Artificial Intelligence Review (2020) 53:3947–3986
https://doi.org/10.1007/s10462-019-09784-7

A survey of regularization strategies for deep models

Reza Moradi[1] · **Reza Berangi**[1] · **Behrouz Minaei**[1]

Published online: 5 December 2019

Abstract
The most critical concern in machine learning is how to make an algorithm that performs well both on training data and new data. No free lunch theorem implies that each specific task needs its own tailored machine learning algorithm to be designed. A set of strategies and preferences are built into learning machines to tune them for the problem at hand. These strategies and preferences, with the core concern of generalization improvement, are collectively known as regularization. In deep learning, because of a considerable number of parameters, a great many forms of regularization methods are available to the deep learning community. Developing more effective regularization strategies has been the subject of significant research efforts in recent years. However, it is difficult for developers to choose the most suitable strategy for their problem at hand, because there is no comparative study regarding the performance of different strategies. In this paper, at the first step, the most effective regularization methods and their variants are presented and analyzed in a systematic approach. At the second step, comparative research on regularization techniques is presented in which the testing errors and computational costs are evaluated in a convolutional neural network, using CIFAR-10 (https://www.cs.toronto.edu/~kriz/cifar.html) dataset. In the end, different regularization methods are compared in terms of accuracy of the network, the number of epochs for the network to be trained and the number of operations per input sample. Also, the results are discussed and interpreted based on the employed strategy. The experiment results showed that weight decay and data augmentation regularizations have little computational side effects so can be used in most applications. In the case of enough computational resources, Dropout family methods are rational to be used. Moreover, in the case of abundant computational resources, batch normalization family and ensemble methods are reasonable strategies to be employed.

Keywords Deep learning · Convolutional neural network · Regularization · Overfitting · Generalization

✉ Reza Berangi
rberangi@iust.ac.ir

Reza Moradi
re_moradi@comp.iust.ac.ir

Behrouz Minaei
b_minaei@iust.ac.ir

[1] Iran University of Science and Technology, Tehran, Iran

1 Introduction

Machine learning techniques are changing traditional algorithm design methodology for complex tasks, in such a way that, complex algorithms are learned automatically based on the training data. The results are so attractive that machine learning usage is spreading rapidly throughout computer science and other fields. A recent report from the McKinsey Global Institute asserts that machine learning will be the driver of the next big wave of innovation (Henke et al. 2016). The fundamental goal of machine learning is to generalize beyond the examples in the training set. The main problem is that the process that generates examples is a stochastic process and it is implausible we see those exact examples again in the future. In order to generalize, a learner must be fed with some expertise or conjecture in addition to the training data. According to the famous "no free lunch" theorem (Wolpert 1996), if we consider learning a particular function over all possible functions, no learner can do better than random guessing. While this is frustrating, the truth is that the function we are looking for is not drawn uniformly from the set of all possible functions in the world. Fortunately, assumptions such as smoothness, complexity constraints, limited dependency and similarity of elements in a class are enough to have good results in machine learning. In this case, induction acts like a knowledge lever that can boost input knowledge. Depending on the kind of knowledge a learning machine can express, an appropriate one is selected for the task at hand. For example, if we have much knowledge about samples similarity in our domain, instance-based learning[1] methods may be a good choice (Aha et al. 1991). On the other hand, probabilistic graphical models[2] are appropriate when we have information about probabilistic dependencies (Ghahramani 2015). Alternatively, if we know there are local spatial dependencies and redundancy information in the input, convolutional networks are good candidates (LeCun et al. 2015).

Regularization is a way of infusing "common sense" knowledge about parameters that the learning algorithm might otherwise not be able to glean from data. Such "side information" is most valuable to a learner when the data itself is insufficient. In general, any component of the learning process or prediction procedure that is added to alleviate data shortage is called regularization. In fact, in the process of regularization, by accepting some bias, the model variance is decreased. In machine learning, we can break data sample points into two components, a pattern plus a stochastic noise. Any machine learning algorithm must be able to model the pattern and ignore the noise. An algorithm that emphasis on fitting the noise and peculiarities in addition to the pattern is going toward overfitting. In this case, Regularization can help to choose an appropriate model complexity, so that the model is better at prediction in the future.

Deep models are capable of learning complex representational spaces, which are necessary for tackling intricate learning tasks. However, due to the model depth and capacity required to capture such representations, they are often susceptible to overfitting. So, the development of new effective regularization techniques is an inevitable need to tackle with overfitting and generalizability of the model. Many successful regularization techniques have been proposed like Dropout (Hinton et al. 2012; Srivastava et al. 2014), Batch normalization (Ioffe and Szegedy 2015). The study and development of new regularization methods and techniques are playing a vital role in the improvement of existing deep

[1] A learning algorithm that compares new problem instances with instances in the training set.

[2] A model in which a graph expresses the conditional dependence structure between random variables.

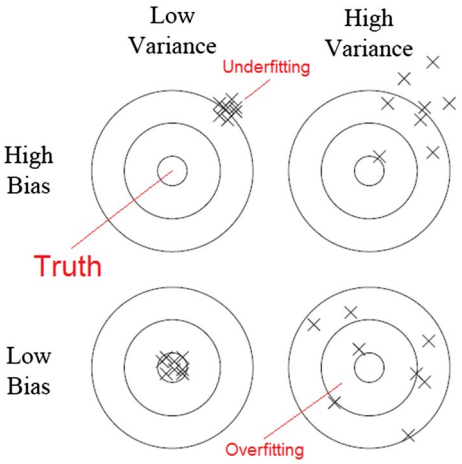

Fig. 1 Bias and variance in dart-throwing analogy (Domingos 2012)

learning systems. However, it is difficult for developers to choose the most suitable strategy for their problem at hand, because there is no comparative study regarding the performance and computational cost of different strategies.

In Peng et al. (2015) four regularization strategies for embedding-based neural networks for NLP[3] are compared systematically. The results of the study are: (1) Regularization methods basically help generalization. (2) Penalizing L_2-norm of embeddings unexpectedly helps optimization. (3) Regularization performance depends mostly on the dataset's size. (4) L_2 penalty mainly acts as a local effect and hyper-parameters can be tuned incrementally. (5) Combining dropout and L_2 penalty further improves generalization, and their coefficients are complementary.

In this paper, the relation between regularization and overfitting is investigated in the next section. Then, the need for regularization in high dimensional feature spaces is considered. Next, deep learning as an effective way to learn representations is introduced. After that, a review of regularization strategies that are heavily used in the literature in recent years is presented. It consists of successful traditional methods like norm penalties that are still popular and more recent architectural approaches that are used effectively in very deep models. In the end, experimental results are presented in which a deep network is presented, and different regularization strategies are compared in terms of classification accuracy and learning computational cost. Finally, in the discussion and conclusion sections, qualitative perspectives and practical recommendations are proposed.

2 Regularization and overfitting

Generally, there is a risk of overfitting for a complex model when the training error is low, and the testing error is high. Sometimes strong false assumptions can be better than weak, true ones because a learner with the latter needs more data to avoid overfitting. Overfitting comes in many forms that are not immediately obvious. One way to understand overfitting is to decompose generalization error into bias and variance (Domingos 2000). The bias is

[3] Natural Language Processing.

a learner's tendency to consistently learn the same wrong thing, while the variance is the tendency to learn random things irrespective of the real pattern.

Figure 1 illustrates this by an analogy with throwing darts at a board. A linear learner has a high bias when the frontier between two classes is not a hyperplane to be induced. On the other hand, deep models do not have this problem because they can represent any complex function, but they can suffer from the high variance. For example, a deep convolutional network learned based on different training sets of the same phenomenon, often find very different values for the learnable parameters of the model.

One way to combat overfitting is cross-validation. However, the model overfits if cross-validation is used to make too many hyper-parameter choices (Ng 1997). On the other hand, doing a statistical significance test like Chi square on a sub-structure before adding it, helps to decide whether the distribution of the class is different with or without this sub-structure. Generally, these techniques are particularly useful when data is very scarce. The most popular way to attack overfitting is regularization. In this way, by penalizing classifiers with more capacity, and favoring smaller ones with less room, we can avoid overfitting.

Nevertheless, we cannot be sure that a particular regularization method solves our overfitting problem safely. It is easy to avoid overfitting by falling into the opposite problem i.e., underfitting. Simultaneously avoiding both require learning a perfect classifier for the problem at hand, that of course, is not known in advance. That is why no single technique will always work perfectly. (no free lunch). So we should employ regularization carefully for every problem and devise some experimentations to make sure about the right capacity of the resulted model.

Generally, in artificial intelligence, the process that generates data is complex and unknown. Deep learning tries to handle this situation by employing models with many parameters that have the required capacity. Although we have millions of samples to train these giant models, they can overfit easily. Regularization methods have an essential role in protecting deep models against overfitting, and almost all deep models have some kind of regularization to handle the complexity of their high dimensional parameter space.

3 Regularization problems in high dimensions

Correct generalization becomes exponentially harder as the dimensionality of the examples grows because of the "curse of dimensionality". This expression was coined first by Bellman in 1961 to refer to the fact that many algorithms that work fine in low dimensions become intractable when the input is high-dimensional. In high dimensions, irrelevant features cause many problems in similarity-based reasoning like nearest neighbor classifiers. As the dimensionality increases, more and more examples become nearest neighbors of a typical sample. In other words, most of the mass of a multivariate Gaussian distribution is not near the mean, but in an increasingly distant "shell" around it. One might think that gathering more features never hurts since at worst they provide no new information about the class. However, their benefits may be outweighed by the curse of dimensionality (Domingos 2012).

Fortunately, there is an effect that partly counteracts the curse, which might be called the "blessing of non-uniformity". In most applications like vision, speech and natural language processing, examples are not spread uniformly throughout the instance space but are concentrated on or near a lower-dimensional manifold. Deep learning can implicitly take

advantage of this lower effective dimension, and there is no need to devise a particular regularization method to handle the situation.

4 Deep learning

Traditional machine-learning techniques were limited in their abilities to take original data in its raw form as input to the model. For decades, making machine-learning systems required careful domain expertise to design feature extractors that transformed the raw data into feature vectors from which the learning subsystem could classify patterns in the input.

In representation learning a machine is fed with raw data and informative features are extracted automatically for the task at hand. Deep learning is one of the most effective ways of representation learning in which a hierarchy of simple non-linear modules are put in layers that each transform the representation at one level into representation at a higher, slightly more abstract level. By putting enough such transformations, very complex functions can be learned. Transformations are designed so that for the task at hand, irrelevant variations are suppressed by higher layers. For example, a raw picture is an array of pixel values. At the first layer, filters for detecting edges with different orientations and color blobs are learned. The second layer learns filters to detect particular arrangements of edges called motifs. Filters that assemble motifs to make parts of objects are learned by the third layer. This trend is followed, and subsequent layers detect objects as combinations of these parts. The key aspect of deep learning is that these layers of features detectors are not designed by hand employing domain knowledge. They are merely learned from data using popular backpropagation and gradient descent.

Deep learning has made significant advances in solving some open problems in artificial intelligence. It can easily discover complex structures in high-dimensional data, and promising results have obtained in business, science and other domains. In addition to beating records in image recognition (Krizhevsky et al. 2012; Szegedy et al. 2014; He et al. 2016) and speech recognition (Hinton et al. 2012), it has beaten other machine learning techniques at predicting the activity of potential drug molecules (Ma et al. 2015), reconstructing brain circuits (Helmstaedter et al. 2013), and predicting genetic determinants of disease (Xiong et al. 2015). Also, deep learning has produced promising results for various tasks in natural language understanding (Collobert et al. 2011), particularly in topics classification, sentiment analysis, question answering (Bordes et al. 2014) and language translation (Sutskever et al. 2014). Because of advances in computation hardware and an increase in the amount of available data, it is anticipated that deep learning will have many more successes in the near future. Besides, new learning algorithms, regularization methods and architectures that are currently being developed will accelerate this progress.

5 Regularization strategies

The primary target of machine learning is to find a model based on the training set that is generalizable to samples not present in the training set. Employing regularization is one of the main strategies to this end. Any arrangement in the model architecture, learning process or inference that is used to compensate the shortage of training data is called regularization. These arrangements are in the form of constraints and penalties on hypothesis search space that guide our learning process to simpler models. In the following sections,

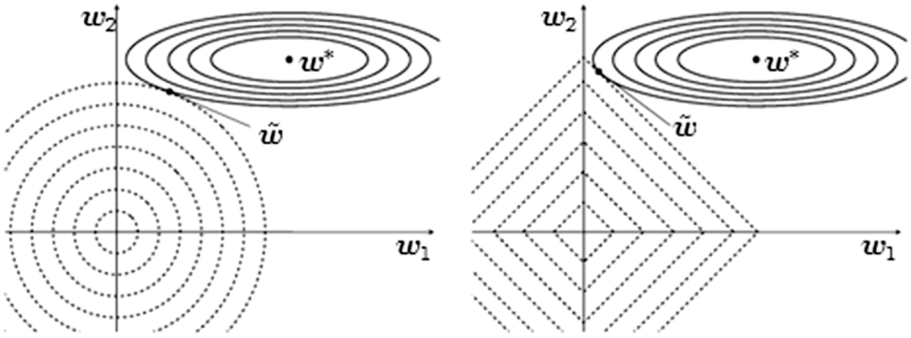

Fig. 2 Comparison of L_2 (left) and L_1 (right) norms in an optimization setting (Goodfellow et al. 2016)

different techniques of regularization employed in deep models are presented. In this study, the target function J(θ; X,y) has a set of learnable parameters, denoted by θ. The learning process is supposed to minimize J over θ based on the training dataset i.e., X,y.

5.1 Weight decay

One of the most original and trivial methods of regularization is to constrain the capacity of the model by adding some penalty function to the original objective function J and making a new objective function J' according to Eq. (1).

$$J'(\theta; X,y) = J(\theta; X,y) + \alpha\Omega(\theta) \tag{1}$$

In Eq. (1), $\Omega(\theta)$ is the regularization term that is controlled by the parameter α. The regularizer, Ω, here consists of the norms of the model trainable parameters. To simplify the presentation, we assume no penalty on bias parameters, so θ is assumed to be the network weights, i.e., *w*. As shown in Fig. 1, depending on the kind of norm, different regularization behavior will happen while training. In Fig. 2 we see that the regularization term moves the optimal point from the point w^* to $\tilde{w}$. Because of the curvature of the main objective function J(θ;X, y), the regularization terms in the Eq. 1 moves the original optimal point w^* along the negative direction of the w_1 axes to $\tilde{w}$. Comparing the two norm types, for L_2 norm, the gradient of the regularization term along w_1 becomes negligible long before that for L_1 norm. So employing L_1 norm penalty usually guides the optimization process to more sparse solutions.

L_1 Regularization has been used to attain sparse solutions. L_1 regularization and its variants such as group sparsity regularization seem to be appropriate for deep neural networks (Wen et al. 2016) in order to attain reduced computation and power consumption. On the other hand, L_2 regularization smooths the parameter distribution and thus reducing the magnitude of parameters, resulting in the model to be less prone to overfitting. L_2 regularization plays an essential role in training deep neural networks (Krizhevsky et al. 2012) and has achieved high performance when combined with dropout regularization (Srivastava et al. 2014). However, it is studied that L_2 regularization does not affect regularization when combined with batch normalization (Laarhoven 2017).

Weight decay regularization has produced excellent results in terms of accuracies when applied to the convolutional neural networks for visual recognition tasks including handwritten digits recognition, gender classification, ethnic origin recognition and, object

recognition (Yu et al. 2009). Also, it has been reported in Krizhevsky et al. (2012) that weight decay regularization, when combined with Dropout, reduces overfitting effectively. Shakeout (Kang et al. 2016) is a simple and effective regularization method for training deep neural networks. Unlike dropout which works by implicitly imposing an L_2 regularizer on weights (Wager et al. 2013), Shakeout uses a combination of L_1 and L_2 regularization imposed on the weights. In other words, Shakeout is equivalent to introducing elastic net-like regularization. Empirical evaluation of Shakeout regularization demonstrated that sparse network weights are obtained and tests on MNIST and CIFAR10 datasets show that shakeout reduced overfitting problem effectively.

5.1.1 Sparse representation

Sparse representation is used to reduce dimensionality and to make interpretable representations by adding a particular penalty term to the model cost function. This penalty lets only a few of the representation layer neurons to be active at the same time. As an example, the following penalty can be used:

$$\text{Sparsity Penalty} = \sum_{j=1}^{L}\left(\rho_0 \log\frac{\rho_0}{\hat{\rho}_j} + (1-\rho_0)\log\frac{1-\rho_0}{1-\hat{\rho}_j}\right), \quad \hat{\rho}_j = \frac{1}{N}\sum_{i=1}^{N}\left(a_j\left(x^{(i)}\right)\right) \tag{2}$$

In this equation, N denotes the number of samples in the mini-batch of the training set, L denotes the number of neurons in the layer, and ρ_0 is the sparsity rate that is usually set to a small number like 0.05. Also, $\hat{\rho}_j$ estimates average activity of the neuron, j, overall N samples. In Eq. 2, the sparsity penalty is defined as the Kullback–Leibler distance between the desired rate ρ_0, and the average activity rate of all M neurons of the representation layer $\hat{\rho}_j$. In this way, during the learning process, the weights of the neurons are so tuned to let only a small portion of the neurons to be active at the same time.

5.1.2 Bayesian interpretation

In the Bayesian viewpoint, parameters of the model are considered as random variables. In this case, before the model experience the input data, every parameter has a prior probability distribution, and estimation is done by integration over all the values of it. In both Bayesian estimation and regularization, prior information and the information that is extracted from the data are blended together. For example, in MAP estimation, prior distribution and likelihood of the data are synthesized as follows

$$\log p\left(\theta|x^{(1)},\ldots,x^{(m)}\right) \propto \log p(\theta) + \sum_i \log p\left(x^{(i)}|\theta\right) \tag{3}$$

In this equation, the prior information $\log p(\theta)$, impacts as an additive term to the log-likelihood of training data that in this case, is the main objective function. So, we can interpret the prior information as a regularization penalty in maximizing the log-likelihood. In this case, Gaussian prior behave like L_2 norm regularization and Laplacian prior is equivalent to L_1 norm regularization. It is important to note that the regularization concept is more general than the priors. That is because a general regularization term does not have probability distribution limitations. Besides, we cannot interpret all main objective functions as the likelihood of the data.

5.1.3 Constrained optimization interpretation

In machine learning, unlike most other optimization problems, we do not have access to the function we want to optimize! Here, we have to use the training error as a surrogate for test error, and this is fraught with danger. On the positive side, since the objective function is only a proxy for the real goal, we may not need to optimize it fully. In fact, a local optimum returned by a simple greedy search algorithm may be better than the global optimum.

An alternative to adding a penalty term to the objective function for regularization is to add a constraint to the main optimization problem. For instance, the constraint $\Omega(\theta) < \mathrm{k}$ can be added to the main problem that k is some constant

$$\begin{aligned} &\min \mathrm{J}(\theta; \mathrm{X}, \mathrm{y}) \\ &\text{st. } \Omega(\theta) < \mathrm{k} \end{aligned} \tag{4}$$

The Lagrangian of the above optimization problem has the following form:

$$\mathcal{L}(\theta, \alpha; \mathrm{X}, \mathrm{y}) = \mathrm{J}(\theta; \mathrm{X}, \mathrm{y}) + \alpha(\Omega(\theta) - \mathrm{k}) \tag{5}$$

Using KKT conditions, the following optimization problem can be derived:

$$\theta^* = \min_{\theta} \left(\max_{\alpha, \alpha \geq 0} \mathcal{L}(\theta, \alpha) \right) \tag{6}$$

Solving the inner problem and finding the optimal value of $\alpha = \alpha^*$ we come to

$$\theta^* = \min_{\theta} \left(\mathcal{L}(\theta, \alpha^*) \right) = \min_{\theta} \mathrm{J}(\theta; \mathrm{X}, \mathrm{y}) + \alpha^* \Omega(\theta) \tag{7}$$

In this equation, some multiple of constraining function is added as a penalty term to the main objective function to mimic the regularization term.

We can also solve the problem of Eq. (4) employing the projection method. In this case, at the first step, the main objective function is solved without any constraint and at the second step, the result is projected onto the added constraint. The idea here is that we can regularize learning problems employing constraints as projections. In this way, we can increase the learning rate, and there is no need to solve the inner problem to find θ^*.

5.2 Adding noise

Adding noise to the systems can judiciously cause the systems to be more generalizable and prevent overfitting (Poole et al. 2014; Hochreiter and Schmidhuber 1995). In deep learning, noise can be added either to the input data or to the weights of the network. Adding noise to the input data has a relatively long history in data pre-processing and in some literature, it is referred to as dithering. In deep learning, adding noise to the input data hinders memorizing but preserves learnability. The idea of adding noise has similar effects on different problems in machine learning. Here, we give an example of the regression problem. In a regression problem, suppose y is the ground truth value and $\hat{\mathrm{y}}(\mathrm{x})$ is the output of the network. The cost function using MSE is defined as follows

$$\mathrm{J} = \mathbb{E}_{\mathrm{p(x,y)}} \left[(\hat{\mathrm{y}}(\mathrm{x}) - \mathrm{y})^2 \right] \tag{8}$$

In this equation, p(x,y) is a non-negative Lebesgue-integrable function (Bartle 1995) whose value at any given sample in the sample space can be interpreted as providing a relative likelihood that the value of the random variable would equal that sample. In the following two sub-sections, adding noise to the data and weights of the regression model are studied separately.

5.2.1 Adding noise to the data for the regression problem

If a Gaussian noise of $\in \sim (0, \nu \mathrm{I})$ is added to the input data, then we have:

$$\bar{\mathrm{J}}_{\mathrm{x}} = \mathbb{E}_{\mathrm{p(x,y,}\epsilon)}\left[(\hat{\mathrm{y}}(\mathrm{x}+\epsilon)-\mathrm{y})^2\right] \tag{9}$$

Expanding the Taylor series for $\hat{\mathrm{y}}(\mathrm{x}+\epsilon)$ and simplifying the right side, we get the following result:

$$\bar{\mathrm{J}}_{\mathrm{x}} = \mathbb{E}_{\mathrm{p(x,y)}}\left[(\hat{\mathrm{y}}(\mathrm{x})-\mathrm{y})^2\right] + \nu \mathbb{E}_{\mathrm{p(x,y)}}\left[\nabla_{\mathrm{x}}\hat{\mathrm{y}}(\mathrm{x})^2\right] + \mathrm{O}\left(\nu^2\right) \tag{10}$$

It is clear that adding Gaussian noise with a small variance to the input data is equivalent to adding the penalty term of $\nu \mathbb{E}_{\mathrm{p(x,y)}}\left[\nabla_{\mathrm{x}}\hat{\mathrm{y}}(\mathrm{x})^2\right]$ to the cost function. In this way, higher values of the gradient of $\hat{\mathrm{y}}(\mathrm{x})$ cause severe penalty to be added to the cost function, tending to reduce the sensitivity of the network to the small changes in the input and also improving the local robustness of the model that can be viewed as a regularization effect. It is noteworthy that for a linear regression problem, the added penalty has the form of the norm 2 function of the weights. Therefore, adding noise to the data has the same effect as the weight decay regularization. In Poole et al. (2014), it is shown that adding noises appropriately to different layers in a neural network, can have a noticeable improvement in the generalizability of the model.

5.2.2 Adding noise to the weights for regression problem

Suppose the Gaussian noise of $\in \mathrm{W} \sim (0, \eta \mathrm{I})$ is added to the weights of the network, then we have:

$$\bar{\mathrm{J}}_{\mathrm{W}} = \mathbb{E}_{\mathrm{p(x,y,}\in \mathrm{W})}\left[\left(\hat{\mathrm{y}}_{\in \mathrm{W}}(\mathrm{x})-\mathrm{y}\right)^2\right] \tag{11}$$

Expanding the Taylor series for $\hat{\mathrm{y}}_{\in \mathrm{W}}(\mathrm{x})$ and simplifying the right side, we get the following result:

$$\bar{\mathrm{J}}_{\mathrm{W}} = \mathbb{E}_{\mathrm{p(x,y)}}\left[(\hat{\mathrm{y}}(\mathrm{x})-\mathrm{y})^2\right] + \eta \mathbb{E}_{\mathrm{p(x,y)}}\left[\nabla_{\mathrm{w}}\hat{\mathrm{y}}(\mathrm{x})^2\right] + \mathrm{O}\left(\eta^2\right) \tag{12}$$

Assuming a small amount of Gaussian noise ($\mathrm{O}\left(\eta^2\right) \approx 0$), leaves a penalty term $\eta \mathbb{E}_{\mathrm{p(x,y)}}\left[\nabla_{\mathrm{w}}\hat{\mathrm{y}}(\mathrm{x})^2\right]$, added to the cost function. This regularizing term causes the optimal weights to be searched in the vicinity of low gradient regions. In other words, there is a little sensitivity to the small variations of the weights in these regions (Hochreiter and Schmidhuber 1995). It is noticeable that for a linear regression problem, the mentioned penalty will be constant and has no effect.

5.3 Dropout

Deep neural nets with lots of learnable parameters are very mighty machine learning models. However, overfitting is a severe problem in such networks. Training large neural networks is also an expensive process, making it difficult to deal with overfitting by combining the predictions of multiple trained neural nets at test time. Dropout (Hinton et al. 2012c) is a technique for addressing this problem. The key idea is that at each training stage, individual nodes are either kept with probability p or dropped out of the net with probability $1-p$. In this case, incoming and outgoing edges to a dropped-out node are also removed. In this way, a large model that overfits easily is considered as the base model, and repeatedly smaller sub-models are sampled and trained. Since all the sub-models share their parameters with the base model, this process trains the large model simultaneously, which is ultimately used at test time.

During training, Dropout technique samples from an exponential number of different "thinned" networks. At test time, it is easy to approximate the effect of averaging the predictions of all these thinned networks by merely using a single un-thinned network that has smaller weights. This prevents neurons from co-adaptation. In this way, we will have an implicit bagging effect with a much smaller cost. This phenomenon significantly reduces overfitting and gives significant advantages over other regularization methods. In Srivastava et al. (2014) it is shown that dropout improves the performance of neural networks on supervised learning tasks in vision, speech recognition, document classification, and computational biology, obtaining state-of-the-art results on many benchmark datasets. Dropout mechanism is illustrated in Fig. 3. In this figure, a(.) denotes the activation functions in the output of neurons and d(•) denotes the zero–one pattern vector that is multiplied element wise. The 0 s and 1 s in the vector correspond to dropped and not dropped neurons.

In the following, for simplicity, a Softmax regression model is considered, and the effect of dropout on the model is studied. Assuming the n-dimensional input variable is represented by the vector v, the output of the regression model without dropout is as follows.

$$\mathrm{P}(\mathrm{y}=\mathscr{y}|\mathrm{v}) = \mathrm{Softmax}\left(\mathrm{W}^{\mathrm{T}}\mathrm{v}+\mathrm{b}\right)_{\mathscr{y}} \tag{13}$$

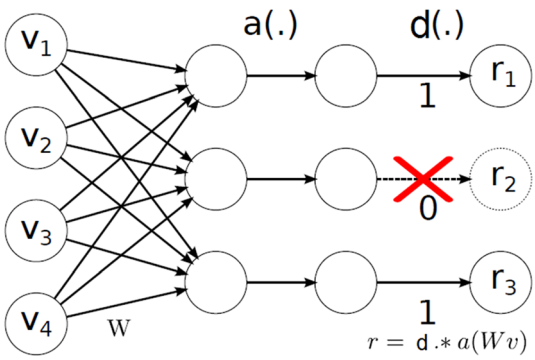

Fig. 3 Visual representation of Dropout (Wan et al. 2013)

The output of the network employing dropout is as follows. In this equation, the 0 s and 1 s in vector d correspond to dropped and not dropped neurons, respectively and $\odot$ denotes element wise or Hadamard product.

$$\mathrm{P}(\mathrm{y} = \mathscr{y}|\mathrm{v}; \mathrm{d}) = \mathrm{Softmax}\left(\mathrm{W}^{\mathrm{T}}(\mathrm{d} \odot \mathrm{v}) + \mathrm{b}\right)_{\mathscr{y}} \tag{14}$$

By taking a Geometric average of different models and normalizing the result, we get the following equations

$$\bar{\mathrm{P}}_{\mathrm{ensemble}}(\mathrm{y} = \mathscr{y}|\mathrm{v}) = \sqrt[2^{\mathrm{n}}]{\prod_{\mathrm{d}' \in \{0,1\}^{\mathrm{n}}} \mathrm{P}\left(\mathrm{y} = \mathscr{y}|\mathrm{v}; \mathrm{d}'\right)} \tag{15}$$

$$\mathrm{P}_{\mathrm{ensemble}}(\mathrm{y} = \mathscr{y}|\mathrm{v}) = \frac{\bar{\mathrm{P}}_{\mathrm{ensemble}}(\mathrm{y} = \mathscr{y}|\mathrm{v})}{\sum_{\mathscr{y}'} \bar{\mathrm{P}}_{\mathrm{ensemble}}(\mathrm{y} = \mathscr{y}'|\mathrm{v})} \tag{16}$$

By putting Eq. 14 into Eq. 15, we come to the following equation:

$$\bar{\mathrm{P}}_{\mathrm{ensemble}}(\mathrm{y} = \mathscr{y}|\mathrm{v}) = \sqrt[2^{\mathrm{n}}]{\prod_{\mathrm{d}' \in \{0,1\}^{\mathrm{n}}} \mathrm{Softmax}\left(\mathrm{W}^{\mathrm{T}}\left(\mathrm{d}' \odot \mathrm{v}\right) + \mathrm{b}\right)_{\mathscr{y}}} \tag{17}$$

By putting the above equation into Eq. 16, we come to the following equation. In this equation, $\mathrm{W}_{\mathscr{y},:}$ denotes row $\mathscr{y}$ of the matrix W and W^{T} dentotes transpose of the matrix W.

$$\begin{aligned}
\bar{\mathrm{P}}_{\mathrm{ensemble}}(\mathrm{y} = \mathscr{y}|\mathrm{v}) &= \frac{\sqrt[2^{\mathrm{n}}]{\prod_{\mathrm{d}' \in \{0,1\}^{\mathrm{n}}} \exp\left(\mathrm{W}^{\mathrm{T}}_{\mathscr{y},:}\left(\mathrm{d}' \odot \mathrm{v}\right) + \mathrm{b}_{\mathscr{y}}\right)}}{\sqrt[2^{\mathrm{n}}]{\prod_{\mathrm{d}' \in \{0,1\}^{\mathrm{n}}} \sum_{\mathscr{y}'} \exp\left(\mathrm{W}^{\mathrm{T}}_{\mathscr{y}',:}\left(\mathrm{d}' \odot \mathrm{v}\right) + \mathrm{b}_{\mathscr{y}'}\right)}} \\
\bar{\mathrm{P}}_{\mathrm{ensemble}}(\mathrm{y} = \mathscr{y}\mathrm{v}) &\propto \sqrt[2^{\mathrm{n}}]{\prod_{\mathrm{d}' \in \{0,1\}^{\mathrm{n}}} \exp\left(\mathrm{W}^{\mathrm{T}}_{\mathscr{y},:}\left(\mathrm{d}' \odot \mathrm{v}\right) + \mathrm{b}_{\mathscr{y}}\right)}
\end{aligned} \tag{18}$$

In Eq. (18), because $\bar{\mathrm{P}}_{\mathrm{ensemble}}$ will be normalized, we can safely ignore multiplication by factors that are constant with respect to $\mathscr{y}$.

$$\begin{aligned}
\bar{\mathrm{P}}_{\mathrm{ensemble}}(\mathrm{y} = \mathscr{y}|\mathrm{v}) &\propto \exp\left(\frac{1}{2^{\mathrm{n}}} \sum_{\mathrm{d}' \in \{0,1\}^{\mathrm{n}}} \left(\mathrm{W}^{\mathrm{T}}_{\mathscr{y},:}\left(\mathrm{d}' \odot \mathrm{v}\right) + \mathrm{b}_{\mathscr{y}}\right)\right) \\
&= \exp\left(\frac{1}{2^{\mathrm{n}}}\left(2^{\mathrm{n}-1}\mathrm{W}^{\mathrm{T}}_{\mathscr{y},:}\mathrm{v} + 2^{\mathrm{n}}\mathrm{b}_{\mathscr{y}}\right)\right) \\
\bar{\mathrm{P}}_{\mathrm{ensemble}}(\mathrm{y} = \mathscr{y}|\mathrm{v}) &\propto \exp\left(\frac{1}{2}\mathrm{W}^{\mathrm{T}}_{\mathscr{y},:}\mathrm{v} + \mathrm{b}_{\mathscr{y}}\right)
\end{aligned} \tag{19}$$

In Eq. (19), because the set $\{0, 1\}^{\mathrm{n}}$ has 2^{n} members and each element of d' is one half of the times, the equation is simplified as follows. According to Eq. 19, we have derived a Softmax regression classifier with the weight matrix of $\frac{1}{2}\mathrm{W}$.

It is noticeable, the above derivation is for a simple Softmax regression classifier, but experimental results show that it can be used safely for general MLP neural networks with non-linear activation functions.

In Goodfellow et al. (2013) it is shown that classification using dropout is better than 1000 sub-networks using Monte Carlo approximation. Also in Srivastava et al. (2014), it is shown that using dropout is better than other low-cost methods like L_1 and L_2 penalties. Dropout has advantages and disadvantages that make it application dependent. Computation and memory complexities of dropout are of order O(n) (n is the number of nodes). Dropout can be used safely in all cases that we have distributed representations such as RNNs and RBMs. Also, stochastic gradient descent can be used safely for training the network. Using dropout moves us to bigger optimal models, but it takes more time to converge. Generally, in situations that we have more than 15 million training samples, using this method is not recommended. On the other hand, if we have less than 5 thousand training sample, other regularization methods like Bayesian neural networks should be used. For small neural networks, we can use fast dropout (Wang and Manning 2013), where the average of all sub-networks is computed using a non-stochastic closed form equation. In Maeda (2014) Dropout is studied from Bayesian viewpoint. Bayesian interpretation enables us to optimize the dropout rate, which is beneficial for learning weight parameters and prediction after learning. In this study, the experiment result also encourages the optimization of the dropout. In Bouthillier et al. (2015) it is shown that using dropout in a network can also be interpreted as a kind of data augmentation in the input space without domain knowledge. Random dropout is described as the procedure to generate samples by back-projecting the dropout noise into the input space, thereby generating augmented versions of the training data. Also, a new dropout noise scheme is proposed based on observations and is shown that it improves dropout results on MNIST and CIFAR-10 datasets without adding significant computational cost. Another research work (Gal and Ghahramani 2016) proposed tools to model uncertainty with dropout and developed a framework using dropout training in deep neural networks as approximate Bayesian inference in deep Gaussian processes.

In the following, different variants of Dropout technique is considered and discussed.

5.3.1 Standout

Standout (Ba et al. 2016) is an adaptive dropout network in which a binary belief network is overlaid on a neural network and is used to regularize its hidden units by selectively setting activities to zero. The network can be trained jointly with the neural network by approximately computing local expectations of binary dropout variables, computing derivatives using back-propagation, and using stochastic gradient descent. Interestingly, experiments show that the learned dropout network parameters recapitulate the neural network parameters, suggesting that a good dropout network regularizes activities according to magnitude. Standout yielded better results on MNIST and Norb datasets than other feature learning methods including denoising auto-encoders, and standard dropout.

5.3.2 DropAll

DropAll (Frazão and Alexandre 2014) is the generalization of Dropout and DropConnect that has both the properties of Dropout and DropConnect. In DropAll, we can drop a randomly selected subset of activations, or we can drop a random subset of weights. DropAll improved the classification errors of networks trained with DropOut or DropConnect on a standard image classification datasets.

5.3.3 Curriculum dropout

Curriculum Dropout (Morerio et al. 2017) using a fixed dropout probability during training is a suboptimal choice. Thus a schedule is proposed for the probability of retaining neurons in the network. This induces an adaptive regularization scheme that smoothly increases the difficulty of the optimization problem. The idea of easy-starting and adaptively increasing the difficulty of the learning problem has its roots in curriculum learning and allows one to train better models. In this study, it is proven that the proposed optimization strategy implements a very general curriculum scheme, by gradually adding noise to both the input and intermediate feature representations within the network architecture. Experiments on seven image classification datasets and different network architectures showed that the proposed method, named Curriculum Dropout, frequently yields better generalization and, at worst, performs just as well as the standard Dropout method.

5.3.4 DropMaps

In Moradi et al. (2019) DropMaps is introduced, where for a training batch, each feature is kept with probability p and is dropped with the probability $1-p$. At test time, all feature maps are kept, and everyone is multiplied by p. Like Dropout, DropMaps has regularization effect that causes coincidence of feature maps to be avoided. As presented in the experimental results, DropMaps technique improves the generalizability of the model and protects the learning procedure against overfitting.

5.4 DropConnect

DropConnect (Wan et al. 2013) is the generalization of Dropout that applies only to fully-connected layers in which each connection, rather than each output unit, can be dropped with probability $1-p$. DropConnect is similar to Dropout as it introduces dynamic sparsity within the model but differs in that the sparsity is on the weightsW, rather than the output vectors of a layer. In other words, the fully connected layer with DropConnect becomes a sparsely connected layer in which the connections are chosen at random during the training stage. Note that this is not equivalent to setting W to be a fixed sparse matrix during training. The

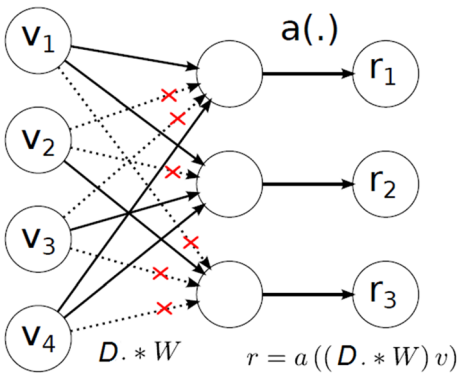

Fig. 4 Visual representation of DropConnct (Wan et al. 2013)

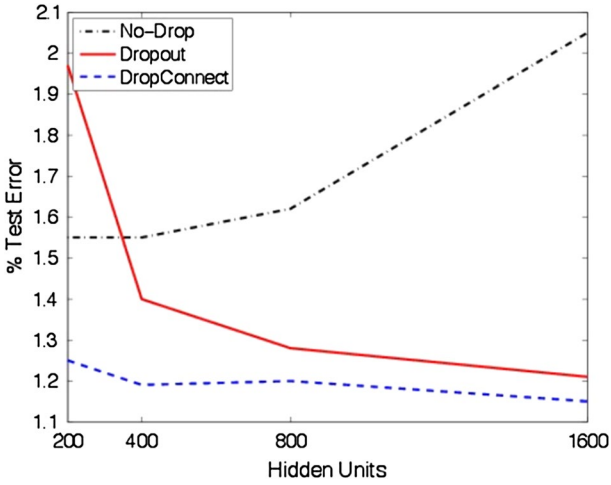

Fig. 5 Regularization effect of Dropout and DropConnect (Wan et al. 2013)

DropConnect mechanism is illustrated in Fig. 4. In this figure, a(•) denotes the activation functions in the output of neurons and D denotes the zero–one pattern matrix that is multiplied element wise. The 0 s and 1 s matrix D correspond to dropped and not dropped weights.

Here, D is a binary matrix of the connection encoding information that $D_{\mathrm{i,j}} \sim Bernoulli(p)$ and operation. * denotes elementwise product. Each element of the mask M is drawn independently for each example during training, essentially instantiating a different connectivity for each example seen. Additionally, the biases are also masked out during training. Regularization effect of dropout and DropConnect is illustrated in Fig. 5. It is noticeable that as the model size increases, No-Drop curve overfits while both Dropout and DropConnect techniques improve performance.

The generalization of a model can be improved by controlling the complexity of the concept class H from which we are choosing a hypothesis. The Rademacher complexity (Shalev-Shwartz and Ben-David 2014) is distribution dependent and can be used to derive data-dependent upper-bounds on the learnability of function classes. Intuitively, a function-class with smaller Rademacher complexity is easier to learn. In Wan et al. (2013) Rademacher complexity after DropConnect $\widehat{R}_{\ell}(\mathcal{F})$ derived as follows

$$\widehat{R}_{\ell}(\mathcal{F}) \leq p\left(2\sqrt{k}mB_s n\sqrt{d}B_h\right)\widehat{R}_{\ell}(\mathcal{G}) \tag{20}$$

In this equation, k is the number of classes, n and m are the dimensionality of the input and output of the DropConnect layer respectively. $\widehat{R}_{\ell}(\mathcal{G})$ and $\widehat{R}_{\ell}(\mathcal{F})$ are the Rademacher complexities before and after DropConnect layer, respectively. Also, we have the conditions $max\left|W_s\right| \leq B_s$, $max|W| \leq B$ where W_s and W are the weights of Softmax and DropConnect layers, respectively. It is noticeable that $\widehat{R}_{\ell}(\mathcal{F})$ is related to p linearly. So in DropConnect, hypothesis complexity of the models decreases linearly as p decreases where p plays the regularization tuning hyper-parameter.

5.5 Ensemble of models

Bagging (Breiman 1994) can decrease the generalization error of the base model by employing multiple models. In this technique, multiple models are trained separately and to determine the final result, outputs of all models are considered. In training different models the main dataset is sampled with replacement, so the main dataset is used differently for different models. This difference causes the same model, train differently, and models with various viewpoints on the dataset are created as a result.

To evaluate this method, consider k different regression models. Suppose error of each model is ϵ_i, with mean, variance and covariance of 0, v, and c respectively. Average of error square is as follows:

$$\begin{aligned} \mathbb{E}\left[\left(\frac{1}{k}\sum_i \varepsilon_i\right)^2\right] &= \frac{1}{k^2}\mathbb{E}\left[\sum_i\left(\varepsilon_i^2 + \sum_{j\neq i}\varepsilon_i\varepsilon_j\right)\right] = \frac{1}{k^2}[kv + k(k-1)c] \\ \mathbb{E}\left[\left(\frac{1}{k}\sum_i \varepsilon_i\right)^2\right] &= \frac{1}{k}v + \frac{k-1}{kc} \end{aligned} \tag{21}$$

In this expression, if covariance is equal to variance, no improvement will happen. However, if the errors of models are independent, the error of ensemble will decrease k times.

5.6 Data augmentation

Increasing the number of training samples is one of the best ways to improve the generalizability of models. However, increasing the size of a dataset is expensive and time-consuming, so datasets are limited in practice. Nevertheless, in some tasks, we can artificially make new data samples using existing dataset. For example, in the image classification task, the model must be robust against some transformations in input images. Hence it is appropriate to apply transition, scale, rotation, reflection and affine transformations to existing images and add the resulted images with the same labels to the dataset. Nowadays, this technique is employed pervasively in various vision applications.

In Taylor and Nitschke (2017), an architecture similar to that described by Zeiler and Fergus (2014), where it is a reasonable tradeoff between topology size and training speed, used to compare various data augmentation schemes. The results are presented in Table 1,

Table 1 Training results for each data augmentation method (in each column the highest accuracy is shown in bold)

Augmentation type	Top-1 accuracy (%)	Top-5 accuracy (%)
Baseline	48.13 ± 0.42	64.50 ± 0.65
Flipping	49.73 ± 0.13	67.36 ± 1.38
Rotating	50.80 ± 0.63	69.41 ± 0.48
Cropping	**61.95 ± 1.01**	**79.10 ± 0.80**
Color Jittering	49.57 ± 0.53	67.18 ± 0.42
Edge Enhancement	49.29 ± 1.16	66.49 ± 0.84
Fancy PCA	49.41 ± 0.84	67.54 ± 1.01

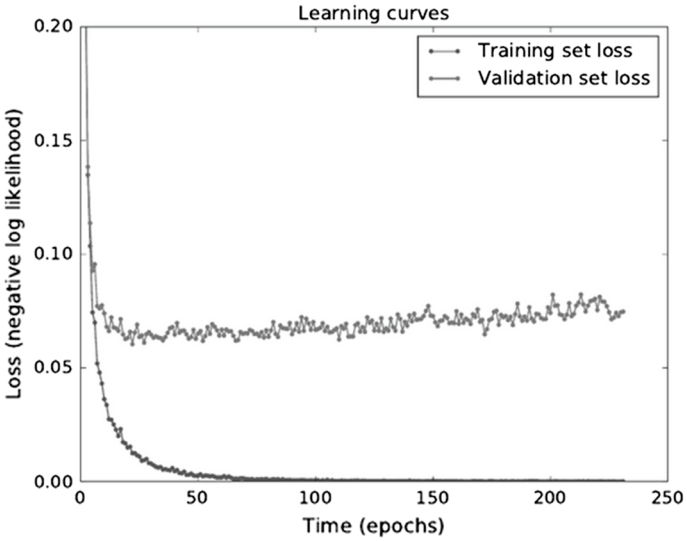

Fig. 6 Training and validation error versus iterations (Goodfellow et al. 2016)

where Top-1 and Top-5 scores were evaluated for each data augmentation method. The Top-1 and Top-5 score is the number of times the correct label is contained within the 1 and 5 highest probabilities, respectively. The standard deviations thus indicate how variable the result is over different testing folds in cross-validation applied to the model. In this table, entries for the geometric and photometric methods are bolded. In all cases, applying data augmentation has improved the performance. Notably, the geometric augmentation schemes outperformed the photometric schemes. The cropping scheme yielded the greatest improvement in Top-1 score, i.e., 13.82%. Results also indicated that Top-5 score has noticeable improvement which conforms the related works (Chatfield et al. 2014; Mash et al. 2016). The reason behind this phenomenon is that cropping generates more training images and allows CNN model exposure to a greater receptive view that reduces the likelihood of over-fitting that in turn increases the overall classification performance. However, the photometric augmentation methods yielded modest improvements in performance compared to the geometric schemes. The most appropriate photometric schemes were found to be color jittering with a Top-1 score improvement of 1.44% and fancy PCA with a Top-5 score improvement of 3.04%. It seems the reason behind the superiority of color jittering is that the augmented images contain more variation compared to the other methods. On the other hand, the edge enhancement method did not yield noticeable performance improvement. The reason behind this observation is that overlaying transformed image onto the source image cannot enhance the contours.

Cutout (DeVries and Taylor 2017) is a technique of randomly masking out rectangular regions of the input image during training. This method can be easily used in conjunction with existing forms of data augmentation and other regularization techniques to improve the performance of convolutional neural networks. Cutout is an extension of dropout but, the comparison between two methods shows that cutout forces the model to take the full image context consideration rather than focusing on a few visual features. Another significant difference between Cutout and dropout is that, in Cutout, the units are dropped at the input stage rather than in the intermediate layers. In this way, the visual features removed

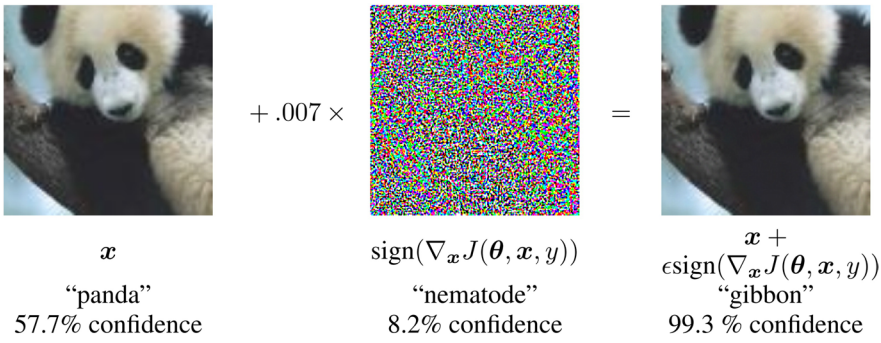

Fig. 7 An adversarial generation example applied to GoogLeNet on ImageNet (Goodfellow et al. 2014)

from input layers are correspondingly removed from all subsequent feature maps. While in case of Dropout, each feature map is considered individually, so the features randomly removed from one feature map may still be present in the others. These inconsistencies produce noise that in turn forces the network to become more robust to noisy inputs. In this way, Cutout method is much closer to the data augmentation than Dropout, as it does not create noise and instead produces images that are novel to the network. Cutout regularization has produced state-of-the-art results when evaluated on the CIFAR-10, CIFAR-100, and SVHN biomarkers.

5.7 Early stopping

In the learning process of big deep models, at first, training and validation errors have a decreasing trend. Continuing with more iterations, decreasing trend of training error continues but at some point; validation error starts to increase (Fig. 6). This phenomenon is due to overfitting of the training model. In early stopping approach, the validation error is checked at every iteration. If the error decreases, the training process continues with more iterations but if the error increases, the training process is forced to stop, and the last registered parameters values are selected as the optimal value for the model parameter. Early stopping is one of the simplest and common methods and can be used simultaneously with other regularization techniques (Goodfellow et al. 2016).

If the cost function is approximated as a quadratic function near the optimal point, early stopping is equivalent to L_2 regularization. In this case, the following relation is established between the number of iterations (τ), the coefficient of L_2 penalty (α) and, learning rate (η).

$$\tau \cong {}^{1}/_{\eta\alpha} \tag{22}$$

In inspecting this relation, it is evident that high values of L_2 penalty corresponds to a small number of iterations, i.e., stopping earlier.

Table 2 Categorization of adversarial example generation techniques

	Black-box	White-box
Targeted	One-pixel attack (Su et al. 2017), ZOO (Chen et al. 2017)	L-BFGS (Szegedy et al. 2013), Hot/Cold* (Rozsa et al. 2016)
Non-targeted	Natural-GAN (Zhao et al. 2017)	FGSM* (Goodfellow et al. 2014), FGVM* (Rozsa et al. 2016), FGSM-Momentum* (Dong et al. 2017)

*Indicates that that method is one-shot. Otherwise, it is iterative

5.8 Adversarial training

Deep models have reached human precision in many applications. Nevertheless, these models have high sensitivity regarding small changes to the input space. In Fig. 7 we see an example that adding just a little noise to the input image confuses the deep model (Goodfellow et al. 2014). This problem is the result of the linear nature of the model in the vicinity of each input data. In other words, changing each input element by ϵ can at most create a change of $\epsilon|w|$ at the output of a linear unit. In an adversarial training, the nature of the model is changed to locally constant instead of being locally linear. In this way, small changes in the input cannot change the output that indicates the network is robust against this abnormal behavior.

In general, the generation of an adversarial example can be described as an optimization problem,

$$\begin{aligned} &\min_{\tilde{x}_i} \left\| x_i - \tilde{x}_i \right\| \\ &s.t. f\left(\tilde{x}_i\right) = \tilde{y}_i \\ &f\left(x_i\right) \neq \tilde{y}_i \end{aligned} \tag{23}$$

where $\tilde{y}_i$ is the assigned class for $\tilde{x}_i$ by classifier. Optimization problem minimizes the distance between x_i and $\tilde{x}_i$ while misclassifying the adversarial example. Inspired from Yuan et al. (2017), we divided the methods in the field according to (1) whether they can specify towards which class the model will be fooled, (2) whether the adversarial generation method knows the model and its parameters or not, and (3) whether the generation of the adversarial perturbations are done in an iterative manner or directly.

1. Targeted versus Non-Targeted: In some of generation methods, the output of the classifier for the adversarial example ($\tilde{y}_i$) can be specified (Targeted generation methods), while in others $\tilde{y}_i$ can be arbitrarily chosen in the optimization (Non-targeted generation methods).
2. White-box versus Black-box: Adversarial example generation methods can be classified into white-box and black-box generation methods. White-box generation methods require the classifier model to be known.
3. Black-box generation methods assume that given an input only the output of the classifier is available. An adversarial example generated by a white-box generation method on a classifier can fool a different classifier model with unknown properties (Szegedy et al. 2013).
4. One-shot versus Iterative: Adversarial example generation methods can be classified by how they solve the optimization problem. In one-shot methods, the optimization prob-

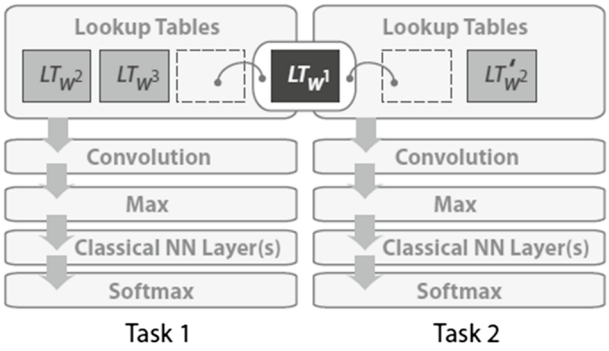

Fig. 8 a model to share representations (Collobert et al. 2011)

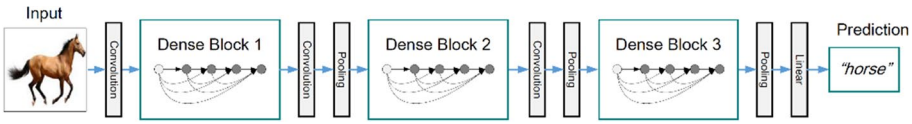

Fig. 9 A deep DenseNet with three dense blocks (Huang et al. 2016a)

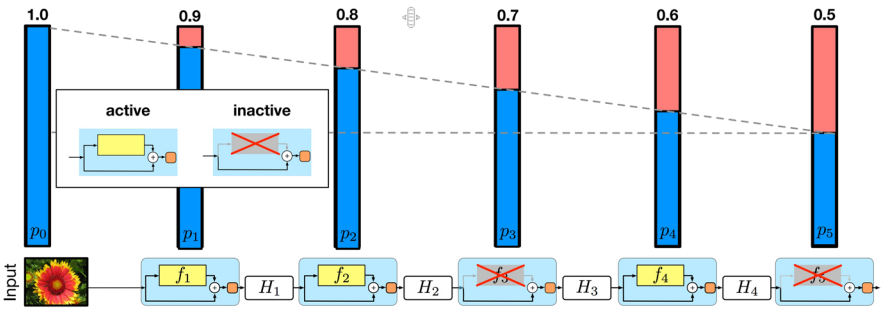

Fig. 10 A Deep Network with Stochastic Depth (Huang et al. 2016b)

lem is solved using a one-shot optimization method. In iterative methods, an iterative optimization method such as gradient descent is used to solve the optimization problem. While one-shot methods can generate examples fast, iterative methods are more robust to defense methods (Table 2).

In Roth et al. (2018) a structured gradient regularizer is proposed that increases the robustness of neural networks using adversarial perturbations. The method leverages the fundamental link between training with noise and regularization, and adds very little computational overhead during learning. In Cohen et al. (2018) the effectiveness of adversarial learning is studied as a cross-domain regularizer in the context of the ranking task. Here an adversarial discriminator is used and trains a neural ranking model on a small set of domains. The discriminator provides a negative feedback signal to discourage the model from learning domain-specific representations. The experiments show up to 30 precision better performance on held out domains in the presence of the adversarial discriminator. In Jakubovitz and Giryes (2018)) regularization is applied using the Frobenius norm of

the Jacobian of the network, which is applied as post-processing, after regular training has finished. Experimental results demonstrate that this method leads to enhanced robustness against adversarial attacks with a minimal change in the original network's accuracy. In Sankaranarayanan et al. (2018) deep neural networks are regularized by efficiently perturbing intermediate layer activations. These perturbations are used to train very deep models such as ResNets, and WideResNets and show improvement in performance across datasets of different sizes such as CIFAR-10, CIFAR-100, and ImageNet. The experiments show that the proposed approach not only provides stronger regularization compared to Dropout but also improves adversarial robustness comparable to traditional adversarial training approaches.

5.9 Multi-task learning

Considering different related works, it is reasonable that informative representations for one task are also informative for other tasks. For example, in NLP applications, useful representations learned for POS[4] task is also useful for NER[5] and SRL[6] tasks (Collobert et al. 2011). Considering the NLP application of Fig. 8 that two tasks are going to be learned in parallel, word representations learned in first layers can be shared between two tasks. Each task affects the initial layers with its own specific perspective (Figs. 9, 10). As the experiments in Collobert et al. (2011) showed, this technique can improve the generalizability of the learned look-up tables.

5.10 Layer-wise pre-training and initialization

In the back-propagation algorithm, the gradient signal moves from the end of the network to the front and is multiplied by small weights along the path. In deep networks, this path is so long that the gradient signal becomes very small near the front of the network. On the other hand, updates in later layers are done based on weights of former layers that are going to be changed. These effects slow the convergence process. An excellent way to manage this hassle is to pre-train the network, layer by layer, in an unsupervised fashion. In this technique, the identity function is thought to each layer from the bottom to the top. In this way, weights find a favorable point in parameter space that will cause regularizing effects in the process of fine-tuning (Erhan et al. 2009).

The drawback of layer-wise pre-training is that it is time-consuming. Initializing weights of the network is an active area of research. A famous recent study (He et al. 2015) recommend that the weights be initialized according to a Gaussian probability distribution with zero mean and $\frac{2}{m+n}$ variance, where m is the number of input neurons and n is the number of output neurons of the layer.

[4] Part-of-speech tagging.
[5] Name entity recognition.
[6] semantic-role labeling.

5.11 Architectural regularization

Devising architectural constraints is one of the most effective ways to control model complexity. For example, domain knowledge of natural images has employed effectively in the design of the architecture of convolutional neural networks. An object in a natural image can exist at any position, so in CNNs every filter is shared and applied to all positions of an image. Parameter sharing causes the number of parameters of the model to decrease dramatically and allows the designer to increase the depth and width of the network without needing a massive training dataset. In this way, in the process of learning the shared parameters, the data of different regions of the input image is used, so overfitting is avoided. Sharing parameters in addition to a sharp decrease in memory and computational costs causes a regularization effect that is appropriate for natural images. In pooling operation, while keeping important information, the number of features decreases severely, and a degree of invariance concerning translations and elastic changed is provided. However, if we simply alternate convolutional layers with max-pooling layers, performance is limited due to the rapid reduction in spatial size, and the disjoint nature of the pooling regions. In Graham (2015) a fractional version of max-pooling is developed in which feature map scaling factor is allowed to take non-integer values. The overall decrease in the size of the feature maps in addition to other advantages, causes a decrease in the number of parameters of subsequent layers, that in turn, have along regularization.

Generally, in deep networks, the ability to propagate gradients back through all the layers in an effective manner is of critical concern. The strong performance of relatively shallower networks suggests that the features produced by the layers in the middle of the network should be very discriminative. In Szegedy et al. (2014) by adding auxiliary classifiers connected to these intermediate layers, the authors encouraged discrimination in the lower stages in the classifier and increased the gradient signal that gets propagated back, that provided some regularization. Also in this research 1×1 convolution are employed intensively through the deep network, that its primary goal is to produce bottleneck of representation. This technique decreases the computational cost and the number of parameters of next convolutional operation, that in turn produces regularization effect in the deep network. In Szegedy et al. (2015) by the same authors, $n \times n$ convolutions are replaced by a sequence of $1 \times n$ and $n \times 1$ convolutions, that decreases the computational cost and number of parameters of $n \times n$ convolution that again has regularization effects.

Recent works have shown that convolutional networks can be substantially deeper, more accurate, and more efficient to train if they contain shorter connections between layers close to the input and those close to the output. Along with this fact, in Huang et al. (2016) DenseNet[7] architecture is introduced in which each layer is connected to every other layer in a feed-forward fashion. This architecture has several advantages: it alleviates the vanishing-gradient problem, strengthens feature propagation, encourages feature reuse, and provides parameter efficiency. Parameter efficiency causes a substantial reduction in the number of parameters that in turn have implicit regularization effect on the learning process. Additionally, through experiences, it is observed that dense connections have a regularizing effect, which reduces overfitting on tasks with smaller dataset size. In DenseNet, each layer has direct access to the gradients from the loss function and the original input signal, leading to implicit deep supervision that also has a regularization effect.

[7] Dense Convolutional Network.

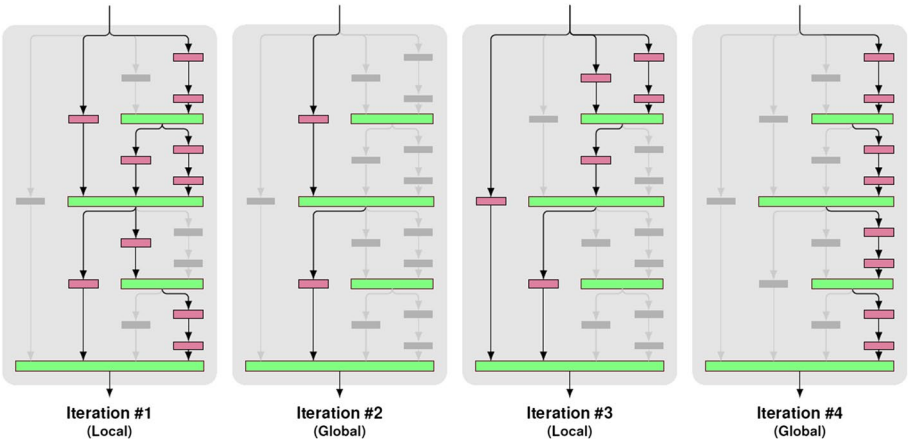

Fig. 11 Drop-path: Each mini-batch flows through just some paths from input to output (Larsson et al. 2017)

Although expressiveness of lots of layers can be highly desirable, diminishing effect in the forward flow of signal and vanishing gradient in backpropagation arises some severe problems. In Huang et al. (2016) Stochastic depth is introduced to solve this problem. It starts with a very deep network, and during training for each mini-batch, a random subset of layers are dropped and bypassed with identity mapping. In this way, short networks are trained, but deep networks are used at the test time. In this way, similar to Dropout (Srivastava et al. 2014), training with stochastic depth acts as a regularizer. In this research, it is observed that by training very deep stochastic depth networks with CIFAR-10, a ResNet with depth beyond 1000 layers obtained that had significant improvements in test error. This experience shows that by employing stochastic depth in very deep networks, overfitting can be avoided.

Drop-path regularization is introduced along with Fractal-Nets in Larsson et al. (2017). These networks contain interacting sub-paths of different lengths, but do not include any pass-through or residual connections. Dropout (Hinton et al. 2012c) and drop-connect (Wan et al. 2013) modify interactions between neighboring layers in order to avoid co-adaptation. Since a fractal network has additional macro-scale structure, this research presented a complementary coarse-scale regularization. Figure 11 illustrates drop-path for four mini-batch iterations. Just as dropout prevents co-adaptation of neurons of a layer, drop-path prevents co-adaptation of parallel paths by randomly dropping operands of join layers. This arrangement prevents the network from using one input path as an anchor and another as a corrective term. In this work, two network sampling strategies are followed. In local sampling, a join drops each input with a fixed probability, but at least one survives. On the other hand, in global sampling, a single path is selected for the entire network. We restrict this path to be a single column, encouraging individual columns to be independent, strong predictors.

As with dropout, signals may need appropriate rescaling. With element-wise means, this is trivial; each join computes the mean of only its active inputs. In the experiments, the network is trained with dropout and a mixture model of 50% local and 50% global sampling for drop-path. Also, a new sub-network is sampled for each mini-batch. With sufficient memory, one local sample and all its global samples can be evaluated

simultaneously for each mini-batch by keeping separate networks in memory and tying them together via weight sharing. While fractal connectivity permits the use of paths of any length, global drop-path forces the use of many paths whose lengths differ by powers of 2. In this way, sub-networks sampled by drop-path technique present vast structural diversity. This property is in contrast to stochastic depth regularization (Huang et al. 2016) of deep ResNets (He et al. 2016), which, by using a fixed drop probability for each layer in a chain, sampled sub-networks exhibit a concentrated depth distribution. In this case, global drop-path not only plays the role of a regularizer, but also it can be used as a trade-off tool. In other words, individually strong columns of various depths also give users trade-off ability between a fast, shallow network and a deep, accurate one.

Channel-Out (Wang and JaJa 2013) is based on encoding information on sparse pathways and recognizing the correct pathway at inference time. In this network, the pathways are not only formed a posteriori, but they are also actively selected according to the inference outputs from the lower layers. From a mathematical perspective, channel-out networks can represent a broader class of piece-wise continuous functions, thereby endowing the network with more expressive power than that of Maxout networks. Channel-out networks are tested on several well-known image classification benchmarks, setting new state-of-the-art performance on CIFAR-100 and STL-10.

Shake–Shake Regularization (Gastaldi 2017) is an attempt to produce softer augmentation than random flips or crops. The idea of shake–shake is to change standard summation of residual branches by a stochastic affine combination in 3-branch ResNet. Comparison of Shakeout and Shake–Shake shows that both methods use the idea of replacing Bernoulli variables by scaling coefficients. While the starting point for shake-out is Dropout and that of Shake–Shake is a mix of FractalNet, Drop-path and Stochastic Depth. Shakeout keeps coefficients the same between forward and backward passes whereas Shake–Shake updates them before each pass. Shake–Shake works by adding up two residual flows and a skip connection whereas shakeout requires only one flow. Applied to 3-branch residual networks, Shake–Shake regularization improved on the best single shot published results on CIFAR-10 and CIFAR-100.

In this way, employing domain knowledge, we can design architectures that are appropriate for our specific application so that it would be able to converge fast using minimum training data without the model experience overfitting.

5.12 Label smoothing

Deep supervised models usually use a cross-entropy cost function to find the optimum values of parameters of the model. In this case, for i-th training example $x^{(i)}$, the model computes the probability of each label $k \in \{1, \ldots, K\}$

$$p\left(k|x^{(i)}\right) = \frac{\exp\left(z_k\right)}{\sum_{j=1}^{K} \exp\left(z_j\right)} \tag{24}$$

Here, z_j s are the logits or un-normalized log probabilities, that are provided by the last layer of the classifier. Also, suppose the ground-truth distribution over labels $q\left(k|x^{(i)}\right)$ for

each training example, normalized so that $\sum_k q\left(k|x^{(i)}\right) = 1$. The cross-entropy cost function is defined as follows:

$$\mathrm{L}_i = \mathrm{H}(q, p|x^{(i)}) = -\sum_{k=1}^{K} \log\left(p(k)|x^{(i)}\right) q\left(k|x^{(i)}\right), \tag{25}$$

Minimizing L_i is equivalent to maximizing the expected log-likelihood of a label, where the label is selected according to its ground-truth distribution $q\left(k|x^{(i)}\right)$. Fortunately, a cross-entropy loss is differentiable with respect to the logits z_k and thus backpropagation and SGD can be used to find the optimal parameters. It is easy to show that the gradient of L_i has a nice simple form as follows, which is bounded between -1 and 1.

$$\frac{\partial \mathrm{L}_i}{\partial z_k} = p\left(k|x^{(i)}\right) - q\left(k|x^{(i)}\right) \tag{26}$$

Consider the case of a single ground-truth label y, so that $q\left(y|x^{(i)}\right) = 1$ and $q\left(k|x^{(i)}\right) = 0$ for all $k \neq y$. In this case, minimizing the cross entropy is equivalent to maximizing the log-likelihood of the correct label. For a particular $x^{(i)}$ with label y the log-likelihood is maximized for Dirac delta:

$$p\left(k|x^{(i)}\right) = q\left(k|x^{(i)}\right) = \delta_{k,y} \tag{27}$$

This maximum is not achievable for finite z_y but is approached asymptotically if $z_y \gg z_k$ for all $k \neq y$. This, can cause two problems. First, it may cause overfitting. That is because, if the model learns to assign a full probability to the ground-truth label for each training example, it is not guaranteed to generalize well. Second, it encourages the differences between the largest logit and the others to become large, and this, combined with the bounded gradient, i.e., Equation (21), reduces the ability of the model to adapt easily. Intuitively, this happens because the model becomes too confident about its predictions. In Szegedy et al. (2015) a mechanism is proposed for encouraging the model to be less confident. While this may not be desired if the goal is to maximize the log-likelihood of training labels, it does regularize the model and makes it more adaptable. To do this, consider a distribution over labels $u(k)$, independent of any training example $x^{(i)}$, and a smoothing parameter ϵ. For a training example with ground-truth label y, the label distribution $q\left(k|x^{(i)}\right) = \delta_{k,y}$ is replaced with

$$q'\left(k|x^{(i)}\right) = (1-\epsilon)\delta_{k,y} + \epsilon u(k) \tag{28}$$

which is a convex combination of the original ground-truth distribution $q\left(k|x^{(i)}\right)$ and the fixed distribution $u(k)$ with weights $1-\varepsilon$ and ε, respectively. This can be seen as the distribution of the label k obtained as a result of doing this experiment: first, set it to the ground-truth label $k=y$; then, with probability ε, replace k with a sample drawn from the distribution $u(k)$. As a simple chose, the uniform distribution $u(k) = \frac{1}{K}$ can be used, so that

$$q'\left(k|x^{(i)}\right) = (1-\epsilon)\delta_{k,y} + \frac{\epsilon}{K} \tag{29}$$

In Szegedy et al. (2015), this change in ground-truth label distribution is referred to as label-smoothing regularization or LSR. It is noticeable that LSR achieves the desired goal of preventing the largest logit from becoming much larger than the others. Indeed, if this were to happen, then a single $q(k|x^{(i)})$ would approach 1 while all others would approach

Table 3 Batch normalizing transform, applied to activation x over a mini-batch (Ioffe and Szegedy 2015)

1 :	Input: values of $x\psi$over a mini-batch : $\mathcal{B} = \{x^{(1)} \ldots x^{(m)}\}$	
2 :	Output : $\{y^{(1)} \ldots y^{(m)}\} = \mathbf{BN}_{\gamma,\beta}\{x^{(1)} \ldots x^{(m)}\}$	
3:	Learnable parameter : γ, β	
4 :	$\mu_{\mathcal{B}} = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}$	% mini-batch mean %
5 :	$\sigma_{\mathcal{B}}^2 = \frac{1}{m}\sum_{i=1}^{m}(x^{(i)} - \mu_{\mathcal{B}})^2$	% mini-batch variance %
6 :	for $i = 1 \ldots m$	
7 :	$\hat{x}^{(i)} \leftarrow \frac{x^{(i)} - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$	% normalize %
8 :	$y^{(i)} \leftarrow \gamma\hat{x}^{(i)} + \beta \equiv \mathbf{BN}_{\gamma,\beta}(x^{(i)})$	% scale and shift %

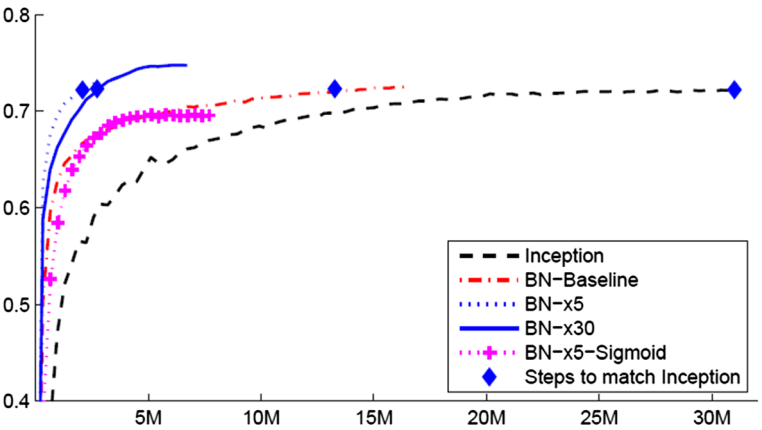

Fig. 12 Validation accuracy of Inception and its BN variants, versus iterations (Ioffe and Szegedy 2015)

0. This would result in a large cross-entropy with $q'\left(k|x^{(i)}\right)$ because, unlike $q\left(k|x^{(i)}\right) = \delta_{k,y}$, all $q'\left(k|x^{(i)}\right)$ have a positive lower bound. Another interpretation of LSR can be obtained by considering the new cross entropy of q' and p

$$\mathrm{L}_i' = H\left(q', p\right) = -\sum_{k=1}^{K} \log\left(p\left(k|x^{(i)}\right)\right) q'\left(k|x^{(i)}\right) = (1-\epsilon)H(q,p) + \epsilon H(u,p), \quad (30)$$

It is clear that LSR is equivalent to replacing a single cross-entropy loss $H(q,p)$ with a pair of such losses, i.e., $H(q, p)$ and $H(u, p)$. The latter loss penalizes the deviation of predicted label distribution p from the assumed prior u, with the relative weight $\frac{\epsilon}{1-\epsilon}$. Note that this deviation could be equivalently captured by the KL divergence since $H(u,p) = D_{KL}(u \parallel p) + H(u)$ which $H(u)$ is fixed. When u is the uniform distribution, $H(u, p)$ is a measure of how dissimilar the predicted distribution p is from uniform. In

another word, label smoothing is a mechanism to regularize the classifier layer by adding some label-dropout effect to the process of model training.

5.13 Batch normalization

Nowadays, offline normalization of input data is an essential preprocessing stage in learning systems. Normalization causes the curvatures of the cost function in different directions to have a gentle and similar form. In this way, the resulting trajectory to the solution has less zigzag movements, and convergence happens faster. This balancing of curvature can be interpreted as regularization of the problem. In recent years, offline normalization of input data has extended, and online normalization layers are added throughout the network based on mini-batches.

While training, the distribution of each layer's inputs changes, as the weights of the previous layers, change. This inconstancy slows down the training by requiring lower learning rates and careful parameter initialization. Also, employing saturating nonlinearities makes the process much harder. In Ioffe and Szegedy (2015)), this phenomenon is referred to as internal covariate shift, and the problem is addressed by batch normalization. In Ioffe and Szegedy (2015) normalization is designed as part of the model architecture and is performed for each training mini-batch. The goal of batch normalization is to achieve a stable distribution of activation values throughout training, and it is applied before the nonlinearity since this is where matching the first and the second moments are more likely to result in a stable distribution. Here, batch normalization is employed so the resulting networks can be trained with saturating nonlinearities without worrying much about more regularization arrangements. In this way, the network is more tolerant of increased training rates that yield a substantial speedup in training.

The batch normalization Algorithm is presented in Table 3. In the algorithm, x denotes a typical layer input and y is its batch normalized form. At first, mini-batch mean and variance are computed at lines 4 and 5, respectively. Then at line 7, all members of the batch are normalized using mini-batch mean and variance. The parameter ε is a small value and is added for numerical stability. Note that simply normalizing each input of a layer may change what the layer can represent. To address this, a transformation is added at line 8 that is able to represent the identity mapping. For each layer input, a scaling γ and a shifting β parameter is defined that is learned along with other learnable parameters of the network ϵ.

In Fig. 12, the validation accuracy of the networks, as a function of the number of training steps is shown. It is noticeable that Inception reaches the accuracy of 72.2% after 31×10^6 training steps whereas BN-x5 need just 2.1×10^6 iterations to get the same accuracy.

In Ioffe and Szegedy (2015) two observations testify the regularization effect of Batch Normalization. First, removing Dropout from modified BN-Inception caused speedup in training, without increasing overfitting. So we can say Batch Normalization has fulfilled the regularization role of Dropout using a different approach. Second, while in Inception the L_2 loss imposed on the model parameters controlled overfitting, in modified BN-Inception the weight of this loss is reduced by a factor of 5, and an improvement in the accuracy of the held-out validation data is observed. In this way, we can say Batch Normalization has taken the regularization effect of L_2 penalty partially.

A recent study shows that in addition to internal covariate shift (ICS), BN adds smoothness to the internal optimization problem of the network (Santurkar et al. 2018). In other words, BN provides more stability and reliability to the gradients and increases the training speed

significantly. The study shows that the batch normalization re-parametrizes the optimization problem to make it more stable and smooth. Thus BN makes the gradients more reliable and predictive, makes the training significantly faster and less sensitive to hyper-parameter choices.

In the following, different variants of batch normalization technique are considered and discussed.

5.13.1 Layer normalization

Layer normalization (Ba et al. 2016) normalizes the inputs across the features, while batch normalization normalizes the input features across the batch dimension. In batch normalization, the statistics are computed across the batch and are the same for each example in the batch. In contrast, in layer normalization, the statistics are computed across each feature and are independent of other examples. This means that layer normalization is not a simple re-parameterization of the network, unlike the case of weight normalization and batch normalization, which both have the same expressive power as an un-normalized neural network. The independence between inputs means that each input has a different normalization operation, allowing arbitrary mini-batch sizes to be used. Because of the dependency of batch normalization on mini-batch size, it is not clear how to apply it on recurrent neural networks. The experimental results show that layer normalization performs well for recurrent neural networks and the method is very effective at hidden state dynamics. Also, unlike batch normalization, the layer normalization substantially reduces the training time by computing the mean and variance used for normalization from all of the summed inputs to the neurons in a layer on a single training case.

5.13.2 Weight normalization

Weight normalization (Salimans and Kingma 2016) instead of normalizing the mini-batch, normalizes the weights of the layer. Weight normalization re-parameterizes the weights w of any layer in the neural network in the following way:

$$w = \frac{g}{||v||} v \tag{31}$$

Similar to batch normalization, weight normalization does not reduce the expressive power of the network. What it does is that it separates the norm of the weight vector from its direction. It then optimizes both g and v using gradient descent. This change in learning dynamics makes optimization easier. Weight normalization is an inspiration from batch normalization but does not introduces its dependencies on mini-batch. This method has an advantage over batch normalization because of its less computation overhead which permits more optimization steps taken in the same amount of time and speed up convergence of stochastic gradient descent. Unlike batch normalization, weight normalization is well suited to recurrent models such as LSTM[8] (Hochreiter and Schmidhuber 1997) and reinforcement learning or generative models. A comparison study between batch normalization and weight normalization for large scale image classification problems such as ImageNet shows that batch normalization has much stronger and stable regularization effect than

[8] Long Short-Term Memory.

weight normalization. Weight normalization is thus limited to shallow networks and cannot replace batch normalization for deep neural networks (Gitman and Ginsburg 2017).

The authors in Salimans and Kingma (2016) claim that this method provides the following benefits:

1. It makes the mean of the activations independent from v—Weight normalization independently cannot isolate the mean of the activations from the weights of the layer, causing high-level dependencies between the means of each layer. Mean-only batch normalization can resolve this problem.
2. It adds "gentler noise" to the activations—One of the side-effects of batch normalization is that it adds some stochastic noise to the activations as a result of using noisy estimates computed on the mini-batches. This has a regularization effect in some applications but can be potentially harmful in some noise-sensitive domains like reinforcement learning. The noise caused by the mean estimations, however, are "gentler" since the law of large numbers ensures the mean of the activations is approximately normally distributed.

The experimental results of the paper show that weight normalization combined with mean-only batch normalization achieves the best results on CIFAR-10.

5.13.3 Orthogonal weight normalization

Orthogonal Weight Normalization (Huang et al. 2017) is the generalization of the square orthogonal matrix to orthogonal square matrix. This technique of normalization is well applicable to recurrent neural networks (RNNs) and feedforward neural networks (FFNs). Orthogonal weight normalization solves the problem of optimization over multiple dependent Stiefel manifolds (OMDSM). The rectangular orthogonal matrix can stabilize the distribution of network activations and regularize FNNs. We also propose a novel orthogonal weight normalization method to solve OMDSM. Notably, it constructs orthogonal transformation over proxy parameters to ensure the weight matrix is orthogonal and back-propagates gradient information through the transformation during training. To guarantee stability, we minimize the distortions between proxy parameters and canonical weights over all tractable orthogonal transformations. Also, an orthogonal linear module (OLM) is designed for learning orthogonal filter banks in practice, which can be used as an alternative to the standard linear module. Extensive experiments demonstrate that by simply substituting OLM for standard linear module without revising any experimental protocols, our method mainly improves the performance of the state-of-the-art networks, including Inception and residual networks on CIFAR-10 and ImageNet datasets.

5.13.4 Group normalization

Group Normalization (GN) (Wu and He 2018) is a simple and effective alternative to batch normalization. In the case of batch normalization (BN), normalizing along the batch direction introduces problems, and the BN's error increases rapidly when the batch size becomes smaller, caused by inaccurate batch statistics estimation. This problem limits the application of BN to the computer vision tasks including video, detection, and segmentation which require small batches constrained by memory consumption. Group normalization fixes this problem by dividing the channels into groups and computes the mean and variance within each group for normalization. Unlike BN, GN's accuracy is stable over

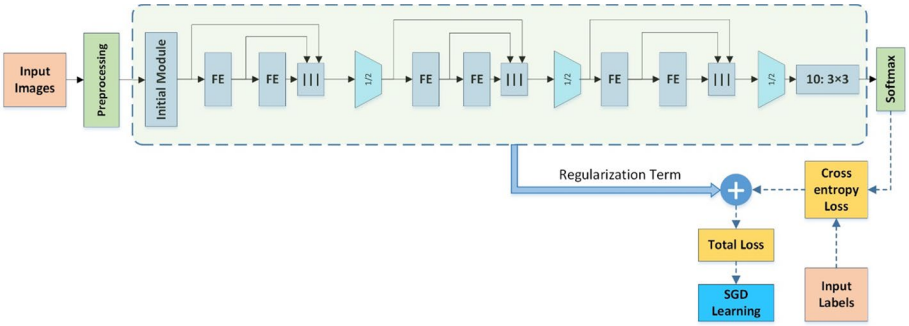

Fig. 13 The architecture of the benchmark model

a wide range of batch sizes, and its computation is independent of batch size. GN can be naturally transferred from pre-training to fine-tuning. GN has attained 10 percent less error than BN when used for ResNet-50 trained on ImageNet when using a batch size of 2. When using typical batch sizes, GN is equivalent to BN and outperforms other normalization variants. GN can effectively replace BN in a variety of computer vision tasks like object detection and segmentation in COCO and, video classification in Kinetics.

5.13.5 Fractional pooling

In CNN models each convolutional layer produces a three-dimensional representation. The size of this representation is usually reduced using some form of pooling. Max-pooling is a procedure that takes an $N_{in} \times N_{in}$ input matrix and returns a smaller output matrix, say $N_{out} \times N_{out}$. This is achieved by dividing the input $N_{in} \times N_{in}$ square into N_{out}^2 pooling regions that each is denoted by $P_{i,j}$ that is defined as follows:

$$P_{i,j} \subset \left\{1, 2 \ldots ., N_{in}\right\}^2 foreach(i,j) \in \left\{1, 2 \ldots ., N_{out}\right\}^2 \tag{32}$$

So each output at (i, j) th position is obtained as follows

$$\text{Output}_{i,j} = \max_{(k,l) \in P_{i,j}} \text{Input}_{k,l} \tag{33}$$

For a regular 2×2 Max-pooling operation, we have $\frac{N_{in}}{N_{out}} = 2$ and $P_{i,j} = \{2i-1, 2i\} \times \{2j-1, 2j\}$, so in this case the size of input is decayed by the factor of $1/2$. Here the decaying factor is too high and informative representations may get lost. To remedy the problem, in Graham (2015) the author is interested in decaying factors in the interval (1,2). Given a particular pair of values $\left(N_{in}, N_{out}\right)$ the problem is to find pooling regions $P_{i,j}$. Here, there are two type of arrangements, overlapping and disjoint pooling squares. To give a formal description of how to generate pooling regions, let $\left(a_i\right)_{i=0}^{N_{out}}$ and $\left(b_i\right)_{i=0}^{N_{out}}$ be two increasing sequences of integers starting at 1, ending with N_{in} and with increments of one or two, in other words, $a_{i+1} - a_i \in \{1, 2\}$. Disjoint and overlapping pooling regions can then be defined as follows:

$$P_{i,j} = \left[a_{i-1}, a_i - 1\right] \times \left[b_{j-1}, b_j - 1\right] \text{or} P_{i,j} = \left[a_{i-1}, a_i\right] \times \left[b_{j-1}, b_j\right] \tag{34}$$

Table 4 The list of kernels and feature map specifications

Module	Stride	Output size	#Parameters	#Operations	#Memory
Initial	1	28×28×64	3712	100,352	6740
FE	1	28×28×64	55,912	43,835,008	195,026
FE	1	28×28×64	55,912	43,835,008	195,026
Pooling	2	14×14×64	0	37,632	25,088
FE	1	14×14×64	55,912	10,958,752	131,378
FE	1	14×14×64	55,912	10,958,752	131,378
Pooling	2	7×7×64	0	9408	6272
FE	1	7×7×64	55,912	2,739,688	117,274
FE	1	7×7×64	55,912	2,739,688	117,274
Pooling	2	3×3×64	0	1728	1152
Final	–	10	5770	5780	6356
		Sum:	344,954	115,221,796	932,964

These integer sequences are generated using random and pseudo-random sequences. The sequence is random if the increments are obtained by taking a random permutation of an appropriate number of one's and two's and is pseudorandom if it takes the form:

$$a_i = \text{ceiling}(\alpha(i+u)), \alpha \in (1,2), \text{with some } u \in (0,1) \tag{35}$$

Some examples of the case N_{in}=25, N_{out}=18 are presented in the following. The increments on the left were generated "randomly", and the increments on the right come from "pseudo-random" sequences:

211112112211112122	112112121121211212
111222121121112121	212112121121121211
121122112111211212	211211212112121121

Although both types of sequences are irregular, the pseudo-random sequences define much more stable and regular pooling regions than the random sequences. Experiences in Graham (2015) show if an image is scaled down using disjoint random pooling regions, the resulted image shows elastic distortion. In contrast, using disjoint pseudo-random pooling regions the result is merely a scaled down of the original image. The effect of random pooling can be interpreted as the effect of Dropout.

In Stochastic Pooling (Zeiler and Fergus 2013) conventional pooling is replaced by stochastic procedure, and the activation within each pooling is picked randomly according to multinomial distribution. This technique can be applied with other regularization techniques such as dropout, weight decay, and data augmentation with negligible computation overhead. Stochastic pooling is tested on a variety of benchmark image datasets (MNIST, CIFAR10, CIFAR100, SVHN) and it is proved that this method is not only effective but also does not require any hyper-parameter tuning.

Table 5 Model accuracy, number of epochs the model needs to be learned and the number of operation per input sample during the training phase

Reg. Method	None	Weight decay	Noisy inputs	Noisy weights	Dropout	DropConnect	Ensemble	Data augmentation	Adversarial	Label smoothing	Batch normalization	Fractional pooling
Accuracy	82.1	88.3	84.8	84.2	90.6	90.4	91.5	88.1	88.4	89.2	90.8	90.5
Epochs	12	12	15	16	28	26	60	17	25	10	6	16
Training Ops/Sam (M)	230	232	230	232	118	118	230	232	230	230	420	430

6 Experimental results

In this section, a deep network is presented, and different regularization strategies are compared in terms of classification accuracy and computational cost required by the model to be learned. Because of space and time limitations, we have compared well-known methods from different regularization categories instead of comparing all regularization methods that is impossible. The model used for this experiment is presented in Fig. 13. The design is based on nine design principles presented in Moradi et al. (2019). The proposed model consists of a pre-processing module, an Initial module, a set of Feature Extractor modules (FEs) and a set of forward feature map concatenations. The input layer is fed from CIFAR-10 image dataset, which is a 10 class image dataset. At the beginning of the network, the pre-processing module does some optional random invariant changes such as hue, contrast, brightness, and saturation distortions and some random transformations like translation, rotation, and flipping. After preprocessing, the images are passed through a sequence of modules that their internal design details are presented in Moradi et al. (2019). The first module is the Initial module that its duty is to create reliable and informative feature maps from the input images. Next, the data is passed through a sequence of FE modules and feature map concatenations. The pooling layers divide the height and width of the input feature maps by 2 that quarters the spatial area of the feature maps. Finally, the fully connected layer that is implemented by a 2×2 convolution, without any padding and stride, delivers the scores. At the end of the network, a Softmax layer converts the scores to probabilities of classes. In Table 4, kernels and feature map specifications are presented. In this experience, different regularization strategies are applied in the model by the regularization module that is shown symbolically in Fig. 13.

The training process consists of running the backpropagation algorithm and Adam updating mechanism based on mini-batch training. The training phase uses 64 images per batch, so each epoch consists of 860 batches to cover all 55,000 samples of the training set. Also, 5,000 images of 60,000 images of the training set are separated at the beginning of the training process that is used as a validation set during the training process of the model. The learning strategy used is early stopping, in which for various regularization strategies the training process continues until the validation set error starts to decrease. The code for the experiment is implemented using Python language and Tensorflow package. The machine specification for the experience is an eight core Core i7 with 2.6 GHz CPU and 32 GB of RAM. The GPU is Nvidia Geforce GTX 980 M with 8 GB of RAM and 2048 CUDA cores.

In Table 5, the model accuracy, the number of epochs the model needs to be converged and the number of operation per input sample during the training phase for each regularization method is presented. As can be seen, in Table 5 the base network without any regularization takes 12 epochs to be trained and gives the accuracy of 82.1. In the following, different regularization methods are added to the base network one by one, and after re-training the network, the results are discussed in separate subsections.

6.1 Weight decay

In this experience, an L_2 norm is added to the cost function, that its associated coefficient is obtained by cross-validation. The resulted accuracy shows that weight decay regularization improves overfitting problem while the number of epochs and the number of operations per

sample is approximately the same as the base network. The reason is that the weight decay penalty compared with other costs is negligible and merely helps the optimal point move to a safer regions. So, based on this observation, the method is considered as an effective regularization method.

6.2 Noisy inputs

Here, a Gaussian noise $\mathcal{N}(0, 0.1)$ is added to each pixel of input images where variance 0.1 is obtained by cross validation. The resulted accuracy shows that noise addition to the input causes little improvement on overfitting problem while the number of epochs has increased and the number of operations per sample is approximately the same as base network. The reason is that, adding Gaussian noise to the input images has negligible cost and tend to make the learning process a little harder. Thus, based on this observation, the method is not very effective.

6.3 Noisy weights

In this experience, a Gaussian noise $\mathcal{N}(0, 0.05)$ is added to each weight of the network where variance 0.05 is obtained by cross validation. The resulted accuracy show that noise addition to the weights turn out to have little improvement on overfitting problem while the number of epochs have increased and the number of operations per sample is approximately the same as the base network. The reason is that, adding Gaussian noise to the weights of the model has negligible cost and tend to make the learning process a little harder. So, based on this observation, the method is not very effective.

6.4 Dropout

Here, Dropout regularization is added to the base network, and the drop rate p is obtained by cross validation. As the results confirms, this method has noticeable improvement on overfitting problem. The drawback of this method is its convergence speed that is among the slowest models. Nevertheless, the number of operations per sample is halved with respect to the base network. The reason is that, dropping approximately half of the neurons of the model in each iteration result in a smaller model that has fewer computations and bring us a model averaging effect during the learning process. In this way, the method is considered as an effective regularization method.

6.5 DropConnect

In this experience, DropConnect regularization is added to the base network, and the drop rate p is obtained by cross validation. As the resulted accuracy shows, this method has a noticeable improvement on overfitting problem. The drawback of this method is its convergence speed that is among the slowest models. Nevertheless, the number of operations per sample like Dropout is halved with respect to the base network. The reason is that, dropping approximately half of the connections of the model in each iteration result in a smaller model that has fewer computations and bring us a model averaging effect during

the learning process. So, we conclude that the method is among effective regularization methods.

6.6 Ensemble

Here, five base networks are trained separately and the final result is obtained by averaging the outputs of five networks. The number of models in the ensemble is obtained by cross-validation. As the resulted accuracy confirms, this method has a noticeable improvement on overfitting problem. Because of training multiple models, four times more computations are required for training and testing, compared to the base model. Nevertheless, the number of operations per sample is the same as the base network. So based on this observation, in the case of abundant computational resources, the method is very effective.

6.7 Data augmentation

In this experience, the pre-processing regularization is added to the base network by activating the pre-processing module in Fig. 13. The preprocessing module does some random invariant changes such as hue, contrast, brightness, and saturation distortions and some random transformations like translation, rotation, and flipping. As the resulted accuracy reveals, this method improves overfitting problem. While the number of operations per sample witnesses a slight increase, the number of epochs for the model to be converged is increased by 50 percent with respect to the base network. The reason is that, while applying preprocessing on input images has negligible cost, it tends to make the learning process capture more information. So, based on this observation, the method is considered as a middle-level regularization method.

6.8 Adversarial

Here, new adversarial samples are generated and added to the training set, using the training set and back-propagation of intentionally generated error in the output to the input. As the resulted accuracy confirms, this method improves the overfitting problem. While the number of epochs for the model to be converged is doubled, the number of operations per sample is the same as the base network. The reason behind this observation is that here we have to spend some epochs to generate adversarial samples, but the model is the same as the base network. So, based on this observation, the method is considered as a middle-level regularization method.

6.9 Label smoothing

In this experience, the label $\delta_{k,y}$ is replaced by $(1-\epsilon)\delta_{k,y} + \frac{\epsilon}{K}$. So, the largest logit is prevented from becoming much larger than the others. Here K is number of classes that is fixed to 10 and ε is smoothness factor that is obtained by cross validation. As the resulted accuracy shows, this method has small improvement on overfitting problem. While the number of epochs for the model to be converged is decreased a little, the number of

operations per sample is the same as the base network. The reason behind this observation is that by decreasing the distance between the largest logit and the rest, the optimization problem becomes just a little simpler. So, based on this observation, the method is not very effective.

6.10 Batch normalization

Here, batch normalization regularization is added to the base network and the hyper-parameters are taken as the defaults of the algorithm. Here scaling hyper-parameter γ and shifting hyper-parameter β are learnable parameters, so their optimal values are obtained during the process of learning. The only fixed hyper-parameter is ε that is a small number and is added for numerical stability. The value $\epsilon = 1e - 8$ is chosen for this hyper-parameter according to the batch normalization paper (Ioffe and Szegedy 2015). As the resulted accuracy shows, this method has a noticeable improvement on overfitting problem. While the number of epochs witnesses a considerable decrease, the number of operations per sample is approximately doubled with respect to the base network. Because of the overhead in training batch normalization learnable parameters in each epoch, much more computational resources are required in this method. So, based on this observation, in the case of abundant computational resources, the method is very effective.

6.11 Fractional pooling

In this experience, fixed max pooling modules are replaced by fractional pooling modules. That way, instead of quartering the area of feature maps they are halved. This change helped us to double the depth of the network and in turn more feature extraction opportunity. As the resulted accuracy confirms, this method has a noticeable improvement on overfitting problem. While the number of epochs for the model to be converged is increased by 35 percent, the number of operations per sample is approximately doubled with respect to the base network. The reason is that doubling the depth of the network, doubles the computations of the network and causes the learning process to become a little harder. So, based on this observation, in the case of abundant computational resources, the method is very effective.

In general, because of well behaved and little side effects of weight decay and data augmentation regularizations, they are popular among deep learning models. In the case of enough computational resources, Dropout family methods are reasonable. And, in the case of abundant computational resources, batch normalization family and ensemble methods are rational. All in all, selecting a regularization method heavily depends on the input data type, architecture of the network and available computational resources.

It is important to note that, different regularization strategies can be employed at same time so that they create a synergistic effect. In general, deep models have large learning capacity and designers use multiple regularization strategies at the same time to harness their overfitting tendency. The set of regularization strategies that is employed at the same time in practice depend on the architecture, the learning method and the application of the proposed deep model. For example, the designers of AlexNet (Krizhevsky et al. 2017) employed weight decay, dropout, early stopping, data augmentation, and ensemble regularization methods at the same time in their proposed model. In this way, based on the model and its objective and the special behavior and effect of each regularization method,

the designer select the appropriate set of regularization strategies that work best for the model at hand.

7 Discussion

Traditionally, the model's capacity is controlled by including or excluding members from the hypothesis space. However, expressing preferences for one function over another implicitly or explicitly is a more general way to do the job, and is known in general as regularization. Regularization is any modification we make to a learning algorithm that is intended to reduce its generalization error but not necessarily its training error. Regularization along with optimization are the central concerns of machine learning. No free lunch theorem has made it clear that there is no best machine learning algorithm, and, in particular, no best form of regularization. Instead, we must choose an appropriate regularization that is well suited to the particular task at hand. The philosophy of deep learning is that a wide range of tasks, such as all the mental tasks that human being can do, may all be solved effectively using general-purpose forms of regularization.

In this survey, regularization techniques that are heavily in use in recent deep learning literature are reviewed. Different kinds of naïve weight decay are still in use as a base method to control the capacity of deep models. Dropout is also one of the most popular methods that are being in use to form an implicit ensemble of models. Although Dropout causes the convergence to take much longer, it is an effective way to control large capacity of deep models. Data augmentation is being used in almost all the researches, because it makes our valuable labeled data an order of magnitude larger and helps to design larger models. The training process of deep models are done based on the error of validation data, and we do not wait for the gradient of the loss get zero, so early stopping is heavily in use. Making ensembles of models usually entail an increase of accuracy by a few percents and usually is done in most researches. Initialization is an important issue that is done based on the type of activity function and the standards that are established. In recent years the design of new deep architectures is developed to make the flow of data and gradient better. These archetectural innovations have helped to decrease the number of parameters that in turn effectively regularizes hypothesis space of deep models. Moreover, finally, Batch Normalization is being in use in most recent papers and is effective in regularizing the shape of cost functions and making the convergence more fluently.

In the presented comparative study, we compared different regularization methods in terms of the accuracy of the network, the number of epochs for the network to be trained and the number of operations per input sample. The experiment results showed that weight decay and data augmentation regularizations have little computational side effects so they can be used in most applications. In the case of enough computational resources, Dropout family methods are rational to be used. And, in the case of abundant computational resources, batch normalization family and ensemble methods are reasonable strategies to be employed in the network as the regularizer.

Generally, the mission of all regularization methods is to prevent the models from overfitting by trying to make the models simpler. In L_1 and L_2 regularization methods due to the addition of a penalty term to the cost function, the values of weights of the model are encouraged to be decreased. In neural networks smaller values for weights means simpler models. In L_1 regularization, unlike L_2 regularization, the weights may be reduced to zero. Hence, it is very useful when we are trying to compress our model. Otherwise, we usually

prefer L_2 over L_1 because it lets the network have many small learnable weights that is essential for a powerful model.

In Dropout, each iteration has a different set of nodes that can be thought of as an ensemble technique in machine learning. Ensemble models usually perform better than a single model as they average the randomness in the model and make the resulted model simpler. DropConnect method generalized this idea that replaced node dropping by weights dropping and obtained similar results.

In data augmentation method, more knowledge is injected to the model by increasing the size of the training data. In this way, the effectiveness of noises in the training data is reduced and consequently the possibility of the overfitting of the model will be reduced. Other strategies like noisy inputs, adversarial training, fractional pooling and batch normalization have generalized this idea and confronted the overfitting problem by expanding the data running in the network. In noisy weights and label smoothing methods the idea of data adjustment is extended to weights and label adjustments.

Early stopping is a kind of cross-validation strategy where we keep one part of the training set as the validation set. Here the performance on the validation set is used as a guide to prevent the model learn noises and uninformative information in the input training set.

It is worthy to note that different regularization strategies have complimentary effect with respect to each other and the weakness of one strategy can be covered by the other. So there is no single approach to be uniformly the best. Each regularization strategy attacks the problem of overfitting with its specific philosophy that are detailed in the corresponding section of the paper that is dedicated to the strategy. In practice, the designer adds the regularization techniques one by one based on the feedbacks from the experimental results.

In recent years, developing more effective regularization strategies has been one of the major research efforts in deep learning. So it is predictable that there are still many more forms of effective regularization methods to be developed in the near future.

8 Conclusion

In this paper, we reviewed favorite regularization techniques in recent years. We started weight decay as the base method to control the capacity of deep models. Next, we discussed dropout as an effective method to disentangle extracted features in deep models. After that, we reviewed data augmentation as one of best practices to increase valuable labeled training data. Next, we considered early stopping as an excellent way to avoid overfitting. After that, the ensembles of deep models as an effective way to increase generalizability is discussed. Next, the effect of initialization in deep models is considered as a way to avoid overfitting. Next, the effect of architectural ideas on generalization is studied. And finally, we discussed batch normalization as an effective way to make the optimization process to be well-behaved. The experiment results showed that weight decay and data augmentation regularizations are safe to be used in most applications. Dropout family methods are appropriate when we have enough computational resources. And, batch normalization family and ensemble methods are reasonable strategies when there are abundant computational resources. It seems the trend of employing more effective regularization techniques will continue and we will experience better and smarter methods in the near future.

References

Aha D, Kibler D, Albert M (1991) Instance-based learning algorithms. Mach Learn 6(1):37–66

Ba JL, Kiros JR, Hinton GE (2016) Layer normalization, arXiv:1607.06450

Bartle RG (1995) The elements of integration and Lebesgue measure. Wiley, New Yor

Bordes A, Chopra S, Weston J (2014) Question answering with subgraph embeddings. In: Empirical methods in natural language processing

Bouthillier X, Konda K, Vincent P, Memisevic R (2015) Dropout as data augmentation, arXiv:1506.08700

Breiman L (1994) Bagging predictors. Mach Learn, pp 123–140

Chatfield K, Simonyan K, Vedaldi A, Zisserman A (2014) Return of the devil in the details: delving deep into convolutional nets. In: British machine vision

Chen PY, Zhang H, Sharma Y, Yi J, Hsieh CJ (2017) Zoo: zeroth order optimization based black-box attacks to deep neural networks without training substitute models. In: Proceedings of the 10th ACM workshop on artificial intelligence and security

Cohen D, Mitra B, Hofmann K, Croft WB (2018) Cross domain regularization for neural ranking models using adversarial learning, arXiv:1805.03403v1

Collobert R, Weston J, Bottou L, Karlen M, Kavukcuoglu K, Kuksa P (2011) Natural language processing (almost) from scratch. J Mach Learn Res 12:2493–2537

DeVries T, Taylor GW (2017) Improved regularization of convolutional neural networks with cutout, arXiv:1708.04552, 2017

Domingos P (2000) A unified bias-variance decomposition and its applications. In: International conference on machine learning

Domingos P (2012) A few useful things to know about machine learning. Commun ACM 55(10):78–87

Dong Y, Liao F, Pang T, Su H, Hu X, Li J, Zhu J (2017) Boosting adversarial attacks with momentum, arXiv:1710.06081

Erhan D, Manzagol PA, Bengio Y, Bengio S, Vincent P (2009) The difficulty of training deep architectures and the effect of unsupervised pre-training. In: AISTATS

Frazão XF, Alexandre LA (2014) DropAll: generalization of two convolutional neural network regularization methods. In: International conference on image analysis and recognition

Gal Y, Ghahramani Z (2016) Dropout as a Bayesian approximation: representing model uncertainty in deep learning. In: Proceedings of the international conference on machine learning

Gastaldi X (2017) Shake–shake regularization, arXiv:1705.07485

Ghahramani Z (2015) Probabilistic machine learning and artificial intelligence. Nature 521:452–459

Gitman I, Ginsburg B (2017) Comparison of batch and weight normalization algorithms for largescale image classification, arXiv:1709.08145

Goodfellow IJ, Warde-Farley D, Mirza M, Courville A, Bengio Y (2013) Maxout networks. In: International conference on machine learning

Goodfellow IJ, Shlens J, Szegedy C (2014) Explaining and harnessing adversarial examples. CoRR. arXiv:1412.6572

Goodfellow I, Bengio Y, Courville A (2016) Deep learning. MIT Press, Cambridge

Graham B (2015) Fractional max-pooling, arXiv:1412.6071

He K, Zhang X, Ren S, Sun J (2015) Delving deep into rectifiers: surpassing human-level performance on ImageNet classification, arXiv:1502.01852

He K, Zhang X, Ren S, Sun J (2016) Deep residual learning for image recognition, arXiv:1512.03385v1

Helmstaedter M, Briggman KL, Turaga SC, Jain V, Seung HS, Denk W (2013) Connectomic reconstruction of the inner plexiform layer in the mouse retina. Nature 500:168–174

Henke N, Bughin J, Chui M, Manyika J, Saleh T, Wiseman B, Sethupathy G (2016) The age of analytics: competing in a data-driven world. In: McKinsey Global Institute

Hinton G, Deng L, Yu D, Dahl G, Mohamed AR, Jaitly N, Senior A, Vanhoucke V, Nguyen P, Sainath TN, Kingsbury B (2012a) Deep neural networks for acoustic modeling in speech recognition: the shared views of four research groups. IEEE Signal Process Mag 29(6):82–97

Hinton G, Deng L, Yu D, Dahl G, Mohamed A, Jaitly N, Senior A, Vanhoucke V, Nguyen P, Sainath T, Kingsbury B (2012b) Deep neural networks for acoustic modeling in speech recognition. IEEE Signal Process Mag 29(6):82–97

Hinton G, Srivastava N, Krizhevsky A, Sutskever I, Salakhutdinov R (2012c) Improving neural networks by preventing co-adaptation of feature detectors, arXiv:1207.0580

Hochreiter S, Schmidhuber J (1995) Simplifying neural nets by discovering flat minima. In: Advances in neural information processing systems, vol 7

Hochreiter S, Schmidhuber J (1997) Long short-term memory. Neural Comput 9(8):1735–1780

Huang G, Liu Z, Weinberger KQ, Maaten L (2016a) Densely connected convolutional networks, arXiv:1608.06993
Huang G, Sun Y, Liu Z, Sedra D, Weinberger K (2016b) Deep networks with stochastic depth, arXiv:1603.09382
Huang L, Liu X, Lang B, Yu AW, Wang W, Li B (2017) Orthogonal weight normalization: solution to optimization over multiple dependent stiefel manifolds in deep neural networks, arXiv:1709.06079
Ioffe S, Szegedy C (2015) Batch normalization: accelerating deep network training by reducing internal covariate shift. arXiv:1502.03167
Jakubovitz D, Giryes R (2018) Improving DNN robustness to adversarial attacks using jacobian regularization. In: European conference on computer vision
Kang G, Li J, Tao D (2016) Shakeout: a new regularized deep neural network training scheme. In: Proceedings of the thirtieth AAAI conference on artificial intelligence
Krizhevsky A, Sutskever I, Hinton G (2012) ImageNet classification with deep convolutional neural networks. In: Advances in neural information processing systems
Krizhevsky A, Sutskever I, Hinton G (2017) ImageNet classification with deep convolutional neural networks. Commun ACM 60(6):84–90
Laarhoven TV (2017) L2 regularization versus batch and weight normalization. arXiv:1706.05350
Larsson G, Maire M, Shakhnarovich G (2017) FractalNet: ultra-deep neural networks without residuals, arXiv:1605.07648
LeCun Y, Bengio Y, Hinton G (2015) Deep learning. Nature 521:436–444
Ma J, Sheridan RP, Liaw A, Dahl GE, Svetnik V (2015) Deep neural nets as a method for quantitative structure-activity relationships. J Chem Inf Model 55(2):263–274
Maeda SI (2014) A Bayesian encourages dropout, arXiv:1412.7003
Mash R, Borghetti B, Pecarina J (2016) Improved aircraft recognition for aerial refueling through data augmentation in convolutional neural networks. In: International symposium on visual computing
Moradi R, Berangi R, Minaei B (2019) SparseMaps: convolutional networks with sparse feature maps for tiny image classification. Expert Syst Appl 119:142–154
Morerio P, Cavazza J, Volpi R, Vidal R, Murino V (2017) Curriculum dropout, arXiv:1703.06229
Ng AY (1997) Preventing "overfitting" of cross-validation data. In: International conference on machine learning
Peng H, Mou L, Li G, Chen Y, Lu Y, Jin Z (2015) A comparative study on regularization strategies for embedding-based neural networks. In: Empirical methods in natural language processing
Poole B, Sohl-Dickstein J, Ganguli S (2014) Analyzing noise in autoencoders and deep networks. In: CoRR
Roth K, Lucchi A, Nowozin S, Hofmann T (2018) Adversarially Robust training through structured gradient regularization. arXiv:1805.08736v1
Rozsa A, Rudd EM, Boult TE (2016) Adversarial diversity and hard positive generation. In: Proceedings of the IEEE conference on computer vision and pattern recognition workshops
Salimans T, Kingma DP (2016) Weight normalization: a simple reparameterization to accelerate training of deep neural networks, arXiv:1602.07868
Sankaranarayanan S, Jain A, Chellappa R, Lim SN (2018) Regularizing deep networks using efficient layerwise adversarial training. In: AAAI conference on artificial intelligence
Santurkar S, Tsipras D, Ilyas A, Madry A (2018) How does batch normalization help optimization? arXiv:1805.11604
Shalev-Shwartz S, Ben-David S (2014) Rademacher complexities. In: Understanding machine learning—from theory to algorithms. Cambridge University Press, Cambridge, pp 325–336
Srivastava N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R (2014a) Dropout: a simple way to prevent neural networks from overfitting. J Mach Learn Res 15:1929–1958
Srivastava N, Hinton GE, Krizhevsky A, Sutskever I (2014b) A simple way to prevent neural network to prevent overfitting. Mach Learn Res 15(1):1929–1958
Su J, Vargas DV, Kouichi S (2017) One pixel attack for fooling deep neural networks, arXiv:1710.08864
Sutskever I, Vinyals O, Le QV (2014) Sequence to sequence learning with neural networks. In: Advances in neural information processing systems
Szegedy C, Zaremba W, Sutskever I, Bruna J, Erhan D, Goodfellow I, Fergus R (2013) Intriguing properties of neural networks, arXiv:1312.6199
Szegedy C, Liu W, Jia Y, Sermanet P, Reed S, Anguelov D, Erhan D, Vanhoucke V, Rabinovich A (2014) Going deeper with convolutions, arXiv:1409.4842
Szegedy C, Vanhoucke V, Ioffe S, Shlens J, Wojna Z (2015) Rethinking the inception architecture for computer vision, arXiv:1512.00567
Taylor L, Nitschke G (2017) Improving deep learning using generic data augmentation, arXiv:1708.06020

Wager S, Wang S, Liang PS (2013) Dropout training as adaptive regularizationauthor. In: Advances in neural information processing systems

Wan L, Zeiler M, Zhang S, LeCun Y, Fergus R (2013) Regularization of neural networks using DropConnect. In: ICML, Department of Computer Science, Courant Institute of Mathematical Science, New York University, [Online]. Available: https://cs.nyu.edu/~wanli/dropc/

Wang Q, JaJa J (2013) From maxout to Channel-Out: Encoding information on sparse pathways, arXiv :1312.1909

Wang S, Manning C (2013) Fast dropout training. In: International conference on machine learning

Wen W, Wu C, Wang W, Chen Y, Li H (2016) Learning structured sparsity in deep neural networks. In: Advances in neural information processing systems

Wolpert D (1996) The lack of a priori distinctions between learning algorithms. Neural Comput 8:1341–1390

Wu Y, He K (2018) Group normalization, arXiv:1803.08494

Xiong HY, Alipanahi B, Lee LJ, Bretschneider H, Merico D, Yuen RKC, Frey BJ (2015) The human splicing code reveals new insights into the genetic determinants of disease. Science 347(6218):1254806

Yu K, Xu W, Gong Y (2009) Deep learning with kernel regularization for visual recognition. In: Advances in neural information processing systems, vol 21

Yuan X, He P, Zhu Q, Bhat RR, Li X (2017) Adversarial examples: attacks and defenses for deep learning, arXiv:1712.07107

Zeiler MD, Fergus R (2013) Stochastic pooling for regularization of deep convolutional neural networks, arXiv:1301.3557

Zeiler M, Fergus R (2014) Visualizing and understanding convolutional networks. In: IEEE European conference on computer vision

Zhao Z, Dua D, Singh S (2017) Generating natural adversarial examples, arXiv:1710.11342