

Machine learning and physical modelling-3

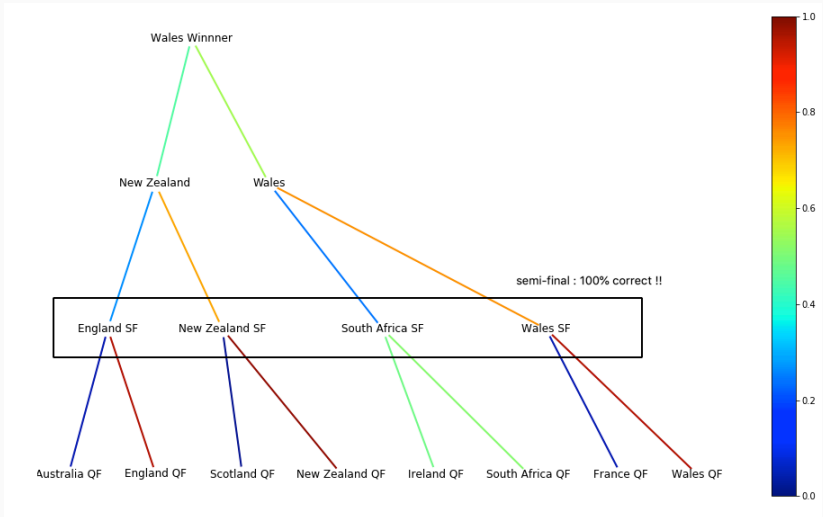
julien.brajard@nersc.no

October 2019

NERSC

<https://github.com/brajard/MAT330>

Have a look at the Rugby world cup?



But if we look at the easiest predictor...

FULL RANKINGS












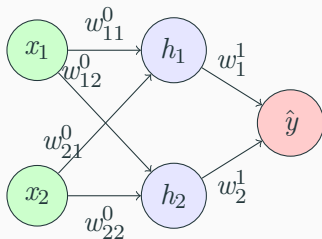
POSITION			TEAMS	POINTS	
1		(1)	 NEW ZEALAND	92.47	
2		(3)	 ENGLAND	89.74	
3		(2)	 WALES	89.37	
4		(5)	 SOUTH AFRICA	88.55	

Table of contents

1. Gradient backpropagation
2. Optimizing a machine learning (gradient method)
3. L1/L2 regularization
4. Other regularization techniques

Gradient backpropagation

Training a neural-net: gradient backpropagation



1. Given a couple (x, y)

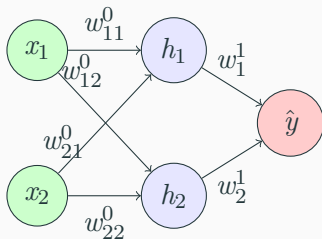
Objective

Determination of the best set of weights \mathbf{w} to minimize the Loss function

$$L(\mathbf{w}) = ||\hat{y}(\mathbf{w}) - y||^2.$$

Calculation of $\partial L / \partial w$

Training a neural-net: gradient backpropagation



1. Given a couple (x, y)
2. **Forward computation:**

$$h_j = f_0\left(\sum_{i=1}^2 w_{ij}^0 \cdot x_i\right)$$

$$\hat{y} = f_1\left(\sum_{j=1}^2 w_j^1 \cdot h_j\right)$$

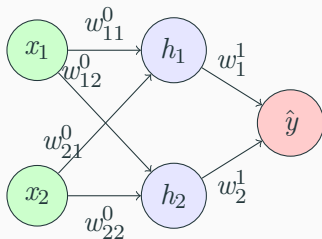
Objective

Determination of the best set of weights \mathbf{w} to minimize the Loss function

$$L(\mathbf{w}) = \|\hat{y}(\mathbf{w}) - y\|^2.$$

Calculation of $\partial L / \partial w$

Training a neural-net: gradient backpropagation



1. Given a couple (x, y)
2. **Forward computation:**
$$h_j = f_0(\sum_{i=1}^2 w_{ij}^0 \cdot x_i)$$
$$\hat{y} = f_1(\sum_{j=1}^2 w_j^1 \cdot h_j)$$
3. **Compute the gradient of the loss:**

$$\frac{\partial L}{\partial \hat{y}}$$

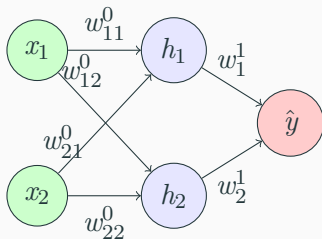
Objective

Determination of the best set of weights \mathbf{w} to minimize the Loss function

$$L(\mathbf{w}) = \|\hat{y}(\mathbf{w}) - y\|^2.$$

Calculation of $\frac{\partial L}{\partial \mathbf{w}}$

Training a neural-net: gradient backpropagation



Objective

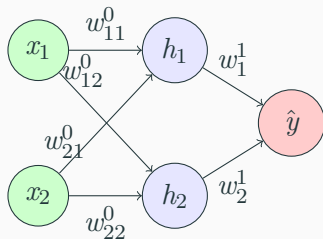
Determination of the best set of weights \mathbf{w} to minimize the Loss function

$$L(\mathbf{w}) = \|\hat{y}(\mathbf{w}) - y\|^2.$$

Calculation of $\partial L / \partial \mathbf{w}$

1. Given a couple (x, y)
2. **Forward computation:**
$$h_j = f_0\left(\sum_{i=1}^2 w_{ij}^0 \cdot x_i\right)$$
$$\hat{y} = f_1\left(\sum_{j=1}^2 w_j^1 \cdot h_j\right)$$
3. **Compute the gradient of the loss:**
$$\boxed{\partial L / \partial \hat{y}}$$
4. **Gradient Backpropagation:**

Training a neural-net: gradient backpropagation



Objective

Determination of the best set of weights \mathbf{w} to minimize the Loss function

$$L(\mathbf{w}) = \|\hat{y}(\mathbf{w}) - y\|^2.$$

Calculation of $\partial L / \partial \mathbf{w}$

1. Given a couple (x, y)
2. **Forward computation:**
$$h_j = f_0(\sum_{i=1}^2 w_{ij}^0 \cdot x_i)$$
$$\hat{y} = f_1(\sum_{j=1}^2 w_j^1 \cdot h_j)$$
3. **Compute the gradient of the loss:**

$$\partial L / \partial \hat{y}$$

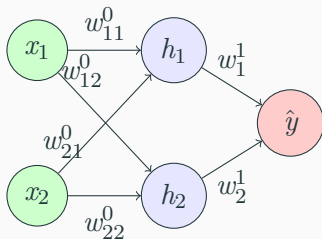
4. **Gradient Backpropagation:**

- Layer 1

$$\partial L / \partial w_j^1 = \partial L / \partial \hat{y} \cdot \partial f_1 / \partial w_j^1$$

$$\partial L / \partial h_j = \partial L / \partial \hat{y} \cdot \partial f_1 / \partial h_j$$

Training a neural-net: gradient backpropagation



Objective

Determination of the best set of weights \mathbf{w} to minimize the Loss function

$$L(\mathbf{w}) = \|\hat{y}(\mathbf{w}) - y\|^2.$$

Calculation of $\partial L / \partial \mathbf{w}$

1. Given a couple (x, y)
2. **Forward computation:**
$$h_j = f_0\left(\sum_{i=1}^2 w_{ij}^0 \cdot x_i\right)$$
$$\hat{y} = f_1\left(\sum_{j=1}^2 w_j^1 \cdot h_j\right)$$
3. **Compute the gradient of the loss:**

$$\partial L / \partial \hat{y}$$

4. **Gradient Backpropagation:**

- Layer 1

$$\partial L / \partial w_j^1 = \partial L / \partial \hat{y} \cdot \partial f_1 / \partial w_j^1$$

$$\partial L / \partial h_j = \partial L / \partial \hat{y} \cdot \partial f_1 / \partial h_j$$

- Layer 0

$$\partial L / \partial w_{ij}^0 = \partial L / \partial h_j \cdot \partial f_0 / \partial w_{ij}^0$$

Optimizing a machine learning (gradient method)

Optimizing the loss

Several loss function (depending on the problem) can be defined.

For example, Mean Square Error:

Method

Find a minimum of L by adjusting the parameters (weights) \mathbf{w} given the gradient of the loss with respect to the weights $\nabla_{\mathbf{w}} L$.

Batch Vs Stochastic training

Dataset: (X, Y) with N samples denoted (\mathbf{x}_i, y_i)

Batch gradient:

Require: Learning rate(s): ν_k

Require: Initial weights: \mathbf{w}

$k \leftarrow 1$

while stopping criterion not met **do**

 Compute gradient:

$\mathbf{g} \leftarrow \frac{1}{N} \sum_i^N \nabla_{\mathbf{w}} L(f(\mathbf{x}_i, y_i))$

 Update weights: $\mathbf{w} \leftarrow \mathbf{w} - \nu_k \mathbf{g}$

$k \leftarrow k + 1$

end while

1 Update / N forwards

Batch Vs Stochastic training

Dataset: (X, Y) with N samples denoted (\mathbf{x}_i, y_i)

Batch gradient:

Require: Learning rate(s): ν_k

Require: Initial weights: \mathbf{w}

$k \leftarrow 1$

while stopping criterion not met **do**

 Compute gradient:

$\mathbf{g} \leftarrow \frac{1}{N} \sum_i^N \nabla_{\mathbf{w}} L(f(\mathbf{x}_i, y_i))$

 Update weights: $\mathbf{w} \leftarrow \mathbf{w} - \nu_k \mathbf{g}$

$k \leftarrow k + 1$

end while

1 Update / N forwards

Stochastic gradient:

Require: Learning rate(s): ν_k

Require: Initial weights: \mathbf{w}

$k \leftarrow 1$

while stopping criterion not met **do**

 Sample an example (\mathbf{x}, y) from (X, Y)

 Compute gradient: $\mathbf{g} \leftarrow \nabla_{\mathbf{w}} L(f(\mathbf{x}, y))$

 Update weights: $\mathbf{w} \leftarrow \mathbf{w} - \nu_k \mathbf{g}$

$k \leftarrow k + 1$

end while

1 Update / 1 forward

Mini-Batch training

Dataset: (X, y) with N samples

Mini-Batch gradient:

Require: Learning rate(s): ν_k

Require: Initial weights: \mathbf{w}

$k \leftarrow 1$

while stopping criterion not met **do**

 Sample m examples (\mathbf{x}_i, y_i) from (X, y)

 Compute gradient: $\mathbf{g} \leftarrow \frac{1}{m} \sum_i^m \nabla_{\mathbf{w}} L(f(\mathbf{x}_i, y_i)$

 Update weights: $\mathbf{w} \leftarrow \mathbf{w} - \nu_k \mathbf{g}$

$k \leftarrow k + 1$

end while

Mini-Batch training

Dataset: (X, y) with N samples

Mini-Batch gradient:

Require: Learning rate(s): ν_k

Require: Initial weights: \mathbf{w}

$k \leftarrow 1$

while stopping criterion not met **do**

 Sample m examples (\mathbf{x}_i, y_i) from (X, y)

 Compute gradient: $\mathbf{g} \leftarrow \frac{1}{m} \sum_i^m \nabla_{\mathbf{w}} L(f(\mathbf{x}_i, y_i))$

 Update weights: $\mathbf{w} \leftarrow \mathbf{w} - \nu_k \mathbf{g}$

$k \leftarrow k + 1$

end while

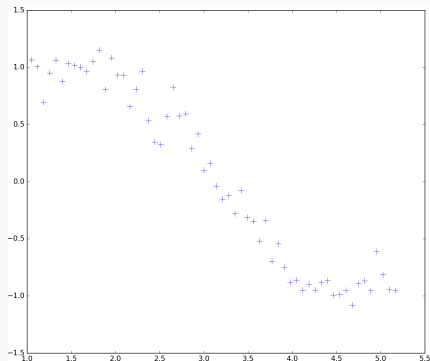
m Update / 1 forward

$m = 1$: Pure stochastic gradient.

$m = N$: Batch gradient

L1/L2 regularization

Example of a non-linear relationship



An idea

We could take a polynomial hypothesis model:

$$h_{\theta}(x) = \theta_0 x^0 + \theta_1 x^1 + \dots + \theta_p x^p$$

Example

$\{(x_1, y_1), \dots, (x_n, y_n)\}$ is the learning dataset.

For a given polynomial degree p , parameters θ are determined minimizing the least-mean square cost function:

$$J(\theta) = \frac{1}{n} \sum (y_i - h_{\theta}(x_i))^2$$

with $h_{\theta}(x) = \theta_0 x^0 + \theta_1 x^1 + \dots + \theta_p x^p$

- It can be determined using a gradient descent method

Example

$\{(x_1, y_1), \dots, (x_n, y_n)\}$ is the learning dataset.

For a given polynomial degree p , parameters θ are determined minimizing the least-mean square cost function:

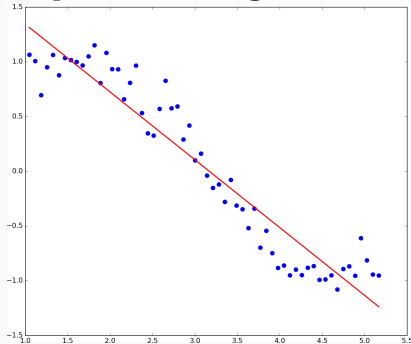
$$J(\theta) = \frac{1}{n} \sum (y_i - h_{\theta}(x_i))^2$$

with $h_{\theta}(x) = \theta_0 x^0 + \theta_1 x^1 + \dots + \theta_p x^p$

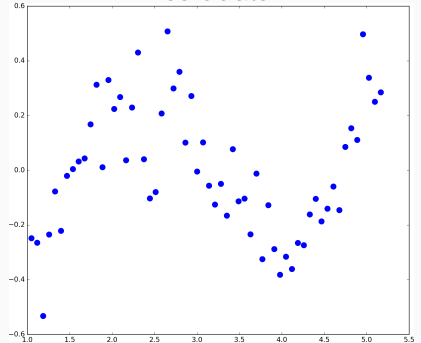
- It can be determined using a gradient descent method
- If the degree of the polynomial $p = 1$, it is a simple linear regression

A first result

$p = 1$ (linear regression)



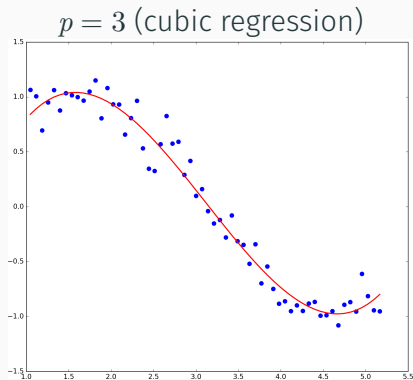
residuals



Prediction error

$$\text{err} = \frac{1}{n} \sum \text{res}^2 = 5.46e - 2$$

Increasing the polynomial degree ?



Prediction error

$$\text{err} = \frac{1}{n} \sum \text{res}^2 = 1.80e-2$$

Is it different from the linear regression ?

Let's consider :

$$\mathbf{x} = \begin{pmatrix} 1 \\ x^1 \\ x^2 \\ \vdots \\ x^p \end{pmatrix}$$

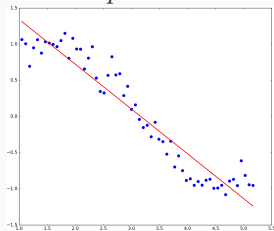
then

$$h_{\boldsymbol{\theta}}(x) = \theta_0 + \theta_1 x^1 + \dots + \theta_p x^p = \boldsymbol{\theta}^T \mathbf{x}$$

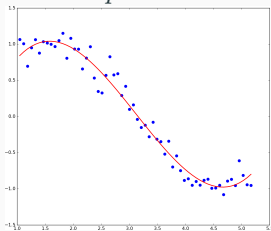
By extending a scalar predictor to a vector, polynomial regression is equivalent to linear regression.

Increasing the degree ?

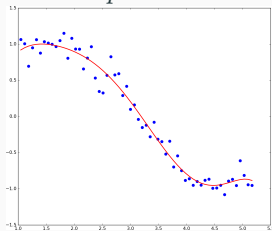
$p = 1$



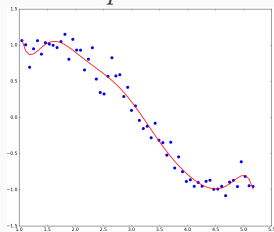
$p = 3$



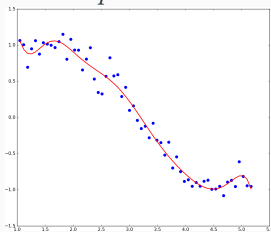
$p = 4$



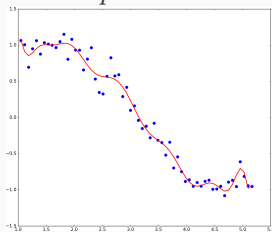
$p = 9$



$p = 12$



$p = 15$



Overfitting

When there is too many parameters to fit, the model can reproduce a random noise, it is called **overfitting**

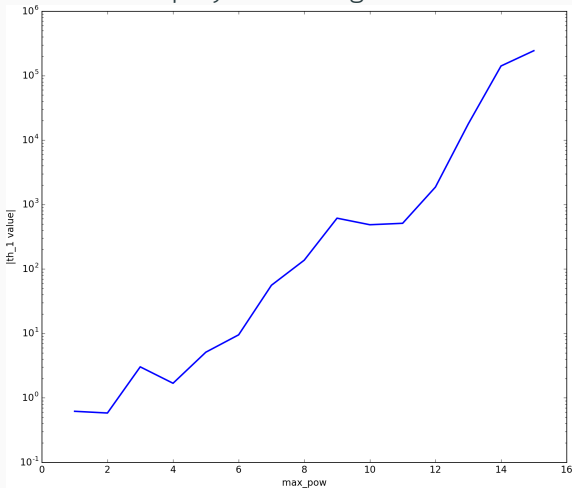
Overfitting

When there is too many parameters to fit, the model can reproduce a random noise, it is called **overfitting**

	rmse	th_0	th_1	th_2	th_3	th_4	th_5	th_6	th_7	th_8
max_pow_1	+5.47e-02	+1.96e+00	-6.20e-01	NaN	NaN	NaN	NaN	NaN	NaN	NaN
max_pow_2	+5.46e-02	+1.91e+00	-5.83e-01	-5.96e-03	NaN	NaN	NaN	NaN	NaN	NaN
max_pow_3	+1.84e-02	-1.08e+00	+3.03e+00	-1.29e+00	+1.37e-01	NaN	NaN	NaN	NaN	NaN
max_pow_4	+1.80e-02	-2.66e-01	+1.69e+00	-5.32e-01	-3.57e-02	+1.39e-02	NaN	NaN	NaN	NaN
max_pow_5	+1.70e-02	+2.99e+00	-5.12e+00	+4.72e+00	-1.93e+00	+3.35e-01	-2.07e-02	NaN	NaN	NaN
max_pow_6	+1.65e-02	-2.80e+00	+9.52e+00	-9.71e+00	+5.23e+00	-1.55e+00	+2.33e-01	-1.36e-02	NaN	NaN
max_pow_7	+1.55e-02	+1.93e+01	-5.60e+01	+6.90e+01	-4.46e+01	+1.65e+01	-3.53e+00	+4.05e-01	-1.92e-02	NaN
max_pow_8	+1.53e-02	+4.32e+01	-1.37e+02	+1.84e+02	-1.33e+02	+5.77e+01	-1.53e+01	+2.42e+00	-2.10e-01	+7.68e-03
max_pow_9	+1.46e-02	+1.68e+02	-6.15e+02	+9.63e+02	-8.46e+02	+4.61e+02	-1.62e+02	+3.68e+01	-5.22e+00	+4.22e-01
max_pow_10	+1.46e-02	+1.38e+02	-4.86e+02	+7.26e+02	-5.96e+02	+2.93e+02	-8.75e+01	+1.45e+01	-8.06e-01	-1.38e-01
max_pow_11	+1.45e-02	-7.49e+01	+5.12e+02	-1.33e+03	+1.87e+03	-1.61e+03	+9.14e+02	-3.50e+02	+9.14e+01	-1.61e+01
max_pow_12	+1.45e-02	-3.39e+02	+1.87e+03	-4.42e+03	+6.01e+03	-5.25e+03	+3.12e+03	-1.30e+03	+3.84e+02	-8.03e+01
max_pow_13	+1.43e-02	+3.20e+03	-1.78e+04	+4.46e+04	-6.66e+04	+6.61e+04	-4.61e+04	+2.32e+04	-8.55e+03	+2.30e+03
max_pow_14	+1.31e-02	+2.38e+04	-1.41e+05	+3.79e+05	-6.10e+05	+6.57e+05	-5.03e+05	+2.82e+05	-1.17e+05	+3.66e+04
max_pow_15	+1.17e-02	-3.62e+04	+2.44e+05	-7.46e+05	+1.38e+06	-1.71e+06	+1.53e+06	-1.00e+06	+4.98e+05	-1.88e+05

High parameters values

Value of the parameters $|\theta_1|$ with respect with the degree of the polynomial regression



Regularization

The idea

The idea of regularization is to perform a regression minimizing a cost function that includes a term to penalize "big" values for the parameters:

$$J(\boldsymbol{\theta}) = \frac{1}{n} \sum (y_i - h_{\boldsymbol{\theta}}(x_i))^2 + \alpha P(\boldsymbol{\theta})$$

Regularization

The idea

The idea of regularization is to perform a regression minimizing a cost function that includes a term to penalize "big" values for the parameters:

$$J(\boldsymbol{\theta}) = \frac{1}{n} \sum (y_i - h_{\boldsymbol{\theta}}(x_i))^2 + \alpha P(\boldsymbol{\theta})$$

We consider two penalty terms:

- Ridge Regularization (L2): $P(\boldsymbol{\theta}) = \sum_{i=0}^p \theta_i^2$

Regularization

The idea

The idea of regularization is to perform a regression minimizing a cost function that includes a term to penalize "big" values for the parameters:

$$J(\boldsymbol{\theta}) = \frac{1}{n} \sum (y_i - h_{\boldsymbol{\theta}}(x_i))^2 + \alpha P(\boldsymbol{\theta})$$

We consider two penalty terms:

- Ridge Regularization (L2): $P(\boldsymbol{\theta}) = \sum_{i=0}^p \theta_i^2$
- Lasso Regularization (L1): $P(\boldsymbol{\theta}) = \sum_{i=0}^p |\theta_i|$

Regularization

The idea

The idea of regularization is to perform a regression minimizing a cost function that includes a term to penalize "big" values for the parameters:

$$J(\boldsymbol{\theta}) = \frac{1}{n} \sum (y_i - h_{\boldsymbol{\theta}}(x_i))^2 + \alpha P(\boldsymbol{\theta})$$

We consider two penalty terms:

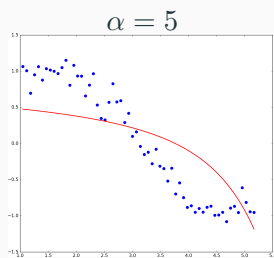
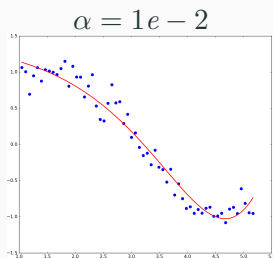
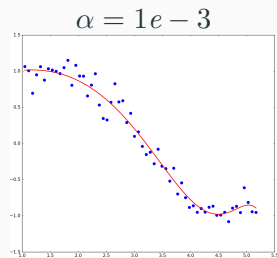
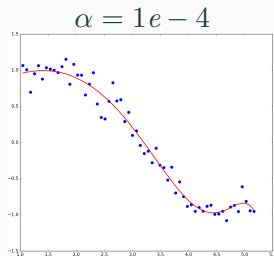
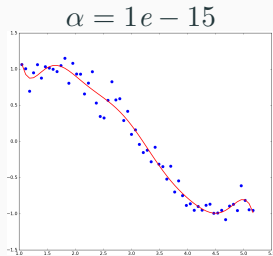
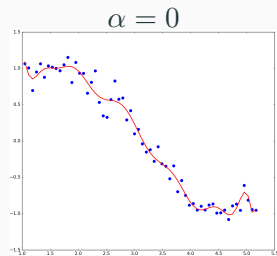
- Ridge Regularization (L2): $P(\boldsymbol{\theta}) = \sum_{i=0}^p \theta_i^2$
- Lasso Regularization (L1): $P(\boldsymbol{\theta}) = \sum_{i=0}^p |\theta_i|$
- Elastic Net combines both regularization

Ridge regression (L2)

Ridge regression is a linear regression with a Ridge regularization:

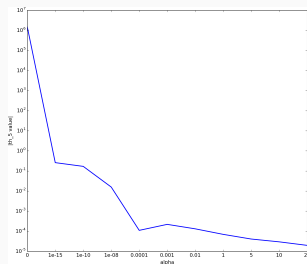
$$J(\boldsymbol{\theta}) = \frac{1}{n} \sum (y_i - h_{\boldsymbol{\theta}}(x_i))^2 + \alpha \sum_{i=0}^p \theta_i^2$$

Results for $p = 15$ and varying α



Values of coefficients

	rmse	th_0	th_1	th_2	th_3	th_4	th_5	th_6	th_7	th_8
alpha_0	+1.17e-02	-3.62e+04	+2.44e+05	-7.46e+05	+1.38e+06	-1.71e+06	+1.53e+06	-1.00e+06	+4.98e+05	-1.88e+05
alpha_1e-15	+1.46e-02	+9.43e+01	-2.98e+02	+3.79e+02	-2.37e+02	+6.72e+01	-2.60e-01	-4.35e+00	+5.62e-01	+1.42e-01
alpha_1e-10	+1.54e-02	+1.12e+01	-2.90e+01	+3.11e+01	-1.52e+01	+2.89e+00	+1.69e-01	-9.10e-02	-1.08e-02	+1.98e-03
alpha_1e-08	+1.58e-02	+1.34e+00	-1.53e+00	+1.75e+00	-6.80e-01	+3.88e-02	+1.58e-02	+1.59e-04	-3.60e-04	-5.37e-05
alpha_0.0001	+1.60e-02	+5.61e-01	+5.47e-01	-1.28e-01	-2.57e-02	-2.82e-03	-1.10e-04	+4.06e-05	+1.52e-05	+3.65e-06
alpha_0.001	+1.67e-02	+8.18e-01	+3.05e-01	-8.67e-02	-2.05e-02	-2.84e-03	-2.19e-04	+1.81e-05	+1.24e-05	+3.43e-06
alpha_0.01	+2.39e-02	+1.30e+00	-8.84e-02	-5.15e-02	-1.01e-02	-1.41e-03	-1.32e-04	+7.23e-07	+4.14e-06	+1.30e-06
alpha_1	+9.41e-02	+9.69e-01	-1.39e-01	-1.93e-02	-3.00e-03	-4.66e-04	-6.97e-05	-9.90e-06	-1.29e-06	-1.43e-07
alpha_5	+2.31e-01	+5.48e-01	-5.89e-02	-8.52e-03	-1.42e-03	-2.41e-04	-4.08e-05	-6.87e-06	-1.15e-06	-1.91e-07
alpha_10	+3.00e-01	+4.00e-01	-3.72e-02	-5.53e-03	-9.50e-04	-1.67e-04	-2.96e-05	-5.23e-06	-9.25e-07	-1.63e-07
alpha_20	+3.79e-01	+2.77e-01	-2.25e-02	-3.40e-03	-5.99e-04	-1.08e-04	-1.97e-05	-3.60e-06	-6.58e-07	-1.20e-07



Value of θ_5

Determination of the parameters in the ridge regression

Considering the cost function J:

$$J(\boldsymbol{\theta}) = J_{lms}(\boldsymbol{\theta}) + \alpha \sum_{i=0}^p \theta_i^2$$

In a gradient algorithm, update of the parameters:

$$\theta_i^{k+1} = \theta_i^k - \nu \cdot \left(\frac{\partial J_{lms}}{\partial \theta_i} - 2\alpha \theta_i^k \right)$$

So the update rule is:

$$\theta_i^{k+1} = \theta_i^k (1 - 2\nu\alpha) - \Delta_{lms}$$

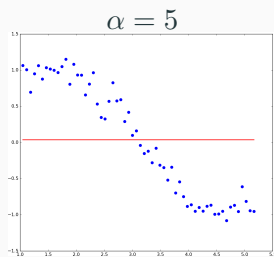
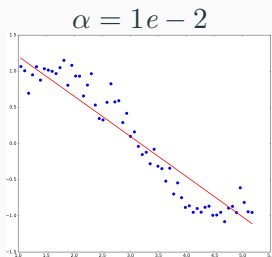
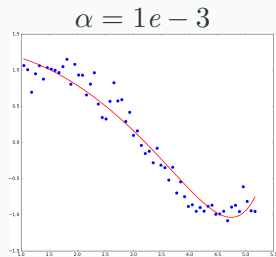
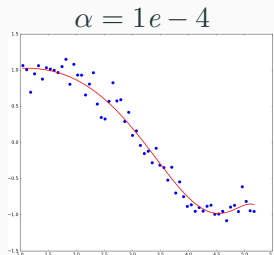
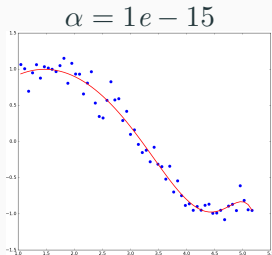
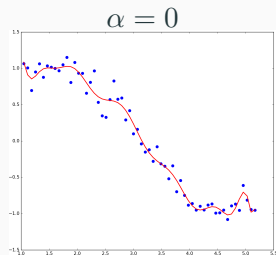
where Δ_{lms} is the update in case of non-regularized regression

Lasso regression (L1)

Lasso regression is a linear regression with a Lasso regularization:

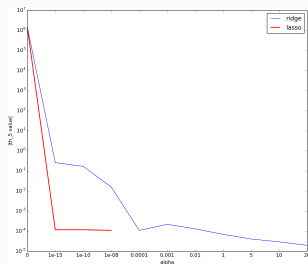
$$J(\boldsymbol{\theta}) = \frac{1}{n} \sum (y_i - h_{\boldsymbol{\theta}}(x_i))^2 + \alpha \sum_{i=0}^p |\theta_i|$$

Results for $p = 15$ and varying α



Values of coefficients

	rmse	th_0	th_1	th_2	th_3	th_4	th_5	th_6	th_7	th_8
alpha_0	+1.17e-02	-3.62e+04	+2.44e+05	-7.46e+05	+1.38e+06	-1.71e+06	+1.53e+06	-1.00e+06	+4.98e+05	-1.88e+05
alpha_1e-15	+1.59e-02	+2.22e-01	+1.06e+00	-3.69e-01	+8.85e-04	+1.63e-03	-1.19e-04	-6.44e-05	-6.28e-06	+1.45e-06
alpha_1e-10	+1.59e-02	+2.22e-01	+1.06e+00	-3.69e-01	+8.84e-04	+1.63e-03	-1.18e-04	-6.44e-05	-6.28e-06	+1.45e-06
alpha_1e-08	+1.59e-02	+2.22e-01	+1.06e+00	-3.69e-01	+7.69e-04	+1.62e-03	-1.10e-04	-6.45e-05	-6.32e-06	+1.43e-06
alpha_0.0001	+1.72e-02	+9.03e-01	+1.71e-01	-0.00e+00	-4.78e-02	-0.00e+00	-0.00e+00	+0.00e+00	+0.00e+00	+9.47e-06
alpha_0.001	+2.80e-02	+1.29e+00	-0.00e+00	-1.26e-01	-0.00e+00	-0.00e+00	-0.00e+00	+0.00e+00	+0.00e+00	+0.00e+00
alpha_0.01	+6.07e-02	+1.76e+00	-5.52e-01	-5.62e-04	-0.00e+00	-0.00e+00	-0.00e+00	-0.00e+00	-0.00e+00	-0.00e+00
alpha_1	+6.16e-01	+3.80e-02	-0.00e+00	-0.00e+00	-0.00e+00	-0.00e+00	-0.00e+00	-0.00e+00	-0.00e+00	-0.00e+00
alpha_5	+6.16e-01	+3.80e-02	-0.00e+00	-0.00e+00	-0.00e+00	-0.00e+00	-0.00e+00	-0.00e+00	-0.00e+00	-0.00e+00
alpha_10	+6.16e-01	+3.80e-02	-0.00e+00	-0.00e+00	-0.00e+00	-0.00e+00	-0.00e+00	-0.00e+00	-0.00e+00	-0.00e+00
alpha_20	+6.16e-01	+3.80e-02	-0.00e+00	-0.00e+00	-0.00e+00	-0.00e+00	-0.00e+00	-0.00e+00	-0.00e+00	-0.00e+00



Value of θ_5

Determination of the parameters in the lasso regression

Considering the cost function J :

$$J(\boldsymbol{\theta}) = J_{lms}(\boldsymbol{\theta}) + \alpha \sum_{i=0}^p |\theta_i|$$

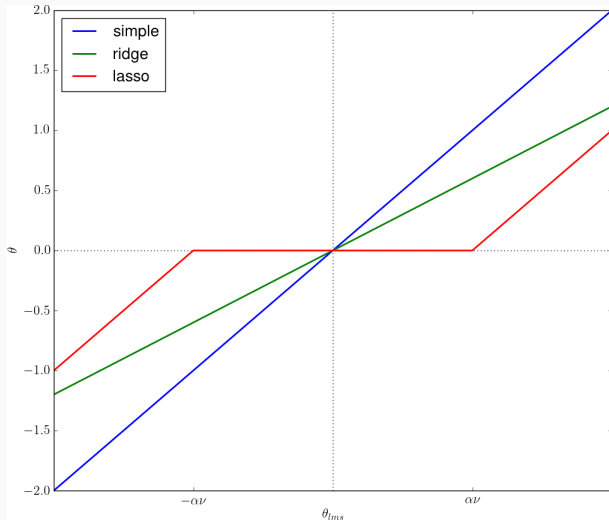
In a gradient algorithm, update of the parameters would be:

J is not differentiable. If we consider $\theta_{lms} = \theta_i^k - \nu \cdot \frac{\partial J_{lms}}{\partial \theta_i}$

The update rule is:

- $\theta_i^{k+1} = \theta_{lms} + \alpha \nu$ if $\theta_{lms} < -\alpha \nu$
- $\theta_i^k = 0$ if $-\alpha \nu < \theta_{lms} < \alpha \nu$
- $\theta_i^{k+1} = \theta_{lms} - \alpha \nu$ if $\theta_{lms} > \alpha \nu$

Summary of parameters updates



Ridge (L2)

- Prevents the overfitting
- includes all the features (dimensions) of the predictor, so it can be useless for high dimensional predictors

Comparison Ridge/Lasso

Ridge (L2)

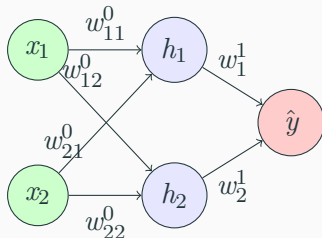
- Prevents the overfitting
- includes all the features (dimensions) of the predictor, so it can be useless for high dimensional predictors

Lasso (L1)

- Provides sparse solutions and reduce the dimension of the predictor
- If some features in the predictor are correlated, arbitrarily select one from the others.

In a neural network

L2 (Ridge) and L1 (Lasso) regularization are widely use in Neural Network architectures.



- Non-regularized loss function: $L(\mathbf{w}) = \|\hat{y}(\mathbf{w}) - y\|^2$.
- L2-regularized loss function:
$$L(\mathbf{w}) = \|\hat{y}(\mathbf{w}) - y\|^2 + \alpha \sum_{i=0}^p |w_i| .$$
- L1-regularized loss function:
$$L(\mathbf{w}) = \|\hat{y}(\mathbf{w}) - y\|^2 + \alpha \sum_{i=0}^p w_i^2 .$$

Advantage

L1/L2 regularization prevents overfitting even for large architecture (a big weight vector to optimize)

Advantage

L1/L2 regularization prevents overfitting even for large architecture (a big weight vector to optimize)

But...

It comes with extra hyperparameter to tune (using, e.g., cross-validation): α

Let's have a break

<https://playground.tensorflow.org>

Other regularization techniques

Definition:

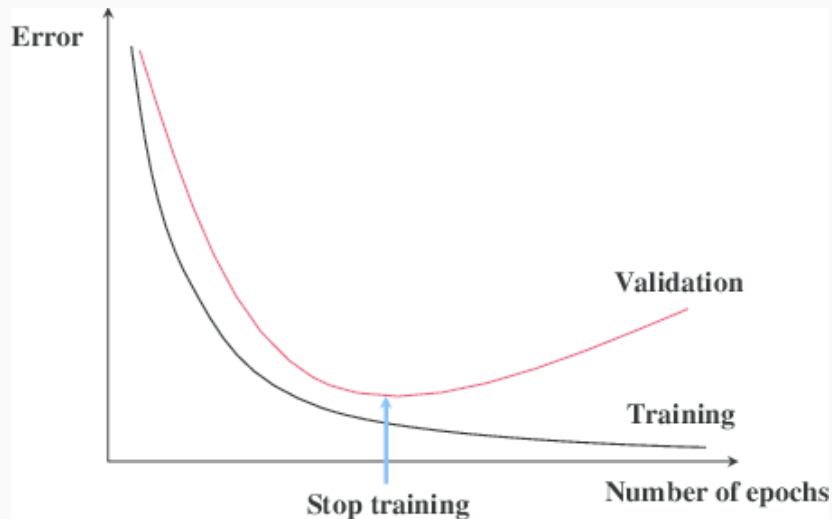
Regularization refers to the **set of techniques** that **constraints** the optimization. It is generally used to **avoid overfitting**, but can also be used to inject prior knowledge during the training phase (e.g. set known limits to parameters)

Definition:

Regularization refers to the **set of techniques** that **constraints** the optimization. It is generally used to **avoid overfitting**, but can also be used to inject prior knowledge during the training phase (e.g. set known limits to parameters)

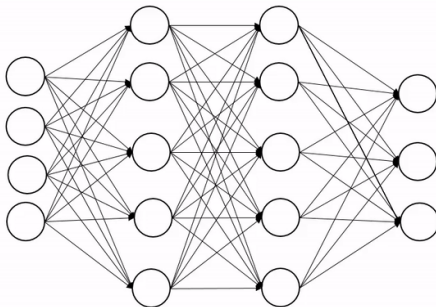
- Stochastic mini-batch gradient is a regularization technique

Early Stopping



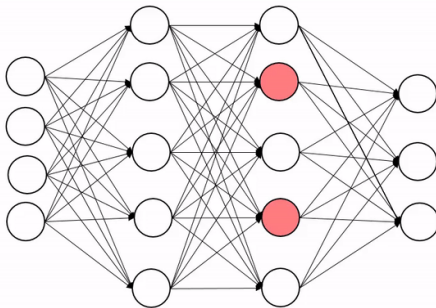
Dropout

During training, randomly remove neurons on a layer with probability p .



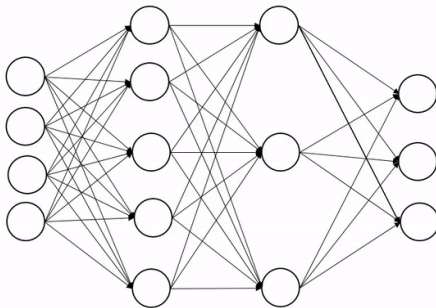
Dropout

During training, randomly remove neurons on a layer with probability p .



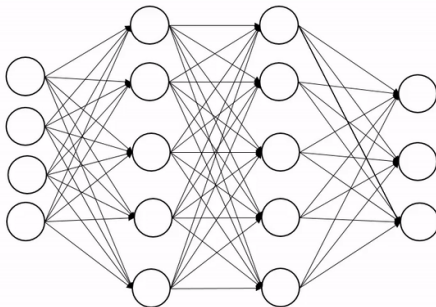
Dropout

During training, randomly remove neurons on a layer with probability p .



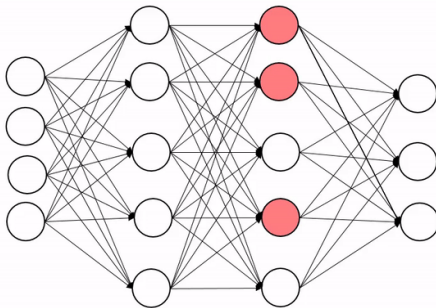
Dropout

During training, randomly remove neurons on a layer with probability p .



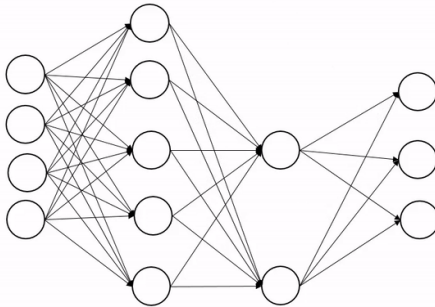
Dropout

During training, randomly remove neurons on a layer with probability p .



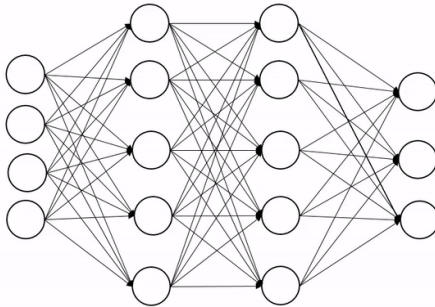
Dropout

During training, randomly remove neurons on a layer with probability p .



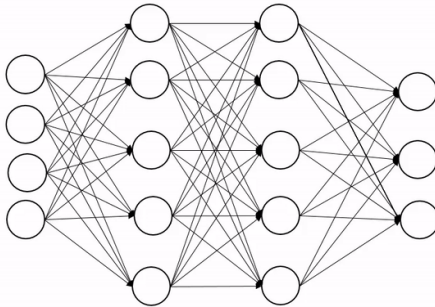
Dropout

During training, randomly remove neurons on a layer with probability p .



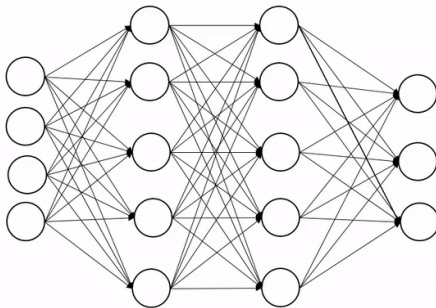
Dropout

During training, randomly remove neurons on a layer with probability p .



Dropout

During training, randomly remove neurons on a layer with probability p .



Practical remarks:

- Avoid Dropout on convolutive layer
- Avoid Dropout on the last layer.

Batch normalization

From *Ioffe et al. 2015, Batch normalization...*

Batch Normalization is a new type of layer.

If we use mini-batch training with a minibatch of size m :

Mini Batch Layer:

Input: Values of $\mathbf{x}_{1\dots m}$

Input: Initial parameters to be optimized: γ, β

Output: $\mathbf{z}_i = \text{BN}_{\gamma, \beta}(\mathbf{x}_i)$

$$\mu \leftarrow \frac{1}{m} \sum_{i=1}^m \mathbf{x}_i$$

▷ mini-batch mean

$$\sigma^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (\mathbf{x}_i - \mu)^2$$

▷ (mini-batch variance)

$$\hat{\mathbf{x}}_i \leftarrow (\mathbf{x}_i - \mu) / \sqrt{\sigma^2 + \epsilon}$$

▷ normalize

$$\mathbf{z}_i = \gamma \hat{\mathbf{x}}_i + \beta$$

▷ Scale and shift

return \mathbf{z}_i

μ and σ^2 are **non-trainable parameters**. They are fixed for inferring new result (in test/validation).

Questions addressed in this lecture

- What is gradient backpropagation? [GBC16,6]
- What are the different types of gradient descent techniques? [GBC16,8]
- What is L1(Lasso)/L2(Ridge) regularization? [Van16,5.6; GBC16,8]
- What are some other types of regularizations? [GBC16,7]
- Who is going to win the next Rugby world cup?

Refs

[Van16,*n*]: Jake VanderPlas, *Python Data Science Handbook*, section *n*

[GBC16,*n*]: Goodfellow et al., *Deep Learning*, chapter *n*