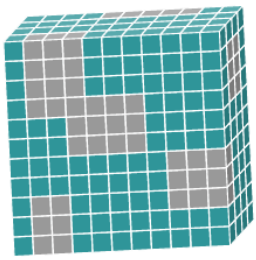
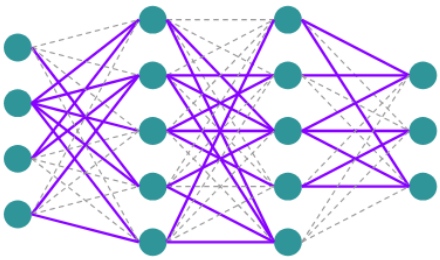
12 Main Dropout Methods: Mathematical and Visual Explanation for DNNs, CNNs, and RNNs

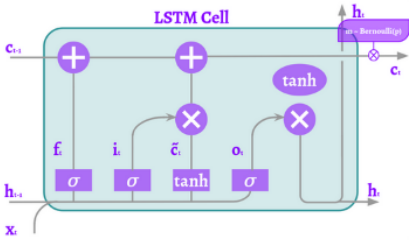
towardsdatascience.com/12-main-dropout-methods-mathematical-and-visual-explanation-58cdc2112293

June 4, 2020

Deep Dive into DNNs, CNNs, and RNNs Dropout Methods for Regularization, Monte Carlo Uncertainty, and Model Compression







Motivations

One of the **major challenges** when training a model in **(Deep) Machine Learning** is **co-adaptation**. This means that the neurons are very **dependent** on each other. They influence each other considerably and are not independent enough regarding their inputs. It is also common to find cases where some neurons have a **predictive capacity** that is more significant than others. In other words, we have an output that is excessively dependent on one neuron.

These effects **must be avoided** and the weight must be distributed to **prevent overfitting**. The co-adaptation and the high predictive capacity of some neurons can be regulated with different **regularization** methods. One of the most used is the **Dropout**. Yet the full capabilities of dropout methods are rarely used.

Depending on whether it is a **DNN**, a **CNN** or an **RNN**, different **dropout methods** can be applied. In practice, **we only use one** (or almost). I think that's a terrible **pitfall**. So in this article, we will dive **mathematically** and **visually** into the world of dropouts to understand :

- the Standard Dropout method
- variants of the Standard Dropout
- dropout methods applied to CNNs
- dropout methods applied to RNNs

- other dropout applications (Monte Carlo and compression)

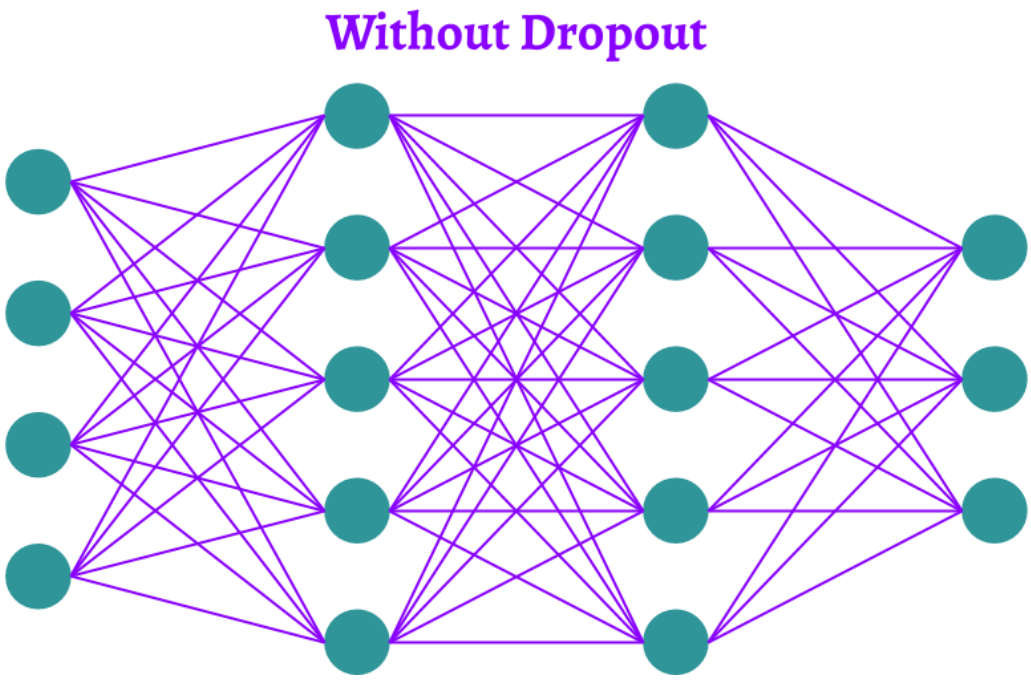
(Sorry I couldn't stop, so it's a little more than 12 methods... 😄)

Notations

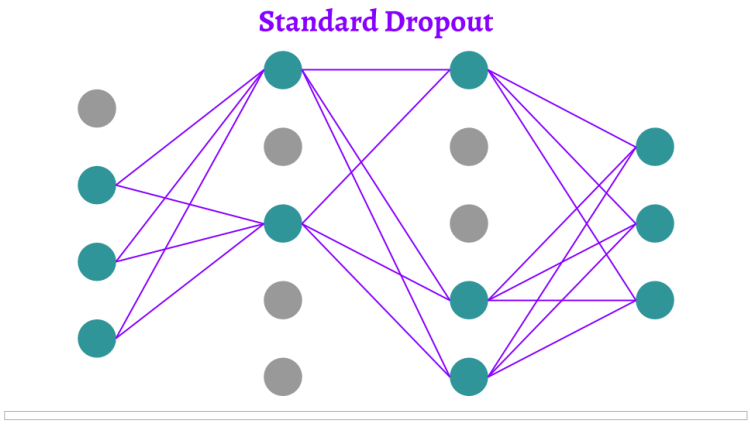
a is a scalar
$\mathbf{a}$ is a vector
$\mathbf{A}$ is a matrix
$\circ$ is the element-wize multiplication (Hadarmard)
f is an activation function
p is the dropout probability
$Bernoulli$ is the Bernoulli distribution
$\mathcal{N}$ is the normal distribution
For simplicity, matrices and vectors are including the bias ($\mathbf{Wx} + \mathbf{b} \longrightarrow \mathbf{Wx}$)

Standard Dropout

The most **well known and used** dropout method is the **Standard Dropout** [1] introduced in 2012 by Hinton et al.. Usually called simply **Dropout**, for obvious reasons, in this article we will call it Standard Dropout.



To **prevent overfitting** in the **training** phase, neurons are **omitted at random**. Introduced in a dense (or fully connected) network, **for each layer** we give a probability `p` of **dropout**. At each iteration, each neuron has a probability `p` of being omitted. The Hinton et al. paper recommends a dropout probability `p=0.2` on the input layer and a probability `p=0.5` on the hidden layers. Obviously, we are interested in the output layer which is our prediction. So we don't apply a dropout on the output layer.



Dropout with p=0.5

$$\text{Training Phase :}$$

$$\mathbf{y} = f(\mathbf{W}\mathbf{x}) \circ \mathbf{m}, \quad m_i \sim Bernoulli(p)$$

$$\text{Testing Phase :}$$

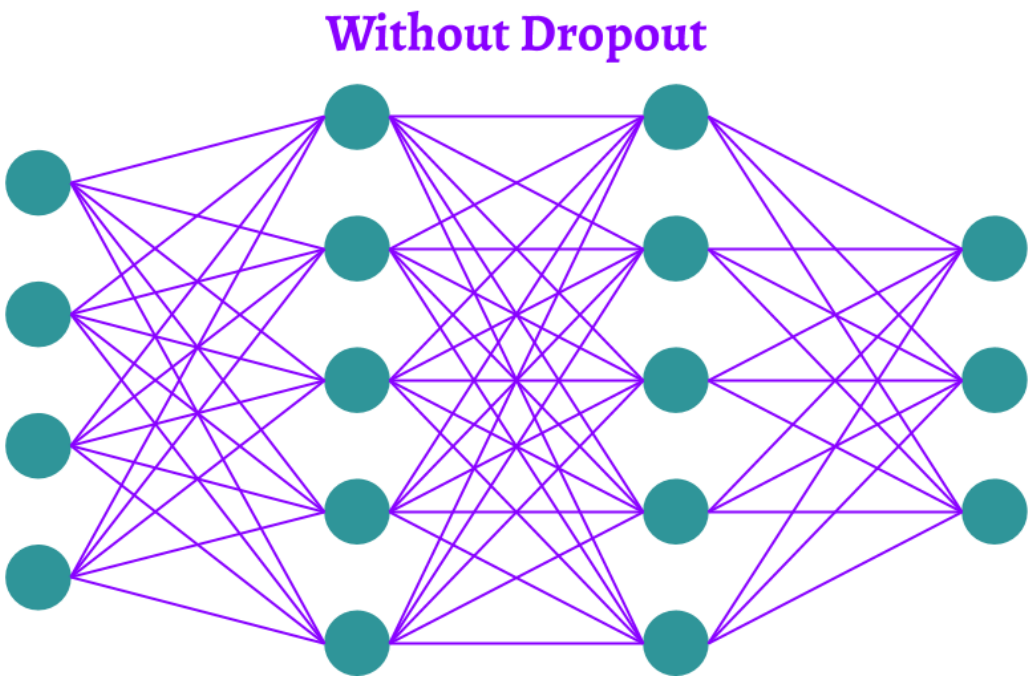
$$\mathbf{y} = (1-p) f(\mathbf{W}\mathbf{x})$$

Mathematically, we say that the probability of omission for each neuron follows a **Bernoulli** distribution of probability `p`. We thus make an **element-wise** of the neuron vector (layer) with a **mask in which each element is a random variable following the Bernoulli distribution**.

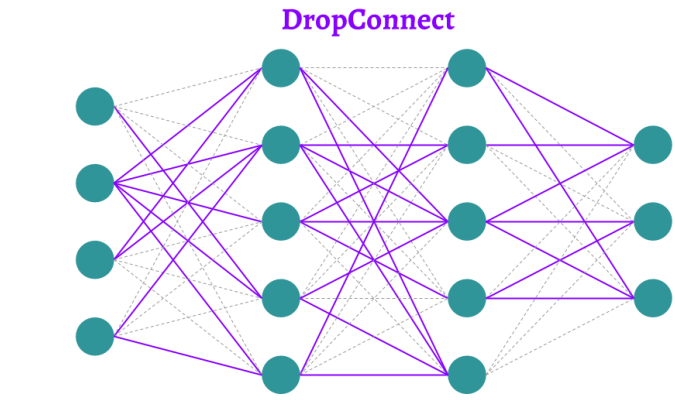
During the **testing** (or inference) phase, there is **no dropout**. All neurons are active. To compensate for the additional information compared to the training phase, we **weight by the probability of presence**. So the probability for a neuron not to be omitted. It is `1-p`.

DropConnect

Perhaps you are already familiar with the Standard Dropout method. But there are many variations. To regularize the forward pass of a Dense network, you can apply a dropout on the neurons. The **DropConnect** [2] introduced by L. Wan et al. does **not** apply a dropout directly on the neurons but **on the weights and bias** linking these neurons.



We, therefore, find the same mechanism as in the Standard Dropout method. Except that the mask (whose elements are random variables following a distribution) is **not** applied on the neuron **vector** of a layer **but on the matrix of weights** connecting the layer to the previous one.



Dropout with p=0.5

$$\text{Training Phase :}$$

$$\mathbf{y} = f((\mathbf{W} \circ \mathbf{M})\mathbf{x}), \quad M_{i,j} \sim Bernoulli(p)$$

$$\text{Testing Phase :}$$

$$\mathbf{y} = (\mathbf{W}\mathbf{x}) \circ \hat{\mathbf{m}}(\mathbf{Z})$$

$$\text{where } \hat{m}_i(Z) = \frac{1}{Z}\sum_{z=0}^{Z} f(\hat{x}_{i,z}), \quad \hat{x}_{i,z} \sim \mathcal{N}(\mu_i, \sigma_i^2)$$

$$\text{and } \boldsymbol{\mu} = p\mathbf{W}\mathbf{x}, \quad \boldsymbol{\sigma}^2 = p(1-p)(\mathbf{W} \circ \mathbf{W})(\mathbf{x} \circ \mathbf{x}), \quad Z \in \mathbb{N}^+$$

For the testing phase, it is possible to have the same logic as for the Standard Dropout method. We can multiply by the **probability of presence**. But this is not the method proposed by L. Wan et al. It is interesting because they propose **a stochastic approach**

of the dropout **even in the testing phase** by applying a **Gaussian approximation** of the DropConnect. Then by **randomly drawing samples** from this Gaussian representation. We will come back to Gaussian approximations just after the Standout.

Standout

As a Standard Dropout method, the **Standout** [3] introduced by L. J. Ba and B. Frey is based on a **Bernoulli** mask (I will call these masks according to the distribution they follow, it will be simpler). The difference is that the probability `p` of omission of the neuron is **not constant on the layer**. It is **adaptative** according to the **value of the weights**.

$$\text{Training Phase :}$$

$$\mathbf{y} = f(\mathbf{W}\mathbf{x}) \circ \mathbf{m}, \quad m_i \sim Bernoulli(g(\mathbf{W_s}\mathbf{x}))$$

$$\text{Testing Phase :}$$

$$\mathbf{y} = (1 - g(\mathbf{W_s}\mathbf{x})) \circ f(\mathbf{W}\mathbf{x})$$

$$\text{where } \mathbf{W_s} \text{ is the belief network's weights}$$
$$\text{and } \quad g \quad \text{is the belief network's activation function}$$

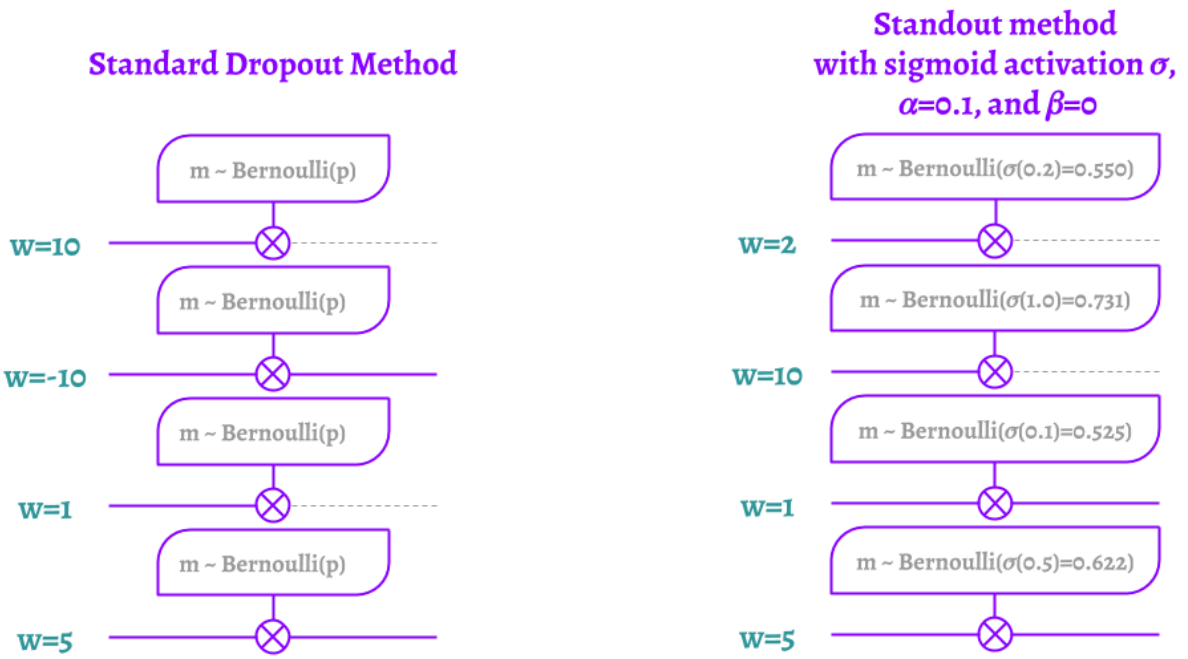
This can work for any `g` activation function or even be a separate neural network. Similarly, for `Ws` which can be a function of `W`. Then for the test phase, we balance by the probability of presence.

Example

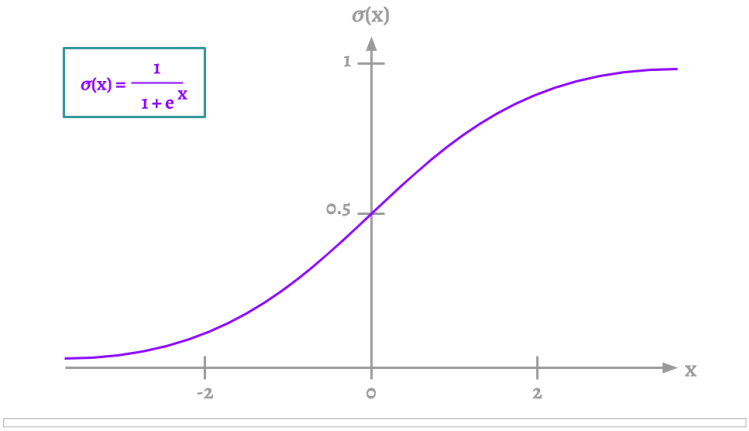
$$g(x) = |\sigma(x)|$$

$$\mathbf{W_s} = \alpha\mathbf{W} + \beta$$

It's a little obscure, so let's take an example. In their paper, they showed that in practice the **belief network weights** can be approximated to an **affine function** of the weights. And for example, I will take the **absolute value of the sigmoid** as the **activation function**.



We can, therefore, see that the greater the **weight**, the greater the **probability** that the neuron will be omitted. This powerfully limits the **high predictive capacity** that some neurons may have.



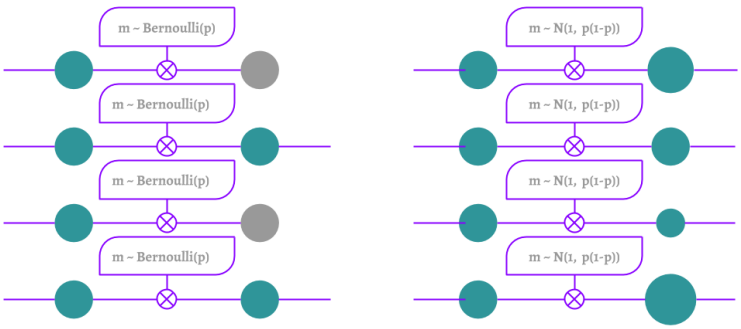
Standard dropout with p=0.5 compared to a Standout Example

Gaussian Dropout

The list of dropout methods applied to neural networks continues to grow. So before moving on to something else than DNNs I would like to talk about a category of Dropout method that is certainly the most **fascinating**.

To take just a few examples, **Fast Dropout** [4], **Variational Dropout** [5] or **Concrete Dropout** [6] are methods interpreting dropout from a **Bayesian perspective**. Concretely, instead of having a Bernoulli mask, we have a mask whose elements are random **variables following a Gaussian distribution** (**Normal** distribution). I won't go into the demonstration of the **law of Large Numbers** here, that's not the point. So let's try to understand this **intuitively**.

Papers [4], [5] and [6] show we can **simulate** a Bernoulli mask for our dropouts with a **normal law**. But what difference does it make. **Everything and nothing** at the same time. It does not change anything concerning the relevance of these methods against **overfitting** due to **co-adaptation** and/or the **predictive capacity** of our neurons. But it changes everything in terms of the **execution time required for the training** phase compared to the methods presented before.



Dropout with p=0.5

Logically, by omitting at each iteration neurons with a dropout, **those omitted on an iteration are not updated** during backpropagation. They do not exist. So the training phase is **slowed down**. On the other hand, by using a Gaussian Dropout method, **all the neurons are exposed at each iteration** and for each training sample. This avoids the slowdown.

$$\text{Training Phase :}$$

$$\mathbf{y} = f(\mathbf{W}\mathbf{x}) \circ \mathbf{m}, \quad m_i \sim \mathcal{N}(1,\ p(1-p))$$

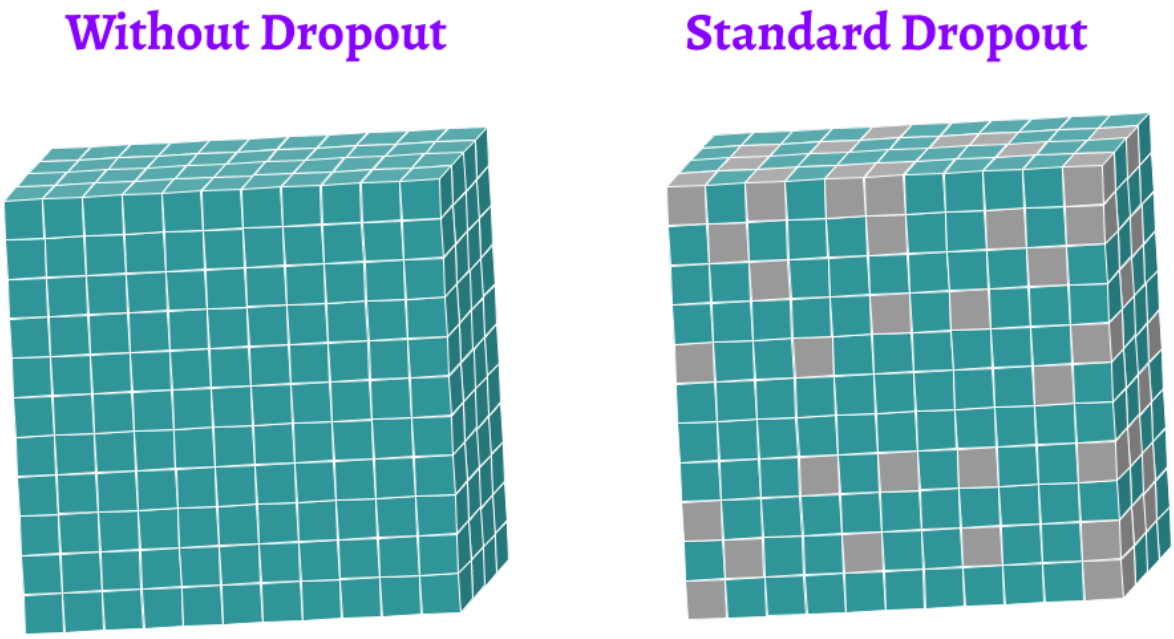
$$\text{Testing Phase :}$$

$$\mathbf{y} = f(\mathbf{W}\mathbf{x})$$

Mathematically, there is a multiplication with a Gaussian mask (for example centered in 1 with Bernoulli's law standard deviation `p(1-p)`). This **simulates** the dropout by randomly **weighting their predictive capacity** by keeping all neurons **active** at each iteration. Another practical advantage of this method centered in 1: during the testing phase there is no modification to be made compared to a model without dropout.

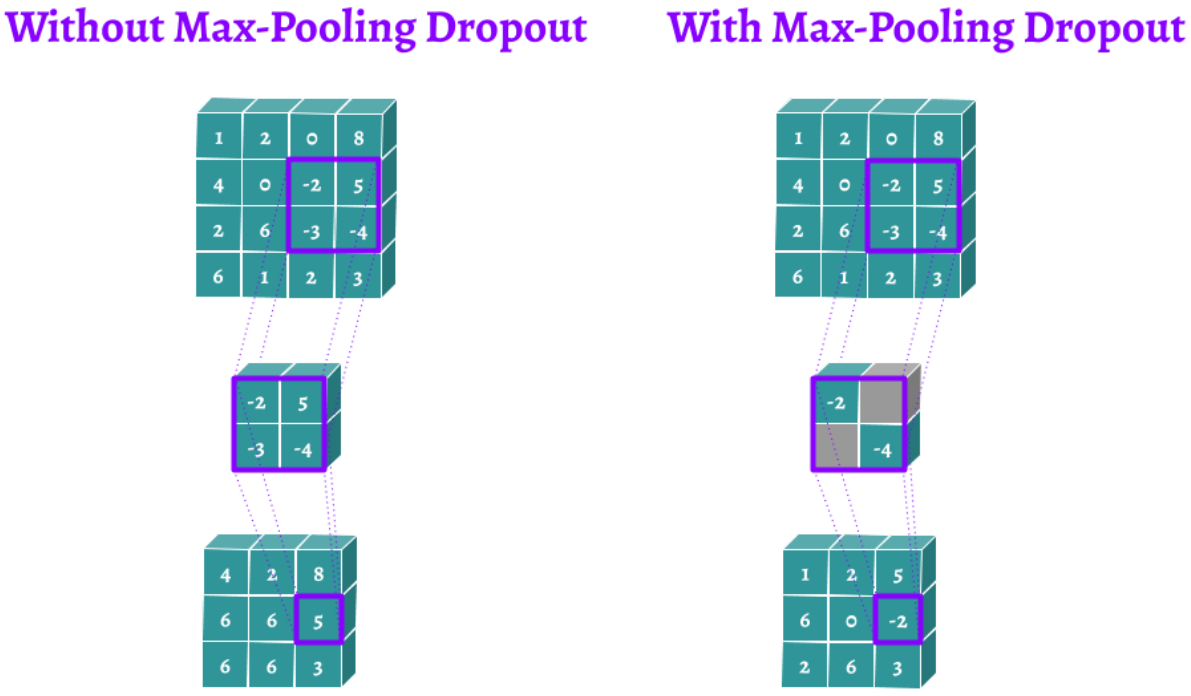
Pooling Dropout

The "difficult" comprehension part of this article is over. Remaining the more **intuitive** part giving to us **better performances**.



The issue with **images** or **feature maps** is that pixels are very **dependent on their neighbors**. To put it simply, on a cat picture, if you take a pixel that corresponds to its coat then all the neighboring pixels will correspond to the same coat. There is little or **no difference**.

So we understand the **limits** of the Standard Dropout method. We could even say it is **inefficient** and the only change it brings is **additional computation time**. If we randomly omit pixels on an image then almost no information is removed. The **omitted pixels are nearly the same as their surroundings**. It means **poor** performance to prevent overfitting.



Why not take advantage of the layers which are **proper** and often used in the **CNNs**. For example the **Max Pooling Layer**. For those who don't know: the Max Pooling Layer is a filter passed on a picture or (feature map) selecting the maximum activation of the overlapping region.

$$\text{Training Phase :}$$
$$\mathbf{Y} = max\{Pool_{size}(\mathbf{Y}) \circ \mathbf{M_{size}}\} \quad M_{ij} \sim Bernoulli(p)$$
$$\text{Testing Phase :}$$
$$\mathbf{Y} = (1-p)\ max\{Pool_{size}(\mathbf{Y})\}$$

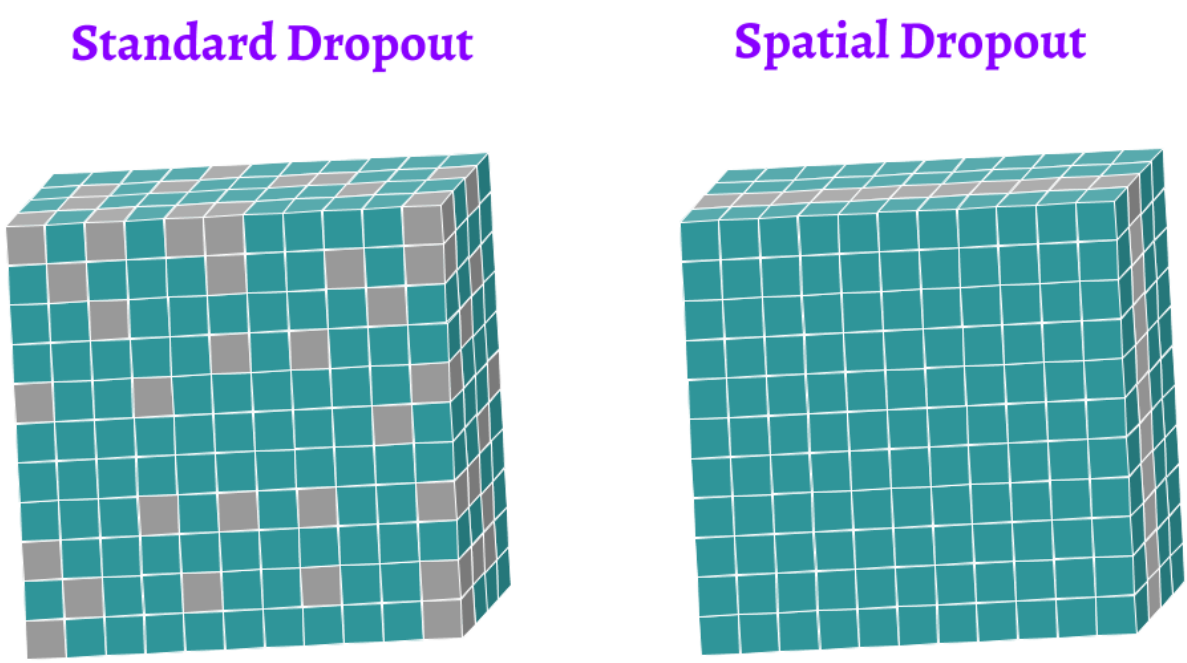
Max-Pooling Dropout [7] is a dropout method applied to CNNs proposed by H. Wu and X. Gu. It applies Bernoulli's mask directly to the **Max Pooling Layer** kernel **before** performing the pooling operation. Intuitively, this allows minimizing the pooling of high activators. It is a very good point to **limit the heavy predictive capacity** of some neurons. During the test phase, you can then weight as for the previous methods by the probability of presence.

$$\text{Training Phase :}$$
$$\mathbf{Y} = avg\{Pool_{size}(\mathbf{Y}) \circ \mathbf{M_{size}}\} \quad M_{ij} \sim Bernoulli(p)$$
$$\text{Testing Phase :}$$
$$\mathbf{Y} = avg\{Pool_{size}(\mathbf{Y})\}$$

The Max Pooling Layer has been taken as an example, but the same could be done with **other Pooling Layers**. For example, with the **Average Pooling Layer**, we could apply a dropout in the same way during the training phase. Then in the test phase, there would be no change since it is already a weighted average.

Spatial Dropout

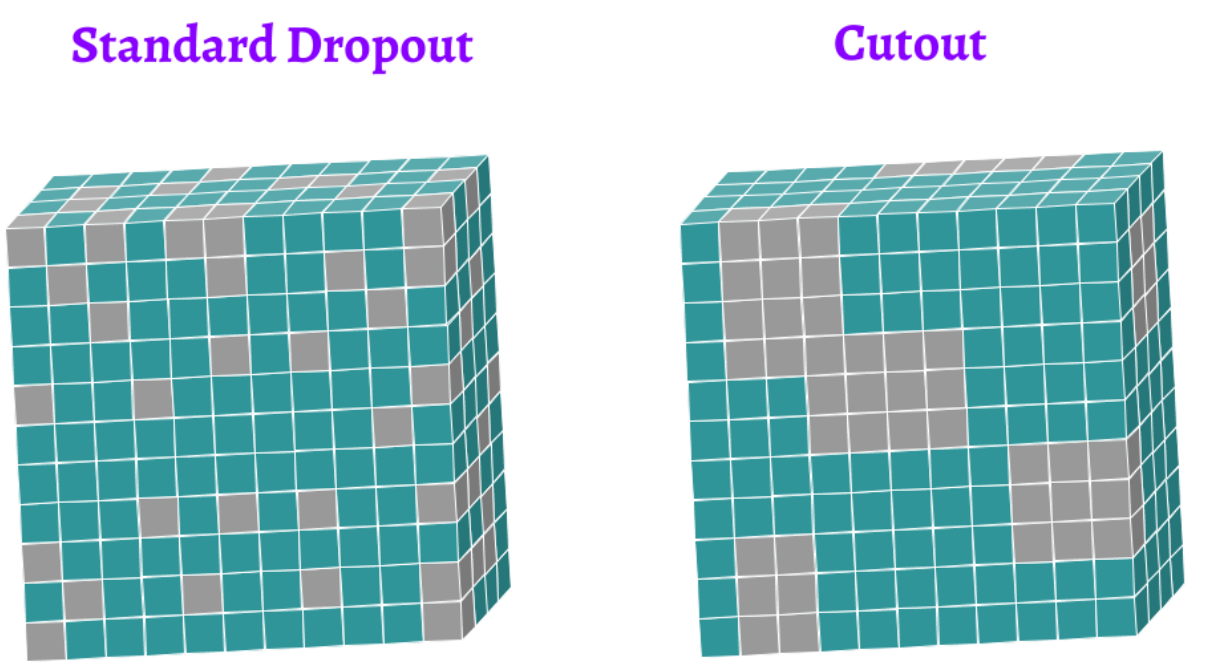
For the CNNs, we can take advantage of the Pooling Layers. But we can also go smarter by following the **Spatial Dropout** [8] method proposed by J. Tompson et al. They propose to **overcome the problem** with classical dropout methods because the **adjacent pixels** are highly **correlated**.



Instead of randomly applying a dropout on the pixels we can think about **applying a dropout per feature map**. If we take the example of our cat, then this is like removing the red from the image and forcing it to generalize on the blue and green of the image. Then randomly other feature maps are dropped on the next iterations.

I did not know how to write properly in mathematics to make it intelligible. But if you understood the previous methods, you won't have any trouble with it. In the **training** phase, a **Bernoulli mask** is applied **per feature map** with a probability of omission `p`. Then during the **testing** phase, there is no dropout but a **weighting** by the probability of presence `1-p`.

Cutout



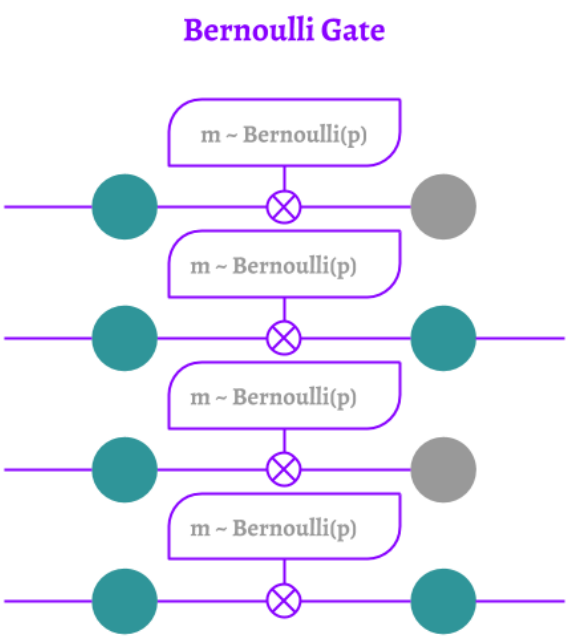
Let's go **deeper** into our approach to overcome the fact that adjacent pixels are highly correlated. Instead of applying Bernoulli masks per feature map, they can be **applied in areas**. This is the **Cutout** method [9] proposed by T. DeVries and G. W. Taylor.

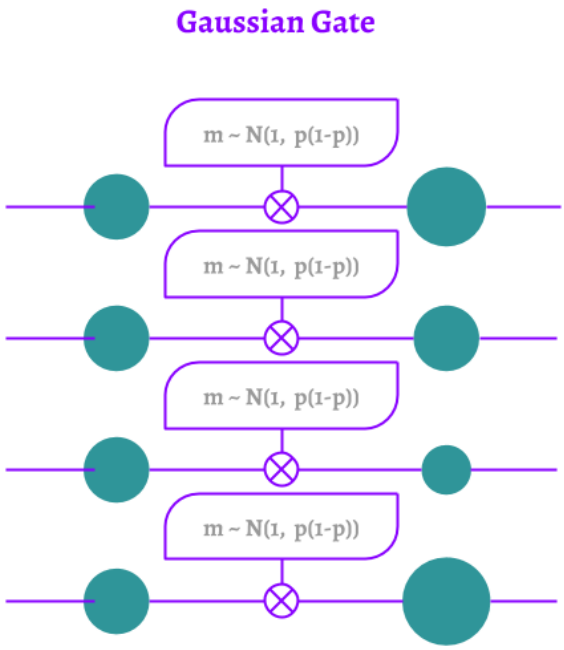
By taking one last time the example of our cat image: this method makes it possible to generalize and thus limit overfitting by **hiding areas of the image**. We end up with images where the cat's head is dropping. This forces the CNN to recognize the less obvious attributes describing a cat.

Again in this section no math. This method depends a lot on our imagination: square areas, rectangles, circles, on all feature maps, on one at a time or possibly on several... It's **up to you**. 😀

Max-Drop

Finally, to conclude this section on the CNNs, I must point out that obviously several **methods can be combined**. This is what makes us strong when we know the different methods: we can take **advantage of their benefits at the same time**. This is what S. Park and N. Kwak propose with their **Max-Drop** method [10].





This approach is in a way a **mixture of Pooling Dropout and Gaussian Dropout**. The dropout is performed on the **Max Pooling Layer** but with a **Bayesian approach**.

$$\text{Training Phase :}$$

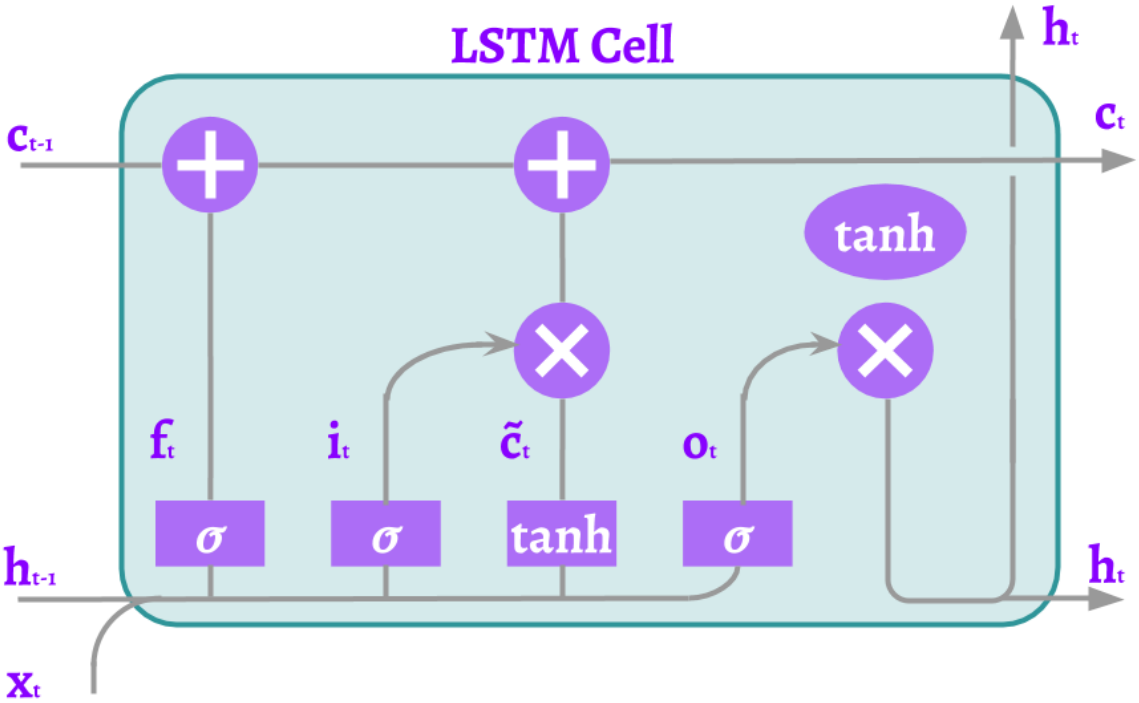
$$\mathbf{Y} = max\{Pool_{size}(\mathbf{Y}) \circ \mathbf{M_{size}}\} \quad M_{ij} \sim \mathcal{N}(\mu,\ \sigma)$$

$$\text{Testing Phase :}$$

$$\mathbf{Y} = (1-\mu)\ max\{Pool_{size}(\mathbf{Y})\}$$

In their paper, they show that this method gives resultsas **efficient** as with a Spatial Dropout. In addition to the fact that at each iteration, all the neurons remain activated which **limits the slowdown** during the training phase.
These results were obtained with $\mu = 0.02$ and $\sigma^2 = 0.05$.

RNNDrop



$$i_t = \sigma\left(W_i \begin{bmatrix} x_t \\ h_{t-1} \end{bmatrix}\right)$$

$$f_t = \sigma\left(W_f \begin{bmatrix} x_t \\ h_{t-1} \end{bmatrix}\right)$$

$$o_t = \sigma\left(W_o \begin{bmatrix} x_t \\ h_{t-1} \end{bmatrix}\right)$$

$$\tilde{c}_t = tanh\left(W_{\tilde{c}} \begin{bmatrix} x_t \\ h_{t-1} \end{bmatrix}\right)$$

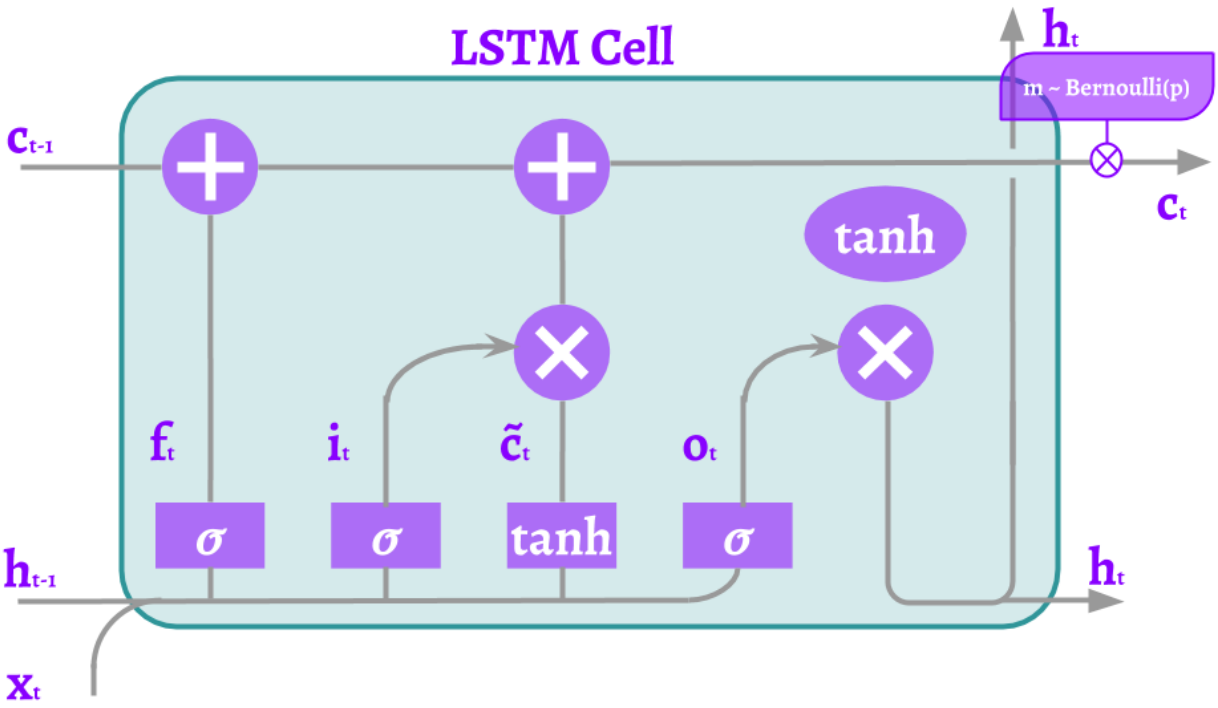
$$c_t = f_t \circ h_{t-1} + i_t \circ \tilde{c}_t$$

$$h_t = o_t \circ tanh(c_t)$$

Well, we have seen some dropout methods for DNNs and CNNs. The research also tried to find out which methods could be **effective for Recurrent Neural Networks** (RNNs). They generally rely on **LSTMs** so I will take this specific case of RNNs. It will be **generalizable** to other RNNs.

The problem is simple: applying a dropout on an RNN is dangerous. In the sense that the purpose of an RNN is to **keep a memory of events over the long term**. But classical dropout methods are not efficient since they **create a noise** that prevents these models

from keeping a memory on the long term. The methods that will be presented allow **preserving this memory** in the long term.



Training Phase :

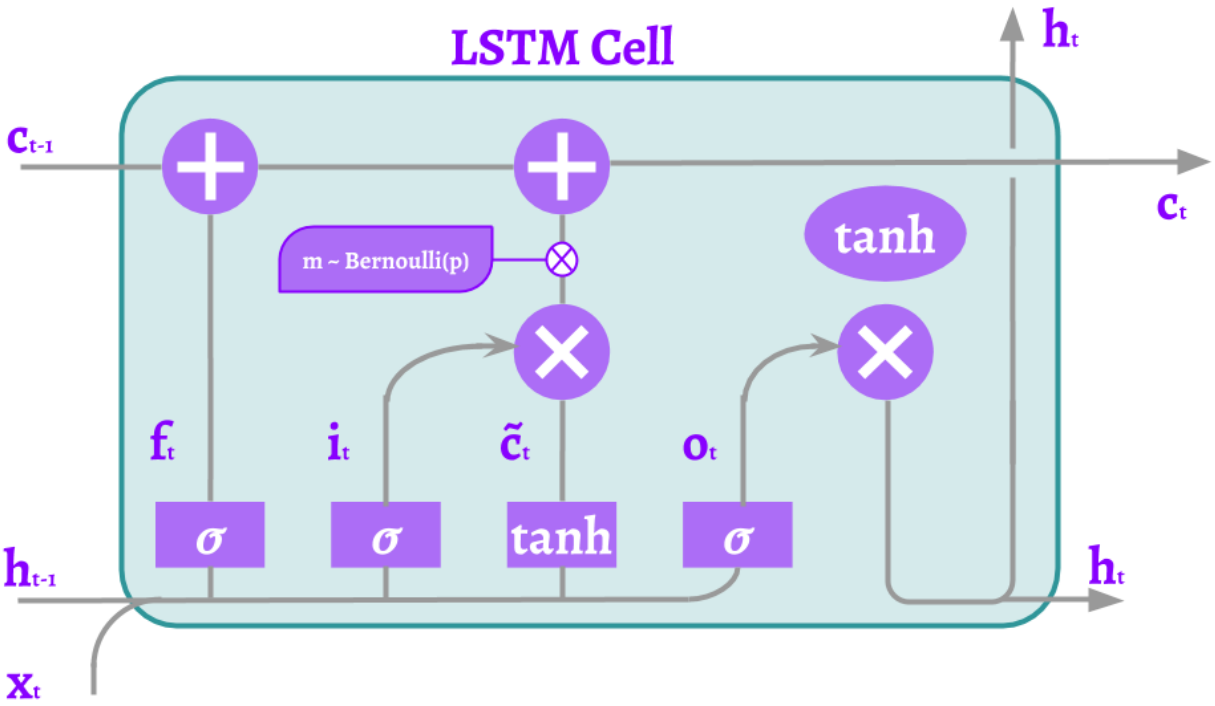
$$\mathbf{c_t} = (\mathbf{f_t} \circ \mathbf{h_{t-1}} + \mathbf{i_t} \circ \tilde{\mathbf{c}}_\mathbf{t}) \circ \mathbf{m}, \quad m_i \sim Bernoulli(p)$$

Testing Phase :

$$\mathbf{c_t} = (1-p)(\mathbf{f_t} \circ \mathbf{h_{t-1}} + \mathbf{i_t} \circ \tilde{\mathbf{c}}_\mathbf{t})$$

RNNDrop [11] proposed by T. Moon et al. is the simplest method. A **Bernoulli** mask is applied only to the hidden **cell states**. But this mask remains **the same from sequence to the other**. This is called the **per-sequence sampling of dropout**. It simply means that at each iteration we create a random mask. Then from one sequence to another, this **mask remains unchanged**. So the dropped elements remain dropped and the present elements remain present. And this on all the sequences.

Recurrent Dropout



Training Phase :

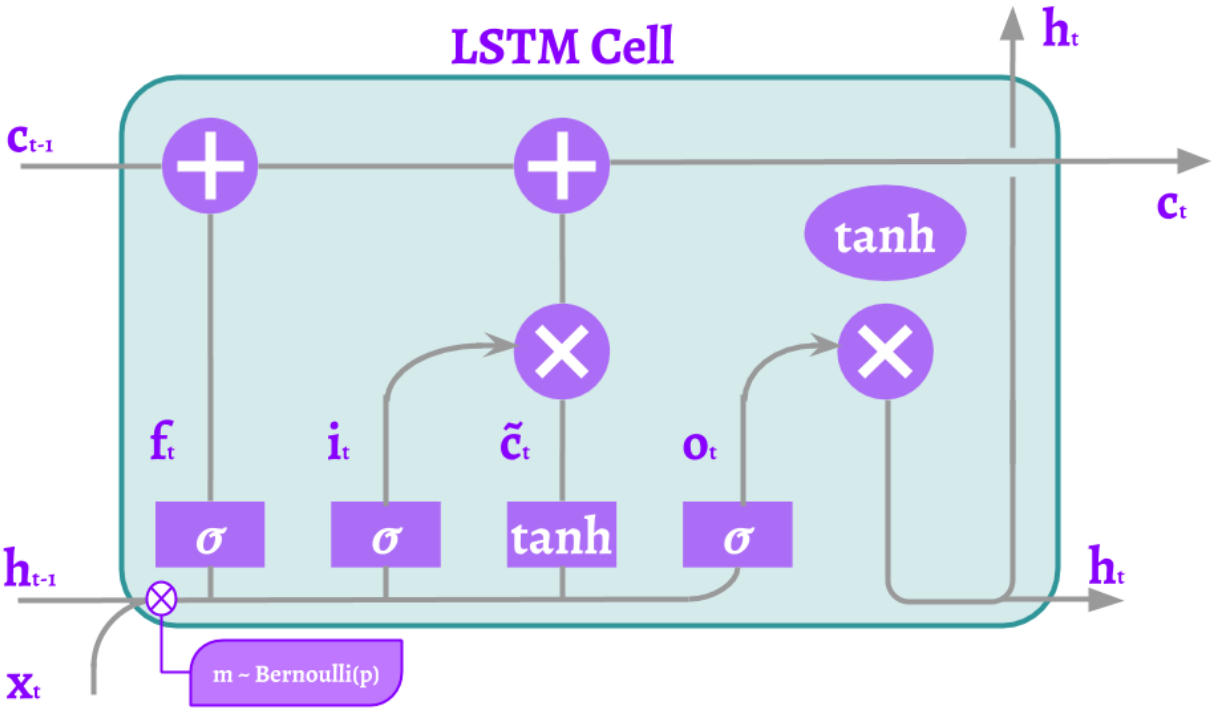
$$\mathbf{c_t} = \mathbf{f_t} \circ \mathbf{h_{t-1}} + \mathbf{i_t} \circ \tilde{\mathbf{c}}_\mathbf{t} \circ \mathbf{m}, \quad m_i \sim Bernoulli(p)$$

Testing Phase :

$$\mathbf{c_t} = \mathbf{f_t} \circ \mathbf{h_{t-1}} + (1-p)\ \mathbf{i_t} \circ \tilde{\mathbf{c}}_\mathbf{t}$$

The **Recurrent Dropout** [12] proposed by S. Semeniuta et al. is an interesting variant. The cell state is left untouched. A dropout is only applied to the **part which updates the cell state**. So at each iteration, Bernoulli's mask makes some elements no longer contribute to the long term memory. But the **memory is not altered**.

Variational RNN dropout



Training Phase :

$$\mathbf{i_t} = \sigma\left(\mathbf{W_i}\left(\begin{bmatrix}\mathbf{x_t}\\ \mathbf{h_{t-1}}\end{bmatrix} \circ \mathbf{m}\right)\right)$$

$$\mathbf{f_t} = \sigma\left(\mathbf{W_f}\left(\begin{bmatrix}\mathbf{x_t}\\ \mathbf{h_{t-1}}\end{bmatrix} \circ \mathbf{m}\right)\right)$$

$$\mathbf{o_t} = \sigma\left(\mathbf{W_o}\left(\begin{bmatrix}\mathbf{x_t}\\ \mathbf{h_{t-1}}\end{bmatrix} \circ \mathbf{m}\right)\right)$$

$$\tilde{\mathbf{c}}_\mathbf{t} = tanh\left(\mathbf{W_{\tilde{c}}}\left(\begin{bmatrix}\mathbf{x_t}\\ \mathbf{h_{t-1}}\end{bmatrix} \circ \mathbf{m}\right)\right)$$

$$m_i \sim Bernoulli(p)$$

Testing Phase :

$$\mathbf{i_t} = \sigma\left((1-p)\ \mathbf{W_i}\begin{bmatrix}\mathbf{x_t}\\ \mathbf{h_{t-1}}\end{bmatrix}\right)$$

$$\mathbf{f_t} = \sigma\left((1-p)\ \mathbf{W_f}\begin{bmatrix}\mathbf{x_t}\\ \mathbf{h_{t-1}}\end{bmatrix}\right)$$

$$\mathbf{o_t} = \sigma\left((1-p)\ \mathbf{W_o}\begin{bmatrix}\mathbf{x_t}\\ \mathbf{h_{t-1}}\end{bmatrix}\right)$$

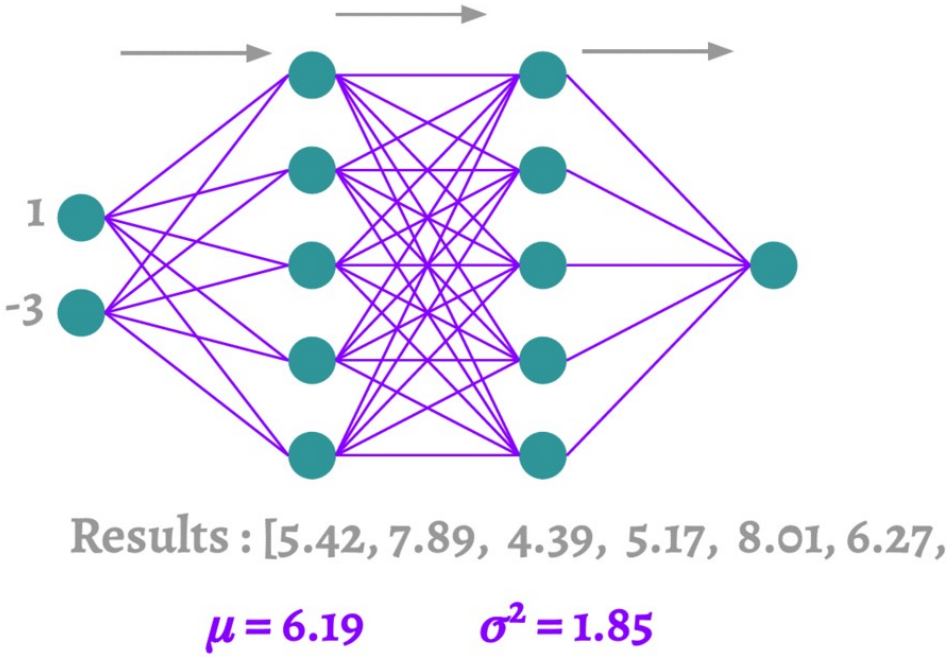
$$\tilde{\mathbf{c}}_\mathbf{t} = tanh\left((1-p)\ \mathbf{W_{\tilde{c}}}\begin{bmatrix}\mathbf{x_t}\\ \mathbf{h_{t-1}}\end{bmatrix}\right)$$

Finally, simple but efficient, **RNN Dropout** [13] introduced by Y. Gal and Z. Ghahramani is the application of a sequence-based dropout before the **internal gates**. This causes a dropout on the **different points** of the LSTM.

Open our minds

There are still **a lot** of different dropout methods but we will stop here for this article. To finish if, I found it was very interesting to know that **Dropout methods are not only regularization methods**.

Monte Carlo Dropout



Dropout methods can also provide an **indicator** of model **uncertainty**. Let me explain. For the same input, the model experiencing a dropout will have a **different architecture** at each iteration. This leads to a **variance** in the **output**. If the network is **fairly generalized** and if the **co-adaptation** is limited then the prediction is **distributed** throughout the model. This leads to **lower variance** on the output at each iteration with the same input. Studying this variance can give an idea of the **confidence** which can be assigned to the model. This can be seen with the Y. Gal and Z. Ghahramani approach [14].

Model Compression

Finally and **intuitively**, by applying dropouts randomly, we can see the **efficiency or inefficiency** of given neurons for the prediction. Following this observation, we can **compress** the model by **reducing the number of parameters while minimizing performance degradation**. K. Neklyudov et al. [15] have proposed such a method using a Variational Dropout to prune DNNs and CNNs.

I hope it will be **useful** for you!
Feel free to follow me or give me **feedback** on what you **liked** or **disliked**. ✏
See you very **soon**! 😘

See why you are so important to me 💗 ?

Sources and References

[1] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, **Improving neural networks by preventing co-adaptation of feature detectors**

[2] L. Wan, M. Zeiler, S. Zhang, Y. LeCun, and R. Fergus, **Regularization of neural networks using dropconnect**

[3] L. J. Ba and B. Frey, **Adaptive dropout for training deep neural networks**

[4] S. Wang and C. Manning, **Fast dropout training**

[5] D. P. Kingma, T. Salimans, and M. Welling, **Variational dropout and the local reparameterization trick**

[6] Y. Gal, J. Hron, A. Kendall, **Concrete Dropout**

[7] H. Wu and X. Gu, **Towards dropout training for convolutional neural networks**

[8] J. Tompson, R. Goroshin, A. Jain, Y. LeCun, and C. Bregler, **Efficient object localization using convolutional networks**

[9] T. DeVries and G. W. Taylor, **Improved regularization of convolutional neural networks with cutout**

[10] S. Park and N. Kwak, **Analysis on the dropout effect in convolutional neural networks**

[11] T. Moon, H. Choi, H. Lee, and I. Song, **Rnndrop**

[12] S. Semeniuta, A. Severyn, and E. Barth, **Recurrent dropout without memory loss**

[13] Y. Gal and Z. Ghahramani, **A theoretically grounded application of dropout in recurrent neural networks**

[14] Y. Gal and Z. Ghahramani, **Dropout as a bayesian approximation: Representing model uncertainty in deep learning**

[15] K. Neklyudov, D. Molchanov, A. Ashukha, and D. P. Vetrov, **Structured bayesian pruning via log-normal multiplicative noise**

[16] A. Labach, H. Salehinejad, **Survey of Dropout Methods for Deep Neural Networks**

All the images and GIFs are homemade and free to use