# Neural Network and Deep Learning

## Introduction to Deep Learning

YANIKA KONGSOROT

# Outline

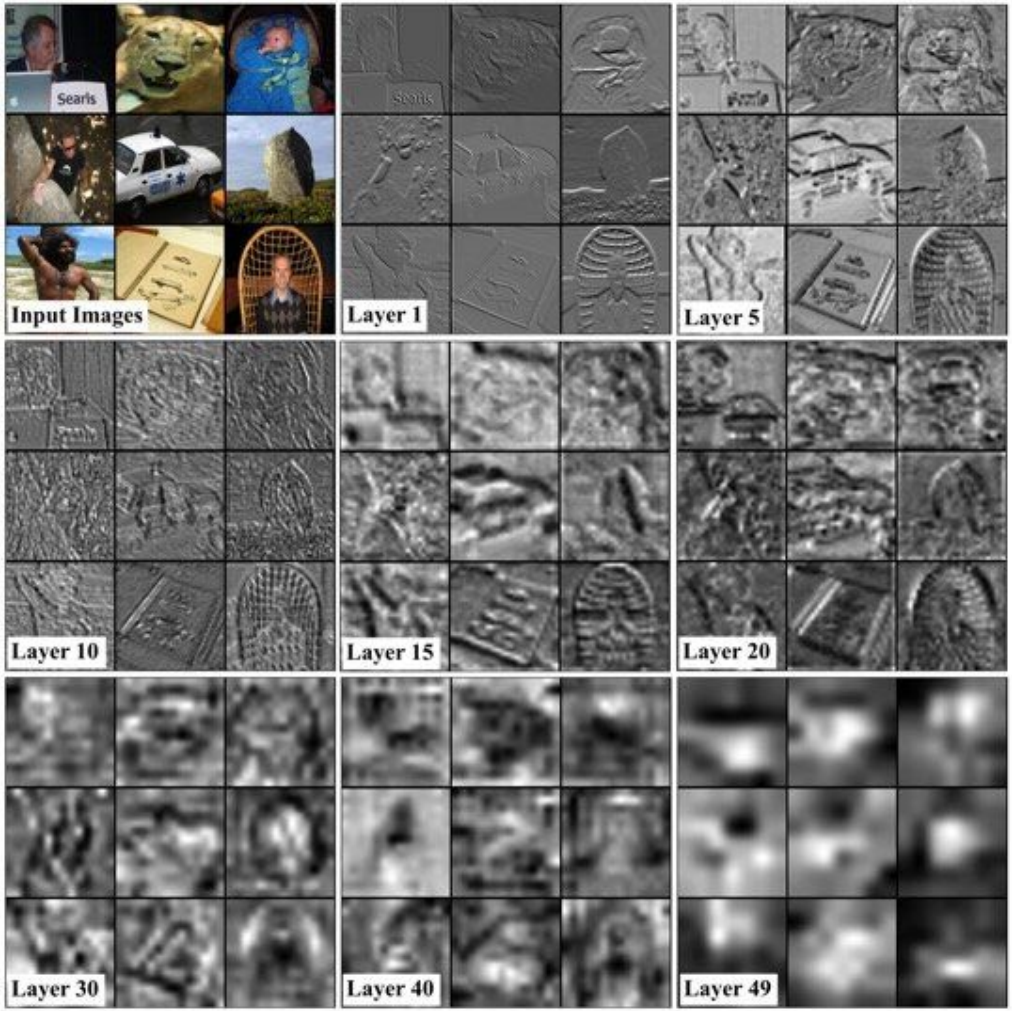
- Timeline of Deep Learning
- Why Increase the Layer Size of MLP?
- Disadvantages of MLP
- 2006 Breakthrough
- Working ideas on how to train deep architectures

# Timeline of Deep Learning

| | | |
|---|---|---|
| 1943 | McCulloch-Pitts neuron model: McCulloch; Pitts | |
| … | … | |
| 1957 | Perceptron: Rosenblatt | |
| 1986 | Backpropagation: Rumelhart; Hinton; Williams | |
| 1987 | Original CNN: Homma; Atlas; Marks II | |
| 1997 | LSTM: Hochreiter; Schmidhuber | |
| 1998 | Applied CNN: LeCun; Bottou; Bengio; Haffner | |
| 2006 | Deep Learning: Hinton; Salakhutdinov/ Deep Belief Nets: Hinton; Osindero; Teh | |
| 2009 | ImageNet: Deng; Dong; Socher; Li; Li; Fei-Fei | |
| 2017 | AlexNet: Krizhevsky; Sutskever; Hinton | Deep Learning Renaissance |

# Why Increase the Layer Size of MLP?

- **Capture More Complex Patterns**
  - Deeper and wider networks can capture more complex patterns and relationships in the data.
- **Hierarchical Feature Learning**
  - Lower layers can learn simple features, while higher layers can combine these into more *abstract representations*.
- **Increased Model Capacity**:
  - Larger networks have a higher capacity to model complex functions.




# Different level of abstraction

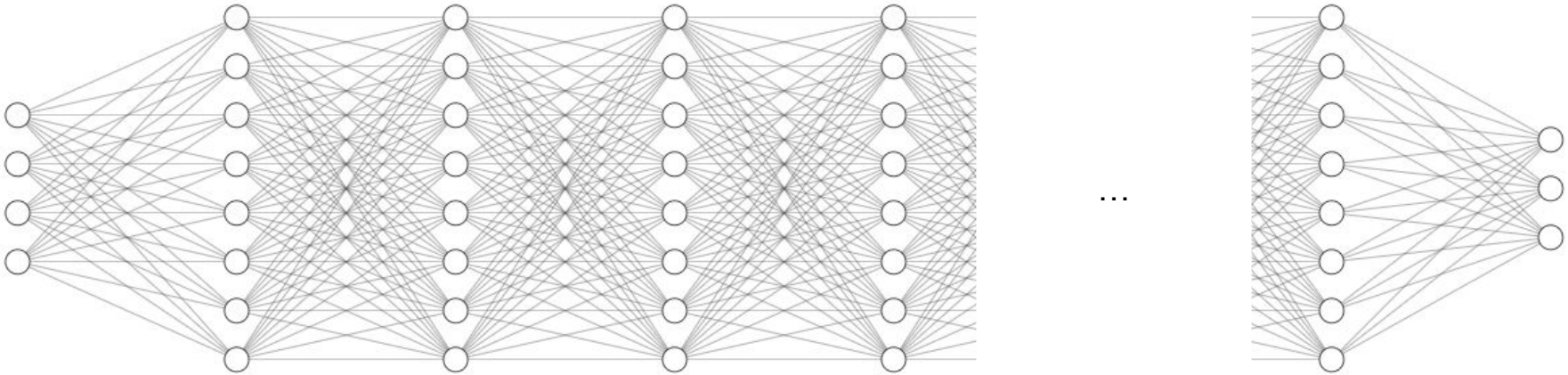
**Low-Level Abstraction (Early Layers)**: These layers (closer to the input) capture basic features like *edges, textures, and simple shapes*.

**Mid-Level Abstraction (Intermediate Layers)**: These layers combine the basic features to detect more complex patterns like *motifs, parts of objects, or textures*.

**High-Level Abstraction (Deeper Layers)**: The layers closer to the output layer capture very abstract and complex features, often corresponding to *entire objects or high-level concepts*.

Mahdi, A., Qin, J., & Crosby, G. (2019). DeepFeat: A bottom-up and top-down saliency model based on deep features of convolutional neural networks. IEEE Transactions on Cognitive and Developmental Systems, 12(1), 54-63.

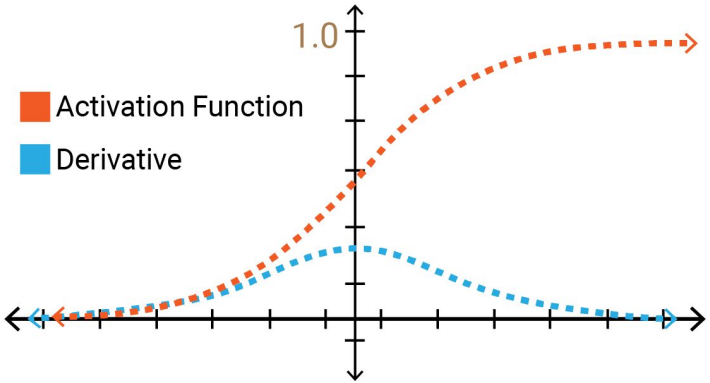
# Why Increase the Layer Size of MLP?




$$\hat{\mathbf{y}} = \sigma(\sigma(...\sigma(\sigma(\mathbf{X}\mathbf{w}^1 + \mathbf{b}^1)\mathbf{w}^2 + \mathbf{b}^2)...)\mathbf{w}^K + \mathbf{b}^K)$$

# Disadvantages of MLP

## Vanishing Gradient Problem




The maximum value of the sigmoid derivative is 0.25, and it occurs at x=0.

For large positive or negative values of x, the sigmoid derivative approaches 0.

https://kku.world/au43p

The vanishing gradient problem occurs in deep neural networks when **gradients of the loss function become very small as they are propagated backward through the network.** During backpropagation, the gradients are computed using the chain rule. **When the network is deep, the gradient at each layer is a product of many small derivatives (each less than 1), leading to an exponentially smaller gradient as it moves back through each layer.**

$$\hat{\mathbf{y}} = \sigma(\sigma(...\sigma(\sigma(\mathbf{X}\mathbf{w}^1 + \mathbf{b}^1)\mathbf{w}^2 + \mathbf{b}^2)...)\mathbf{w}^K + \mathbf{b}^K)$$

$$\frac{\partial L}{\partial \mathbf{w}^1} = \left(\frac{\partial L}{\partial \hat{\mathbf{y}}} \cdot \sigma'(\mathbf{z}^K) \cdot \mathbf{w}^K \cdots \sigma'(\mathbf{z}^2) \cdot \mathbf{w}^2 \cdot \sigma'(\mathbf{z}^1)\right) \cdot \mathbf{X}$$

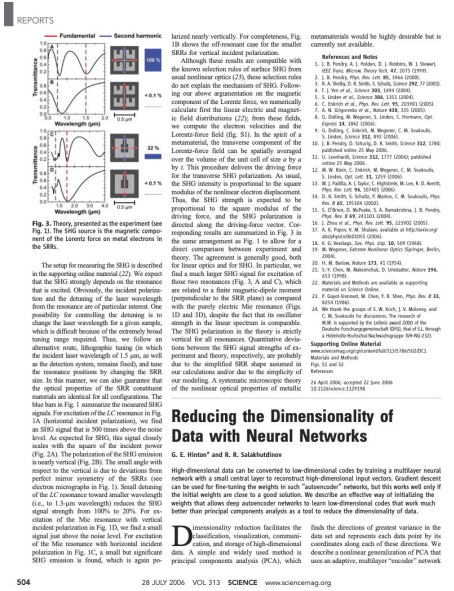
$$\approx 0.000.....001$$

$$\mathbf{w}^1 = \mathbf{w}^1 + \eta \times 0.000.....001$$

$$\mathbf{w}^1 \approx \mathbf{w}^1$$

# 2006 Breakthrough

# 2006 Breakthrough – Hinton and Salakhutdinov



REPORTS

Reducing the Dimensionality of Data with Neural Networks

G. E. Hinton* and R. R. Salakhutdinov

High-dimensional data can be converted to low-dimensional codes by training a multilayer neural network with a small central layer to reconstruct high-dimensional input vectors. Gradient descent can be used for fine-tuning the weights in such "autoencoder" networks, but this works well only if the initial weights are close to a good solution. We describe an effective way of initializing the weights that allows deep autoencoder networks to learn low-dimensional codes that work much better than principal components analysis as a tool to reduce the dimensionality of data.

***The first solution to the vanishing gradient problem.***

***Build the model layer by layer using unsupervised learning.***

- The early layers have features that are already initialized or "pretrained" with appropriate weights.
- During supervised learning, these pretrained features in the early layers require only slight adjustments to achieve good results.

https://www.cs.toronto.edu/~hinton/absps/science.pdf

# 2006 Breakthrough – Hinton, Osindero and Teh

LETTER Communicated by Yann Le Cun

## A Fast Learning Algorithm for Deep Belief Nets

**Geoffrey E. Hinton**
*hinton@cs.toronto.edu*
**Simon Osindero**
*osindero@cs.toronto.edu*
*Department of Computer Science, University of Toronto, Toronto, Canada M5S 3G4*

**Yee-Whye Teh**
*tehyw@comp.nus.edu.sg*
*Department of Computer Science, National University of Singapore, Singapore 117543*

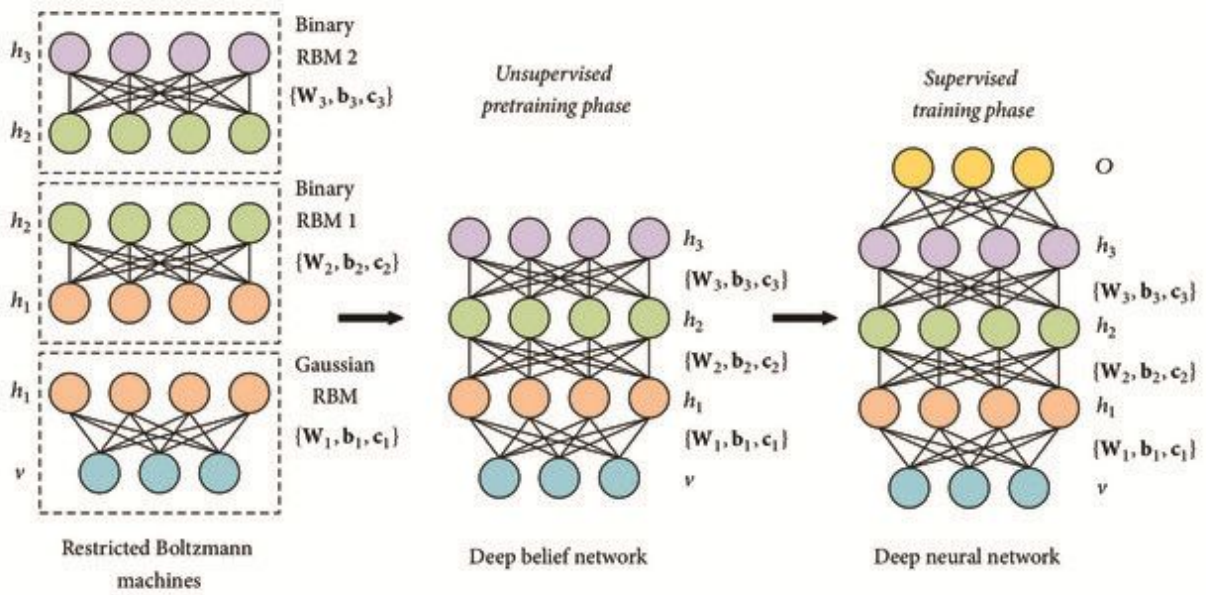
**We show how to use "complementary priors" to eliminate the explaining-away effects that make inference difficult in densely connected belief nets that have many hidden layers. Using complementary priors, we derive a fast, greedy algorithm that can learn deep, directed belief networks one layer at a time, provided the top two layers form an undirected associative memory. The fast, greedy algorithm is used to initialize a slower learning procedure that fine-tunes the weights using a contrastive version of the wake-sleep algorithm. After fine-tuning, a network with three hidden layers forms a very good generative model of the joint distribution of handwritten digit images and their labels. This generative model gives better digit classification than the best discriminative learning algorithms. The low-dimensional manifolds on which the digits lie are modeled by long ravines in the free-energy landscape of the top-level associative memory, and it is easy to explore these ravines by using the directed connections to display what the associative memory has in mind.**

### 1 Introduction

Learning is difficult in densely connected, directed belief nets that have many hidden layers because it is difficult to infer the conditional distribution of the hidden activities when given a data vector. Variational methods use simple approximations to the true conditional distribution, but the approximations may be poor, especially at the deepest hidden layer, where the prior assumes independence. Also, variational learning still requires all of the parameters to be learned together and this makes the learning time scale poorly as the number of parameters increases.

We describe a model in which the top two hidden layers form an undirected associative memory (see Figure 1) and the remaining hidden layers

*Neural Computation* **18, 1527–1554** (2006) 




Hinton, G. E., Osindero, S., & Teh, Y. W. (2006). A fast learning algorithm for deep belief nets. *Neural computation*, *18*(7), 1527-1554.

# 2006 Breakthrough

Hinton et al. presented **Deep Belief Network (DBN)**

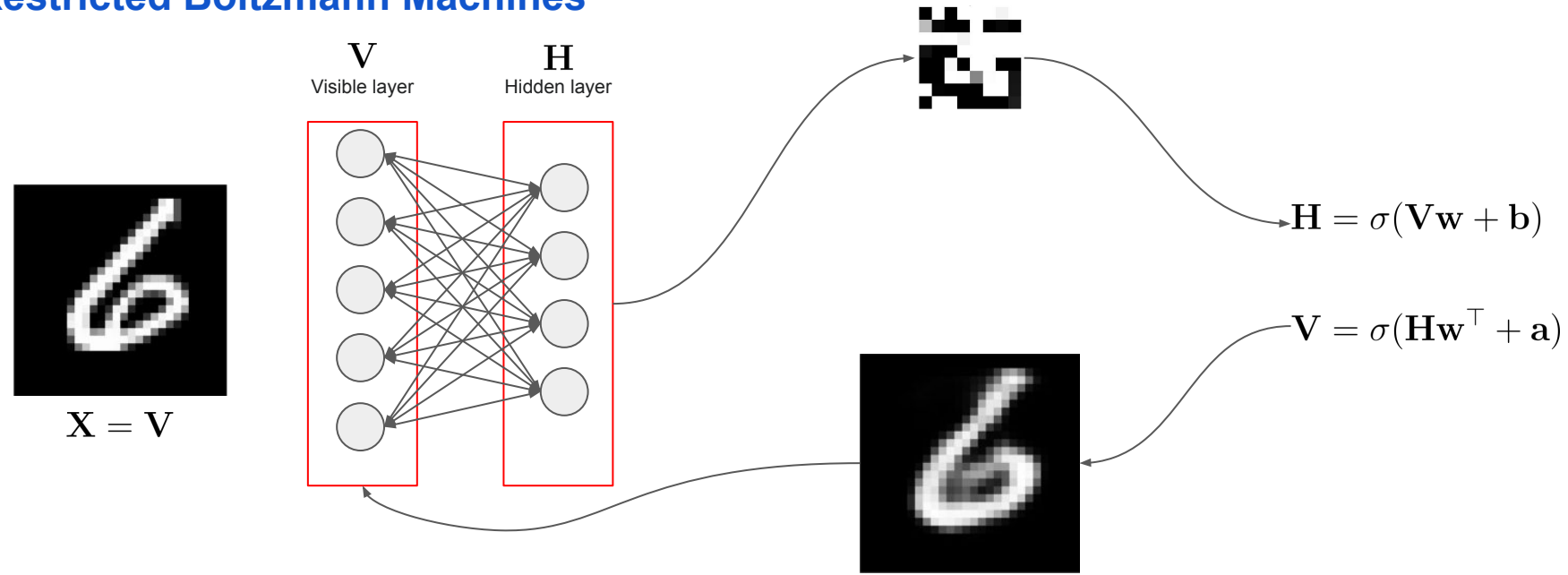
- A powerful models for learning hierarchical representations of data
- It leverages *unsupervised learning* for pretraining and can effectively capture complex features.
- *DBNs use Restricted Boltzmann Machines (RBMs)* for layer-wise pretraining.

Hinton, G. E., Osindero, S., & Teh, Y. W. (2006). A fast learning algorithm for deep belief nets. *Neural computation*, *18*(7), 1527-1554.

# 2006 Breakthrough

*Layer-Wise Unsupervised Pretraining*

## Restricted Boltzmann Machines



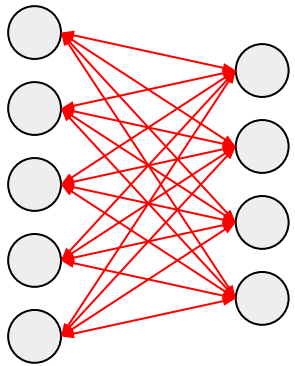

Smolensky, P. (1986). Information processing in dynamical systems: Foundations of harmony theory.

# 2006 Breakthrough

*Layer-Wise Unsupervised Pretraining*

## Restricted Boltzmann Machines

$$\mathbf{V} = \mathbf{X}$$

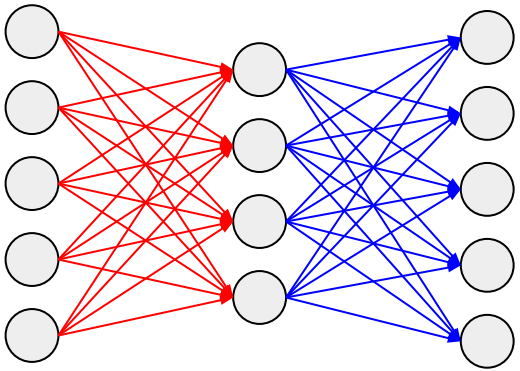


$$\mathbf{H} = \sigma(\mathbf{V}\mathbf{w} + \mathbf{b})$$

$$\mathbf{V} = \sigma(\mathbf{H}\mathbf{w}^{\top} + \mathbf{a})$$

Smolensky, P. (1986). Information processing in dynamical systems: Foundations of harmony theory.

## Autoencoder



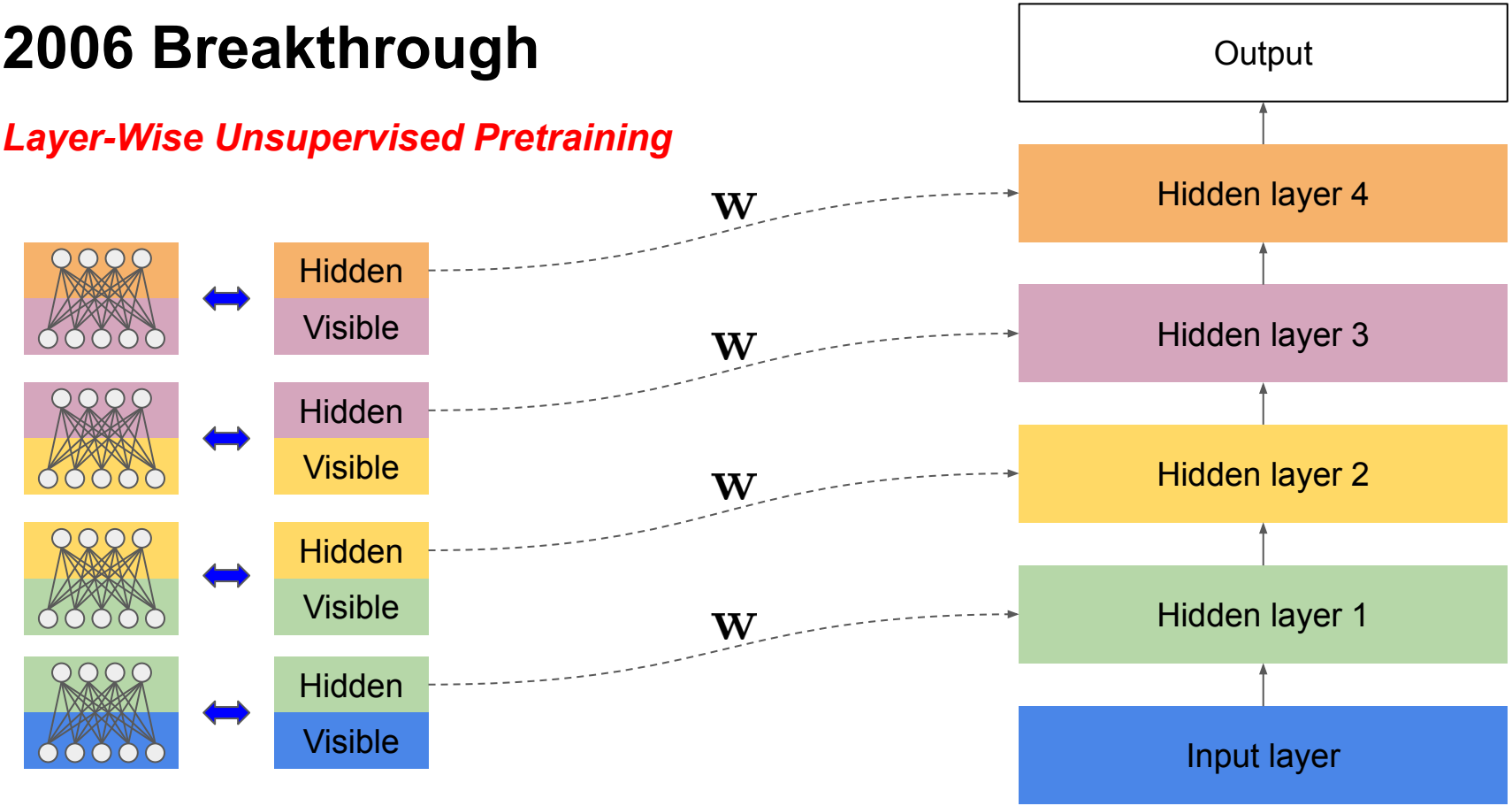
$$\mathbf{H} = \sigma(\mathbf{X}\mathbf{w} + \mathbf{b})$$

$$\hat{\mathbf{X}} = \sigma(\mathbf{H}\boldsymbol{\beta} + \mathbf{a})$$

Kramer, M. A. (1991). Nonlinear principal component analysis using autoassociative neural networks. AIChE journal, 37(2), 233-243.

# 2006 Breakthrough

***Layer-Wise Unsupervised Pretraining***



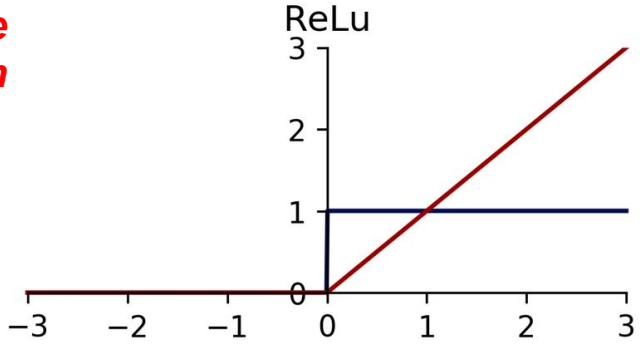
# Working ideas on how to train deep architectures

Working ideas on how to train deep architectures

# Activation Function

# Working ideas on how to train deep architectures

***The early solution to the vanishing gradient problem of sigmoid function.***




**Rectified Linear Units (ReLU)** are a type of activation function commonly used in neural networks, particularly in deep learning.

$$\sigma(x) = \begin{cases} 0 & , x \leq 0 \\ x & , x > 0 \end{cases}$$

$$\frac{d\sigma(x)}{dx} = \sigma'(x) = \begin{cases} 0 & , x \leq 0 \\ 1 & , x > 0 \end{cases}$$

Deep Sparse Rectifier Neural Networks

**Xavier Glorot**
DIRO, Université de Montréal
Montréal, QC, Canada
glorotxa@iro.umontreal.ca

**Antoine Bordes**
Heudiasyc, UMR CNRS 6599
UTC, Compiègne, France
and
DIRO, Université de Montréal
Montréal, QC, Canada
antoine.bordes@hds.utc.fr

**Yoshua Bengio**
DIRO, Université de Montréal
Montréal, QC, Canada
bengioy@iro.umontreal.ca

**Abstract**

While logistic sigmoid neurons are more biologically plausible than hyperbolic tangent neurons, the latter work better for training multi-layer neural networks. This paper shows that rectifying neurons are an even better model of biological neurons and yield equal or better performance than hyperbolic tangent networks in spite of the hard non-linearity and non-differentiability at zero, creating sparse representations with true zeros, which seem remarkably suitable for naturally sparse data. Even though they can take advantage of semi-supervised setups with extra-unlabeled data, deep rectifier networks can reach their best performance without requiring any unsupervised pre-training on purely supervised tasks with large labeled datasets. Hence, these results can be seen as a new milestone in the attempts at understanding the difficulty in training deep but purely supervised neural networks, and closing the performance gap between neural networks learnt with and without unsupervised pre-training.

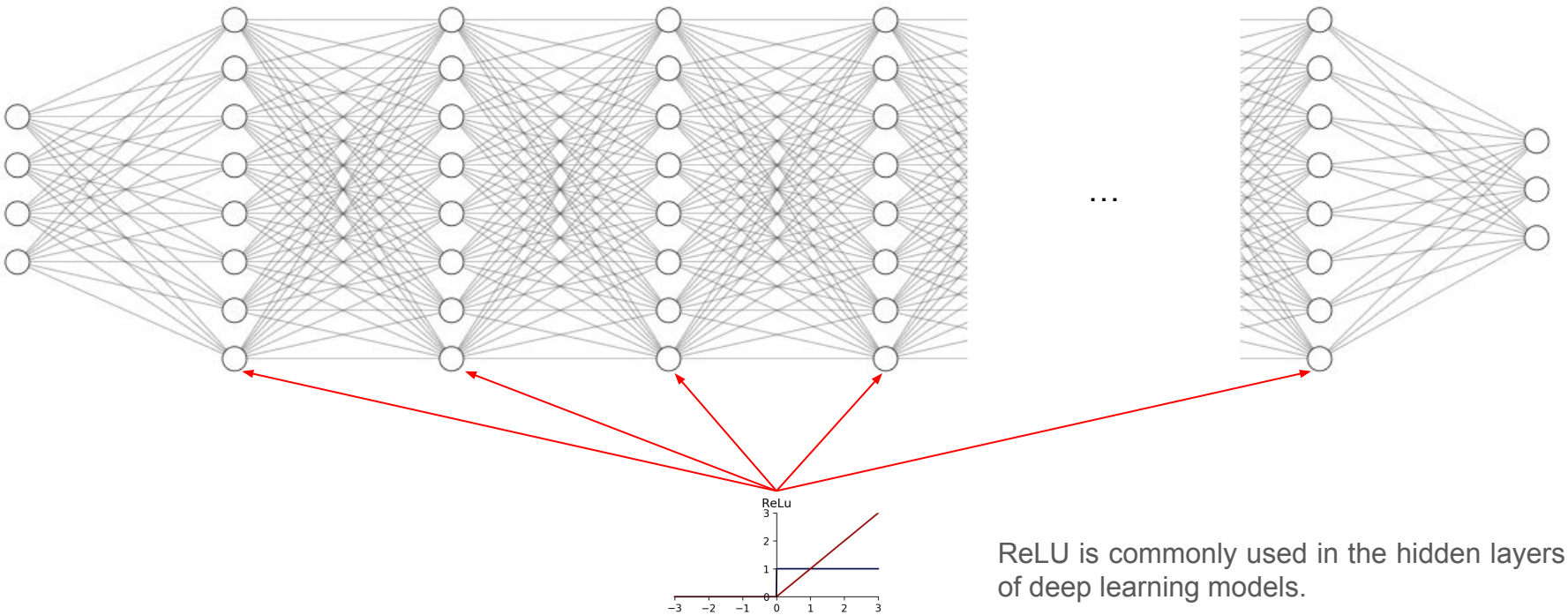
1 Introduction

Many differences exist between the neural network models used by machine learning researchers and those used by computational neuroscientists. This is in part because the objective of the former is to obtain computationally efficient learners, that generalize well to new examples, whereas the objective of the latter is to abstract out neuroscientific data while obtaining explanations of the principles involved, providing predictions and guidance for future biological experiments. Areas where both objectives coincide are therefore particularly worthy of investigation, pointing towards computationally motivated principles of operation in the brain that can also enhance research in artificial intelligence. In this paper we show that two common gaps between computational neuroscience models and machine learning neural network models can be bridged by using the following linear by part activation : $\max(0, x)$, called the rectifier (or hinge) activation function. Experimental results will show engaging training behavior of this activation function, especially for *deep architectures* (see Bengio (2009) for a review), i.e., where the number of hidden layers in the neural network is 3 or more.

Recent theoretical and empirical work in statistical machine learning has demonstrated the importance of learning algorithms for deep architectures. This is in part inspired by observations of the mammalian visual cortex, which consists of a chain of processing elements, each of which is associated with a different representation of the raw visual input. This is particularly clear in the primate visual system (Serre *et al.*, 2007), with its sequence of processing stages: detection of edges, primitive shapes, and moving up to gradually more complex visual shapes. Interestingly, it was found that the features learned in deep architectures resemble those observed in the first two of these stages (in areas V1 and V2 of visual cortex) (Lee *et al.*, 2008), and that they become increasingly invariant to factors of variation (such as camera movement) in higher layers (Goodfellow *et al.*, 2009).

Appearing in Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS) 2011, Fort Lauderdale, FL, USA. Volume 15 of JMLR: W&CP 15. 

# Working ideas on how to train deep architectures




ReLU is commonly used in the hidden layers of deep learning models.

Working ideas on how to train deep architectures

# Regularization

# Working ideas on how to train deep architectures

Deep neural nets with a large number of parameters are very powerful machine learning systems. However, overfitting is a serious problem in such networks. Large networks are also slow to use, making it difficult to deal with overfitting by combining the predictions of many different large neural nets at test time. Dropout is a technique for addressing this problem. The key idea is to randomly drop units (along with their connections) from the neural network during training. This prevents units from co-adapting too much. During training, dropout 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 simply using a single unthinned network that has smaller weights. This significantly reduces overfitting and gives major improvements over other regularization methods. We show 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 data sets.

## *Better Learning Regularization (e.g. Dropout)*

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. Journal of Machine Learning Research, 15(1), 1929-1958.

Journal of Machine Learning Research 15 (2014) 1929-1958 Submitted 11/13; Published 6/14

**Dropout: A Simple Way to Prevent Neural Networks from Overfitting**

**Nitish Srivastava** NITISH@CS.TORONTO.EDU
**Geoffrey Hinton** HINTON@CS.TORONTO.EDU
**Alex Krizhevsky** KRIZ@CS.TORONTO.EDU
**Ilya Sutskever** ILYA@CS.TORONTO.EDU
**Ruslan Salakhutdinov** RSALAKHU@CS.TORONTO.EDU
*Department of Computer Science*
*University of Toronto*
*10 Kings College Road, Rm 3302*
*Toronto, Ontario, M5S 3G4, Canada.*

**Editor:** Yoshua Bengio

**Abstract**

Deep neural nets with a large number of parameters are very powerful machine learning systems. However, overfitting is a serious problem in such networks. Large networks are also slow to use, making it difficult to deal with overfitting by combining the predictions of many different large neural nets at test time. Dropout is a technique for addressing this problem. The key idea is to randomly drop units (along with their connections) from the neural network during training. This prevents units from co-adapting too much. During training, dropout 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 simply using a single unthinned network that has smaller weights. This significantly reduces overfitting and gives major improvements over other regularization methods. We show 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 data sets.

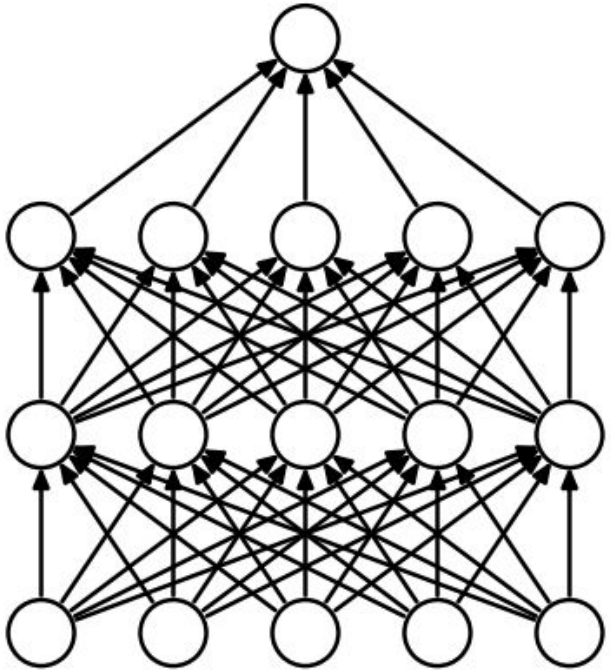
**Keywords:** neural networks, regularization, model combination, deep learning

**1. Introduction**

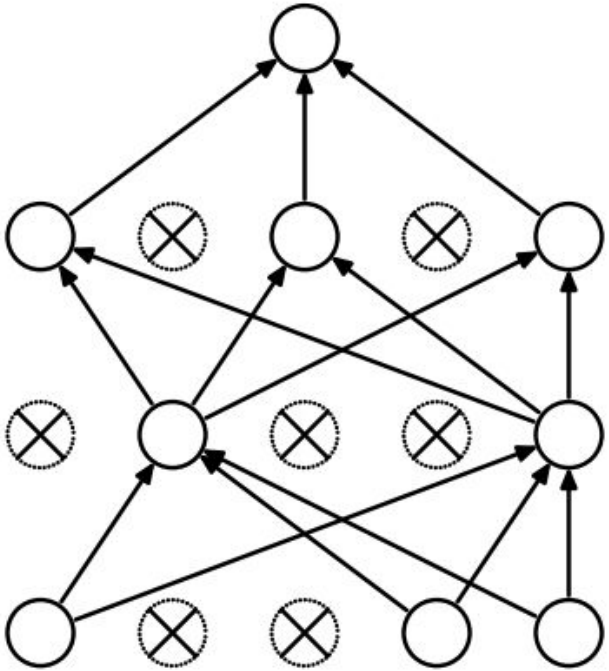
Deep neural networks contain multiple non-linear hidden layers and this makes them very expressive models that can learn very complicated relationships between their inputs and outputs. With limited training data, however, many of these complicated relationships will be the result of sampling noise, so they will exist in the training set but not in real test data even if it is drawn from the same distribution. This leads to overfitting and many methods have been developed for reducing it. These include stopping the training as soon as performance on a validation set starts to get worse, introducing weight penalties of various kinds such as L1 and L2 regularization and soft weight sharing (Nowlan and Hinton, 1992).

With unlimited computation, the best way to "regularize" a fixed-sized model is to average the predictions of all possible settings of the parameters, weighting each setting by

# Working ideas on how to train deep architectures



(a) Standard Neural Net



(b) After applying dropout.

Working ideas on how to train deep architectures

# Optimization

# Working ideas on how to train deep architectures

Training Deep Neural Networks is complicated by the fact that the distribution of each layer's inputs changes during training, as the parameters of the previous layers change. This slows down the training by requiring lower learning rates and careful parameter initialization, and makes it notoriously hard to train models with saturating nonlinearities. We refer to this phenomenon as *internal covariate shift*, and address the problem by normalizing layer inputs. Our method draws its strength from making normalization a part of the model architecture and performing the normalization *for each training mini-batch*. Batch Normalization allows us to use much higher learning rates and be less careful about initialization, and in some cases eliminates the need for Dropout. Applied to a state-of-the-art image classification model, Batch Normalization achieves the same accuracy with 14 times fewer training steps, and beats the original model by a significant margin. Using an ensemble of batch-normalized networks, we improve upon the best published result on ImageNet classification: reaching 4.82% top-5 test error, exceeding the accuracy of human raters.

## *Better Optimization Conditioning (e.g. Batch Normalization)*

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Proceedings of the 32nd International Conference on Machine Learning (ICML-15) (pp. 448-456).

**Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift**

**Sergey Ioffe** IOFFE@GOOGLE.COM
**Christian Szegedy** SZEGEDY@GOOGLE.COM
Google, 1600 Amphitheatre Pkwy, Mountain View, CA 94043

**Abstract**

Training Deep Neural Networks is complicated by the fact that the distribution of each layer's inputs changes during training, as the parameters of the previous layers change. This slows down the training by requiring lower learning rates and careful parameter initialization, and makes it notoriously hard to train models with saturating nonlinearities. We refer to this phenomenon as *internal covariate shift*, and address the problem by normalizing layer inputs. Our method draws its strength from making normalization a part of the model architecture and performing the normalization *for each training mini-batch*. Batch Normalization allows us to use much higher learning rates and be less careful about initialization, and in some cases eliminates the need for Dropout. Applied to a state-of-the-art image classification model, Batch Normalization achieves the same accuracy with 14 times fewer training steps, and beats the original model by a significant margin. Using an ensemble of batch-normalized networks, we improve upon the best published result on ImageNet classification: reaching 4.82% top-5 test error, exceeding the accuracy of human raters.

**1. Introduction**

Deep learning has dramatically advanced the state of the art in vision, speech, and many other areas. Stochastic gradient descent (SGD) has proved to be an effective way of training deep networks, and SGD variants such as momentum (Sutskever et al., 2013) and Adagrad (Duchi et al., 2011) have been used to achieve state of the art performance. SGD optimizes the parameters $\Theta$ of the network, so as to minimize the loss

$$\Theta = \arg\min_{\Theta} \frac{1}{N} \sum_{i=1}^{N} \ell(x_i, \Theta)$$

where $x_{1\ldots N}$ is the training data set. With SGD, the training proceeds in steps, at each step considering a *mini-batch* $x_{1\ldots m}$ of size $m$. Using mini-batches of examples, as opposed to one example at a time, is helpful in several ways. First, the gradient of the loss over a mini-batch $\frac{1}{m}\sum_{i=1}^{m} \frac{\partial \ell(x_i,\Theta)}{\partial \Theta}$ is an estimate of the gradient over the training set, whose quality improves as the batch size increases. Second, computation over a mini-batch can be more efficient than $m$ computations for individual examples on modern computing platforms.

While stochastic gradient is simple and effective, it requires careful tuning of the model hyper-parameters, specifically the learning rate and the initial parameter values. The training is complicated by the fact that the inputs to each layer are affected by the parameters of all preceding layers – so that small changes to the network parameters amplify as the network becomes deeper.

The change in the distributions of layers' inputs presents a problem because the layers need to continuously adapt to the new distribution. When the input distribution to a learning system changes, it is said to experience *covariate shift* (Shimodaira, 2000). This is typically handled via domain adaptation (Jiang, 2008). However, the notion of covariate shift can be extended beyond the learning system as a whole, to apply to its parts, such as a sub-network or a layer. Consider a network computing

$$\ell = F_2(F_1(u, \Theta_1), \Theta_2)$$

where $F_1$ and $F_2$ are arbitrary transformations, and the parameters $\Theta_1, \Theta_2$ are to be learned so as to minimize the loss $\ell$. Learning $\Theta_2$ can be viewed as if the inputs $x = F_1(u, \Theta_1)$ are fed into the sub-network

$$\ell = F_2(x, \Theta_2).$$

*Proceedings of the 32nd International Conference on Machine Learning*, Lille, France, 2015. JMLR: W&CP volume 37. 

# Working ideas on how to train deep architectures

**Input:** Values of $x$ over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: $\gamma$, $\beta$

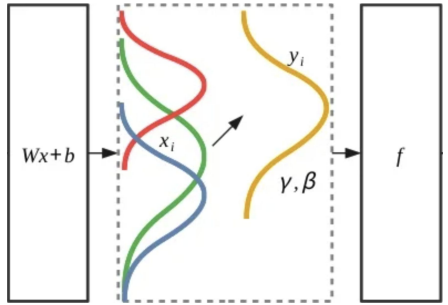
**Output:** $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

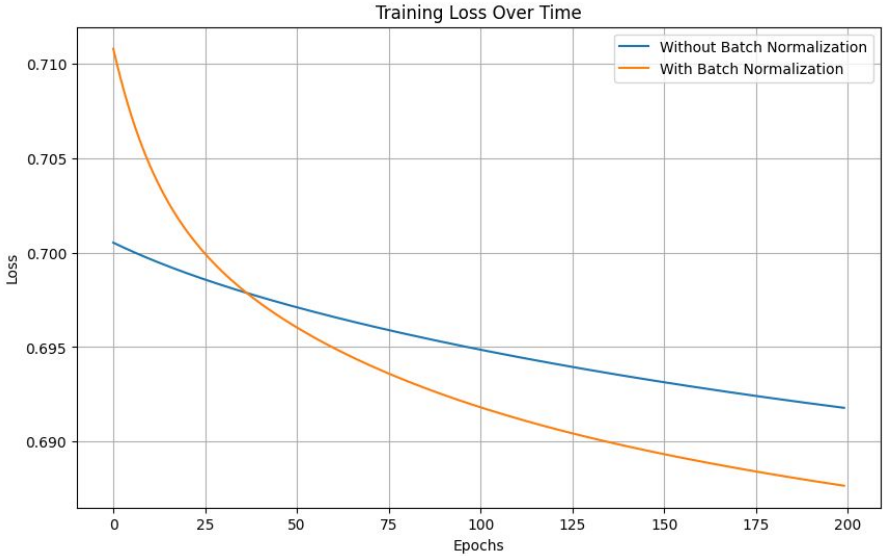
$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$



Working ideas on how to train deep architectures

# Architectures

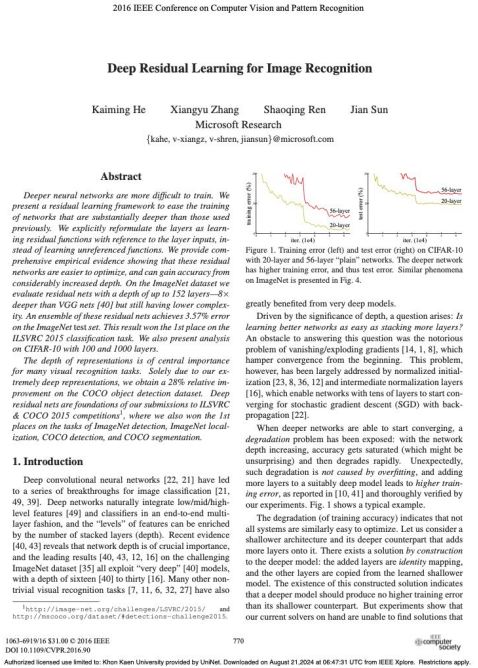
# Working ideas on how to train deep architectures

*Deeper neural networks are more difficult to train. We present a residual learning framework to ease the training of networks that are substantially deeper than those used previously. We explicitly reformulate the layers as learning residual functions with reference to the layer inputs, instead of learning unreferenced functions. We provide comprehensive empirical evidence showing that these residual networks are easier to optimize, and can gain accuracy from considerably increased depth. On the ImageNet dataset we evaluate residual nets with a depth of up to 152 layers—8× deeper than VGG nets [41] but still having lower complexity. An ensemble of these residual nets achieves 3.57% error on the ImageNet test set. This result won the 1st place on the ILSVRC 2015 classification task. We also present analysis on CIFAR-10 with 100 and 1000 layers.*

*The depth of representations is of central importance for many visual recognition tasks. Solely due to our extremely deep representations, we obtain a 28% relative improvement on the COCO object detection dataset. Deep residual nets are foundations of our submissions to ILSVRC & COCO 2015 competitions[1], where we also won the 1st places on the tasks of ImageNet detection, ImageNet localization, COCO detection, and COCO segmentation.*

## *Better neural architectures (e.g. ResNet)*

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (pp. 770-778).



2016 IEEE Conference on Computer Vision and Pattern Recognition

**Deep Residual Learning for Image Recognition**

Kaiming He Xiangyu Zhang Shaoqing Ren Jian Sun
Microsoft Research
{kahe, v-xiangz, v-shren, jiansun}@microsoft.com

**Abstract**

*Deeper neural networks are more difficult to train. We present a residual learning framework to ease the training of networks that are substantially deeper than those used previously. We explicitly reformulate the layers as learning residual functions with reference to the layer inputs, instead of learning unreferenced functions. We provide comprehensive empirical evidence showing that these residual networks are easier to optimize, and can gain accuracy from considerably increased depth. On the ImageNet dataset we evaluate residual nets with a depth of up to 152 layers—8× deeper than VGG nets [40] but still having lower complexity. An ensemble of these residual nets achieves 3.57% error on the ImageNet test set. This result won the 1st place on the ILSVRC 2015 classification task. We also present analysis on CIFAR-10 with 100 and 1000 layers.*

*The depth of representations is of central importance for many visual recognition tasks. Solely due to our extremely deep representations, we obtain a 28% relative improvement on the COCO object detection dataset. Deep residual nets are foundations of our submissions to ILSVRC & COCO 2015 competitions[1], where we also won the 1st places on the tasks of ImageNet detection, ImageNet localization, COCO detection, and COCO segmentation.*

**1. Introduction**

Deep convolutional neural networks [22, 21] have led to a series of breakthroughs for image classification [21, 49, 39]. Deep networks naturally integrate low/mid/high-level features [49] and classifiers in an end-to-end multi-layer fashion, and the "levels" of features can be enriched by the number of stacked layers (depth). Recent evidence [40, 43] reveals that network depth is of crucial importance, and the leading results [40, 43, 12, 16] on the challenging ImageNet dataset [35] all exploit "very deep" [40] models, with a depth of sixteen [40] to thirty [16]. Many other non-trivial visual recognition tasks [7, 11, 6, 32, 27] have also greatly benefited from very deep models.

Driven by the significance of depth, a question arises: *Is learning better networks as easy as stacking more layers?* An obstacle to answering this question was the notorious problem of vanishing/exploding gradients [14, 1, 8], which hamper convergence from the beginning. This problem, however, has been largely addressed by normalized initialization [23, 8, 36, 12] and intermediate normalization layers [16], which enable networks with tens of layers to start converging for stochastic gradient descent (SGD) with back-propagation [22].

When deeper networks are able to start converging, a *degradation* problem has been exposed: with the network depth increasing, accuracy gets saturated (which might be unsurprising) and then degrades rapidly. Unexpectedly, such degradation is *not caused by overfitting*, and adding more layers to a suitably deep model leads to *higher training error*, as reported in [10, 41] and thoroughly verified by our experiments. Fig. 1 shows a typical example.

The degradation (of training accuracy) indicates that not all systems are similarly easy to optimize. Let us consider a shallower architecture and its deeper counterpart that adds more layers onto it. There exists a solution *by construction* to the deeper model: the added layers are *identity* mapping, and the other layers are copied from the learned shallower model. The existence of this constructed solution indicates that a deeper model should produce no higher training error than its shallower counterpart. But experiments show that our current solvers on hand are unable to find solutions that

Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

[1] http://image-net.org/challenges/LSVRC/2015/ and http://mscoco.org/dataset/#detections-challenge2015.

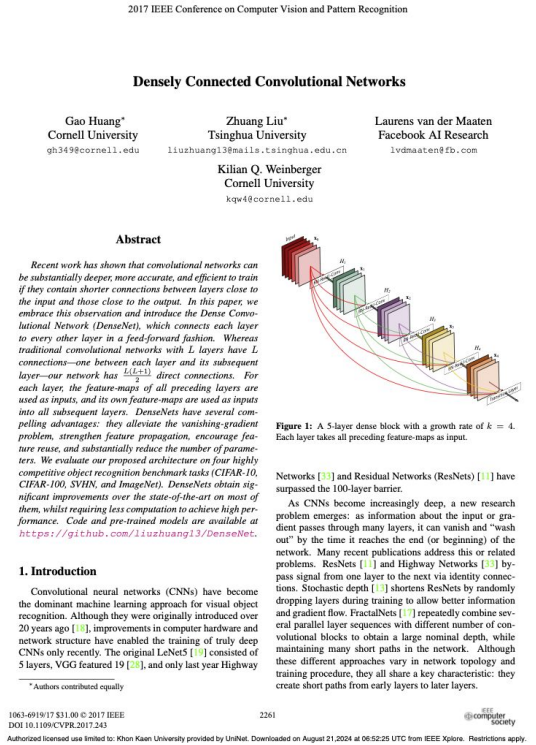
1063-6919/16 $31.00 © 2016 IEEE
DOI 10.1109/CVPR.2016.90

# Working ideas on how to train deep architectures

*Recent work has shown that convolutional networks can be substantially deeper, more accurate, and efficient to train if they contain shorter connections between layers close to the input and those close to the output. In this paper, we embrace this observation and introduce the Dense Convolutional Network (DenseNet), which connects each layer to every other layer in a feed-forward fashion. Whereas traditional convolutional networks with L layers have L connections—one between each layer and its subsequent layer—our network has $\frac{L(L+1)}{2}$ direct connections. For each layer, the feature-maps of all preceding layers are used as inputs, and its own feature-maps are used as inputs into all subsequent layers. DenseNets have several compelling advantages: they alleviate the vanishing-gradient problem, strengthen feature propagation, encourage feature reuse, and substantially reduce the number of parameters. We evaluate our proposed architecture on four highly competitive object recognition benchmark tasks (CIFAR-10, CIFAR-100, SVHN, and ImageNet). DenseNets obtain significant improvements over the state-of-the-art on most of them, whilst requiring less computation to achieve high performance. Code and pre-trained models are available at* https://github.com/liuzhuang13/DenseNet.

## *Better neural architectures (e.g. DenseNet)*

Huang, G., Liu, Z., Van Der Maaten, L., & Weinberger, K. Q. (2017). Densely connected convolutional networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (pp. 4700-4708).



2017 IEEE Conference on Computer Vision and Pattern Recognition

**Densely Connected Convolutional Networks**


Gao Huang*
Cornell University
gh349@cornell.edu

Zhuang Liu*
Tsinghua University
liuzhuang13@mails.tsinghua.edu.cn

Laurens van der Maaten
Facebook AI Research
lvdmaaten@fb.com

Kilian Q. Weinberger
Cornell University
kqw4@cornell.edu



**Abstract**

*Recent work has shown that convolutional networks can be substantially deeper, more accurate, and efficient to train if they contain shorter connections between layers close to the input and those close to the output. In this paper, we embrace this observation and introduce the Dense Convolutional Network (DenseNet), which connects each layer to every other layer in a feed-forward fashion. Whereas traditional convolutional networks with L layers have L connections—one between each layer and its subsequent layer—our network has $\frac{L(L+1)}{2}$ direct connections. For each layer, the feature-maps of all preceding layers are used as inputs, and its own feature-maps are used as inputs into all subsequent layers. DenseNets have several compelling advantages: they alleviate the vanishing-gradient problem, strengthen feature propagation, encourage feature reuse, and substantially reduce the number of parameters. We evaluate our proposed architecture on four highly competitive object recognition benchmark tasks (CIFAR-10, CIFAR-100, SVHN, and ImageNet). DenseNets obtain significant improvements over the state-of-the-art on most of them, whilst requiring less computation to achieve high performance. Code and pre-trained models are available at* https://github.com/liuzhuang13/DenseNet.


**1. Introduction**

Convolutional neural networks (CNNs) have become the dominant machine learning approach for visual object recognition. Although they were originally introduced over 20 years ago [18], improvements in computer hardware and network structure have enabled the training of truly deep CNNs only recently. The original LeNet5 [19] consisted of 5 layers, VGG featured 19 [28], and only last year Highway Networks [33] and Residual Networks (ResNets) [11] have surpassed the 100-layer barrier.

**Figure 1:** A 5-layer dense block with a growth rate of $k = 4$. Each layer takes all preceding feature-maps as input.

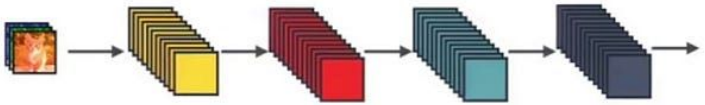
As CNNs become increasingly deep, a new research problem emerges: as information about the input or gradient passes through many layers, it can vanish and "wash out" by the time it reaches the end (or beginning) of the network. Many recent publications address this or related problems. ResNets [11] and Highway Networks [33] bypass signal from one layer to the next via identity connections. Stochastic depth [13] shortens ResNets by randomly dropping layers during training to allow better information and gradient flow. FractalNets [17] repeatedly combine several parallel layer sequences with different number of convolutional blocks to obtain a large nominal depth, while maintaining many short paths in the network. Although these different approaches vary in network topology and training procedure, they all share a key characteristic: they create short paths from early layers to later layers.

*Authors contributed equally


1063-6919/17 $31.00 © 2017 IEEE
DOI 10.1109/CVPR.2017.243

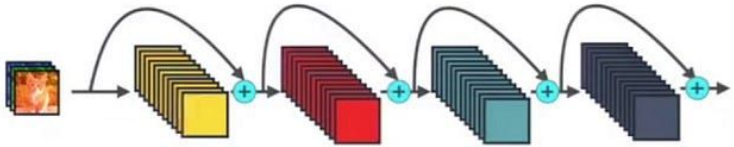
# Working ideas on how to train deep architectures

**Standard Connectivity**



Successive convolutions

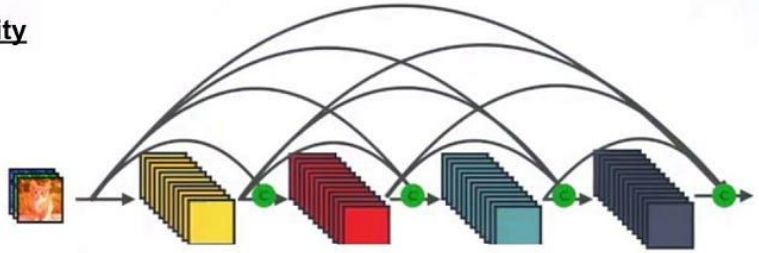
**Resnet Connectivity**



Element-wise feature summation

⊕ : Element-wise addition

**DenseNet Connectivity**



Feature concatenation

© : Channel-wise concatenation

# *Different deep learning architectures have emerged.*