# Neural networks

# Cours 2/3

**Machine Learning**
ModIA 2022

*Enseignant: Sixin Zhang*
sixin.zhang@toulouse-inp.fr

Under-fitting and over-fitting

Regularization methods for generalization

Stochastic optimization methods

# Challenge


As the size (width or depth) of a neural network increases, certain problems appear (under-fitting or over-fitting, *vanishing* and *exploding gradients*, etc.) which requires careful analysis and practice.

# Empirical error and generalization error

- **Empirical error** : prediction error on a training set (with a finite number of samples).
- **Generalization error** : prediction error on a test set (ideally with infinite number of samples). ... **Unknown !**

The goal of a machine learning algorithm is to minimize the generalization error, but we are only able to optimize the **empirical error**.

# Under-fitting vs. over-fitting

We say a model is *under-fitting* if it learns too badly on a training set.

We say a model is *over-fitting* if it learns very well on a training set, but generalizes badly to test set.

Example: `https://playground.tensorflow.org/`
• Learn a classifier on the spiral data using a MLP of size $(2, 4, 1)$, tanh activation, 20% training / test set ratio.
• Learn a classifier on the spiral data using a MLP of size $(2, 8, 8, 8, 1)$, tanh activation, 10% training / test set ratio.

# How to avoid under-fitting?

One interesting property of Neural networks is their universal approximation.

**Théorème d'approximation universelle (Cybenko 1989)**

Toute fonction $h$, continue, de $[0,1]^m$ dans $\mathbb{R}$, peut être approximée par un perceptron à une couche cachée comportant suffisamment de neurones (avec une fonction d'activation sigmoïde).

In theory, one can avoid under-fitting by increasing the width of an 1-layer perceptron !

Question: Can we also avoid over-fitting?

# How to detect over-fitting: Cross-validation

In practice, one may not have access to the whole test data (such as in Kaggle), one can then split the training data into 2 parts to estimate whether there is over-fitting: one for training, the other for validation.

One often uses $N$ fold cross-validation:

- Step 1: Partition the training set into $N$ subsets (folds).
- Step 2: Take one subset among the $N$ subsets as a validation set, use the rest $N-1$ subsets for training.
- Step 3: Repeat Step 2 multiple times, each time using a different subset for validation. Check if the average error on the validation sets is very different to the average error on the training sets.

How to avoid over-fitting?

- Early stopping (optimization specific)
- Data augmentation (data specific)
- Dropout (neural network specific)
- Ensemble methods (classification specific)

# Early stopping

To avoid over-fitting, one can trace how the validation error changes during the training: stop the training when it starts to increase while the training error still goes down.
In practice : the validation error is noisy if using stochastic gradient descent methods, one should train for a longer time to decide whether to stop or not.

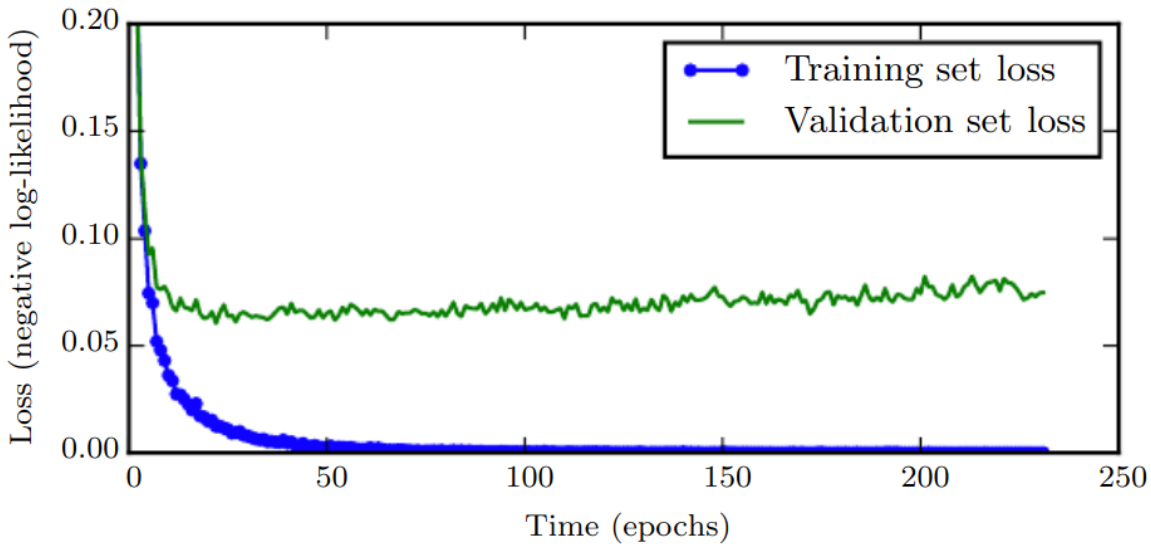



Image from [Goodfellow et al. 2015] Deep Learning

# Data augmentation

Use a small set of training data may result in over-fitting in classification
$\rightarrow$ Increase artificially the training data by controlled transformations.

Useful in *image processing* based on affine transformations:

- Translation:
  $x(u) \rightarrow x(u - \tau), \quad \tau \in \mathbb{R}^2$.
- Rotation: $x(u) \rightarrow x(r_\theta u)$,
  rotation matrix $r_\theta$ in 2d.
- Scaling: $x(u) \rightarrow x(u/s)$, $s > 0$.

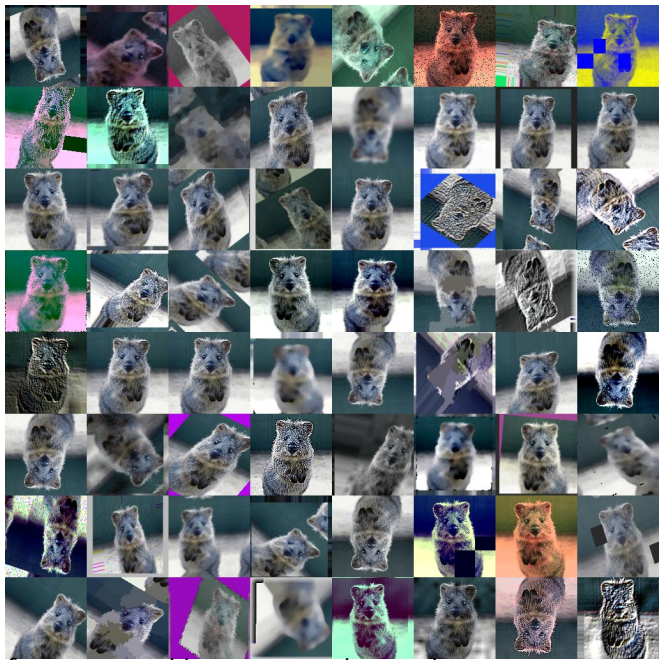


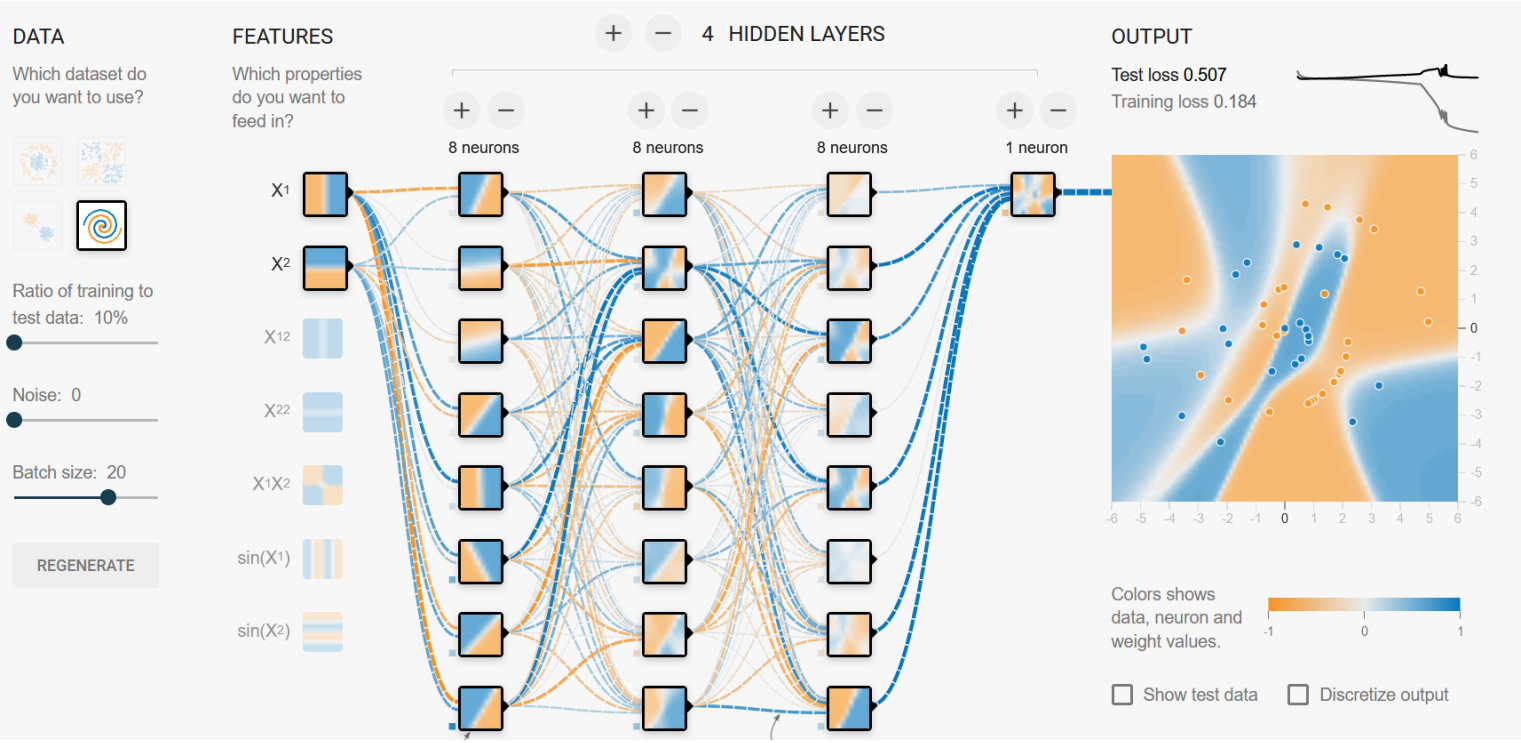
Image from https://github.com/aleju/imgaug

Results on MNIST from 'Regularization of Neural Networks using DropConnect', Li et al. 2013

| crop | rotation scaling | model | error(%) 5 network | voting error(%) |
|---|---|---|---|---|
| no | no | No-Drop | 0.77±0.051 | 0.67 |
| | | Dropout | 0.59±0.039 | **0.52** |
| | | DropConnect | 0.63±0.035 | 0.57 |
| yes | no | No-Drop | 0.50±0.098 | 0.38 |
| | | Dropout | 0.39±0.039 | 0.35 |
| | | DropConnect | 0.39±0.047 | **0.32** |
| yes | yes | No-Drop | 0.30±0.035 | **0.21** |
| | | Dropout | 0.28±0.016 | 0.27 |
| | | DropConnect | 0.28±0.032 | **0.21** |

*Table 3.* MNIST classification error. Previous state of the art is *0.47%* (Zeiler and Fergus, 2013) for a single model without elastic distortions and *0.23%* with elastic distortions and voting (Ciresan et al., 2012).

One cause of over-fitting is due to some preferential paths learnt in neural networks, e.g. MLP:



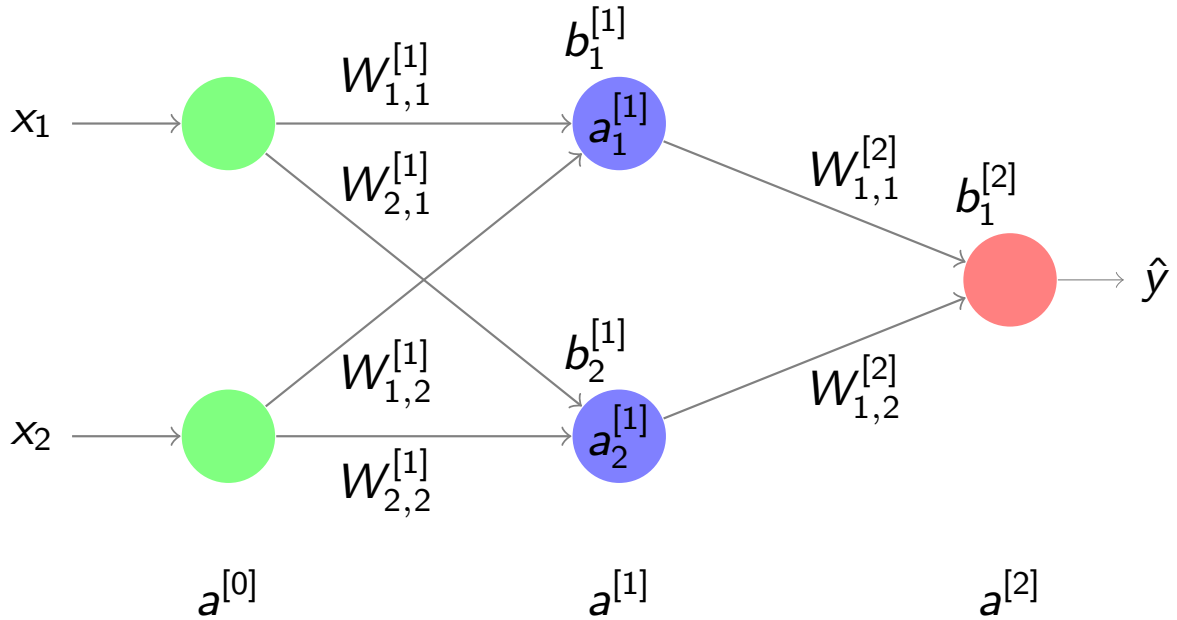
In order to avoid this phenomenon, one may randomly "dis-connect" some neurons in a network during the training. This allows to favor various paths to make decisions collectively.

*How it works during training?*
For an element-wise layer $a = f(z)$, introduce a random mask $M \in \{0,1\}^N$ such that

$$a_i = f_M(z_i) = M_i f(z_i), \quad i \leq N$$

We also write: $a = f_M(z) = M \cdot f(z)$.

- During training: $a_i^{[1]} = M_i f_i^{[1]}(W^{[1]}x + b^{[1]})$ may be dropped randomly




In practice, $M_i$ is drawn from a Bernoulli random variable (with probability $p = 0.5$, $M_i = 1$).

- During test: $a_i^{[1]} = pf_i^{[1]}(W^{[1]}x + b^{[1]})$ is fixed to be an average value.

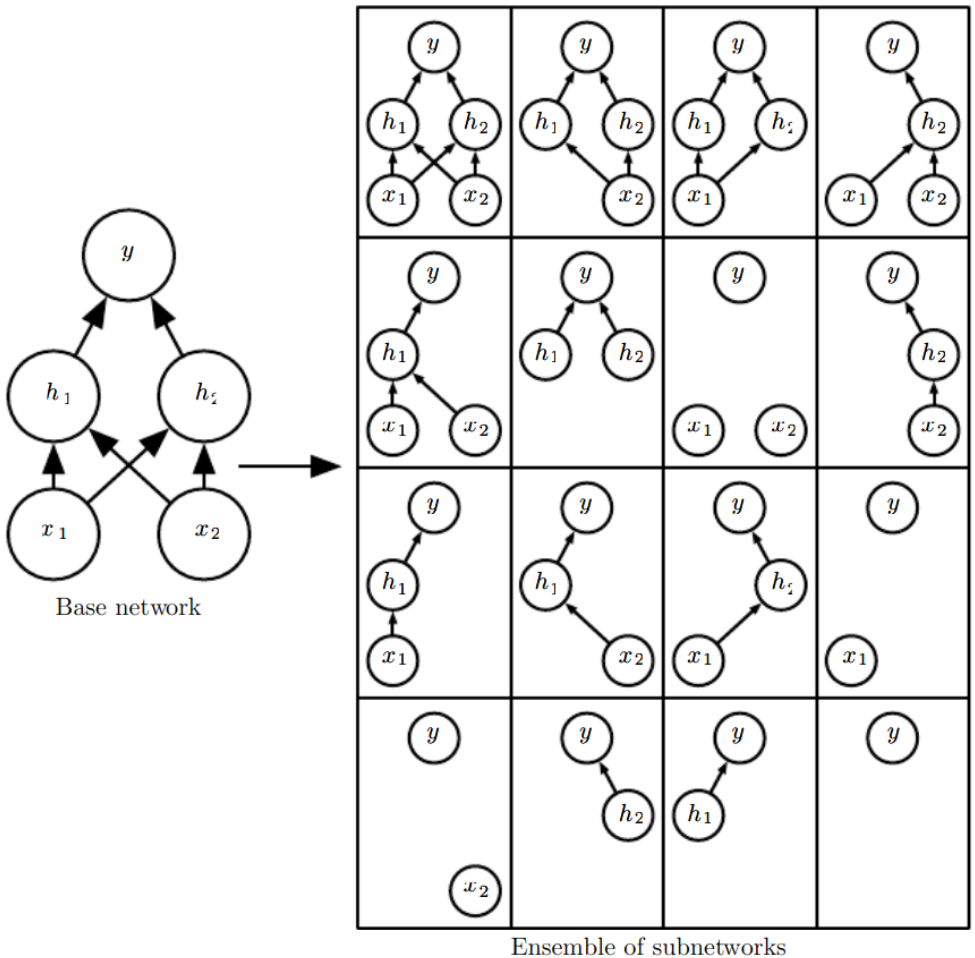



Image from [Goodfellow et al. 2015] Deep Learning

- Let $X \in \mathbb{R}^{m \times d}$ and $Y \in \mathbb{R}^m$: $m$ samples.
- Objective: find $w \in \mathbb{R}^d$ such that

$$\min_w \|y - Xw\|^2 = \sum_{j \leq m} |y^{\{j\}} - \langle w, x^{\{j\}} \rangle|^2$$

- With dropout, the objective becomes

$$L_p = \sum_{j \leq m} \mathbb{E}_{M^j} |y^{\{j\}} - \langle w, M^{\{j\}} \cdot x^{\{j\}} \rangle|^2$$

- $L_p$ behaves like a ridge regression using a dropout rate $p \in (0, 1)$:

$$L_p = \|y - pXw\|^2 + p(1-p) w^T \mathrm{Diag}(X^T X) w$$

# Dropout performance on MLP

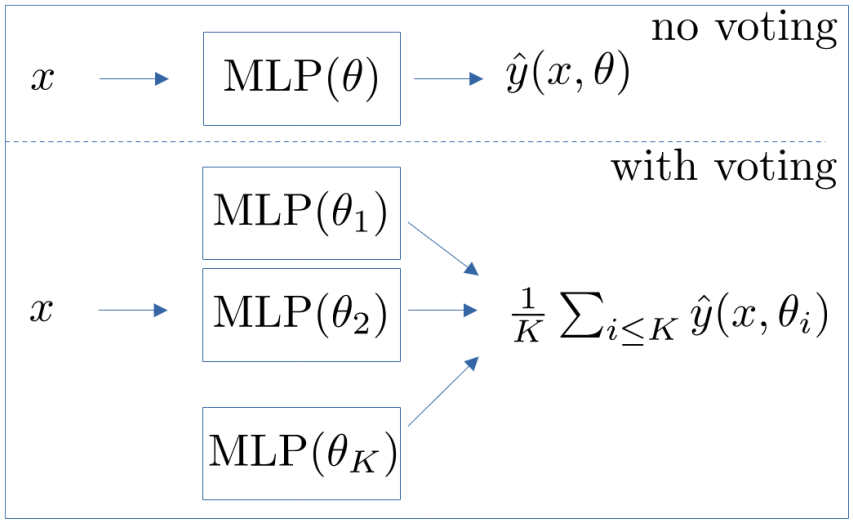
| crop | rotation scaling | model | error(%) 5 network | voting error(%) |
|---|---|---|---|---|
| no | no | No-Drop | 0.77±0.051 | 0.67 |
| | | Dropout | 0.59±0.039 | **0.52** |
| | | DropConnect | 0.63±0.035 | 0.57 |
| yes | no | No-Drop | 0.50±0.098 | 0.38 |
| | | Dropout | 0.39±0.039 | 0.35 |
| | | DropConnect | 0.39±0.047 | **0.32** |
| yes | yes | No-Drop | 0.30±0.035 | **0.21** |
| | | Dropout | 0.28±0.016 | 0.27 |
| | | DropConnect | 0.28±0.032 | **0.21** |

*Table 3.* MNIST classification error. Previous state of the art is *0.47%* (Zeiler and Fergus, 2013) for a single model without elastic distortions and *0.23%* with elastic distortions and voting (Ciresan et al., 2012).

# Ensemble methods: voting (classification specific)

*Average the probabilities of several models, and decide based on the maximal average probability*

Let $\hat{y}(x, \theta)$: softmax probability of input $x$ with model (parameter $\theta$).




The averaging over $K$ MLP is similar to Bagging, but the multiple-layer perceptron (MLP) is trained independently with the same training data.

# Voting performance on MLP

| crop | rotation scaling | model | error(%) 5 network | voting error(%) |
|---|---|---|---|---|
| no | no | No-Drop | 0.77±0.051 | 0.67 |
| | | Dropout | 0.59±0.039 | **0.52** |
| | | DropConnect | 0.63±0.035 | 0.57 |
| yes | no | No-Drop | 0.50±0.098 | 0.38 |
| | | Dropout | 0.39±0.039 | 0.35 |
| | | DropConnect | 0.39±0.047 | **0.32** |
| yes | yes | No-Drop | 0.30±0.035 | **0.21** |
| | | Dropout | 0.28±0.016 | 0.27 |
| | | DropConnect | 0.28±0.032 | **0.21** |

*Table 3.* MNIST classification error. Previous state of the art is *0.47%* (Zeiler and Fergus, 2013) for a single model without elastic distortions and *0.23%* with elastic distortions and voting (Ciresan et al., 2012).

Training Neural networks

- Gradient-descent
- Stochastic gradient-descent (mini-batch)

Compute the gradient of $J(\theta)$ is computationally **costly** as it requires to evaluate all the $m$ training samples (when $m$ is very big or each evaluation is costly).

To reduce the cost of gradient computation, there is an alternative:

- **Stochastic** / **Mini-Batch** gradient-descent (SGD): estimate the gradient by using $k$ samples among the $m$ (Monte-Carlo).
- Typically, we choose $m >> k \geq 1$.

Let $J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \ell(y^{(i)}, \hat{y}(x^{(i)}, \theta)$. SGD takes a random subset $B \subseteq \{1, \cdots, m\}$ of fixed size $|B|$ to compute an estimation of $J(\theta)$ using

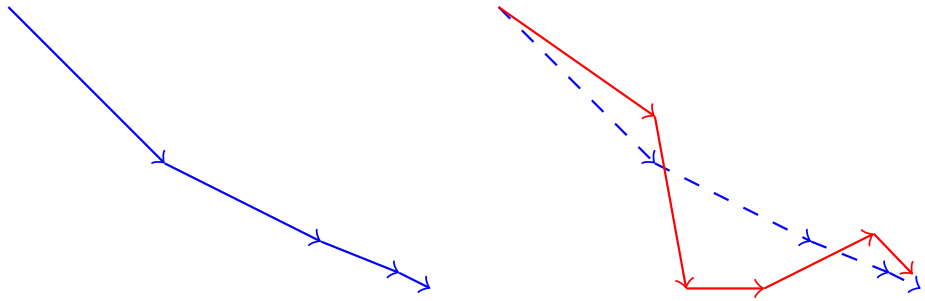
$$J_B(\theta) = \frac{1}{|B|}\sum_{i \in B} \ell(y^{(i)}, \hat{y}(x^{(i)}, \theta))$$

At iteration $t$, a subset $B^{(t)}$ is taken, and

$$\theta^{(t+1)} = \theta^{(t)} - \eta_t \nabla J_{B^{(t)}}(\theta^{(t)})$$

**Remark 1 :** There is no guarantee that $J(\theta^{(t+1)}) < J(\theta^{(t)})$, in particular when the mini-batch size $|B|$ is small.
**Remark 2 :** One often says that SGD runs an epoch when it sees the whose training data once, i.e. 1 epoch equals to $m/|B|$ iterations.

Unlike GD, which often has a guarantee that $J(\theta^{(t+1)}) < J(\theta^{(t)})$, SGD does not have a monotone decreasing property.



GD SGD

One important convergence condition of SGD is about the learning rate:

$$\eta_t > 0, \quad \sum_{t=1}^{\infty} \eta_t = \infty, \quad \sum_{t=1}^{\infty} \eta_t^2 < \infty.$$

- Analyze an ideal case ($m = \infty$): $J(\theta) = \mathbb{E}_{\xi \sim \mathcal{N}(\mu, \sigma^2 I_d)} \|\theta - \xi\|^2/2$, with $\theta, \xi \in \mathbb{R}^d$.

1. Sufficient conditions of SGD convergence: assume $J(\theta) = \mathbb{E}_{\xi \sim \mathcal{N}(\mu, \sigma^2 I_d)} h\|\theta - \xi\|^2/2$, with $\theta, \xi \in \mathbb{R}^d$, $h > 0$.
   - At each iteration, sample independently $\xi^{(t)} \sim \mathcal{N}(\mu, \sigma^2 I_d)$
   - Update $\theta^{(t)}$ with $\theta^{(t+1)} = \theta^{(t)} - \eta_t h(\theta^{(t)} - \xi^{(t)})$

   What are the following conditions sufficient for SGD convergence, i.e. $\mathbb{E}\|\theta^{(t)} - \mu\|^2 \to 0$ as $t \to \infty$?
   1. $\sum_{t=1}^{\infty} \eta_t^2 < \infty$
   2. $\eta_t > 0, \sum_{t=1}^{\infty} \eta_t = \infty, \sum_{t=1}^{\infty} \eta_t^2 < \infty$
   3. $0 < \eta_t < 2/h$, $\sum_{t=1}^{\infty} \eta_t = \infty$, $\sum_{t=1}^{\infty} \eta_t^2 < \infty$,
   4. $\eta_t > 0, \sum_{t=1}^{\infty} \eta_t = \infty$