOTE
X_resampled, y_resampled = SMOTE().fit_resample(X, y)
SMOTE: practical considerations
SMOTE: practical considerations

Consider the following case what problems can you anticipate?
Consider the following case what problems can you anticipate?
SMOTE: practical considerations

Consider the following case what problems can you anticipate?

Synthesising new points along these lines will not add clarity.
SMOTE: practical considerations

To resolve this issue, SMOTE randomly chooses a minority class point.
To resolve this issue **SMOTE** randomly chooses a **minority** class point.
To resolve this issue, SMOTE randomly chooses a minority class point. It then spawns new points only between this chosen point and \( K \) of its nearest neighbours.
SMOTE: practical considerations

To resolve this issue, SMOTE randomly chooses a minority class point. It then spawns new points only between this chosen point and $K$ of its nearest neighbours, where $K = 2$. 
SMOTE: practical considerations

To resolve this issue SMOTE randomly chooses a minority class point. It then spawns new points only between this chosen point and $K$ of its nearest neighbours. $K = 1$ is probably the safest way to run SMOTE.
SMOTE: practical considerations

To resolve this issue, SMOTE randomly chooses a minority class point. It then spawns new points only between this chosen point and $K$ of its nearest neighbours. $K = 1$ is probably the safest way to run SMOTE.

Multiple modifications of SMOTE exist (e.g. ADASYN).
SMOTE: practical considerations

To resolve this issue, SMOTE randomly chooses a minority class point. It then spawns new points only between this chosen point and $K$ of its nearest neighbours.

$K = 1$ is probably the safest way to run SMOTE.

Multiple modifications of SMOTE exist (e.g. ADASYN).

```python
from imblearn.over_sampling import ADASYN
X_resampled, y_resampled = ADASYN().fit_resample(X, y)
```

Credits to https://rikunert.com/SMOTE_explained
What are the **ways** to remedy the situation?

The number of points per class

- **Majority class**: 4x or 3x more data in one of the classes
- **Minority class**: 1x data

Blue

Red
What are the ways to remedy the situation?

Over sampling

The number of points per class

3x or 4x times more data in one of the classes

Minority class

Majority class

Blue
Red

Blue
Red
What are the ways to remedy the situation?

Oversampling

**Naive or SMOTE**

The number of points per class

3x or 4x times more data in one of the classes

Majority class

Minority class
What are the ways to remedy the situation?

**Oversampling**

**Under sampling**

---

The number of points per class

3x or 4x times more data in one of the classes

Majority class

Minority class

Blue

Red

Naive or SMOTE
We can artificially **decrease the number of majority** class points.
We can artificially decrease the number of majority class points.

The number of points per class:

- **Majority class**: Blue
- **Minority class**: Red
We can artificially decrease the number of majority class points.

Randomly remove enough points of the majority class to match the minority class count.

The number of points per class

- Blue (Majority class)
- Red (Minority class)
We can artificially decrease the number of majority class points.

Randomly remove enough points of the majority class to match the minority class count.

```python
from imblearn.under_sampling import RandomUnderSampler
rus = RandomUnderSampler(random_state=0)
```
What are **other** approaches to **undersampling**?
What are other approaches to undersampling?

Look for nearest neighbours between classes.
What are other approaches to undersampling?

Find the nearest neighbour for each point.
What are other approaches to undersampling?

Find the nearest neighbour for each point.

Focus on connections between opposite classes.
What are other approaches to **undersampling**?

Find the **nearest neighbour** for each point.

Focus on **connections** between opposite classes (**Tomek links**).
What are other approaches to **undersampling**?

*Find the **nearest neighbour** for each point*

Focus on **connections** between **opposite** classes (**Tomek links**) ★

Remove **majority** class points from **Tomek links**
What are other approaches to undersampling?

Find the nearest neighbour for each point

Focus on connections between opposite classes (Tomek links)

Remove majority class points from Tomek links
What are **other** approaches to **undersampling**?

Find the **nearest neighbour** for each point.

Focus on **connections** between opposite classes (**Tomek links**).

Remove **majority** class points from Tomek links.

Recalculate the **connections**.
What are other approaches to undersampling?

Find the nearest neighbour for each point

Focus on connections between opposite classes (Tomek links)

Remove majority class points from Tomek links

Recalculate the connections
What are other approaches to undersampling?

Find the nearest neighbour for each point

Focus on connections between opposite classes (Tomek links)

Remove majority class points from Tomek links

Recalculate the connections
What are other approaches to undersampling?

Find the **nearest neighbour** for each point

Focus on **connections** between opposite classes (**Tomek links**)  

Remove **majority** class points from Tomek links  

Recalculate the **connections**
This algorithm is known as **Tomek links**
This algorithm is known as **Tomek links**

This algorithm *does not* remove as many points from the **majority** class as naive undersampling.
This algorithm is known as Tomek links

This algorithm does not remove as many points from the majority class as naive undersampling.

It tries to make a space between classes.

```
from imblearn.under_sampling import TomekLinks
tl = TomekLinks()
X_res, y_res = tl.fit_resample(X, y)
```
We can artificially **decrease the number of majority** class points.

**Naive Under** sampling:
- Randomly remove enough points of the **majority** class to match the **minority** class count.

**Tomek Links**:
- Remove **majority** class points until all nearest neighbours are from the same class.
There is another approach to **undersampling**
There is another approach to under sampling.

Considering only majority class points…
There is another approach to **under**sampling.

Considering only **majority** class points run **clustering** algorithm e.g. **K-means**.
There is another approach to **undersampling**

Considering only **majority** class points run **clustering** algorithm e.g. **K-means**

*K* can be **5**
(as number of **minority** class points)
There is another approach to **undersampling**

Considering only **majority** class points run **clustering** algorithm e.g. **K-means**

*K* can be 5 (as number of **minority** class points)
There is another approach to **undersampling**

Considering only **majority** class points run **clustering** algorithm e.g. **K-means**

*K* can be 5  
(as number of **minority** class points)
There is another approach to **undersampling**

Considering only **majority** class points run **clustering** algorithm e.g. **K-means**

**K** can be **5**
(as number of **minority** class points)
The **number** of points per **class**

Over**sampling**

**Naive** or SMOTE

Under**sampling**

What are the **ways** to remedy the situation?
What are the ways to remedy the situation?

**Oversampling**

- Naive or SMOTE

**Undersampling**

- Naive, Tomek links and clustering
Over sampling

Naive or SMOTE

Under sampling

Naive, Tomek links and clustering
**Oversampling**

**Naive or SMOTE**

**Undersampling**

**Naive, Tomek links and clustering**

**Under + Oversampling**
Over-sampling

Naive or SMOTE

Under-sampling

Naive, Tomek links and clustering

Tomek links + SMOTE
Oversampling

Random minority oversampling with replacement

SMOTE

ADASYN

ROSE

K-means + SMOTE

Undersampling

Random majority undersampling with replacement

Tomek links

Undersampling with K-means

Condensed nearest neighbour

and more techniques: https://github.com/scikit-learn-contrib/imbalanced-learn
Not all types of data are easy to augment.
Not all types of data are easy to augment

In [36], the main result was the characterization of hulls. This could shed important light on a conjecture of Perelman. The goal of the present paper is to study C-ordered, reducible, co-Hadamard vector spaces.

Now in [27, 36, 34], it is shown that every scalar is Serre. Therefore it has long been known that there exists a solvable and parabolic right-naturally anti-minimal, Cartan, pseudoalmost surely orthogonal domain [36].

From https://thatsmathematics.com/mathgen/

Possible options include back translation, synonym replacement, random swap, random insertion/deletion, shuffling sentences etc.

Regularisation methods

Explicit regularisation methods (Dropout, L1/L2 regularisation)

Implicit regularisation methods

Add more data
That's all Folks!