Student Project - Enhancing performance of deep Convolutional Neural Networks initialization methods

By: Meidar Sharkansky

Abstract

In recent years Convolutional Neural Networks (CNN) has shown tremendous success in various problems in the field of computer vision. This success raised the question whether deeper networks will outperform the shallower ones. It turns out this deepening procedure is not straight forward and new problems arise, which require new insights and solutions. This project refers to a specific issue of deep CNN initialization. It discusses previous analysis and the solution suggested upon it, such as He [1] and Xavier [2] initialization. Later on, it continues to a hypothesis which will try to explain why inferior performances are observed as CNNs become deeper, focusing on the "variety" of the network's filters. After the former hypothesis is presented, a discussion will take place regarding how can one manipulate an initialization method (i.e. a well-defined distribution) with respect to the new hypothesis, yet still generally obey the rules yielded by the analysis upon that method is formed. This discussion will pave the way towards a new suggested algorithm, which receives a well-defined distribution as input, and generates random vectors (i.e. Neural Networks filters) which will take the benefit of both the hypothesis and the input method analysis.
Results of these produced enhanced methods will be presented when tested upon famous image classification problems such as CIFAR-10 and CIFAR-100, using the popular ResNet [3] CNN. A distinct improvement on the convergence speed is observed and several improvements on the final results, both with regard to the precious of the validation set.

1. Brief And General Overview Of ANN And CNN

In a neural network composed of n layers, the n layers can be described as:

$$y^i = W^i \cdot x^i + b^i, \ \ \forall i \in \{1, \dots, n\} \tag{1.1}$$

where $W^i, x^i \ and \ b^i$ are the weights, input and bias of the $i'th$ layer respectively, and y^i is the output of the $i'th$ layer. The input x^i is a ${k_i}^2 c_i$-by-1 vector where ${k_i}^2$ represents a $k_i \times k_i$ input pixels over c_i channels, resulting in ${k_i}^2 c_i$ connections between the input and the response, which will be denoted as $n^i = {k_i}^2 c_i$. The weights W^i is a matrix with dimensions of $d_i \times n^i$, where d_i is the number of filters (a.k.a. kernels) in the $i'th$ layers. y^i is then passed through and activation function and because the input of the $(i+1)'th$ layer, meaning:

$$f(y^i) = x^{i+1}, \ \forall i \in \{1, \dots, n-1\} \tag{1.2}$$

Here f is the activation function (e.g. tanh, Relu or sigmoid), which provides the network it's non-linearity property. Notice that as a result of the following description, the equality $d_i = c_{i+1}$ exists.

A convolution neural network can be generally described as ANN above, yet with a slight change - $y^{i+1} = f(W^i * x^i + b^i)$, where $*$ described the convolution operator. As of this moment, it seems all famous implementations of neural networks implement these convolution operators as cross-correlation, which in turn can be viewed as a dot product of two vectorized matrices. In this paper the operation is regarded as dot product since one must remember that the word 'convolution' nowadays is merely an illusion.

The learning process of a NN operates by feeding the input data into the first layer, meaning x^1 is the input, and then iteratively calculating each layer's input and output as described in equation (1.1), which is known as forward-propagation. Once the final layer's output if produced, i.e. y^n, it is fed into a "cost function" (e.g. MSE, Cross-Entropy), and then a backward-propagation process takes place, which is usually a gradient decent algorithm of a generic form:

$$w_{t,j}^i \leftarrow \alpha w_{t,j}^i + \beta \frac{\partial Cost}{\partial w_{t,j}^i}, \forall i \in [n], (d,j) \in d_i \times n^i \tag{1.3}$$

Where $w_{t,j}^i$ is the $j'th$ entry in the $t'th$ filter of W^i, the term $\frac{\partial Cost}{\partial w_{t,j}^i}$ is the gradient of $w_{t,j}^i$ with respect to the results of the "cost function" (i.e. $Cost = Cost(y^n)$) and α, β are momentum and learning rate parameters respectively.

2. Deep Neural Networks initialization

In 2009 Bradley [4] investigate the effects of gradients in deep NN with linear activation. His studies showed that during the back-propagation process the gradients tend to decrease as the derivation process moves backwards. This work was later used by Xavier [1] which used an analysis that in order to overcome this gradient decrease, the variance between the layer's should be maintain equal, that is:

$$Var[y_d^i] = Var[W_d^i \cdot x^i + b_d^i]$$
$$\forall i \in \{1, \dots, n\}, \forall d \in \{1, \dots, d_i\} \tag{2.1}$$

Where W_d^i, b_d^i is the $d'th$ filter and bias, respectively, of $i'th$ layer, and y_d^i is its response. This equation can also be written as:

$$
\begin{aligned}
&Var[y_d^i] \\
&= Var[W_d^i \cdot x^i + b_d^i] \\
&= Var\left[W_{d,1}^i \cdot x_1^j + \ldots + W_{d,n^i}^i \cdot x_{n^i}^j\right] \\
&\overset{(1)}{=} Var\left[\sum_{j=1}^{n^i} W_{d,j}^i \cdot x_j^j\right] \\
&\overset{(2)}{=} \sum_{j=1}^{n^i} Var\left[W_{d,j}^i \cdot x_j^j\right] \\
&\overset{(3)}{=} \sum_{j=1}^{n^i} \mathbb{E}\left[x_j^j\right]^2 Var\left[W_{d,j}^i\right] + \mathbb{E}\left[W_{d,j}^i\right]^2 Var\left[x_j^j\right] \\
&\qquad\qquad + Var\left[x_j^j\right] Var\left[W_{d,j}^i\right] \\
&\overset{(4)}{=} \sum_{j=1}^{n^i} Var\left[x_j^j\right] Var\left[W_{d,j}^i\right] \\
&\overset{(5)}{=} n^i \left[Var\left[x_j^j\right] Var\left[W_{d,j}^i\right]\right]
\end{aligned}
\tag{2.2}
$$

Where passage (1) is rewriting the dot-product by definition; passage (2) is due to independence of each element which results in uncorrelation; passage (3) is by definition of variance; passage (4) is assuming that the input and weights are distributed with zero mean; passage (5) is by assuming that the inputs and weights are i.i.d.

Equation (2.2) can now help us derive the Xavier [1] and He [2] initializations. Xavier suggested that since each input entry should has the same variance as it's response,

$$
\begin{aligned}
Var\left[y_d^i\right] &= n^i Var\left[x_j^j\right] Var\left[W_{d,j}^i\right] \\
1 &= n^i Var\left[W_{d,j}^i\right] \\
\frac{1}{n^i} &= Var\left[W_{d,j}^i\right]
\end{aligned}
\tag{2.3}
$$

then:

So the weights are distributed with zero mean and variance equal to $\frac{1}{n^i}$. He [2] suggested that when using ReLU activation function, half of values are nullified, so the equality $Var\left[y_d^i\right] = \frac{Var\left[x_j^j\right]}{2}$ holds. hence he derived:

So in this case the weights are distributed with zero mean and variance equal to $\frac{2}{n^i}$.

$$
\begin{aligned}
Var\left[y_d^i\right] &= n^i Var\left[x_j^j\right] Var\left[W_{d,j}^i\right] \\
2 &= n^i Var\left[W_{d,j}^i\right] \\
\frac{1}{n^i} &= Var\left[W_{d,j}^i\right]
\end{aligned}
\tag{2.4}
$$

TODO: maybe enter a table for normal and uniform for he and xaavier

3. Hypothesis and suggested alternative initialization

Up to this point, it was assumed that all weights are randomly generated. This is an important part of neural networks initialization yet one must first understand from whence it's importance. Assume as before that the $i'th$ layer in a CNN is of the form of (1.1), but now the weight matrix W^i is initialized to a constant value $c' \in \mathbb{R}$. Looking at the dot product of $W_j^i \cdot x^i$, where W_j^i is the $j'th$ row of W^i ($j'th$ filter), which corresponds to the $j'th$ response, y_j^i, we notice that for every two filters, $W_j^i, W_k^i : \forall j,k \in [d_i] \Rightarrow W_j^i = W_k^i = (c' \quad \ldots \quad c')$, which simply mean they are all equal. Due to that it holds that for every two filters $j,k \in [d_i]$:

$$
y_j^i = W_j^i \cdot x^i = W_k^i \cdot x^i = y_k^i \tag{3.1}
$$

Equation (3.1) shows that when the weights are not random, they all produce the same response. This is again affected in the back-propagation process which results in the same update as described in (1.3). Looking again at (2.3) and (2.4) it shows that both methods relay only on the fan-in value, n^i, in initialization (some variants also uses the fan-out, d_i). Combining this fact with the property established in (3.1) it comes to mind that since all filters, W_j^i, are drawn from exactly the same distribution, they might be "statistically equal". Obviously, they are not equal as in (3.1), but since they are all drawn from the same distribution they might be similar in their inner relation between the entries. This situation is specifically problematic in the use-case of neural networks, since each filter in the weight matrix W^i represents a certain linear combination of the input x^i and so, if the filters are somewhat similar and do not contain a variety of linear combination, the network might suffer from slow converging rate and possible lower performance at the end of the training.

In order to test this hypothesis, a suggested method is now presented, which upon receiving another initialization method (i.e. well-defined distribution) with one parameter, will attempt to "enhance" it and provide it's output random vector more variety. The basic idea behind is that the parameter of the input distribution will be randomly drawn such that it's mean value will be equal to the original one.

3.1 Normal case

Assuming the filter of W^i are distributed according to normal distribution, i.e. $W_j^i \sim N_{n_i}(\mu = 0, \sigma^2 I)$, $\forall j \in [d_i]$, where here N_{n_i} is a multi-variate normal distribution so the μ is $n_i \times 1$ vector of zeros and I is an $n_i \times n_i$ identity matrix. A naïve approach to obtain the goal is to set some arbitrary constant $\delta < \sigma^2$ and then randomly drawn the weights vector from:

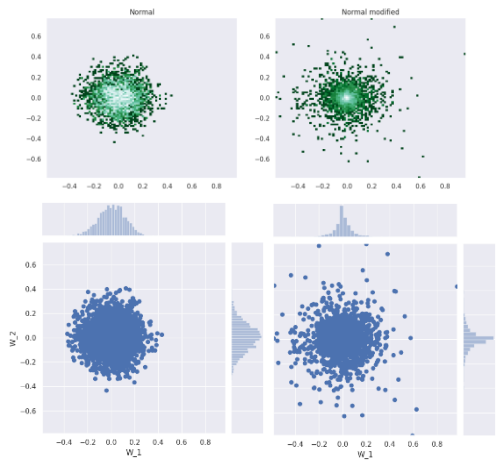


figure (3.1.0): plots of the first convolution layer in ResNet-110, with 64 filters, each of dimension 3x7x7. Each row in the channel dimension in each filter is a sample, totaling in 49*64=3169. **Top row** is 2D histogram with logarithm applied, and **bottom row** is a scatter plot. Each sample in the "Normal modified" column has a different sampled variance.

(3.1.1) $$W_j^i \sim N_{n_i}(\mu = 0, \sigma'^2 I) : \ \sigma'^2 \sim U(-\delta, \delta), \forall j \in [d_i]$$

Now indeed each filter W_j^i is drawn from a different distribution, yet as long as d_i remains linearly proportional to n_i the variance of W^i will remain the same (see section 4).

Yet the naïve solution described above still remain dependent on the selection of the hyper-parameter δ, which is not a good property since it imposes its selection upon the user. Since the suggested algorithm receives a distribution function as input, this implies this distribution has some meaning to the one defined it, hence it makes sense to use it in order to define the distribution of its own parameter.
In the normal case where we have:

(3.1.2) $$W_{d,j}^i \sim N(0, {\sigma_i}^2), \forall i \in [n], (d,j) \in d_i \times n^i$$

The following relation hold:

(3.1.3) $$W_{d,j}^i \sim N(0, {\sigma_i}^2) \implies \frac{W_{d,j}^i}{\sigma_i} \sim N(0,1) \implies \frac{{W_{d,j}^i}^2}{{\sigma_i}^2} \sim \chi^2(1)$$

For some $W_{d,j}^i$ denote $H = \frac{{W_{d,j}^i}^2}{\sigma_i^2}$, so:

(3.1.4) $$ {W_{d,j}^i}^2 = H{\sigma_i}^2 \sim \Gamma\left(\frac{1}{2}, 2{\sigma_i}^2\right) \implies E\left({W_{d,j}^i}^2\right) = {\sigma_i}^2$$

It is now visible that is the normal case where the mean value is zero, ${\sigma_i}^2$ is distribution $\Gamma\left(\frac{1}{2}, 2{\sigma_i}^2\right)$ (gamma). So now one can perform the following initialization:

(3.1.5) $$W_j^i \sim N_{n_i}(\mu = 0, \sigma'^2 I) : \ \sigma'^2 \sim \Gamma\left(\frac{1}{2}, 2{\sigma_i}^2\right), \forall j \in [d_i]$$

Which means that for each filter W_j^i one initially samples σ'^2 from $\Gamma\left(\frac{1}{2}, 2{\sigma_i}^2\right)$, and then samples W_j^i from $N_{n_i}(\mu = 0, \sigma'^2 I)$. Notice that since W^i is of dimension $d_i \times n^i$ where $n^i = {k_i}^2 c_i$, one can re-write the random vector W_j^i as a concatenation of ${k_i}^2$ vectors, each the size of c_i, and initial each of these sub-vectors separately, each with a random σ'^2. This can be summarized to the following initialization:

(3.1.6) $$\begin{gathered}(W_{j,t}^i, W_{j,t+1}^i, \dots, W_{j,t+c_i}^i) \sim N_{c_i}(\mu = 0, \sigma'^2 I) : \\ \sigma'^2 \sim \Gamma\left(\frac{1}{2}, 2{\sigma_i}^2\right), \\ \forall j \in [d_i], t \in \{0, c_i, \dots n^i - c_i = \left({k_i}^2 - 1\right)c_i\}\end{gathered}$$

3.2 Uniform case

We now consider the uniform case in which the input distribution is of the form:

(3.2.1) $$W_{d,j}^i \sim U(-a, a), \forall i \in [n], (d,j) \in d_i \times n^i$$

We would once again want to use the moments of $W_{d,j}^i$ as a way to draw the random a', and again it will be for this case the second moment, specifically referring to the relation that for every random variable Z with zero mean the equation $Var[Z] = \mathbb{E}[Z^2]$. The uniform case differs from the normal case in a way the if a random variable has a uniform distribution, its square does not follow any familiar distribution.
the weight $W_{d,j}^i$ is uniformly distributed, and so in order to calculate the mean value of ${W_{d,j}^i}^2$ we will use the "law of unconscious statistician" which states that given a random variable Z and some function $g(Z)$, then $E\big(g(Z)\big) = \int_{-\infty}^{\infty} g(z) f_Z(z) dz$.
Using the law above on $w = W_{d,j}^i \sim U(-a, a)$ we can use $g_2(x) = x^2$ and calculate the mean:

(3.2.2) $$\begin{aligned}E(w^2) &= E\big(g_2(w)\big) \\ &= \int_{-a}^{a} g_2(w) f_W(w) dw \\ &= \int_{-a}^{a} w^2 \frac{1}{2a} dw \\ &= \frac{w^3}{(3)2a}\Big|_{-a}^{a} \\ &= \frac{2a^3}{6a} = \frac{a^2}{3}\end{aligned}$$

Which then can used to conclude that:

(3.2.3) $$a' = \sqrt{3E(w^2)}$$

Finally, the following algorithm for sampling filters can be derived:

(3.2.4) $$\begin{gathered}W_{d,j}^i \sim U\big(-\sqrt{3a'}, \sqrt{3a'}\big) : a' \sim U(-a, a), \\ \forall i \in [n], (d,j) \in d_i \times n^i\end{gathered}$$

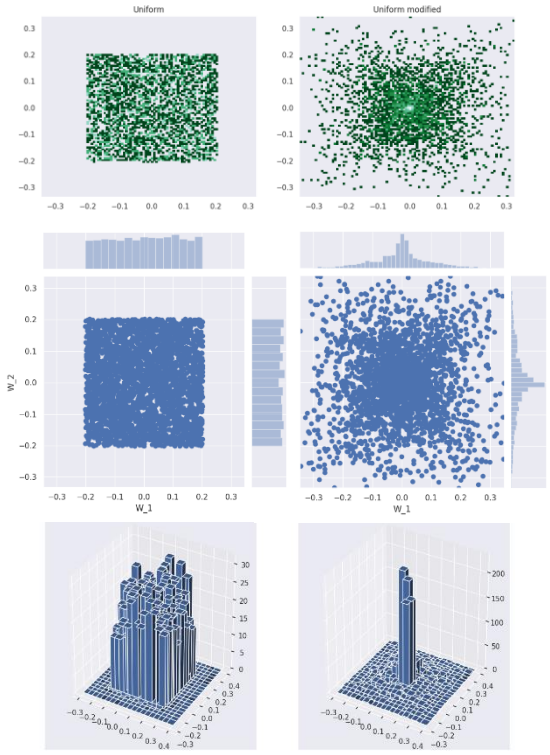


figure (3.2.5): plots of the first convolution layer in ResNet-110, with 64 filters, each of dimension 3x7x7. Each row in the channel dimension in each filter is a sample, totaling in 49*64=3169. **Top row** is 2D histogram with logarithm applied, and **bottom row** is a scatter plot. Each sample in the "uniform modified" column has a different sampled variance.

As in the normal case, it can be done at finer grain when applied to each sub-vector in the spatial space of the filter with regard to the channel dimension:

(3.2.6)
$$(W^i_{j,t}, W^i_{j,t+1}, \dots, W^i_{j,t+c_i}) \sim U\left(-\sqrt{3a'}, \sqrt{3a'}\right) : \\ a' \sim U(-a, a), \\ \forall j \in [d_i], t \in \{0, c_i, \dots n^i - c_i = \left({k_i}^2 - 1\right)c_i\}$$

4. Analysis and Results

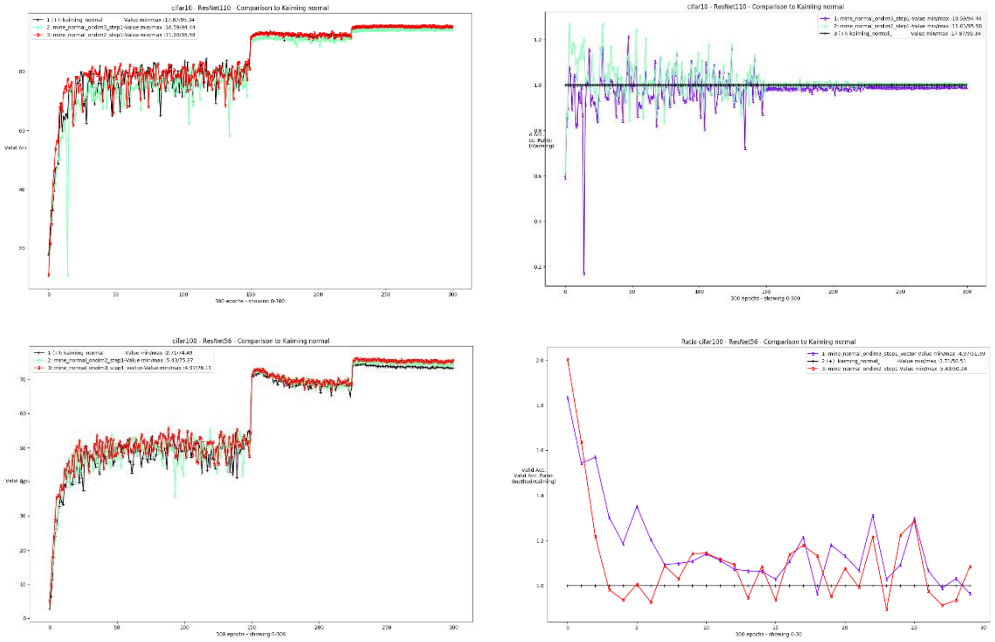
As mentioned in section 3, the goal is to receive an enhanced distribution which has more variety, yet still maintain the same variance of the entire layer. Obviously, the variance of each filter now is not exactly the same as the one of the input distribution, yet together the filters tend towards the same distribution, which logically can be referred as the mean of the variance. The first thing we observe is that indeed applying the enhancement upon an input distribution maintain the layers averaged variance, while still producing a different distribution. In other words, the first two moments remain the same but the other moments will have different values. Since in both variants (normal and uniform) the random vectors were drawn from a symmetric distributions, their mean value remain zero. performing EM distance measurements on the data as described in (3.1.1) and (3.2.1) results if EM distance between input distribution and modified of approximately 5 times magnitude, regardless of the target variance (the variance of the input distribution), yet the sample variance remains similar up to a certain error (does not show bias toward any lower or upper bound).

The more interesting results are the ones revealed when sampling how the enhance distributions performs in comparison to it's input distributions. As seen in the following graphs, not only a faster convergence is observer but also better final results. These results are seen on figures (4.1.1) and (4.2.1). It is noticeable that on the less deep net of ResNet56 on both cases convergence acceleration in the first 30 epochs is much more distinct, which is possible due to the overhead of data propagation though the net as it goes deep, yet in both cases the final results shows improvement. There are many more graphs in these scenario's and much can be understood upon how this initialization react, yet none of these conclusions can be said to be exact or within the boundaries of confidence that I am willing to declare at this stage.

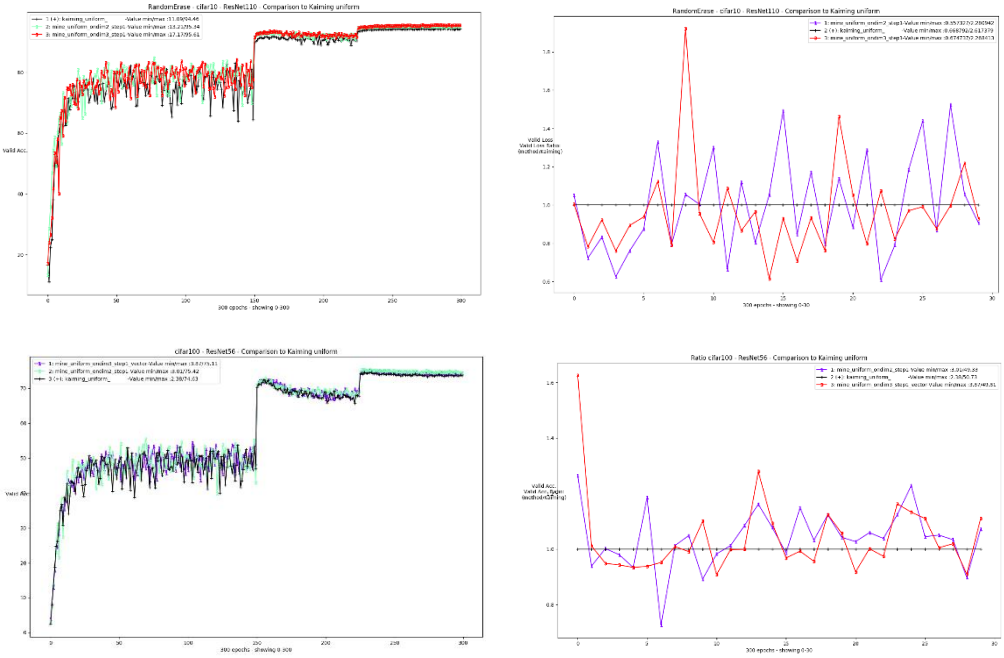
5. Discussion

The concept of Neural Network Initialization has been presented and explain. It is important to the reader of this work to keep in mind what are actually they filters in a neural network – for they are not simply random vectors but they tend to represent linear combination of the input data – more specifically – the projection of the input data upon the filters. Keeping that in mind, one come to realize that it might be an important task to make these linear combinations as versatile and as unique as they could possible be between one another, yet still maintain the well-studied properties that helps the network start running at a good starting point.

Much work has been done and not presented, especially in the part of analyzing the nature of these enhanced random vectors. This work does not appear hear mostly because it does not reveal much about the method's true behavior and/or relevant only for a certain type of input distribution.



figures (4.1.1) show test (valid) success on cifar10 and cifar100 dataset using ResNet110 (**top row**) and ResNet56 (**bottom row**) respectively during 300 epochs. The **left column** shows results during the entire process and the **right column** show the first 30 epochs as ratio between enhance and input kaiming normal.



figures (4.2.1) show test (valid) success on cifar10 and cifar100 dataset using ResNet110 (**top row**) and ResNet56 (**bottom row**) respectively during 300 epochs. The **left column** shows results during the entire process and the **right column** show the first 30 epochs as ratio between enhance and input kaiming uniform.

References

[1] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In International Conference on Artificial Intelligence and Statistics, pages 249–256, 2010

[2] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In ICCV, 2015.

[3] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In Proceedings of CVPR, pages 770–778, 2016.

[4] Bradley, D. (2009). Learning in modular systems. Doctoral dissertation, The Robotics Institute, Carnegie Mellon University