Machine learning and physical modelling-3

julien.brajard@nersc.no

October 2021

NERSC

https://github.com/brajard/MAT330

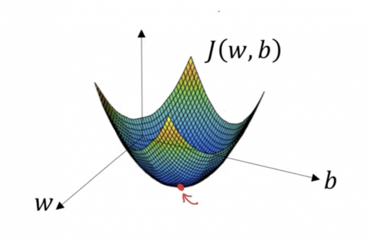
Table of contents

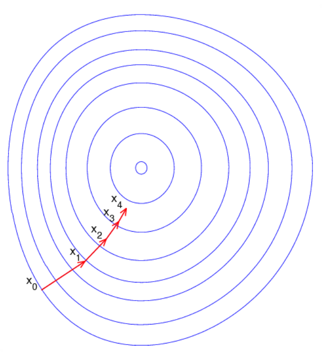
Optimization using gradient descent

Objective

Minimize the function $L(\boldsymbol{\theta})$ where $\boldsymbol{\theta}$ is a vector of parameters (e.g. weights of a neural net).

Example with $\boldsymbol{\theta} = (w, b)$





Iterative algorithm

1. We start with a ”first guess” of the parameter $\boldsymbol{\theta}_0$

Iterative algorithm

1. We start with a ”first guess” of the parameter $\boldsymbol{\theta}_0$
2. we iterate over several values of the parameters following the update rule:

   $$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \gamma \nabla L(\boldsymbol{\theta}_k),$$

   γ is called the learning rate.

Iterative algorithm

1. We start with a "first guess" of the parameter $\boldsymbol{\theta}_0$
2. we iterate over several values of the parameters following the update rule:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \gamma \nabla L(\boldsymbol{\theta}_k),$$

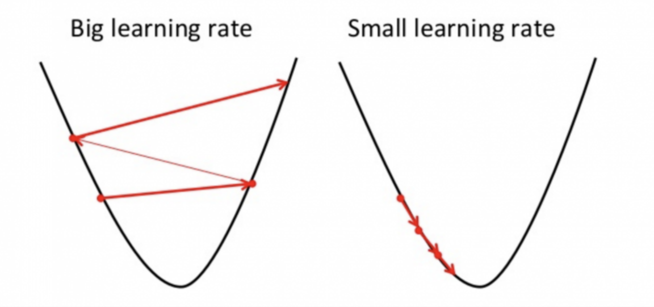
   γ is called the learning rate.

Iterative algorithm

1. We start with a "first guess" of the parameter $\boldsymbol{\theta}_0$
2. we iterate over several values of the parameters following the update rule:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \gamma \nabla L(\boldsymbol{\theta}_k),$$

γ is called the learning rate.



Few comments

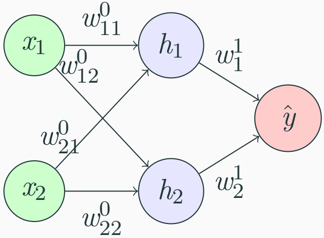
- The learning γ is an important hyperparameter that needs to be tuned using, e.g random search

Few comments

- The learning γ is an important hyperparameter that needs to be tuned using, e.g random search
- The key part of the formula is the computation of the gradient $\nabla L(\boldsymbol{\theta}_k)$

Gradient backpropagation

Training a neural-net: gradient backpropagation

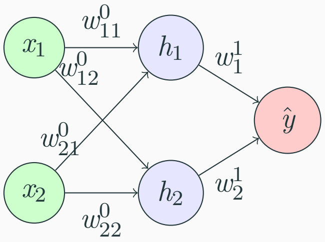


Objective

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 $\partial L / \partial w$

1. Given a couple (x, y)

Training a neural-net: gradient backpropagation

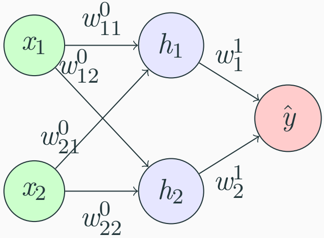


1. Given a couple (x, y)
2. Forward computation:
   $h_j = f_0(\sum_{i=1}^2 w_{ij}^0 . x_i)$
   $\hat{y} = f_1(\sum_{j=1}^2 w_j^1 . h_j)$

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

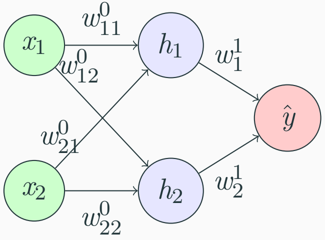


1. Given a couple (x, y)
2. Forward computation:
   $h_j = f_0(\sum_{i=1}^{2} w^0_{ij}.x_i)$
   $\hat{y} = f_1(\sum_{j=1}^{2} w^1_j.h_j)$
3. Compute the gradient of the loss:
   $\boxed{\partial L/\partial \hat{y}}$

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 $\partial L/\partial w$

Training a neural-net: gradient backpropagation

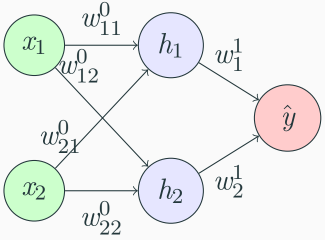


Objective

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 $\partial L/\partial w$

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

Training a neural-net: gradient backpropagation



Objective

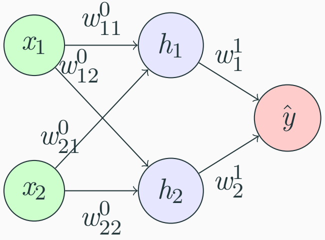
Determination of the best set of weights $\mathbf{w}$ to minimize the Loss function

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

Calculation of $\partial L/\partial w$

1. Given a couple (x, y)
2. Forward computation:
   $h_j = f_0(\sum_{i=1}^{2} w^0_{ij}.x_i)$
   $\hat{y} = f_1(\sum_{j=1}^{2} w^1_j.h_j)$
3. Compute the gradient of the loss:
   $\boxed{\partial L/\partial \hat{y}}$
4. Gradient Backpropagation:
   - Layer 1
     $\partial L/\partial w^1_j = \boxed{\partial L/\partial \hat{y}}.\partial f_1/\partial w^1_j$
     $\boxed{\partial L/\partial h_j} = \boxed{\partial L/\partial \hat{y}}.\partial f_1/\partial h_j$

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 $\partial L / \partial w$

1. Given a couple (x, y)
2. Forward computation:
   $h_j = f_0(\sum_{i=1}^{2} w^0_{ij}.x_i)$
   $\hat{y} = f_1(\sum_{j=1}^{2} w^1_j.h_j)$
3. Compute the gradient of the loss:
   $\boxed{\partial L / \partial \hat{y}}$
4. Gradient Backpropagation:
   - Layer 1
     $\partial L / \partial w^1_j = \boxed{\partial L / \partial \hat{y}} . \partial f_1 / \partial w^1_j$
     $\boxed{\partial L / \partial h_j} = \boxed{\partial L / \partial \hat{y}} . \partial f_1 / \partial h_j$
   - Layer 0
     $\partial L / \partial w^0_{ij} = \boxed{\partial L / \partial h_j} . \partial f_1 / \partial w^0_{ij}$

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 adjustig 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

1 Update / m forward

$m = 1$: Pure stochastic gradient.

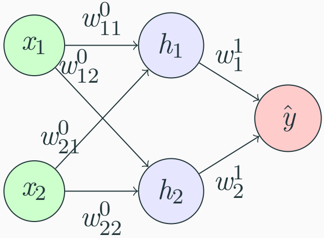
$m = N$: Batch gradient

Let's have a break

`https://playground.tensorflow.org`

Gradient backpropagation

Training a neural-net: gradient backpropagation

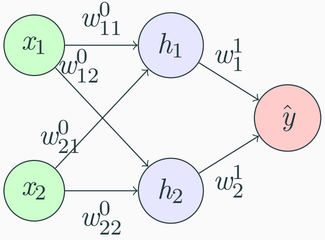


Objective

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 $\partial L / \partial w$

1. Given a couple (x, y)

Training a neural-net: gradient backpropagation

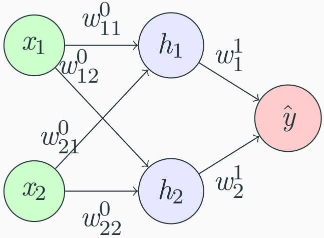


1. Given a couple (x, y)
2. Forward computation:
   $h_j = f_0(\sum_{i=1}^{2} w^0_{ij}.x_i)$
   $\hat{y} = f_1(\sum_{j=1}^{2} w^1_j.h_j)$

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

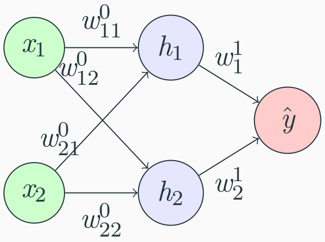


Objective

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 $\partial L/\partial w$

1. Given a couple (x, y)
2. Forward computation:
   $h_j = f_0(\sum_{i=1}^2 w^0_{ij}.x_i)$
   $\hat{y} = f_1(\sum_{j=1}^2 w^1_j.h_j)$
3. Compute the gradient of the loss:
   $\boxed{\partial L/\partial \hat{y}}$

Training a neural-net: gradient backpropagation



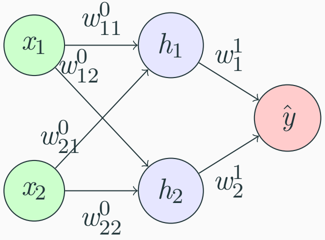
Objective

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 $\partial L / \partial w$

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

Training a neural-net: gradient backpropagation

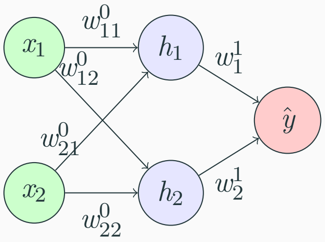


Objective

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 $\partial L / \partial w$

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

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 $\partial L/\partial w$

1. Given a couple (x, y)
2. Forward computation:
   $h_j = f_0(\sum_{i=1}^2 w^0_{ij}.x_i)$
   $\hat{y} = f_1(\sum_{j=1}^2 w^1_j.h_j)$
3. Compute the gradient of the loss:
   $\boxed{\partial L/\partial \hat{y}}$
4. Gradient Backpropagation:
   - Layer 1
     $\partial L/\partial w^1_j = \boxed{\partial L/\partial \hat{y}}.\partial f_1/\partial w^1_j$
     $\boxed{\partial L/\partial h_j} = \boxed{\partial L/\partial \hat{y}}.\partial f_1/\partial h_j$
   - Layer 0
     $\partial L/\partial w^0_{ij} = \boxed{\partial L/\partial h_j}.\partial f_1/\partial w^0_{ij}$

Probabilistic interpretation

Maximum likelihood estimator and loss

We can assume that the observation y follows a Gaussian law:

$$p(y/x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp -\frac{(y-\mu(x))^2}{2\sigma^2},$$

where x is observed and $\mu(x)$ is a function of x.
Given a set of samples $(x_k, y_k)_{1:K}$, the negative log-likelihood is defined by

$$L = \sum_{k=1}^{K} \left(\frac{\log 2\pi\sigma^2}{2} + \frac{(y_k - \mu(x_k))^2}{2\sigma^2} \right)$$

Minimizing L is maximizing the probability of having the observations y_k given x_k.

Loss function of a neural net

First case: σ is constant

$\mu(x)$ is parametrized by a neural net $G_\mu(x, \boldsymbol{\theta}_\mu)$
The Maximum likelihood estimator is found by minizming:

$$L(\boldsymbol{\theta}_\mu) = \sum_{k=1}^{K} (y_k - G_\mu(x, \boldsymbol{\theta}_\mu))^2$$

which is exactly the regression loss already introduced.

Loss function of a neural net

First case: σ is constant

$\mu(x)$ is parametrized by a neural net $G_\mu(x, \boldsymbol{\theta}_\mu)$
The Maximum likelihood estimator is found by minizming:

$$L(\boldsymbol{\theta}_\mu) = \sum_{k=1}^{K} (y_k - G_\mu(x, \boldsymbol{\theta}_\mu))^2$$

which is exactly the regression loss already introduced.

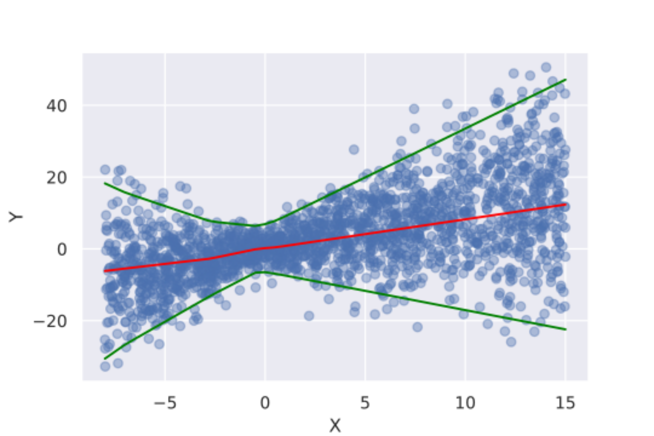
Second case: $\sigma(x)$ is a function of x.

In addition to $G_\mu(x, \boldsymbol{\theta}_\mu)$, $\sigma(x)$ is parametrized by $G_\sigma(x, \boldsymbol{\theta}_\sigma)$. The loss to minimize is then:

$$L(\boldsymbol{\theta}_\mu, \boldsymbol{\theta}_\sigma) = \sum_{k=1}^{K} \left(\frac{\log 2\pi G_\sigma(x_k, \boldsymbol{\theta}_\sigma)^2}{2} + \frac{(y_k - G_\mu(x_k, \boldsymbol{\theta}_\mu))^2}{2 G_\sigma(x_k, \boldsymbol{\theta}_\sigma)^2} \right)$$

The neural net $G = (G_\mu, G_\sigma)$ gives also the uncertainty of its estimation in the form of the standard deviation.

Illustration



In red the estimation of the mean $G_\mu(x, \boldsymbol{\theta}_\mu)$

In green the confidence interval $G_\mu(x, \boldsymbol{\theta}_\mu) \pm \sigma(x_k, \boldsymbol{\theta}_\sigma)$

Other regularization techniques

Regularization

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)

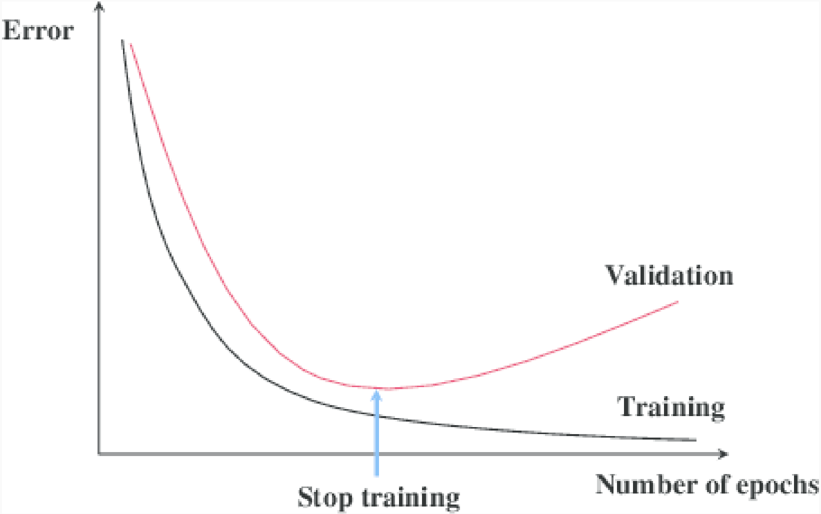
Regularization

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)

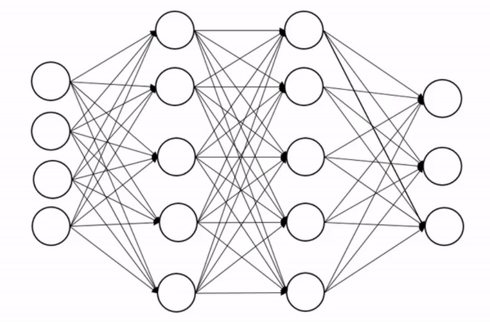
- Stochastic mini-batch gradient is a regularization technique

Early Stopping



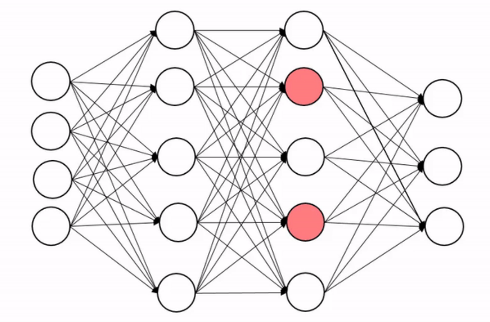
Dropout

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



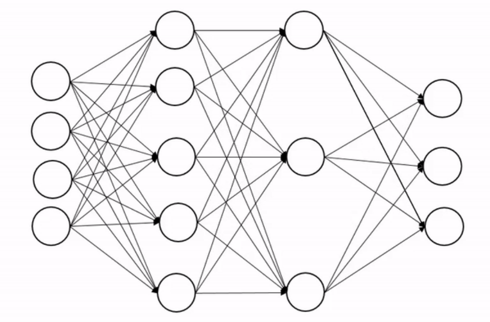
Dropout

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



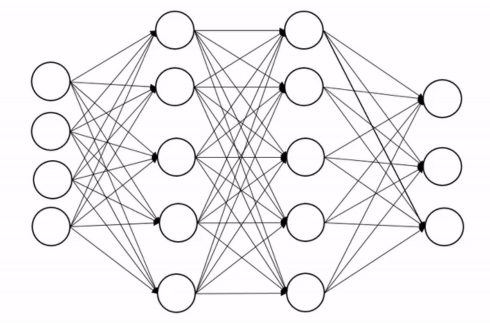
Dropout

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



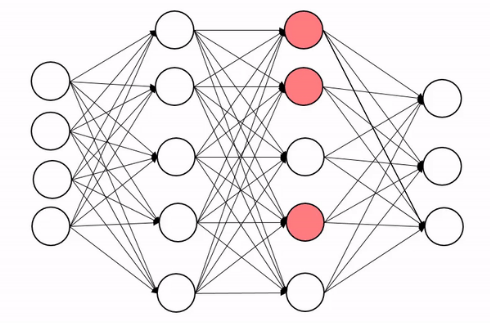
Dropout

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



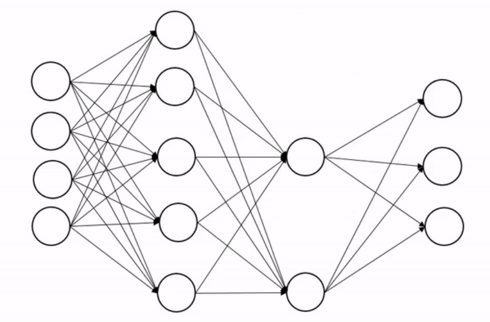
Dropout

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



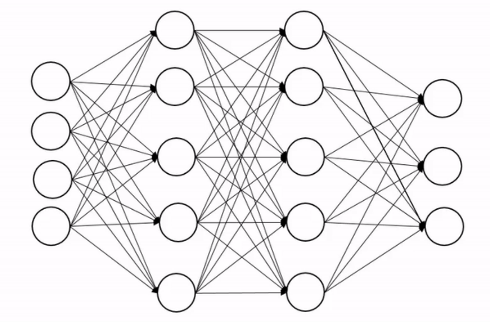
Dropout

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



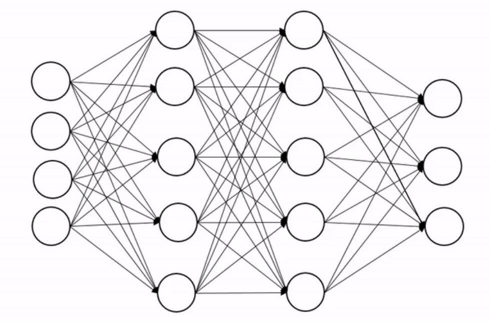
Dropout

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



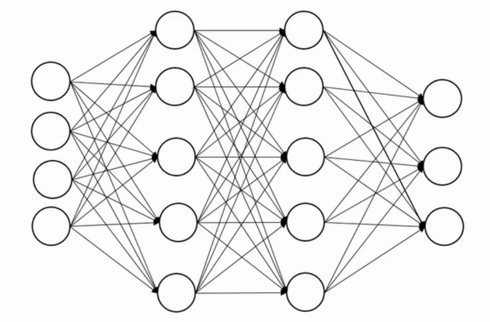
Dropout

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 normalizaion...*

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 $\mathbf{x_{1\ldots m}}$

Input: Initial parameters to be optimized: γ, β

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

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

Link with data assimilation

Data Assimilation

Example of BLUE: Best Linear Unbiased Estimator

Given a state vector $\mathbf{x} \in \mathbb{R}^n$ and a data vector $\mathbf{d} \in \mathbb{R}^m$:

$$\begin{aligned} &\mathbf{x}^{\mathrm{f}} = \mathbf{x}^{\mathrm{t}} + \mathbf{p}, && \overline{\mathbf{p}} = 0, && \overline{\mathbf{p}\mathbf{p}^T} = \mathbf{C}_{xx}. \\ &\mathbf{d} = \mathbf{H}\mathbf{x}^{\mathrm{t}} + \boldsymbol{\epsilon}, && \overline{\boldsymbol{\epsilon}} = 0, && \overline{\boldsymbol{\epsilon}\boldsymbol{\epsilon}^T} = \mathbf{C}_{\epsilon\epsilon}. \end{aligned}$$

Example of BLUE: Best Linear Unbiased Estimator

Given a state vector $\mathbf{x} \in \mathbb{R}^n$ and a data vector $\mathbf{d} \in \mathbb{R}^m$:

$$\mathbf{x}^{\mathrm{f}} = \mathbf{x}^{\mathrm{t}} + \mathbf{p}, \qquad \overline{\mathbf{p}} = 0, \qquad \overline{\mathbf{p}\mathbf{p}^T} = \mathbf{C}_{xx}.$$
$$\mathbf{d} = \mathbf{H}\mathbf{x}^{\mathrm{t}} + \boldsymbol{\epsilon}, \qquad \overline{\boldsymbol{\epsilon}} = 0, \qquad \overline{\boldsymbol{\epsilon}\boldsymbol{\epsilon}^T} = \mathbf{C}_{\epsilon\epsilon}.$$

Estimating $\mathbf{x}^{\mathrm{t}}$ by minimizing the estimation error leads to minizing the following function:

$$\mathcal{J}(\mathbf{x}) = (\mathbf{d} - \mathbf{H}\mathbf{x})^T \mathbf{C}_{\epsilon\epsilon}^{-1} (\mathbf{d} - \mathbf{H}\mathbf{x}) + (\mathbf{x} - \mathbf{x}^{\mathrm{f}})^T \mathbf{C}_{xx}^{-1} (\mathbf{x} - \mathbf{x}^{\mathrm{f}})$$

Example of BLUE: Best Linear Unbiased Estimator

Given a state vector $\mathbf{x} \in \mathbb{R}^n$ and a data vector $\mathbf{d} \in \mathbb{R}^m$:

$$\mathbf{x}^{\mathrm{f}} = \mathbf{x}^{\mathrm{t}} + \mathbf{p}, \qquad \overline{\mathbf{p}} = 0, \qquad \overline{\mathbf{p}\mathbf{p}^T} = \mathbf{C}_{xx}.$$

$$\mathbf{d} = \mathbf{H}\mathbf{x}^{\mathrm{t}} + \boldsymbol{\epsilon}, \qquad \overline{\boldsymbol{\epsilon}} = 0, \qquad \overline{\boldsymbol{\epsilon}\boldsymbol{\epsilon}^T} = \mathbf{C}_{\epsilon\epsilon}.$$

Estimating $\mathbf{x}^{\mathrm{t}}$ by minimizing the estimation error leads to minizing the following function:

$$\mathcal{J}(\mathbf{x}) = (\mathbf{d} - \mathbf{H}\mathbf{x})^T \mathbf{C}_{\epsilon\epsilon}^{-1} (\mathbf{d} - \mathbf{H}\mathbf{x}) + (\mathbf{x} - \mathbf{x}^{\mathrm{f}})^T \mathbf{C}_{xx}^{-1} (\mathbf{x} - \mathbf{x}^{\mathrm{f}})$$

This is data assimilation!

We correct a forecast $\mathbf{x}^{\mathrm{f}}$ given some observational data $\mathbf{d}$

Is it machine learning?

- Ridge regression: $J(\boldsymbol{\theta}) = (\mathbf{y} - h_{\boldsymbol{\theta}}(\mathbf{x}))^T(\mathbf{y} - h_{\boldsymbol{\theta}}(\mathbf{x})) + \alpha\theta^T\theta$
- BLUE: $\mathcal{J}(\mathbf{x}) = (\mathbf{d} - \mathbf{H}\mathbf{x})^T\mathbf{C}_{\epsilon\epsilon}^{-1}(\mathbf{d} - \mathbf{H}\mathbf{x}) + (\mathbf{x} - \mathbf{x}^{\mathrm{f}})^T\mathbf{C}_{xx}^{-1}(\mathbf{x} - \mathbf{x}^{\mathrm{f}})$

Is it machine learning?

- Ridge regression: $J(\boldsymbol{\theta}) = (\mathbf{y} - h_{\boldsymbol{\theta}}(\mathbf{x}))^T(\mathbf{y} - h_{\boldsymbol{\theta}}(\mathbf{x})) + \alpha\theta^T\theta$
- BLUE: $\mathcal{J}(\mathbf{x}) = (\mathbf{d} - \mathbf{H}\mathbf{x})^T\mathbf{C}_{\epsilon\epsilon}^{-1}(\mathbf{d} - \mathbf{H}\mathbf{x}) + (\mathbf{x} - \mathbf{x}^{\mathrm{f}})^T\mathbf{C}_{xx}^{-1}(\mathbf{x} - \mathbf{x}^{\mathrm{f}})$

BLUE	Ridge
data $\mathbf{d}$	target $\mathbf{y}$
Observation operator $\mathbf{H}$	feature $\mathbf{x}$
State $\mathbf{x}$	parameters $\boldsymbol{\theta}$
$\mathbf{C}_{\epsilon\epsilon}\mathbf{C}_{xx}^{-1}$	α

Is it machine learning?

- Ridge regression: $J(\boldsymbol{\theta}) = (\mathbf{y} - h_{\boldsymbol{\theta}}(\mathbf{x}))^T(\mathbf{y} - h_{\boldsymbol{\theta}}(\mathbf{x})) + \alpha\theta^T\theta$
- BLUE: $\mathcal{J}(\mathbf{x}) = (\mathbf{d} - \mathbf{H}\mathbf{x})^T\mathbf{C}_{\epsilon\epsilon}^{-1}(\mathbf{d} - \mathbf{H}\mathbf{x}) + (\mathbf{x} - \mathbf{x}^{\mathrm{f}})^T\mathbf{C}_{xx}^{-1}(\mathbf{x} - \mathbf{x}^{\mathrm{f}})$

BLUE	Ridge
data $\mathbf{d}$	target $\mathbf{y}$
Observation operator $\mathbf{H}$	feature $\mathbf{x}$
State $\mathbf{x}$	parameters $\boldsymbol{\theta}$
$\mathbf{C}_{\epsilon\epsilon}\mathbf{C}_{xx}^{-1}$	α

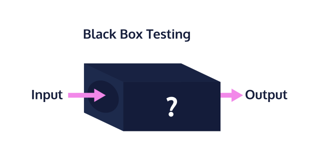
Further links...

If a numerical model is integrated over several time steps, it can be related to successive layers of a Neural Network.

A black box?

The black box paradigm

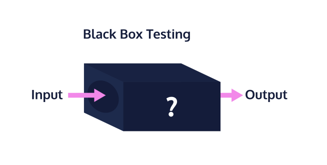
The machine-learning based model is a black box. It gives some results but we don't understand how.



Do we need to understand the model?

The black box paradigm

The machine-learning based model is a black box. It gives some results but we don't understand how.



Do we need to understand the model?

"Every time I fire a linguist, the performance of our speech recognition system goes up."

F. Jelinek, 1988

Frederick Jelinek 1932-2010

Beyond the black box

Motivation

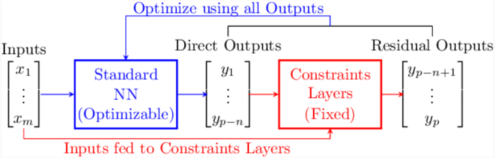
- Build models that can be trusted
- Use other source of knowledge (e.g. physical properties) when data are not sufficient.

Two directions:

- Add physical constraints to ML models
- Explainable/Transparent ML.

Add physical constraints to ML models

- Simple example: enforce the positivity of some quantities (e.g. concentration)
- More complex: enforce conservation laws
  Beucler, T., Pritchard, M., Rasp, S., Ott, J., Baldi, P. and Gentine, P., 2021. Enforcing analytic constraints in neural networks emulating physical systems. *Physical Review Letters*, 126(9), p.098302.



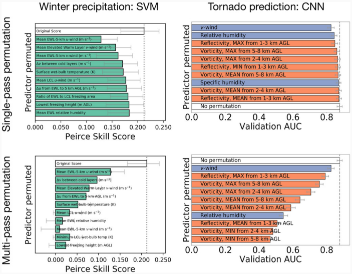
Beucler et al.

Explainable/Transparent ML.

The objective is to understand how the machine learning makes a prediction (e.g. which feature is important for the prediction).

McGovern et al., 2019, Making the black box more transparent: Understanding the physical implications of machine learning. *Bulletin of the American Meteorological Society*

Sonnewald et al.,2021. Bridging observation, theory and numerical simulation of the ocean using Machine Learning. *Environ. Res. Lett.*



McGovern et al.

Questions addressed in this lecture

- What is gradient backpropagation? [GBC16,6]
- What are the different types of gradient descent techniques? [GBC16,8]
- What are some other types of regularizations? [GBC16,7]
- How machine learning can be related to data assimilation?

Refs

[Van16,n]: Jake VanderPlas, *Python Data Science Handbook*, section n
[GBC16,n]: Goodfellow etal., Deep Learning, chapter n