Deep Learning in Biomedical Informatics

Dr. Fayyaz ul Amir Afsar Minhas
PIEAS Biomedical Informatics Research Lab
Department of Computer and Information Sciences
Pakistan Institute of Engineering & Applied Sciences
PO Nilore, Islamabad, Pakistan
http://faculty.pieas.edu.pk/fayyaz/

Lecture Plan

- What are Neural Networks?
  - Fundamentals
    - Connectionism
    - Multilayer Perceptron
    - Understanding issues in MLP
    - Making neural networks in Keras
- Why go Deep?
- Issues in deep learning
  - How to solve those issues?
- Modern practices in Deep Learning
  - Convolutional Neural Networks
  - Residual Networks
  - Generative Models
    - Auto-encoders: VAE, NAE
    - Generative Adversarial Networks
    - Recurrent Neural Networks
  - Recurrent Models: RNN, LSTM
  - Neural Networks with Stochastic Depth
  - Other architectures: Ladder, Highway, etc.
  - Transfer Learning
  - Zero-shot and One-shot learning
  - Non-Neural Deep Learning
    - Multilayer Kernel Machines
    - Convolutional Kernel Networks: https://arxiv.org/abs/1406.3332
    - When Correlation Filters Meet Convolutional Neural Networks for Visual Tracking
- Applications
  - Biomedical applications of deep learning

- **Deep learning in bioinformatics**
  - https://academic.oup.com/bib/article-abstract/doi/10.1093/bib/bbw068/2562808/Deep-learning-in-bioinformatics?redirectedFrom=fulltext
- **Deep learning for computational biology**
  - http://msb.embopress.org/content/12/7/878

Interesting codes

- skFlow: http://www.kdnuggets.com/2016/02/scikit-flow-easy-deep-learning-tensorflow-scikit-learn.html
- Trained image classification models for Keras [COVER]
  - https://github.com/fchollet/deep-learning-models
- Transfer Learning: Recognition of traffic lights [COVER] (https://medium.freecodecamp.com/recognizing-traffic-lights-with-deep-learning-23dae23287cc)
- OR Building powerful image classification models using very little data: https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html
- Building auto-encoders: https://blog.keras.io/building-autoencoders-in-keras.html [COVER]
- Deep Networks with Stochastic Depth [COVER]
- Visualization: https://blog.keras.io/how-convolutional-neural-networks-see-the-world.html [COVER]
- Time Series Prediction with LSTM Recurrent Neural Networks in Python with Keras [COVER]
  - http://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras/
- Takken from: https://github.com/fchollet/keras-resources
- Cycle GAN: https://github.com/junyanz/CycleGAN
-

Papers

- Deep Learning Papers Reading Roadmap from : https://github.com/songrotek/Deep-Learning-Papers-Reading-Roadmap
  - Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "**Imagenet classification with deep convolutional neural networks**." Advances in neural information processing systems. 2012. [pdf] **(AlexNet, Deep Learning Breakthrough)**
  - He, Kaiming, et al. "**Deep residual learning for image recognition**." arXiv preprint arXiv:1512.03385 (2015). [pdf]**(ResNet,Very very deep networks, CVPR best paper)**
  - Andrychowicz, Marcin, et al. "**Learning to learn by gradient descent by gradient descent**." arXiv preprint arXiv:1606.04474 (2016). [pdf] **(Neural Optimizer,Amazing Work)**
  - Han, Song, Huizi Mao, and William J. Dally. "**Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding**." CoRR, abs/1510.00149 2 (2015). [pdf] **(ICLR best paper, new direction to make NN running fast,DeePhi Tech Startup)**
  - Iandola, Forrest N., et al. "**SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 1MB model size**." arXiv preprint arXiv:1602.07360 (2016). [pdf] **(Also a new direction to optimize NN,DeePhi Tech Startup)**
  - Le, Quoc V. "**Building high-level features using large scale unsupervised learning**." 2013 IEEE international conference on acoustics, speech and signal processing. IEEE, 2013. [pdf] **(Milestone, Andrew Ng, Google Brain Project, Cat)**
  - Goodfellow, Ian, et al. "**Generative adversarial nets**." Advances in Neural Information Processing Systems. 2014. [pdf]**(GAN,super cool idea)**
  - Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le. "**Sequence to sequence learning with neural networks**." Advances in neural information processing systems. 2014. [pdf] **(Outstanding Work)**
  - Graves, Alex, Greg Wayne, and Ivo Danihelka. "**Neural turing machines**." arXiv preprint arXiv:1410.5401 (2014). [pdf]**(Basic Prototype of Future Computer)**
  - Silver, David, et al. "**Mastering the game of Go with deep neural networks and tree search**." Nature 529.7587 (2016): 484-489. [pdf] **(AlphaGo)**
  - Silver, Daniel L., Qiang Yang, and Lianghao Li. "**Lifelong Machine Learning Systems: Beyond Learning Algorithms**." AAAI Spring Symposium: Lifelong Machine Learning. 2013. [pdf] **(A brief discussion about lifelong learning)**
  - Hariharan, Bharath, and Ross Girshick. "**Low-shot visual object recognition**." arXiv preprint arXiv:1606.02819 (2016). [pdf]**(A step to large data)**
  - **Deep Networks with Stochastic Depth**
  - **Failures of Gradient-Based Deep Learning https://arxiv.org/abs/1703.07950**
  - **Understanding deep learning requires rethinking generalization: https://openreview.net/forum?id=Sy8gdB9xx&noteId=Sy8gdB9xx**
- https://github.com/sbrugman/deep-learning-papers

Papers

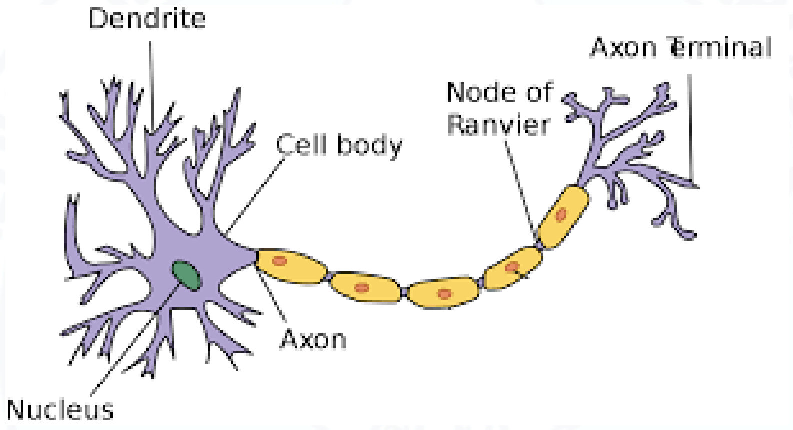
- Failures of Gradient-Based Deep Learning (https://arxiv.org/abs/1703.07950) [COVER]
- Efficient Processing of Deep Neural Networks: A Tutorial and Survey (https://arxiv.org/abs/1703.09039) [COVER]
- Deeo Networks with Stochastic Depth (https://arxiv.org/abs/1603.09382, https://github.com/dblN/stochastic_depth_keras)
- Deep residual learning for image recognition (2016), K. He et al. (http://arxiv.org/pdf/1512.03385)
- CNN Application: http://www.kdnuggets.com/2017/04/medical-image-analysis-deep-learning-part-2.html [COVER]
- The Shattered Gradients Problem: If resnets are the answer, then what is the question?
- Scaling the Scattering Transform: Deep Hybrid Networks : We use the scattering network as a generic and fixed initialization of the first layers of a supervised hybrid deep network. We show that early layers do not necessarily need to be learned, providing the best results to-date with pre-defined representations while being competitive with Deep CNNs. [COVER]

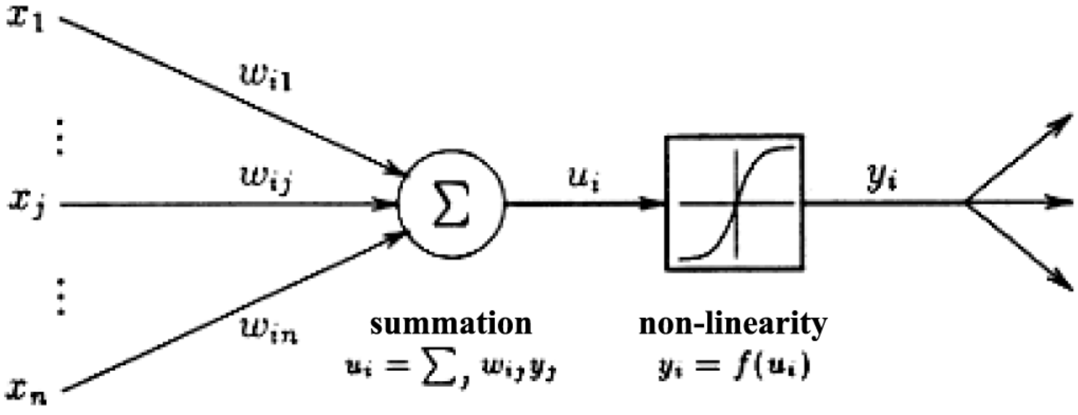
- Applications (summarized)
  - https://github.com/gokceneraslan/awesome-deepbio
- FractalNet: Ultra-Deep Neural Networks without Residuals (https://arxiv.org/abs/1605.07648)
- How to Scale Up Kernel Methods to Be As Good As Deep Neural Nets
- Deep Semi-Random Features for Nonlinear Function Approximation
- Steps toward deep kernel methods from infinite neural networks
- Progressive Neural Networks
- Learning Infinite–Layer Networks. Beyond the Kernel Trick.
- Gradient Descent Learns Linear Dynamical Systems
- Unsupervised representation learning with deep convolutional generative adversarial networks (https://arxiv.org/pdf/1511.06434v2)
  - GAN-Zoo: https://deephunt.in/the-gan-zoo-79597dc8c347
- Highway Networks (https://arxiv.org/abs/1505.00387) : Highway networks with hundreds of layers can be trained directly using stochastic gradient descent and with a variety of activation functions, opening up the possibility of studying extremely deep and efficient architectures.
- Lensless Imaging with Compressive Ultrafast Sensing
- NIPS 2016 Tutorial: Generative Adversarial Networks Ian Goodfellow OpenAI, ian@openai.com Abstract, C:/Users/afsar/Downloads\1701.00160v1.pdf
- Image De-raining Using a Conditional Generative Adversarial Network
- On the Origin of Deep Learning
- Deep Semi-Random Features for Nonlinear Function Approximation
- On-line Learning with Abstention
- One-ShotImitationLearning
- Comparisons of Sequence Labeling Algorithms and Extensions

- **Style Transfoer:** Real-time style transfer
- **Image Analogies:** Image analogies
- **Deep Dreams**
- **Deep Jazz: https://github.com/jisungk/deepjazz**
- **Image Colorization**
- **Caption Generation: https://arxiv.org/pdf/1411.4555.pdf**
- **Lip Reading: LipNet: End-to-End Sentence-level Lipreading https://arxiv.org/abs/1611.01599**
- **2012-07** | Deep architectures for protein contact map prediction | *Pietro Di Lena, Ken Nagata and Pierre Baldi*Bioinformatics
- **2016-11** | Accurate De Novo Prediction of Protein Contact Map by Ultra-Deep Learning Model | *Sheng Wang, Siqi Sun, Zhen Li, Renyu Zhang, Jinbo Xu* | bioRxiv
- Dermatologist-level classification of skin cancer with deep neural networks | *Andre Esteva, Brett Kuprel, Roberto A. Novoa, Justin Ko, Susan M. Swetter, Helen M. Blau & Sebastian Thrun* | Nature
- Deep Recurrent Neural Network for Protein Function Prediction from Sequence | *Xueliang Leon Liu* | bioRxiv
- Deep Motif Dashboard: Visualizing and Understanding Genomic Sequences Using Deep Neural Networks | *Lanchantin, Jack, Ritambhara Singh, Beilun Wang, and Yanjun Qi* | Pacific Symposium on Biocomputing 2017
- Convolutional neural network architectures for predicting DNA–protein binding | *Haoyang Zeng, Matthew D. Edwards, Ge Liu and David K. Gifford* | Bioinformatics | code
- Predicting protein residue–residue contacts using deep networks and boosting | *Jesse Eickholt and Jianlin Cheng* | Bioinformatics
- https://lyrebird.ai/demo
- Medical Image Analysis with Deep Learning: http://www.kdnuggets.com/2017/04/medical-image-analysis-deep-learning-part-2.html

Neural Networks

- An abstraction of the biological neuron

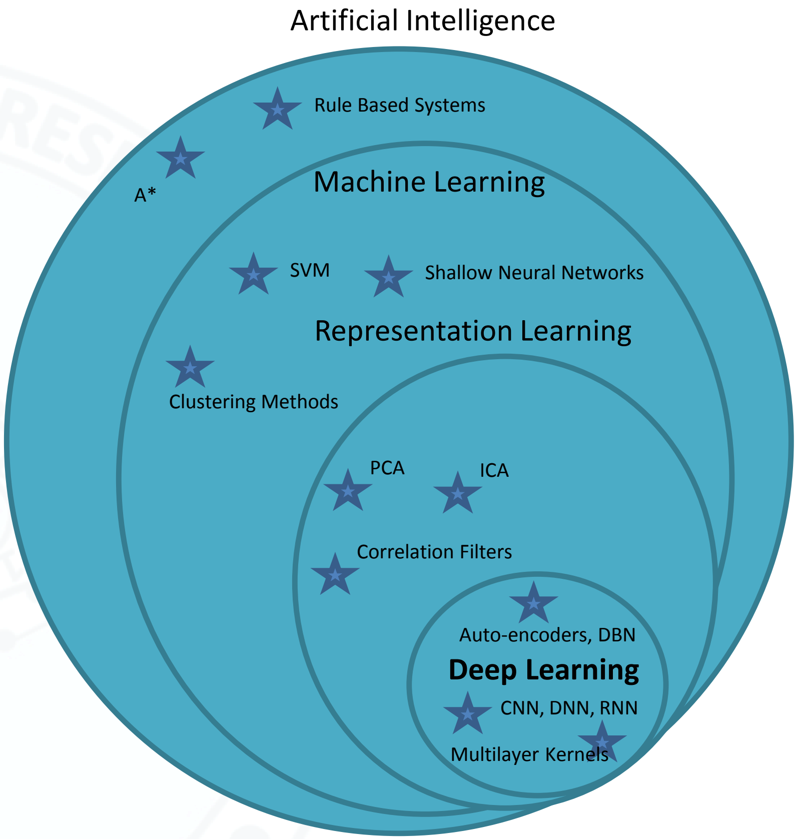




(b)

Deep Learning

- Traditional machine learning focuses on feature engineering
- Deep learning is a branch of machine learning
  - That uses a cascade of many layers of non-linear units for feature extraction and transformation
  - Based on “automatic” learning of multiple levels of features or representations of the data
- Re-branding of neural networks!
  - Massive growth in efficient algorithms for solving AI challenges!
- Many Applications in Biomedical Informatics



AlphaGo

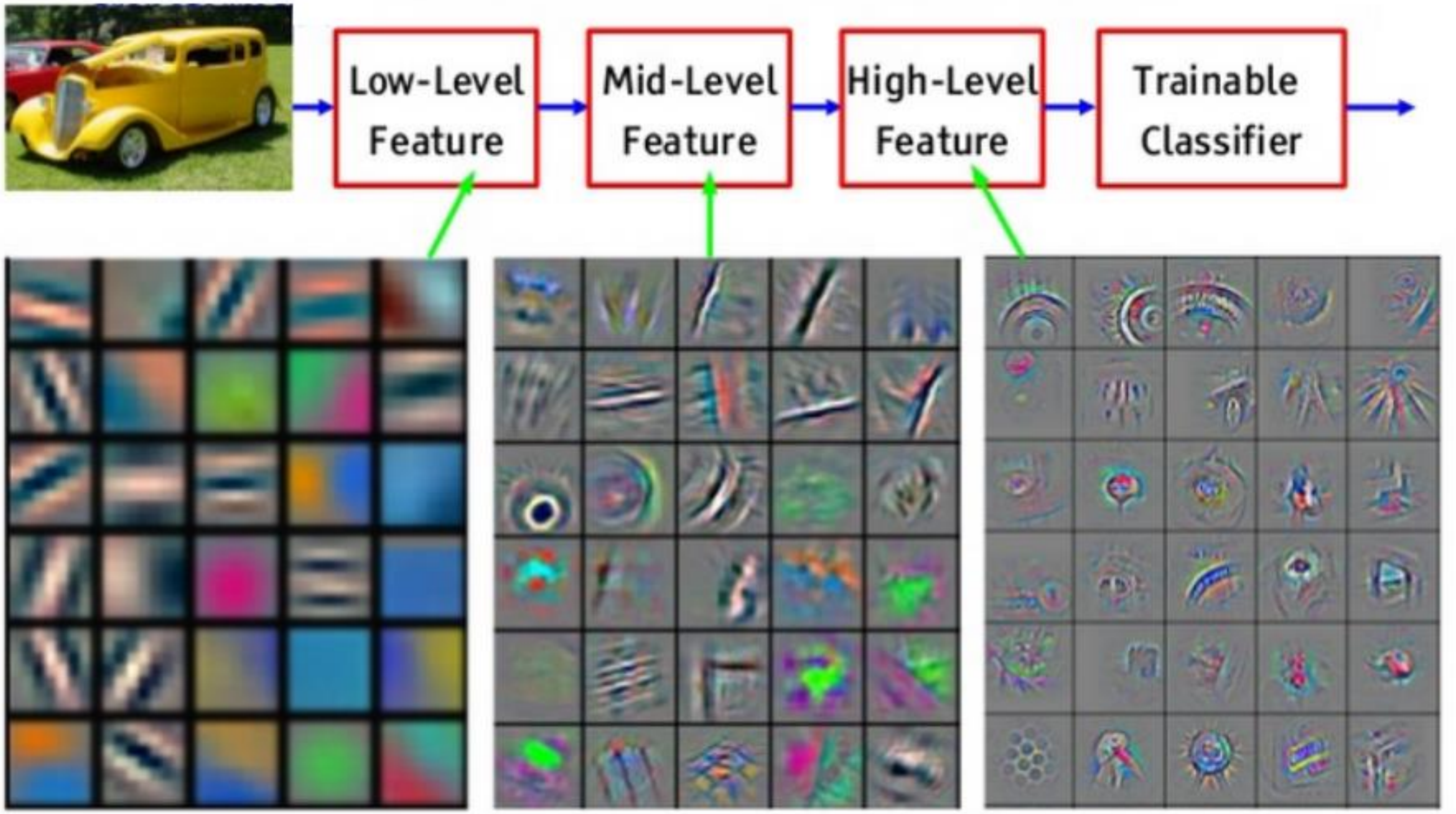
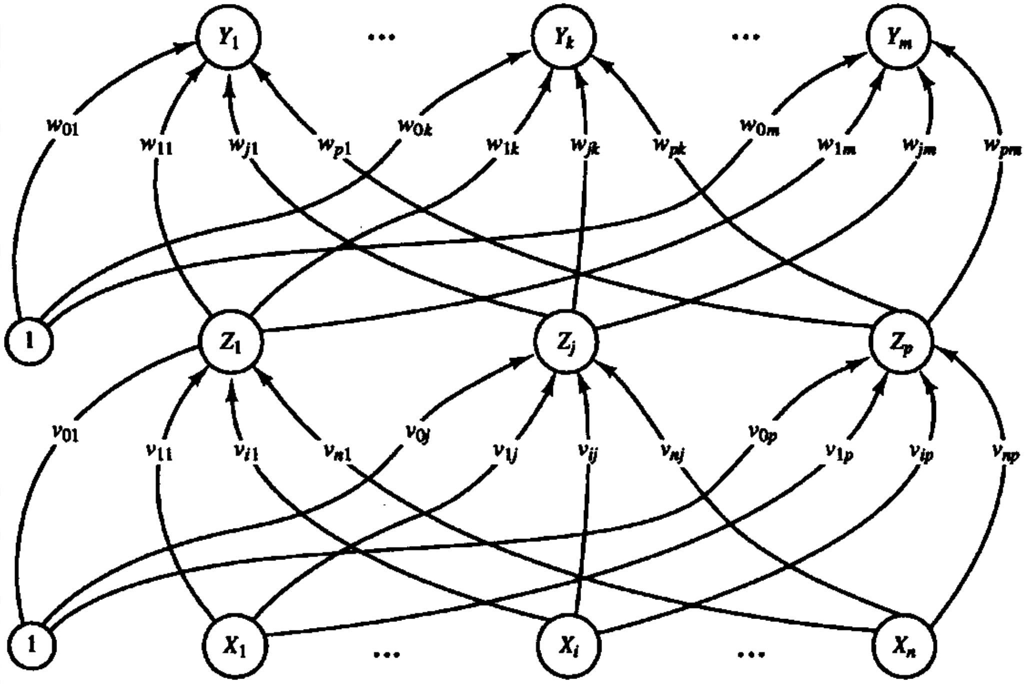


Figure: Picture from Yann LeCun's tutorial, based on Zeiler and Fergus [2014].

Multilayer Perceptron

- Consists of multiple layers of neurons
- Layers of units other than the input and output are called hidden units
- Unidirectional weight connections and biases
- Activation functions
  - Use of activation functions
    - Sigmoidal activations
    - Nonlinear Operation: Ability to solve practical problems
    - Differentiable: Makes theoretical assessment easier
    - Derivative can be expressed in terms of functions themselves: Computational Efficiency
  - Activation function is the same for all neurons in the same layer
  - Input layer just passes on the signal without processing (linear operation)



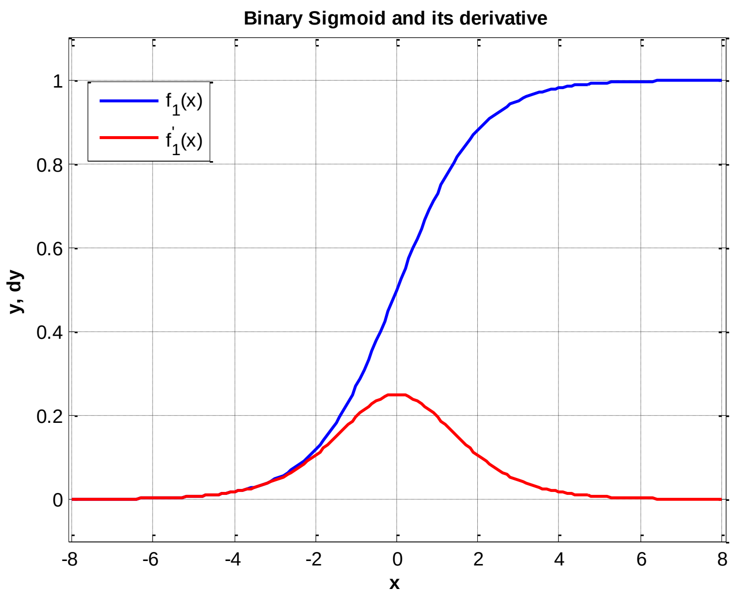
$$z_j = f\left(z\_in_j\right)$$

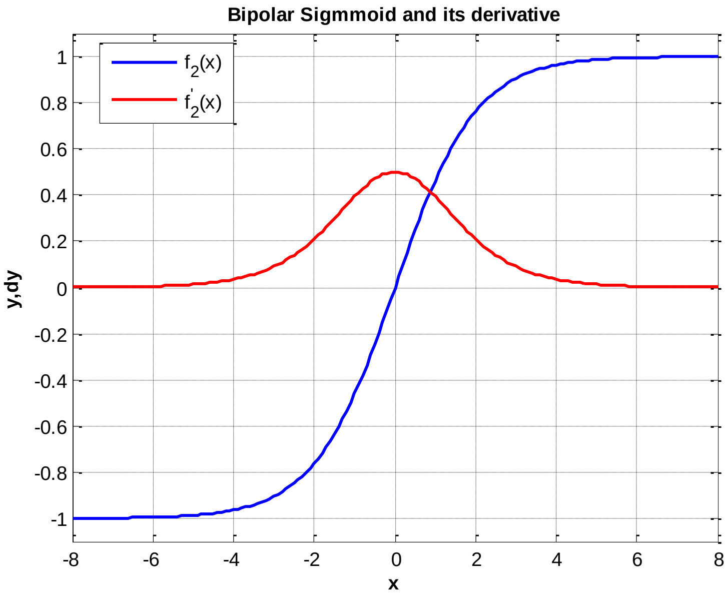
$$z\_in_j = \sum_{i=0}^{n} x_i v_{ij}, \qquad x_0 = 1, \qquad j = 1 \ldots p$$

$$y_k = f\left(y\_in_k\right)$$

$$y\_in_k = \sum_{j=0}^{p} z_j w_{jk}, \qquad z_0 = 1, \qquad k = 1 \ldots m$$

Architecture: Activation functions





$$f_1(x) = \frac{1}{1 + \exp(-x)}$$

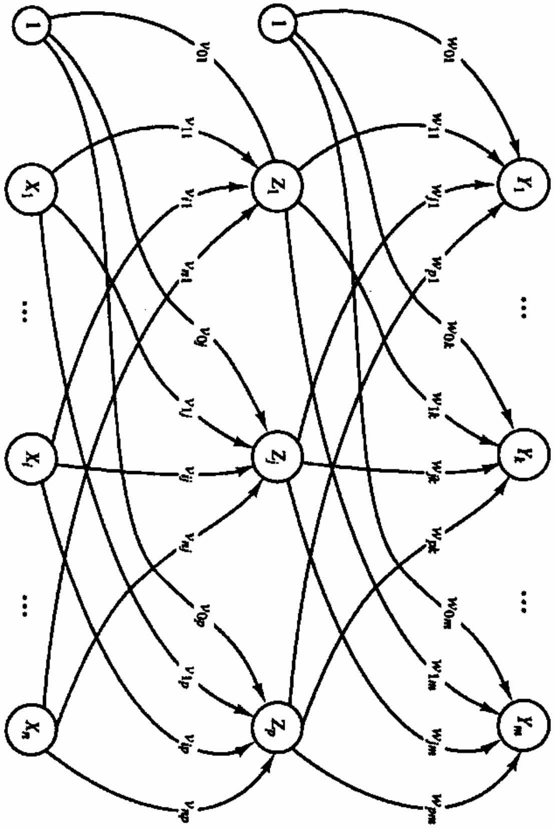
$$f'_1(x) = f_1(x)[1 - f_1(x)]$$

$$f_2(x) = \frac{2}{1 + \exp(-x)} - 1,$$

$$f'_2(x) = \frac{1}{2}[1 + f_2(x)][1 - f_2(x)].$$

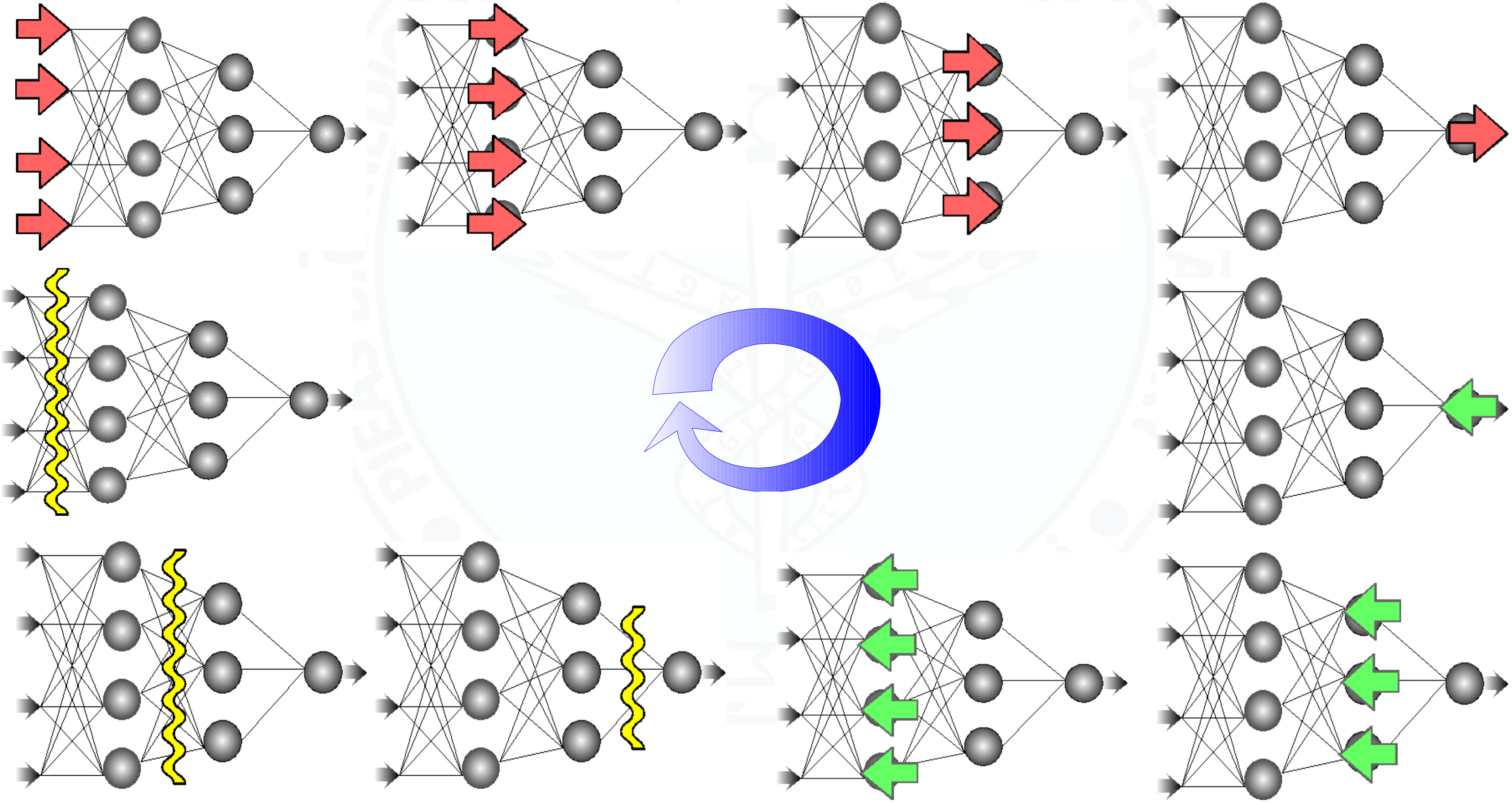
Training

- During training we are presented with input patterns and their targets
- At the output layer we can compute the error between the targets and actual output and use it to compute weight updates through the Delta Rule
- But the Error cannot be calculated at the hidden input as their targets are not known
- Therefore we propagate the error at the output units to the hidden units to find the required weight changes (Backpropagation)
- 3 Stages
  - Feed-forward of the input training pattern
  - Calculation and Backpropagation of the associated error
  - Weight Adjustment
- Based on minimization of SSE (Sum of Square Errors)



Backpropagation training cycle

Feed forward



Weight Update

Backpropagation

Proof for the Learning Rule

$$E = .5 \sum_{k} [t_k - y_k]^2.$$

By the Chain rule, we have:

$$\frac{\partial E}{\partial w_{JK}} = \frac{\partial}{\partial w_{JK}} .5 \sum_{k} [t_k - y_k]^2$$

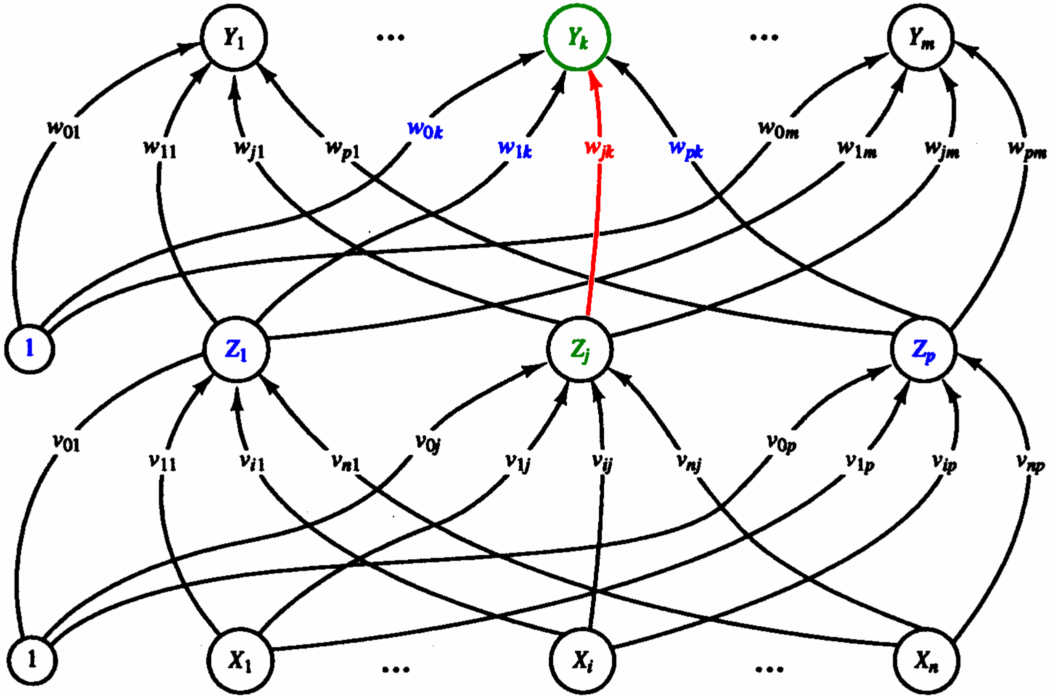
Change in w_{jk} affects only Y_k

$$= \frac{\partial}{\partial w_{JK}} .5[t_K - f(y\_in_K)]^2$$

$$= -[t_K - y_K] \frac{\partial}{\partial w_{JK}} f(y\_in_K)$$

$$= -[t_K - y_K] f'(y\_in_K) \frac{\partial}{\partial w_{JK}} (y\_in_K)$$

$$= -[t_K - y_K] f'(y\_in_K) z_J.$$



Define:

$$\delta_K = [t_K - y_K] f'(y\_in_K).$$

$$\Delta w_{jk} = -\alpha \frac{\partial E}{\partial w_{jk}}$$

$$= \alpha[t_k - y_k]f'(y\_in_k)z_j$$

$$= \alpha \delta_k z_j;$$

Use of Gradient Descent Minimization

Proof for the Learning Rule…

For the weight from x_i to z_j:

$$\frac{\partial E}{\partial v_{IJ}} = -\sum_k [t_k - y_k] \frac{\partial}{\partial v_{IJ}} y_k$$

Change in v_{ij} affects all $Y_{1..m}$

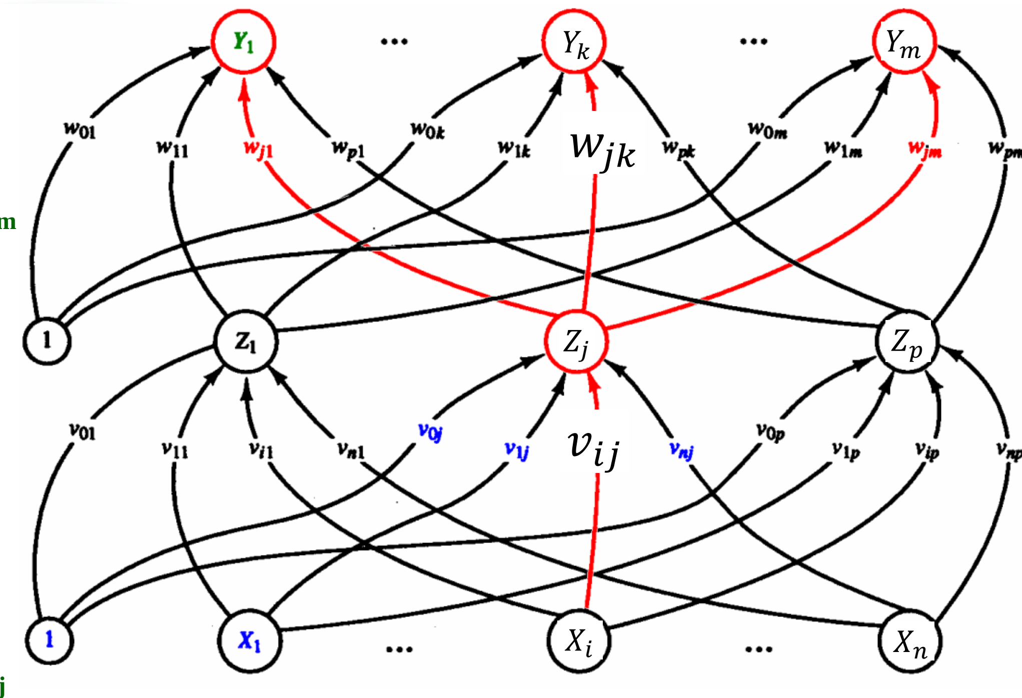
$$= -\sum_k [t_k - y_k]\, f'(y\_in_k) \frac{\partial}{\partial v_{IJ}} y\_in_k$$

$$= -\sum_k \delta_k \frac{\partial}{\partial v_{IJ}} y\_in_k$$

$$= -\sum_k \delta_k w_{Jk} \frac{\partial}{\partial v_{IJ}} z_J$$

Change in v_{ij} affects only z_j

$$= -\sum_k \delta_k w_{Jk}\, f'(z\_in_J)[x_I].$$



Define:

$$\delta_J = \sum_k \delta_k w_{Jk} f'(z\_in_J)$$

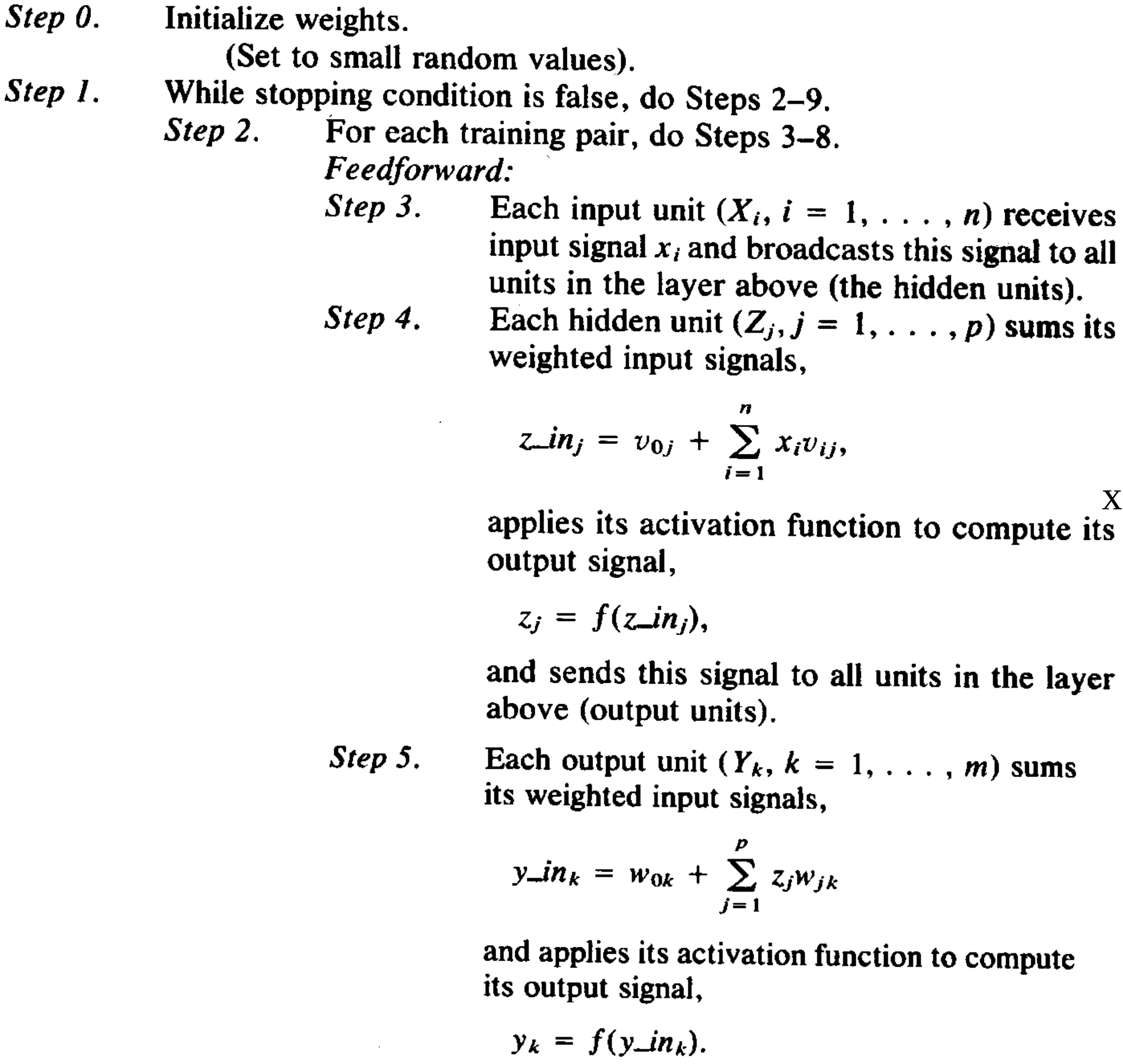
$$\Delta v_{ij} = -\alpha \frac{\partial E}{\partial v_{ij}}$$

$$= \alpha f'(z\_in_j) x_i \sum_k \delta_k w_{jk},$$

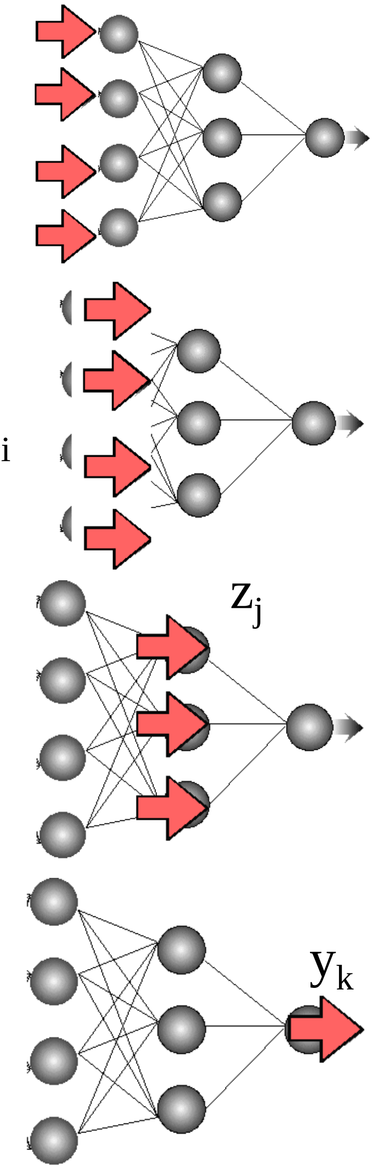
$$= \alpha \delta_j x_i.$$

Use of Gradient Descent Minimization

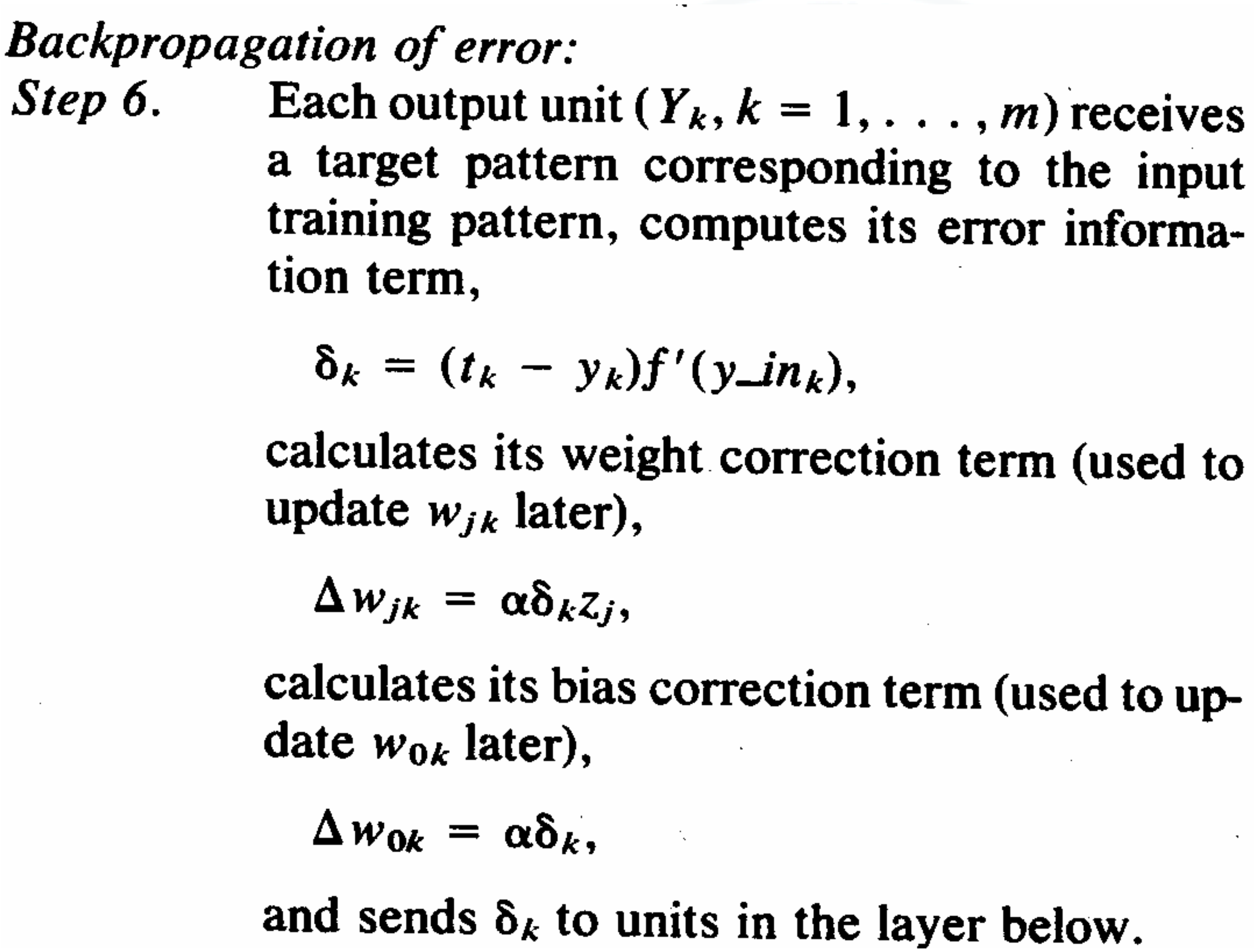
Training Algorithm



```
Step 0. Initialize weights.
            (Set to small random values).
Step 1. While stopping condition is false, do Steps 2–9.
        Step 2. For each training pair, do Steps 3–8.
                Feedforward:
                Step 3. Each input unit ($X_i, i = 1, \dots, n$) receives
                        input signal $x_i$ and broadcasts this signal to all
                        units in the layer above (the hidden units).
                Step 4. Each hidden unit ($Z_j, j = 1, \dots, p$) sums its
                        weighted input signals,
                            $z\_in_j = v_{0j} + \sum_{i=1}^{n} x_i v_{ij}$,
                        applies its activation function to compute its
                        output signal,
                            $z_j = f(z\_in_j)$,
                        and sends this signal to all units in the layer
                        above (output units).
                Step 5. Each output unit ($Y_k, k = 1, \dots, m$) sums
                        its weighted input signals,
                            $y\_in_k = w_{0k} + \sum_{j=1}^{p} z_j w_{jk}$
                        and applies its activation function to compute
                        its output signal,
                            $y_k = f(y\_in_k)$.
```



Training Algorithm…



Backpropagation of error:

Step 6. Each output unit ($Y_k, k = 1, \ldots, m$) receives a target pattern corresponding to the input training pattern, computes its error information term,

$$\delta_k = (t_k - y_k)f'(y\_in_k),$$

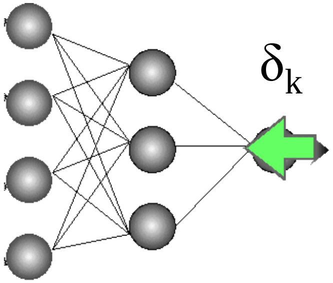
calculates its weight correction term (used to update w_{jk} later),

$$\Delta w_{jk} = \alpha\delta_k z_j,$$

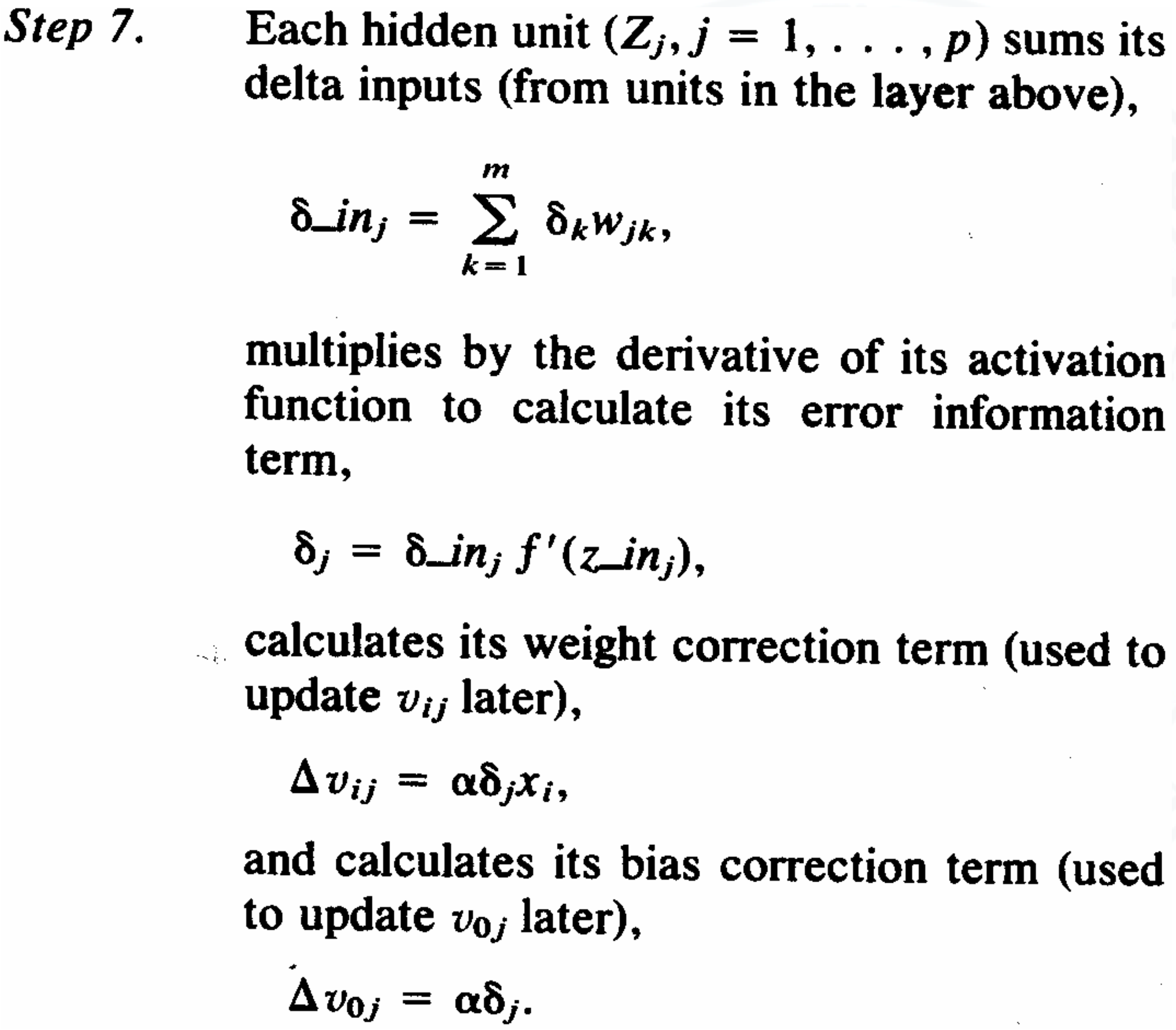
calculates its bias correction term (used to update w_{0k} later),

$$\Delta w_{0k} = \alpha\delta_k,$$

and sends δ_k to units in the layer below.



Training Algorithm…



Step 7. Each hidden unit ($Z_j, j = 1, \ldots, p$) sums its delta inputs (from units in the layer above),

$$\delta\_in_j = \sum_{k=1}^{m} \delta_k w_{jk},$$

multiplies by the derivative of its activation function to calculate its error information term,

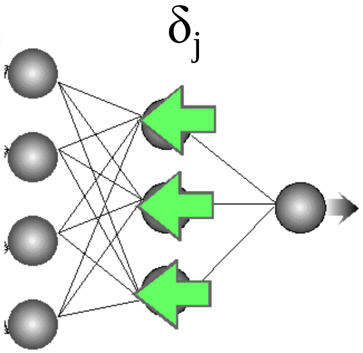
$$\delta_j = \delta\_in_j \, f'(z\_in_j),$$

calculates its weight correction term (used to update v_{ij} later),

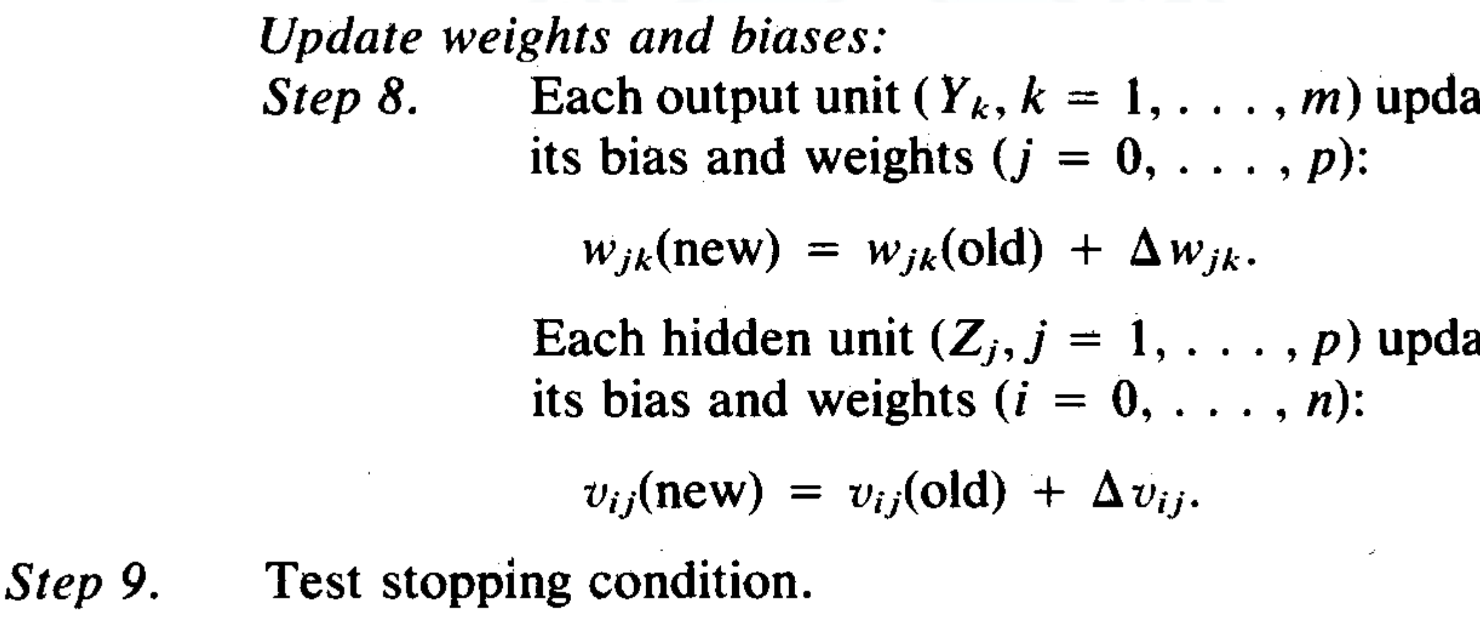
$$\Delta v_{ij} = \alpha \delta_j x_i,$$

and calculates its bias correction term (used to update v_{0j} later),

$$\Delta v_{0j} = \alpha \delta_j.$$



Training Algorithm…



Update weights and biases:

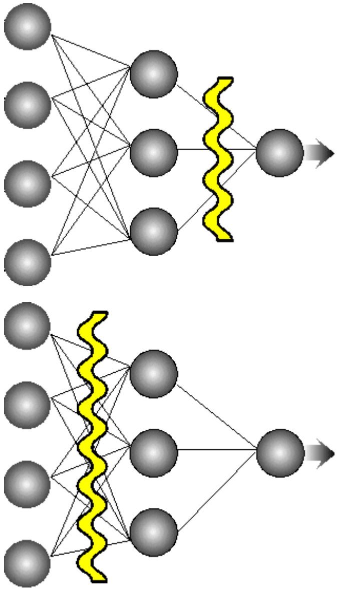
Step 8. Each output unit ($Y_k, k = 1, \ldots, m$) updates its bias and weights ($j = 0, \ldots, p$):

$$w_{jk}(\text{new}) = w_{jk}(\text{old}) + \Delta w_{jk}.$$

Each hidden unit ($Z_j, j = 1, \ldots, p$) updates its bias and weights ($i = 0, \ldots, n$):

$$v_{ij}(\text{new}) = v_{ij}(\text{old}) + \Delta v_{ij}.$$

Step 9. Test stopping condition.



Optimization in minibatches

- We can do a full scale optimization across all examples or take a few examples at a time to determine the gradients
  - Mini-batches

Things to note

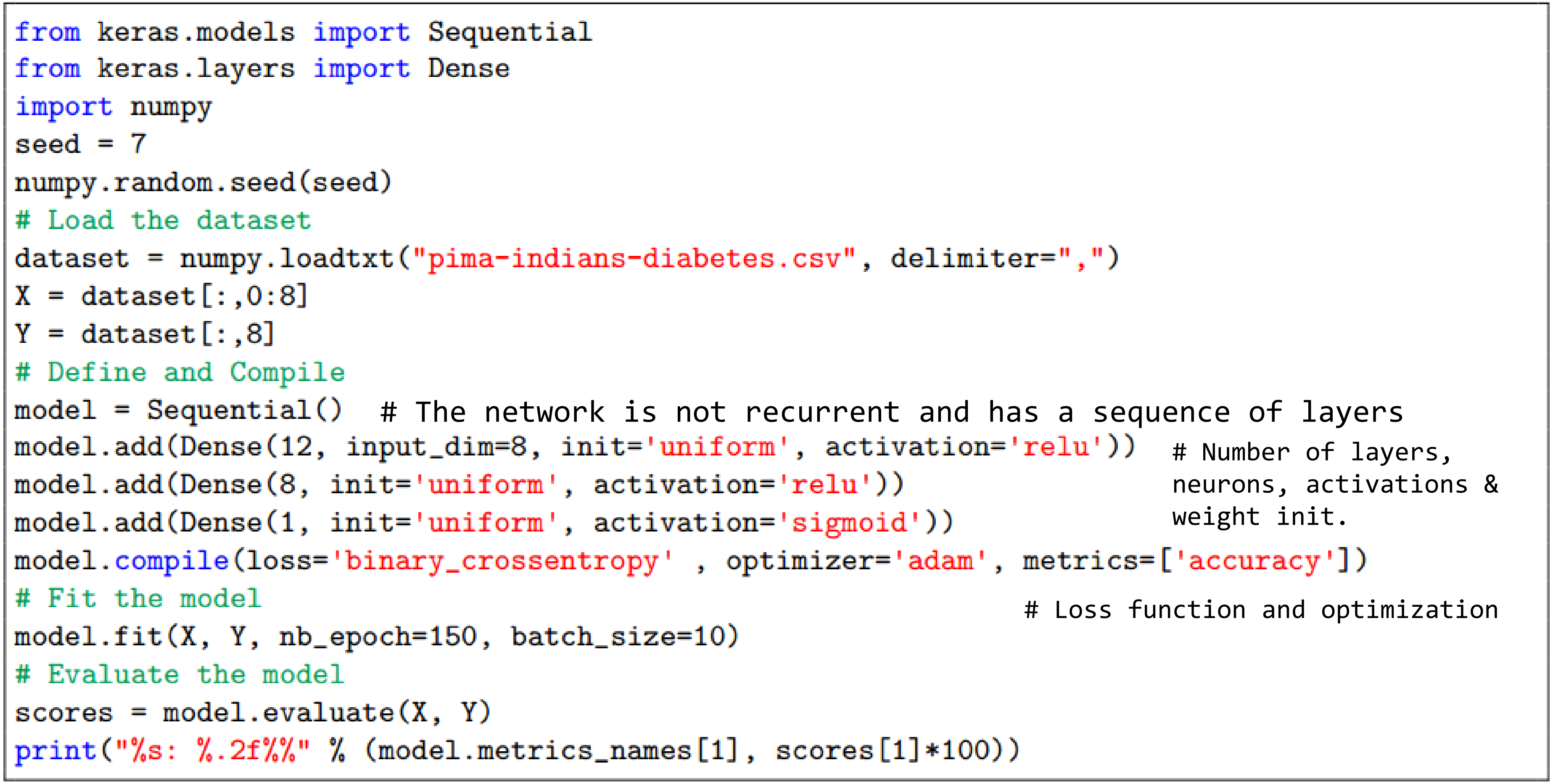
- A large number of derivatives will be computed
  - For every input
  - For every weight at every layer
- The update is dependent upon
    - The activation function value
    - The input
    - The target
    - The current weight value
    - The value of the derivative of the activation function of the current layer
    - The value of the derivative of the activation function of the following layers
    - The derivatives are multiplied: Vanishing gradients!
    - The error value

Parameter Selection

- A MLP has a large number of parameters
  - Number of Neurons in Each Layer
  - Number of Layers
  - Activation Function for each neuron: ReLU, logsig…
  - Layer Connectivity: Dense, Dropout…
  - Objective function
    - Loss Function: MSE, Entropy, Hinge loss, …
    - Regularization: L1, L2…
  - Optimization Method
    - SGD, ADAM, RMSProp, LM …
  - Parameters for the Optimization method
    - Weight initialization
    - Momentum, weight decay, etc.

Implementation

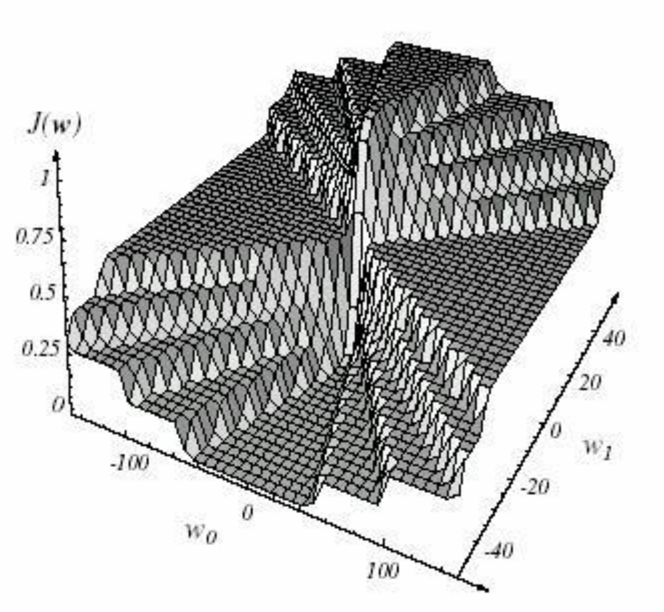
- Keras! https://keras.io/



```
from keras.models import Sequential
from keras.layers import Dense
import numpy
seed = 7
numpy.random.seed(seed)
# Load the dataset
dataset = numpy.loadtxt("pima-indians-diabetes.csv", delimiter=",")
X = dataset[:,0:8]
Y = dataset[:,8]
# Define and Compile
model = Sequential()  # The network is not recurrent and has a sequence of layers
model.add(Dense(12, input_dim=8, init='uniform', activation='relu'))  # Number of layers,
model.add(Dense(8, init='uniform', activation='relu'))                 neurons, activations &
model.add(Dense(1, init='uniform', activation='sigmoid'))              weight init.
model.compile(loss='binary_crossentropy' , optimizer='adam', metrics=['accuracy'])
# Fit the model                                                  # Loss function and optimization
model.fit(X, Y, nb_epoch=150, batch_size=10)
# Evaluate the model
scores = model.evaluate(X, Y)
print("%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
```

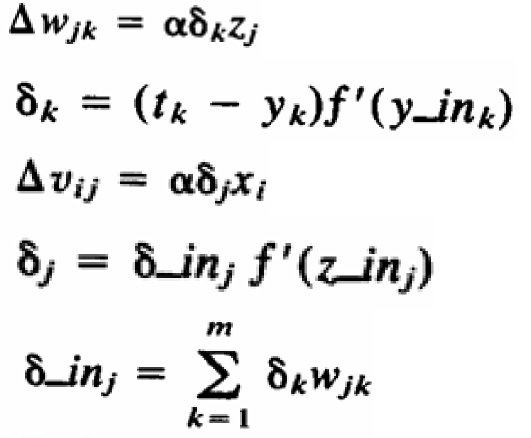
Issues with Neural Networks with non-linear activations

- Unlike an SVM, which has a single global optimum due to its convex loss function, the error surface of a neural network is not as smooth
- This complicates the optimization
- A number of “tricks” are used to make the neural network learn



How to improve MLP?

- Don’t let the network stop learning prematurely!
  - For example: Don’t let the neurons saturate!
    - If the input or the gradient goes to zero, the learning stops!



$$\Delta w_{jk} = \alpha \delta_k z_j$$

$$\delta_k = (t_k - y_k) f'(y\_in_k)$$

$$\Delta v_{ij} = \alpha \delta_j x_i$$

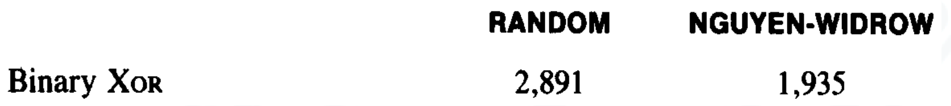
$$\delta_j = \delta\_in_j \, f'(z\_in_j)$$

$$\delta\_in_j = \sum_{k=1}^{m} \delta_k w_{jk}$$

$$\Delta v_{ij} = \alpha x_i f'\left(\boldsymbol{v}_{\boldsymbol{j}}^{\boldsymbol{T}} \boldsymbol{x}\right) \sum_{k=1}^{m} w_{jk} \left(t_k - f\left(\sum_{j=0}^{p} w_{jk} f\left(\boldsymbol{v}_{\boldsymbol{j}}^{\boldsymbol{T}} \boldsymbol{x}\right) \right) \right) f'\left(\sum_{j=0}^{p} w_{jk} f\left(\boldsymbol{v}_{\boldsymbol{j}}^{\boldsymbol{T}} \boldsymbol{x}\right) \right)$$

How to improve MLP?

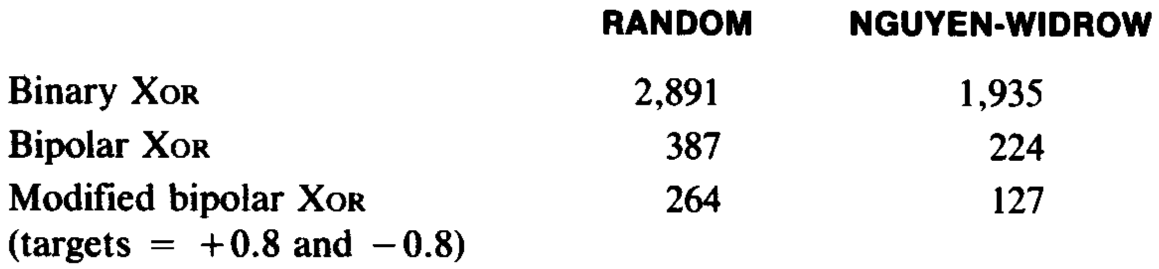
- How to achieve?
  - Weight initialization
    - Use Nguyen-Widrow or more sophisticated weight initialization methods
    - Start with small random weights
    - Large weights will cause saturation
    - Implicit regularization!



	RANDOM	NGUYEN-WIDROW
Binary Xor	2,891	1,935

How to improve MLP?

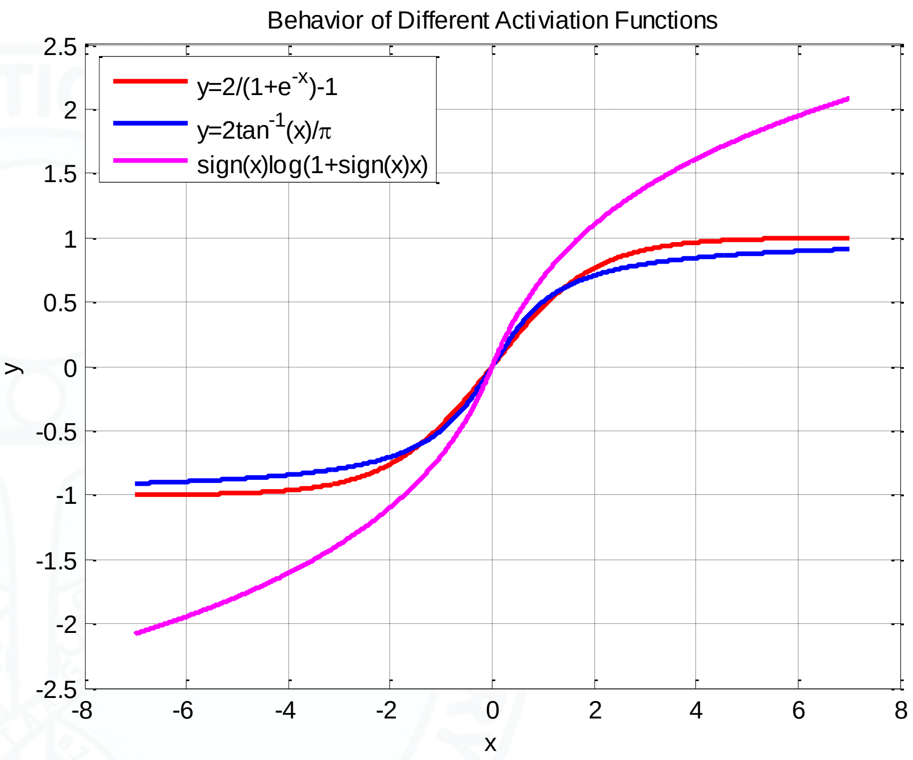
- Changes in Data Representation
  - Bipolar inputs/targets are better than binary
    - Zeros in inputs can cause stalls
  - Using clipped bipolar outputs instead of bipolar ones
    - Sigmoidal activation functions will produce a 1.0 or 0.0 only in the asymptote



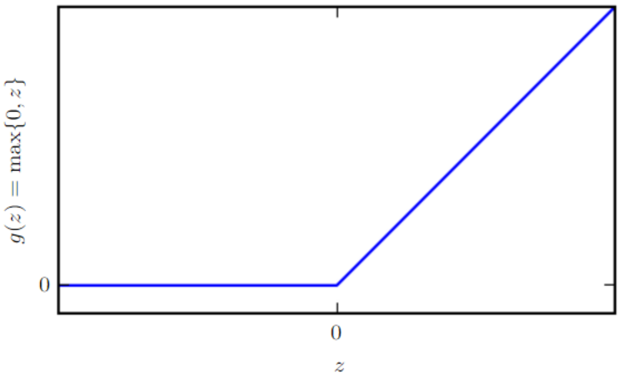
	RANDOM	NGUYEN-WIDROW
Binary Xor	2,891	1,935
Bipolar Xor	387	224
Modified bipolar Xor (targets = +0.8 and −0.8)	264	127

How to improve MLP?

- Use slowly saturating or non-saturating nonlinear activation functions
  - Examples: ReLU, Log Activation

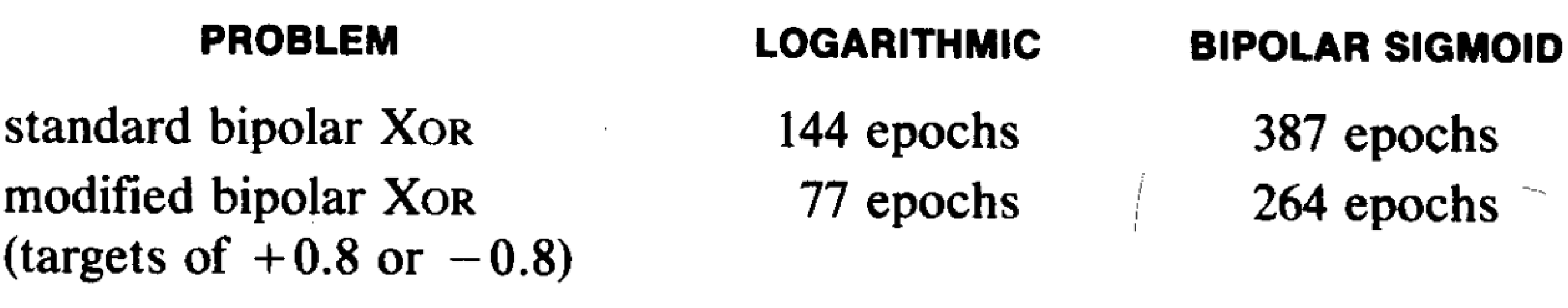


$$f(x) = \begin{cases} \log(1 + x) & \text{for } x > 0 \\ -\log(1 - x) & \text{for } x < 0. \end{cases}$$



How to improve MLP?

- Effect of log activation



PROBLEM	LOGARITHMIC	BIPOLAR SIGMOID
standard bipolar XOR	144 epochs	387 epochs
modified bipolar XOR (targets of +0.8 or −0.8)	77 epochs	264 epochs

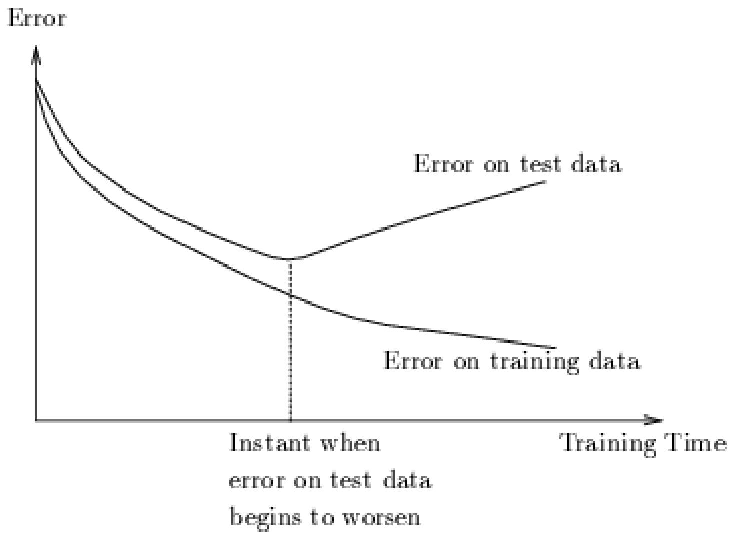
Improvements in Optimization

- Use stochastic gradient updates with mini-batches
  - Easy parallelization
- Change learning rate adaptively
- Use momentum (0≤μ ≤1) based update
  - but too much momentum may cause you to overshoot the local minima

$$\Delta w_{jk}(t+1) = \alpha \delta_k z_j + \mu \Delta w_{jk}(t),$$

Improving MLP

- Data Augmentation
  - Create artificial examples
    - Addition of noise
    - Translation of images or other transforms
- Drop-Off
- Batch Normalization
- Use Early Stopping
  - Keep track of generalization error and stop if the generalization error does not improve enough even when the error on training data is going down



Doing all this in Keras

- Layers

```
model = Sequential()
model.add(Dense(32, input_shape=(500,)))
model.add(Dense(10, activation='softmax'))
model.compile(optimizer='rmsprop',
loss='categorical_crossentropy', metrics=['accuracy'])
```

Useful attributes of Model

`model.layers`: is a flattened list of the layers comprising the model graph.
`model.inputs`: is the list of input tensors
`model.outputs`: is the list of output tensors.

Doing all this in Keras

- Activations

```
from keras.layers import Activation, Dense
model.add(Dense(64))
model.add(Activation('tanh'))

model.add(Dense(64, activation='tanh'))
```

- Available Activation
  - Softmax
  - Elu
  - Softplus
  - Softsign
  - Relu
  - Tanh
  - Sigmoid
  - Hard Sigmoid
  - Linear

Doing all this in Keras

- Losses

```
model.compile(loss='mean_squared_error', optimizer='sgd')

from keras import losses model.compile(loss=losses.mean_squared_error, optimizer='sgd')
```

- Available
  - Mean Squared Error
  - Mean Absolute Error
  - Mean Absolute Percentage Error
  - Mean Squared Logarithmic Error
  - Squared Hinge
  - Hinge
  - Categorical Cross Entropy
  - Sparse categorical crossentropy
  - Binary Crossentropy
  - Kullback Leibler Divergence
  - Posison
  - Cosine Proximity

Doing all this in Keras

- Metrics
  - Used to evaluate model performance

```
from keras import metrics
model.compile(loss='mean_squared_error',
              optimizer='sgd',
              metrics=[metrics.mae, metrics.categorical_accuracy])
```

- Available
  - Binary Accuracy
  - Categorical Accuracy
  - Sparse Categorical Accuracy
  - Top K Categorical Accuracy
  - Custom

Doing all this in Keras

- Optimizers

```
from keras import optimizers
model = Sequential()
model.add(Dense(64, init='uniform', input_shape=(10,)) model.add(Activation('tanh'))
model.add(Activation('softmax'))
sgd = optimizers.SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
model.compile(loss='mean_squared_error', optimizer=sgd)
```

- Available
  – SGD
  – RMSprop
  – Adagrad
  – AdaDelta
  – Adam
  – Adamax
  – Nadam

Doing all this in Keras

- Initializers

```
model.add(Dense(64,
       kernel_initializer='random_uniform',
       bias_initializer='zeros'))
```

Doing all this in Keras

- Regularization
  - L1 and L2
- Drop-Out
- Batch Normalization

Doing all this in Keras

- Data Augmentation
  - Noise Layer
  - ImageDataGenerator

Class Exercise!

- Requires Keras based computers
- Solve the XOR using a single hidden layer BPNN with sigmoid activations
  - See what is the effect of different parameters on the convergence characteristics of the neural network

Universal Function Approximation

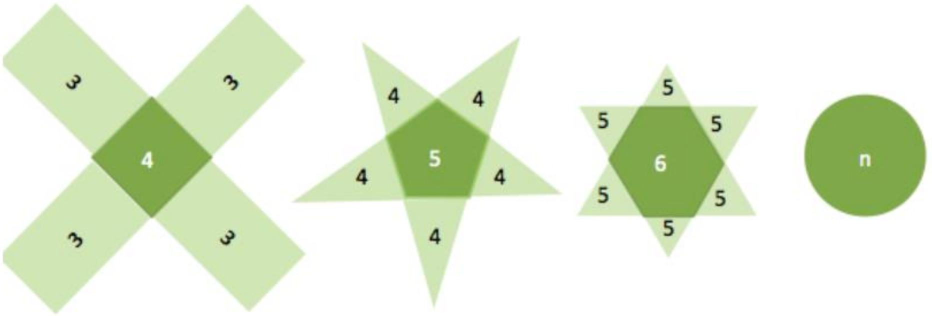
- Universal Approximation
  - Any function $f(\boldsymbol{x})$ over m inputs can be represented as follows:

$$F(\boldsymbol{x}) = \sum_{i=1}^{N} \boldsymbol{v}_i \varphi\left(\boldsymbol{w}_i^T \boldsymbol{x} + b_i\right)$$

    - $\varphi(\cdot)$ is a non-constant, bounded and monotonically-increasing continuous "basis" function
    - *N* is the number of functions
    - $F(\boldsymbol{x})$ is an approximation of $f(\boldsymbol{x})$, i.e., $|f(\boldsymbol{x}) - F(\boldsymbol{x})| < \epsilon$
- A neural network with a single hidden layer is a universal approximator
- A single hidden layer neuron with randomly initialized weights is a universal approximator
  - Extreme Learning Machine
  - Random Fourier Features as Kernel Approximators

Universal Function Approximation

- A neural network with one hidden layer can be used to approximate any shape
  - – However, the approximation might require exponentially many neurons
  - – How can we reduce the number of computations?



Wang, Haohan, and Bhiksha Raj. “On the Origin of Deep Learning.” *arXiv:1702.07800 [Cs, Stat]*, February 24, 2017. http://arxiv.org/abs/1702.07800.

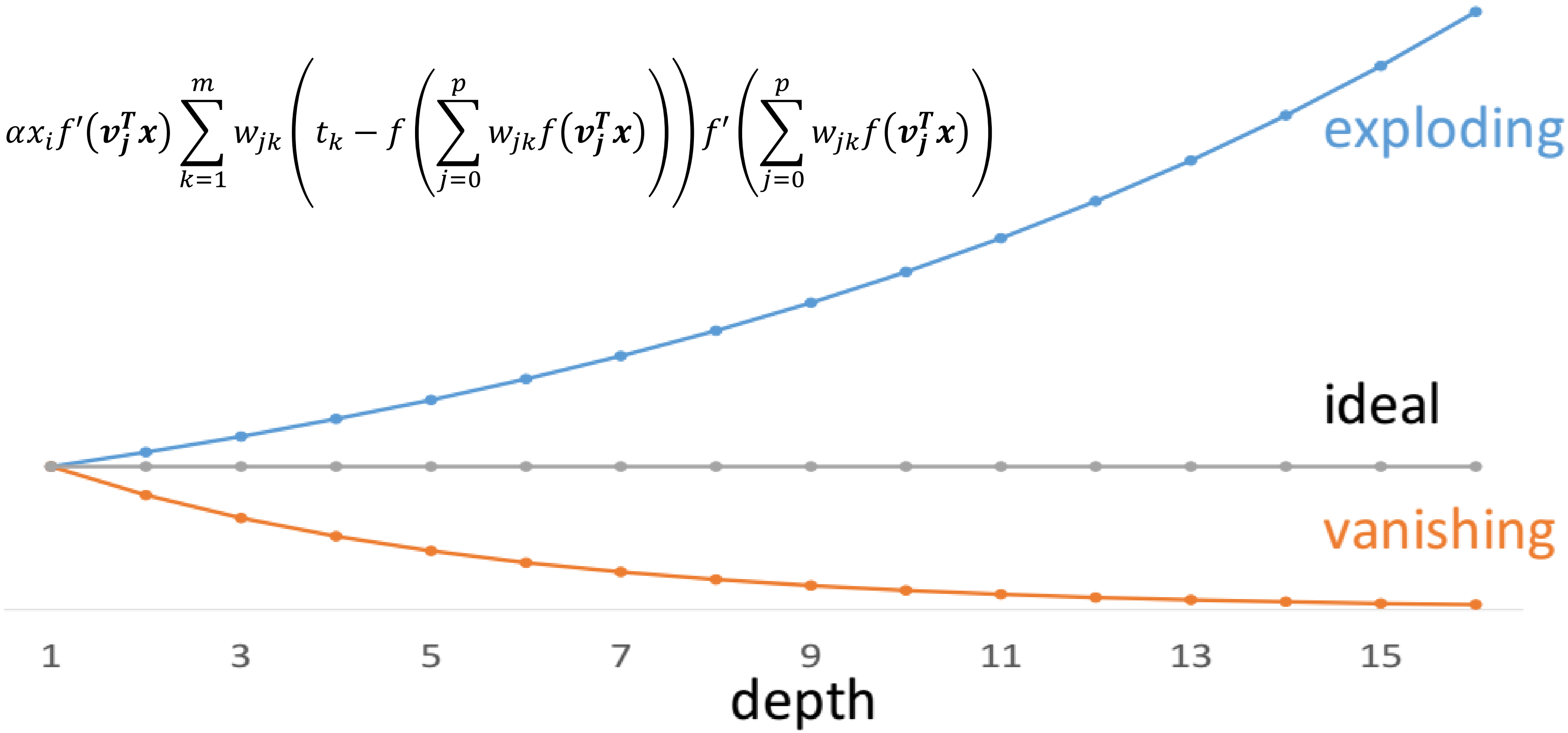
Practical Issues in Universal Approximation

- The universal approximation theorem means that regardless of what function we are trying to learn, we know that a large MLP will be able to represent this function.
- However, we are not guaranteed that the training algorithm will be able to “learn” that function.
  - Optimization can fail
  - Learning is different from optimization
    - The primary requirement for learning is generalization

The case of the exploding (or vanishing) gradients

- Effect of gradient multiplication

$$\Delta v_{ij} = \alpha x_i f'(\boldsymbol{v}_j^T \boldsymbol{x}) \sum_{k=1}^{m} w_{jk} \left(t_k - f\left(\sum_{j=0}^{p} w_{jk} f(\boldsymbol{v}_j^T \boldsymbol{x}) \right) \right) f'\left(\sum_{j=0}^{p} w_{jk} f(\boldsymbol{v}_j^T \boldsymbol{x}) \right)$$



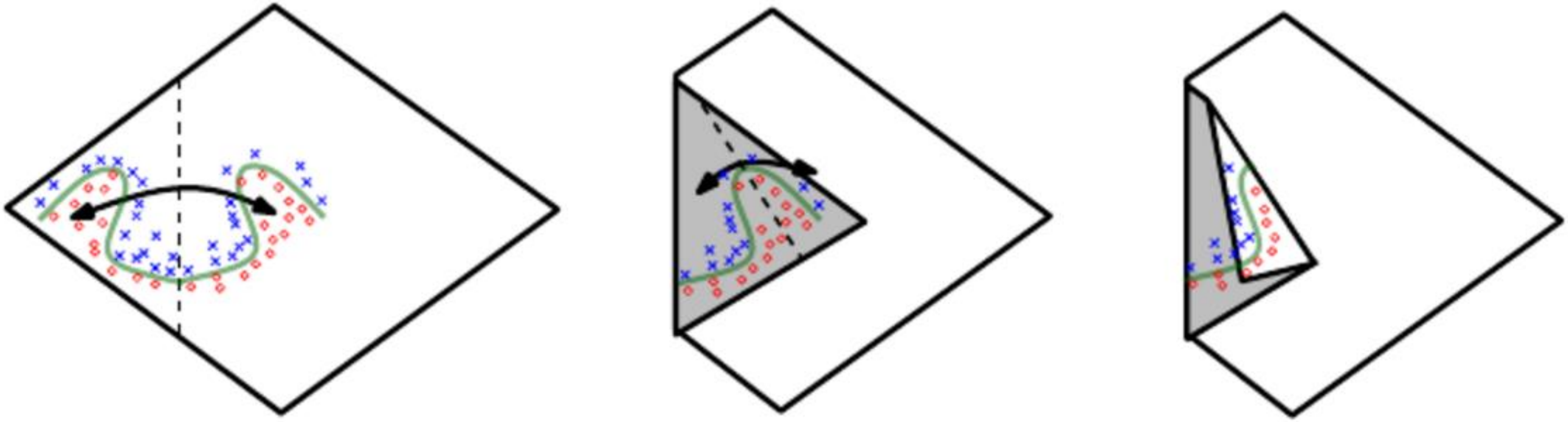
Glorot & Bengio 2010 “Understanding the difficulty of training deep feedforward neural networks”.

Width vs. Depth

- An MLP with a single hidden layer is sufficient to represent any function
  - But the layer may be infeasibly large
  - May fail to learn and generalize correctly
- Using a deeper model can reduce the number of units required to represent the desired function and can reduce the amount of generalization error
- A function that could be expressed with O(n) neurons on a network of depth k required at least $O(2^{\sqrt{n}})$ and $O((n -1)^k)$ neurons on a two-layer neural network: Delalleau and Bengio (2011)
- Functions representable with a deep rectifier net can require an exponential number of hidden units with a shallow (one hidden layer) network: Montufar (2014)
- For a shallow network, the representation power can only grow polynomially with respect to the number of neurons, but for deep architecture, the representation can grow exponentially with respect to the number of neurons: Bianchini and Scarselli (2014)
- Depth of a neural network is exponentially more valuable than the width of a neural network, for a standard MLP with any popular activation functions: Eldan and Shamir (2015)

Width vs. Depth

- Exponential advantage of deeper networks



Montufar (2014)

Width vs. Depth

- Empirical results for some data showed that depth increases generalization performance in a variety of applications

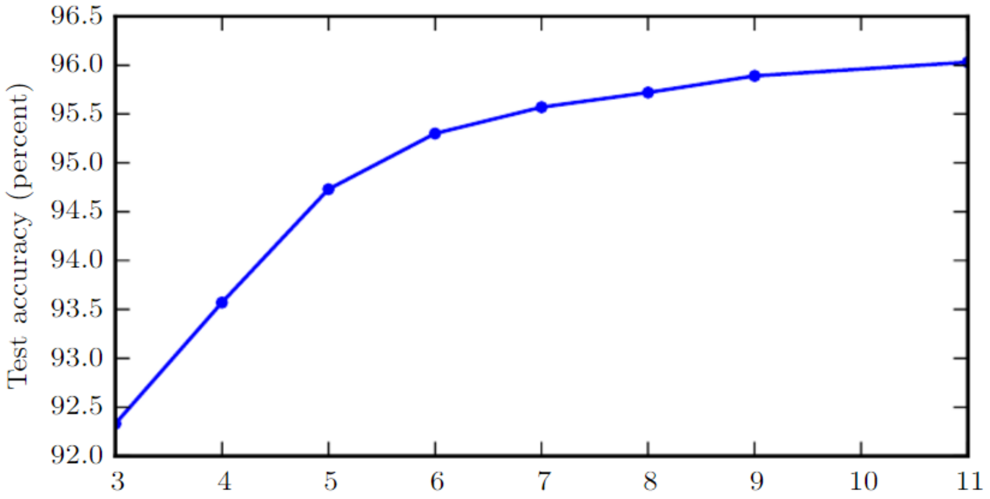


Figure 6.6: Empirical results showing that deeper networks generalize better when used to transcribe multi-digit numbers from photographs of addresses. Data from Goodfellow *et al.* (2014d). The test set accuracy consistently increases with increasing depth. See figure 6.7 for a control experiment demonstrating that other increases to the model size do not yield the same effect.

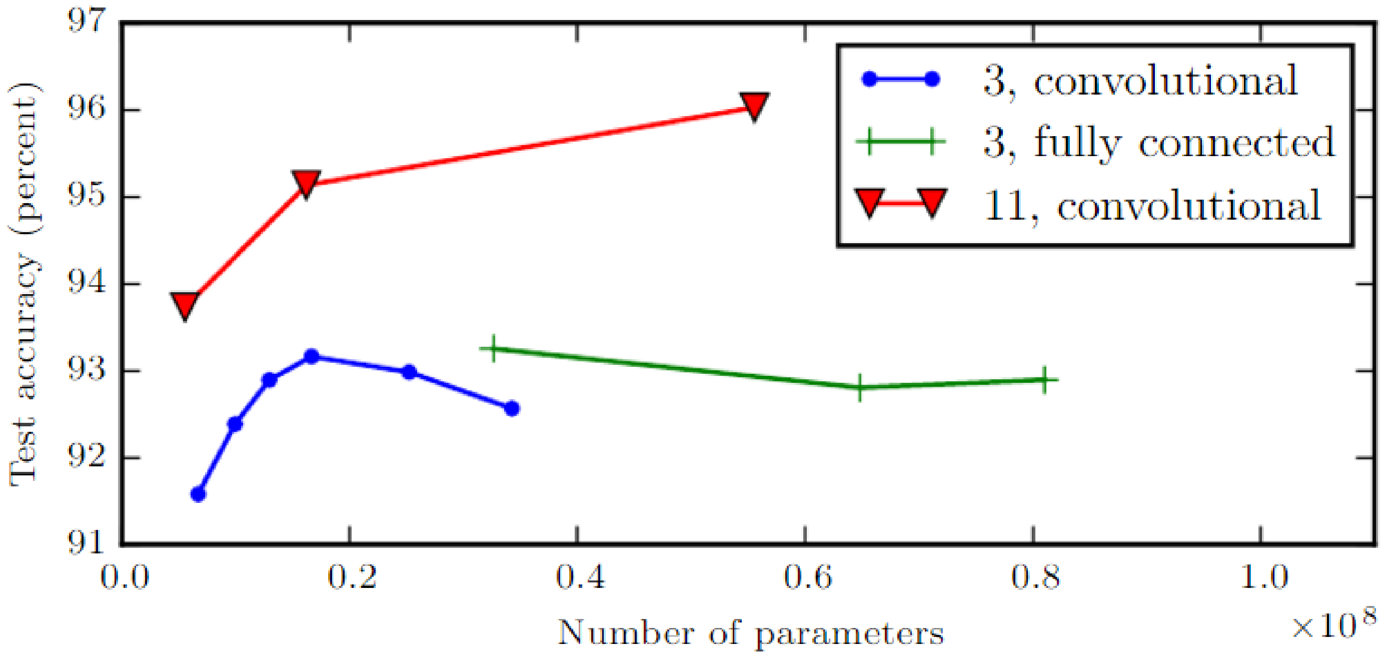
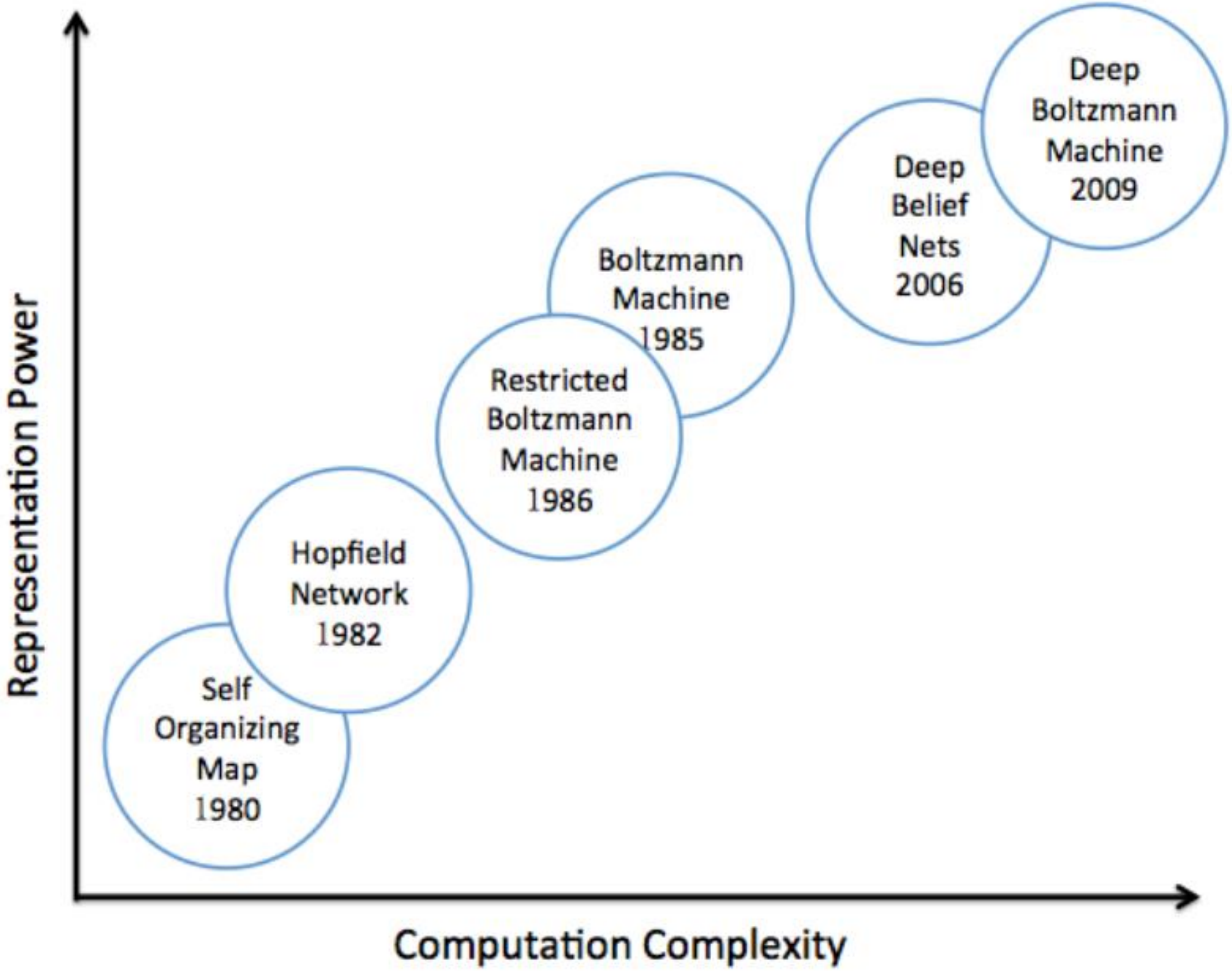


Figure 6.7: Deeper models tend to perform better. This is not merely because the model is larger. This experiment from Goodfellow *et al.* (2014d) shows that increasing the number of parameters in layers of convolutional networks without increasing their depth is not nearly as effective at increasing test set performance. The legend indicates the depth of network used to make each curve and whether the curve represents variation in the size of the convolutional or the fully connected layers. We observe that shallow models in this context overfit at around 20 million parameters while deep ones can benefit from having over 60 million. This suggests that using a deep model expresses a useful preference over the space of functions the model can learn. Specifically, it expresses a belief that the function should consist of many simpler functions composed together. This could result either in learning a representation that is composed in turn of simpler representations (e.g., corners defined in terms of edges) or in learning a program with sequentially dependent steps (e.g., first locate a set of objects, then segment them from each other, then recognize them).

Increasing representation power with depth

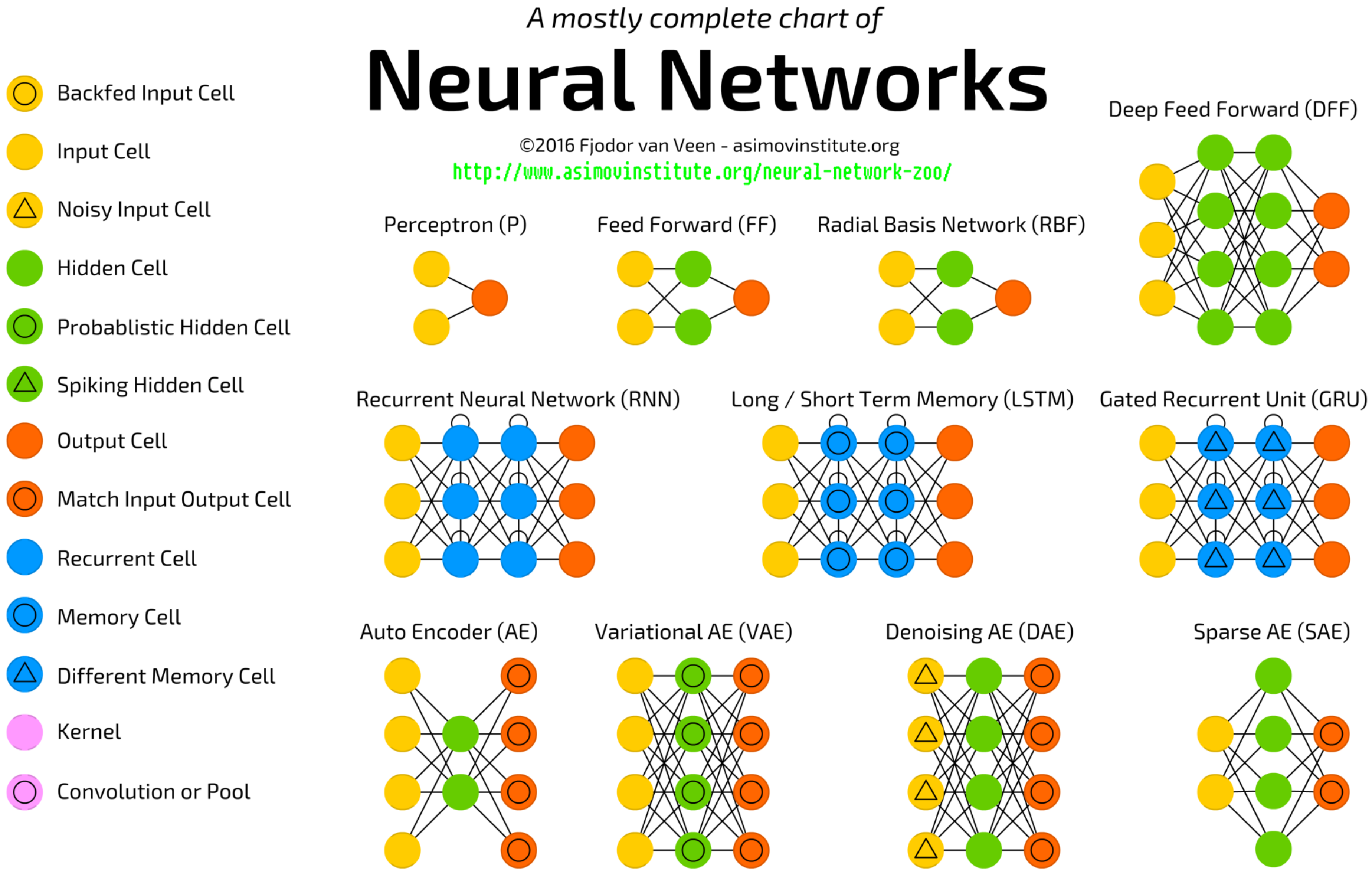


Issues with Depth

- Generalization
  - Large networks are large capacity machines
  - Remember: Learning requires generalization and goes beyond mere minimization of an objective function!
- Failure to Optimize
  - Random initialization leads to the network being stuck in poor solutions
  - Deeper networks are more prone to vanishing/exploding gradients and optimization failures
    - "Greedy Layer-Wise Training of Deep Networks" by Bengio et al., 2006
      - Uses unsupervised pre-training to initialize the weights of a network such that the optimization becomes easier
    - Since 2010, this has been replaced with Drop-out and batch-normalization schemes which improve the optimization performance
      - Rectified Linear Units get rid of the vanishing gradient problem
      - Drop-out improves generalization
      - Batch Normalization accelerates deep learning and improves generalization
        - » Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift by Ioffe and Segedy, 2015.
- Large scale optimization is tricky in deep learning
  - Computationally demanding
  - Requires efficient methods
    - Stochastic gradient and sub-gradient methods

Issues with depth

- Handling variety of neural network architectures
  - How can we develop a framework of learning in which we can add layers, have a large diversity of layer connectivity, change objective functions and losses, layer connectivity, regularization, etc.?
  - And still solve the optimization problem in an efficient manner!
  - Symbolic Computation and Automatic Differentiation
  - GPU
    - Efficient matrix operations
    - Higher bandwidth

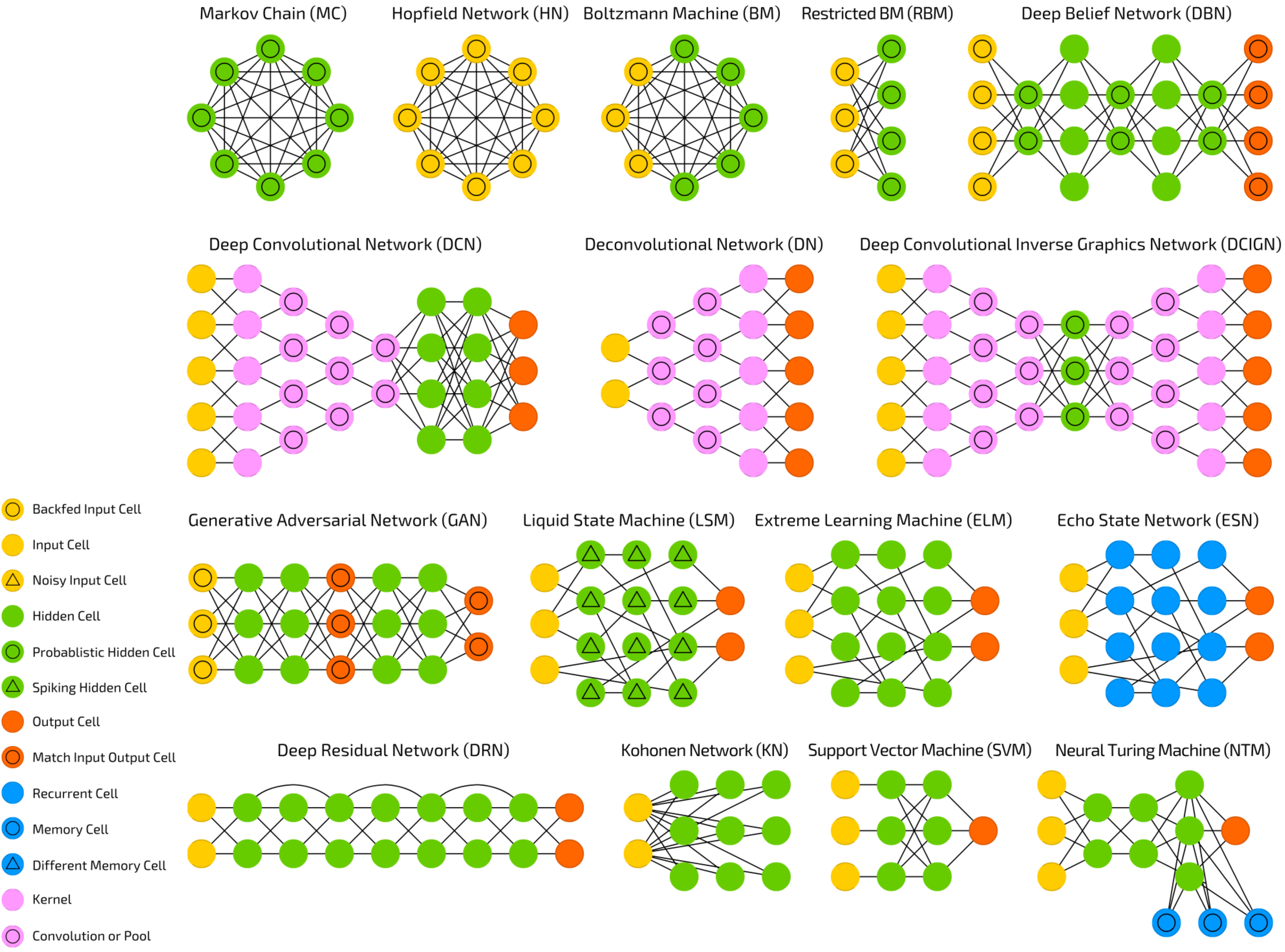


A mostly complete chart of

Neural Networks

©2016 Fjodor van Veen - asimovinstitute.org

http://www.asimovinstitute.org/neural-network-zoo/



Markov Chain (MC)
Hopfield Network (HN)
Boltzmann Machine (BM)
Restricted BM (RBM)
Deep Belief Network (DBN)
Deep Convolutional Network (DCN)
Deconvolutional Network (DN)
Deep Convolutional Inverse Graphics Network (DCIGN)
Backfed Input Cell
Input Cell
Noisy Input Cell
Hidden Cell
Probablistic Hidden Cell
Spiking Hidden Cell
Output Cell
Match Input Output Cell
Recurrent Cell
Memory Cell
Different Memory Cell
Kernel
Convolution or Pool
Generative Adversarial Network (GAN)
Liquid State Machine (LSM)
Extreme Learning Machine (ELM)
Echo State Network (ESN)
Deep Residual Network (DRN)
Kohonen Network (KN)
Support Vector Machine (SVM)
Neural Turing Machine (NTM)

Making deeper neural networks practical

- Optimization
- Handling vanishing (or exploding) gradients
  - Pre-training (old!)
  - Drop-out
  - Batch Normalization
- Computational challenges
  - Use of computational graphs for automatic differentiation of neural networks
    - Allows for different types of architectures
  - Using GPU parallelization

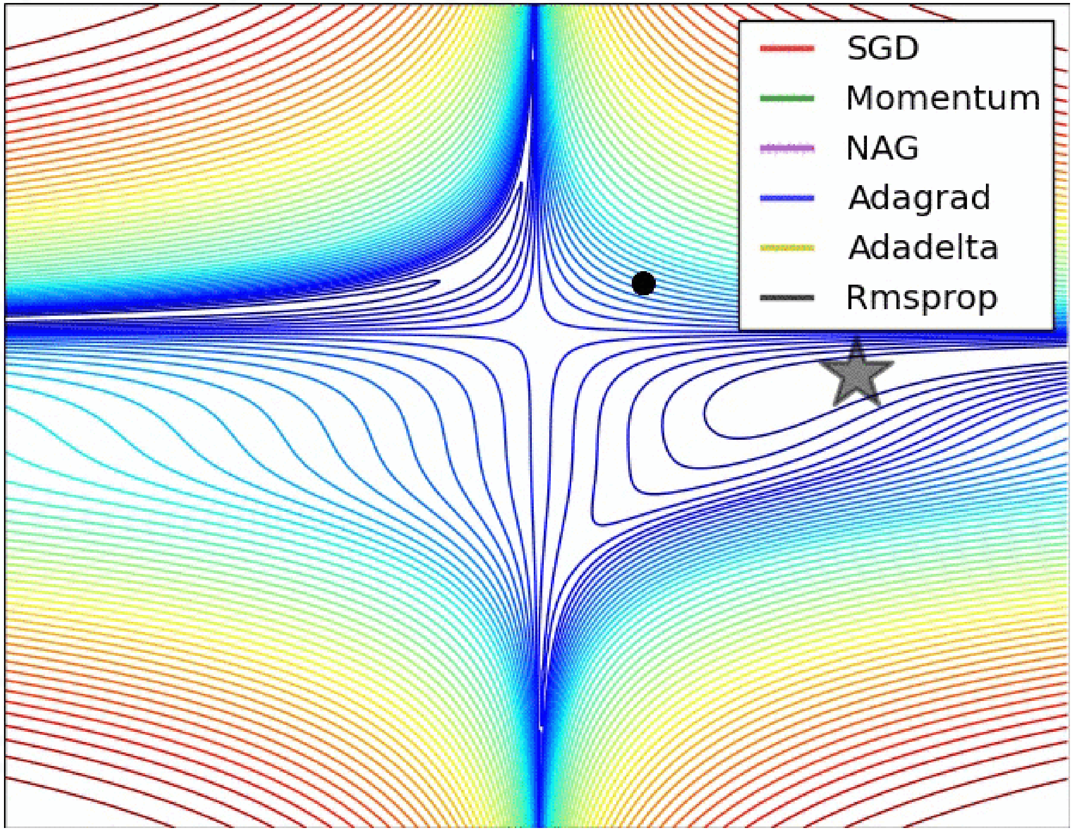
Optimization Methods

- Gradient Descent: Go down!
- Stochastic Gradient Descent
- Mini-batch Gradient Descent
- SGD with momentum: accelerate if going downhill for a long time
- Nesterov momentum: accelerate but not indefinitely
- Adagrad: Adaptive Learning Rate by accumulating past gradients
- AdaDelta/RMSProp: Adaptive Learning rate but does not accumulate all past gradients
- Adam: Adaptive learning rate with momentum

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta)$$

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i)}; y^{(i)})$$

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i:i+n)}; y^{(i:i+n)})$$



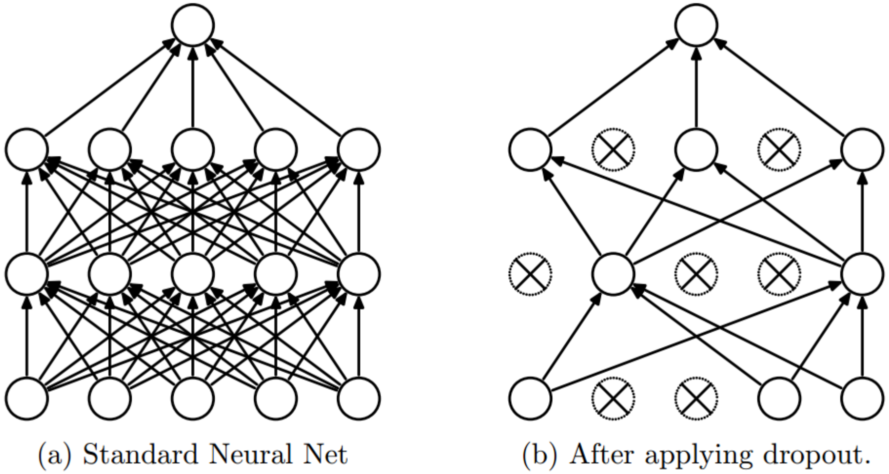
An overview of gradient descent optimization algorithms by Sebastian Ruder, 20-16
http://sebastianruder.com/optimizing-gradient-descent/ , https://arxiv.org/abs/1609.04747

Optimization Methods

- Sparse data
  - Adaptive Learning rate
- RMSProp, AdaDelta and Adam are very similar
  - Do well in general
  - Adam slightly outperforms RMSProp and is a good choice
- Parallelizing and distributing SGD
  - TensorFlow uses computational graph distribution
  - Other parallel schemes include: HogWild! Delayed SGD, etc.

Understanding Drop-out

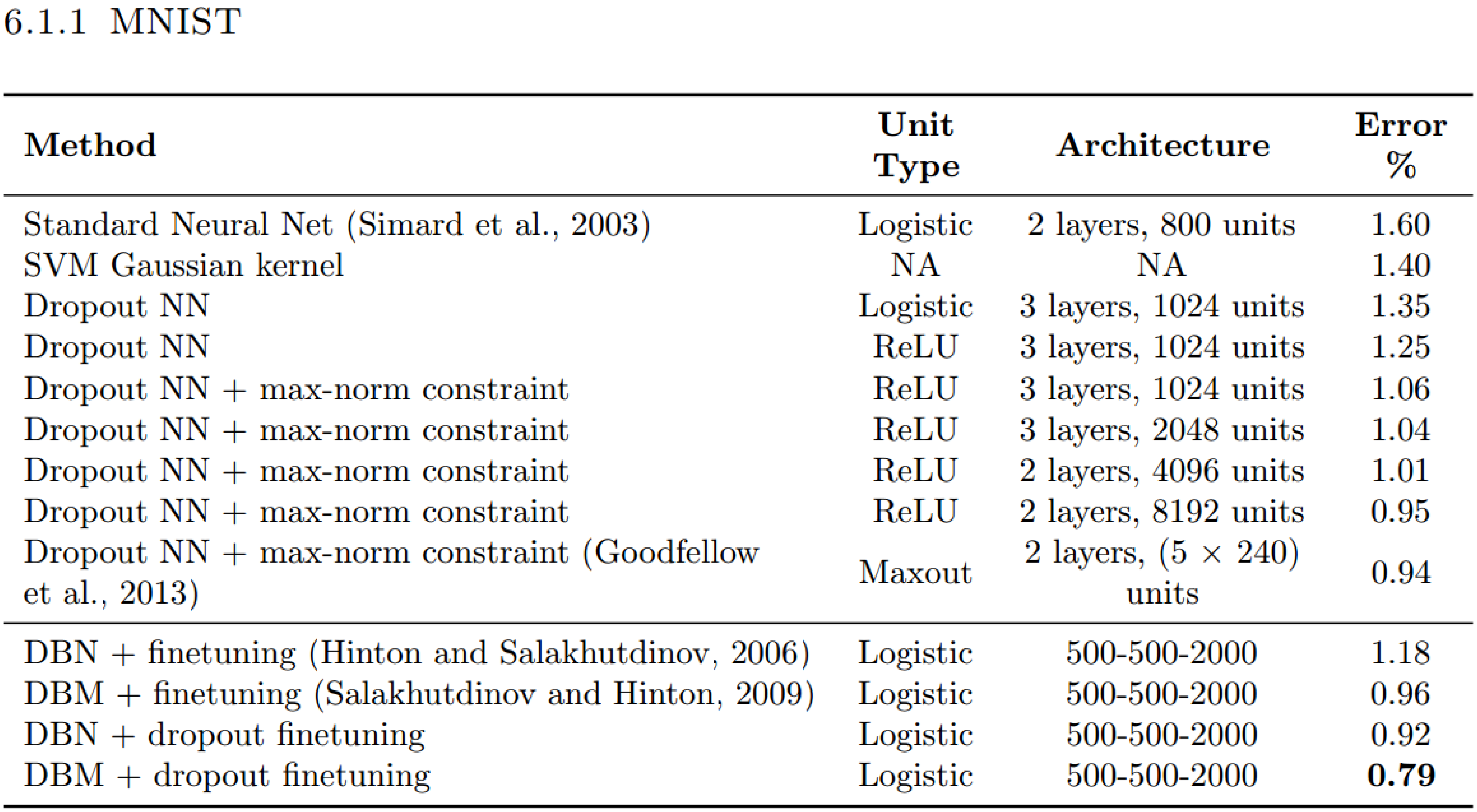
- "Dropout: A Simple Way to Prevent Neural Networks from Overfitting" by Srivastava et al., 2014.
  - Randomly drop units (along with their connections) from the neural network during training
  - Average weights across all "thinned" networks
  - Replaces explicit regularization and produces faster learning
  - Drop-out layer in keras!


(a) Standard Neural Net

(b) After applying dropout.

Dropout Neural Net Model. **Left**: A standard neural net with 2 hidden layers. **Right**: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

Effect of Dropout



6.1.1 MNIST

Method	Unit Type	Architecture	Error %
Standard Neural Net (Simard et al., 2003)	Logistic	2 layers, 800 units	1.60
SVM Gaussian kernel	NA	NA	1.40
Dropout NN	Logistic	3 layers, 1024 units	1.35
Dropout NN	ReLU	3 layers, 1024 units	1.25
Dropout NN + max-norm constraint	ReLU	3 layers, 1024 units	1.06
Dropout NN + max-norm constraint	ReLU	3 layers, 2048 units	1.04
Dropout NN + max-norm constraint	ReLU	2 layers, 4096 units	1.01
Dropout NN + max-norm constraint	ReLU	2 layers, 8192 units	0.95
Dropout NN + max-norm constraint (Goodfellow et al., 2013)	Maxout	2 layers, (5 × 240) units	0.94
DBN + finetuning (Hinton and Salakhutdinov, 2006)	Logistic	500-500-2000	1.18
DBM + finetuning (Salakhutdinov and Hinton, 2009)	Logistic	500-500-2000	0.96
DBN + dropout finetuning	Logistic	500-500-2000	0.92
DBM + dropout finetuning	Logistic	500-500-2000	**0.79**

Understanding Batch-Normalization

- Re-normalization of weight parameters after every mini-batch to zero-norm and unit-variance with backpropagation
  - Note: This is not re-initialization to random values, rather the current weights are updated
- Reduces the effects of weight normalization and enforces regularization leading to faster learning
  - More effective than drop-out
  - No need for “pre-training”
- Available in Keras!

Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv preprint arXiv:1502.03167v3*, 2015.

Effect of Batch Normalization

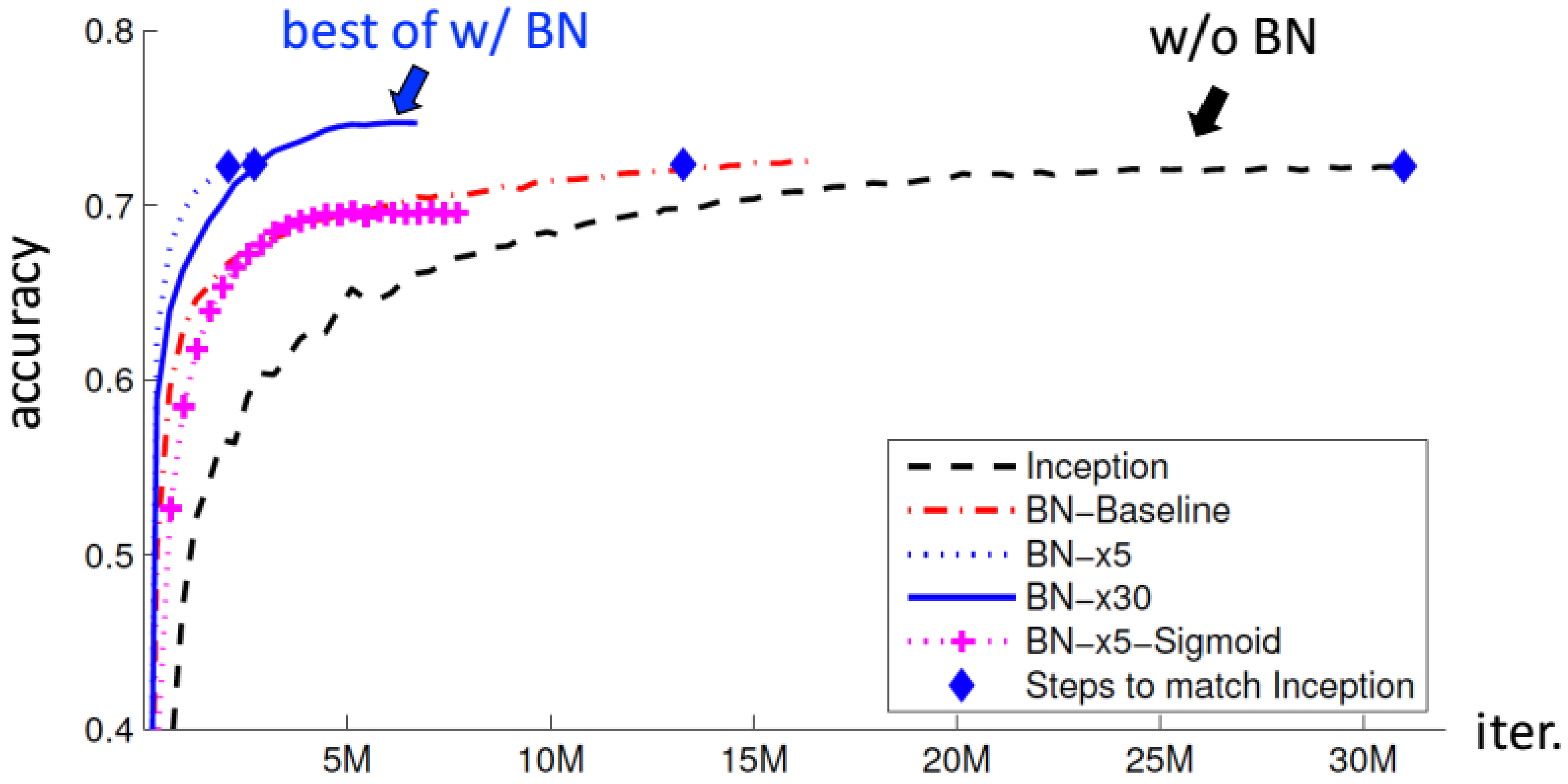


Figure taken from [S. Ioffe & C. Szegedy]

Computational Graphs

- Making a generic package for multi-layer neural networks requires
  - An abstract way of representing various computational operations involved in the network
  - Distributed Evaluations
  - Calculation of gradients
- Computational Graphs allow us to do all this!

Computational Graphs

- CGs are an easy way to think about mathematical expressions
- Formalizes the idea of neural networks and generalizes backpropagation and makes it computationally efficient
- The “compile” in keras builds a Computational Graph for the Network (can take time!)

Example

- Differentiation via the chain rule can be represented as the computational graph
- Symbolic derivatives
- Parallelization
  - Compute independent components in parallel
- Avoiding re-computation
  - Re-use symbolic derivatives
  - Store previous values

$$z = f\left(f\left(f(w)\right)\right) \qquad \frac{dz}{dw} = \frac{dz}{dy}\frac{dy}{dx}\frac{dx}{dw}$$

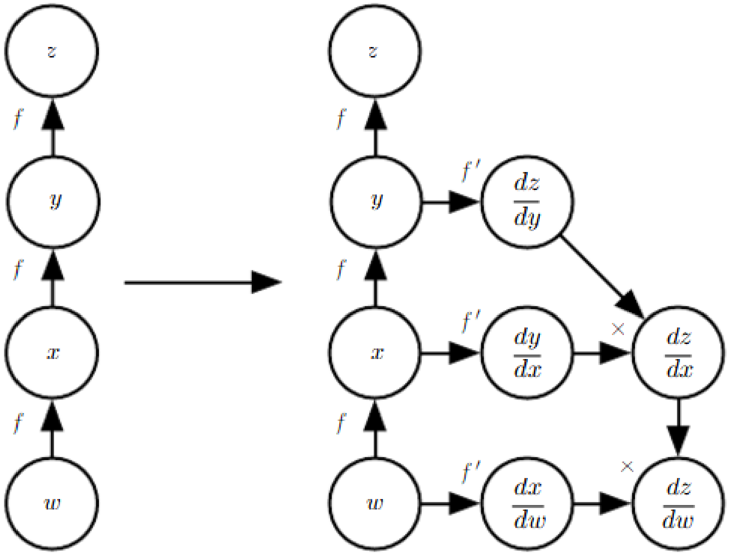
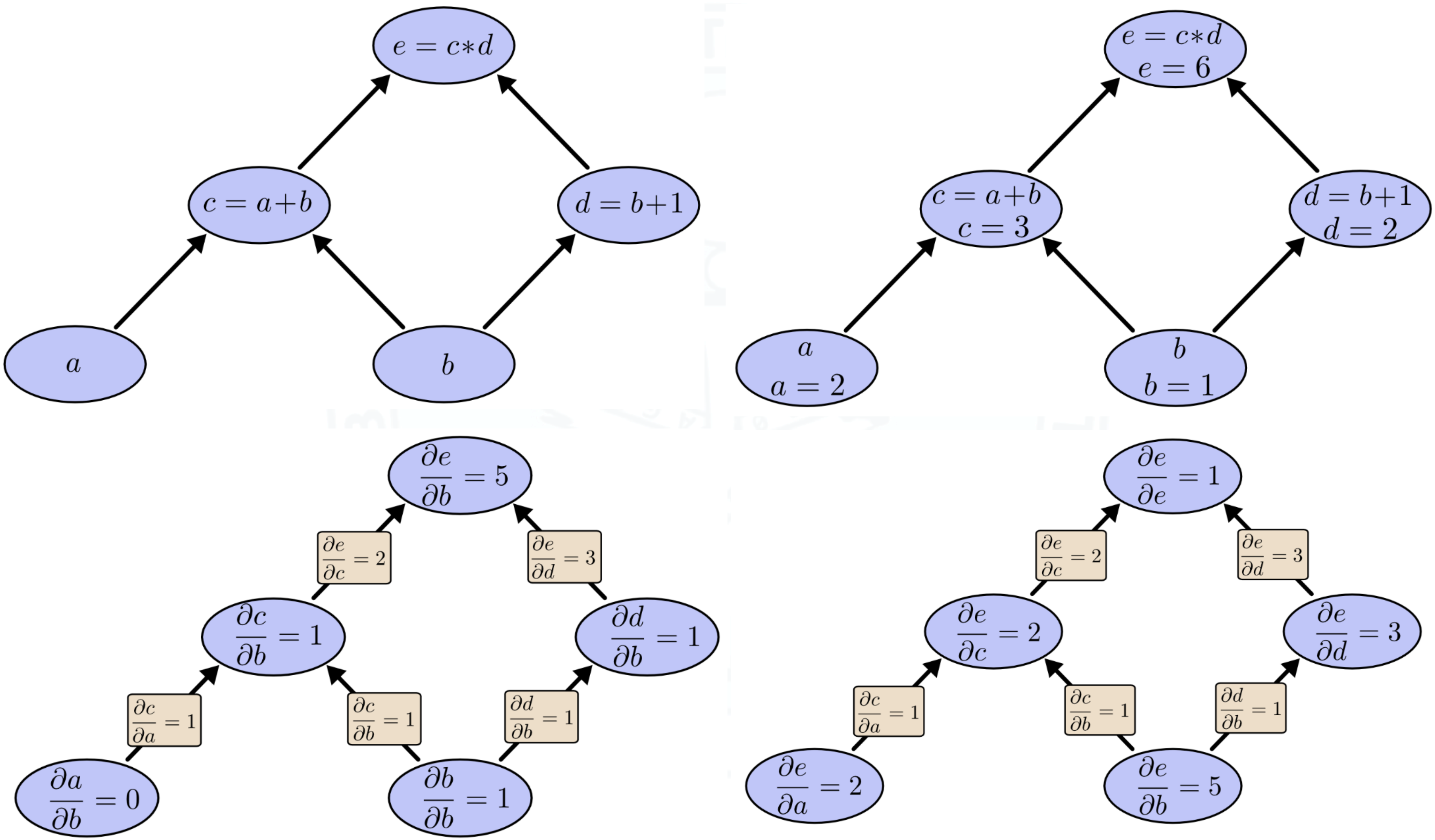


Figure 6.10: An example of the symbol-to-symbol approach to computing derivatives. In this approach, the back-propagation algorithm does not need to ever access any actual specific numeric values. Instead, it adds nodes to a computational graph describing how to compute these derivatives. A generic graph evaluation engine can later compute the derivatives for any specific numeric values. *(Left)*In this example, we begin with a graph representing $z = f(f(f(w)))$. *(Right)*We run the back-propagation algorithm, instructing it to construct the graph for the expression corresponding to $\frac{dz}{dw}$. In this example, we do not explain how the back-propagation algorithm works. The purpose is only to illustrate what the desired result is: a computational graph with a symbolic description of the derivative.

The magic of computational graphs



$e = c*d$
$c = a+b$
$d = b+1$
a
b
$e = c*d$ $e = 6$
$c = a+b$ $c = 3$
$d = b+1$ $d = 2$
a $a = 2$
b $b = 1$
$\frac{\partial e}{\partial b} = 5$
$\frac{\partial e}{\partial c} = 2$
$\frac{\partial e}{\partial d} = 3$
$\frac{\partial c}{\partial b} = 1$
$\frac{\partial d}{\partial b} = 1$
$\frac{\partial c}{\partial a} = 1$
$\frac{\partial c}{\partial b} = 1$
$\frac{\partial d}{\partial b} = 1$
$\frac{\partial a}{\partial b} = 0$
$\frac{\partial b}{\partial b} = 1$
$\frac{\partial e}{\partial e} = 1$
$\frac{\partial e}{\partial c} = 2$
$\frac{\partial e}{\partial d} = 3$
$\frac{\partial e}{\partial c} = 2$
$\frac{\partial e}{\partial d} = 3$
$\frac{\partial c}{\partial a} = 1$
$\frac{\partial c}{\partial b} = 1$
$\frac{\partial d}{\partial b} = 1$
$\frac{\partial e}{\partial a} = 2$
$\frac{\partial e}{\partial b} = 5$

Forward-mode differentiation

reverse-mode differentiation

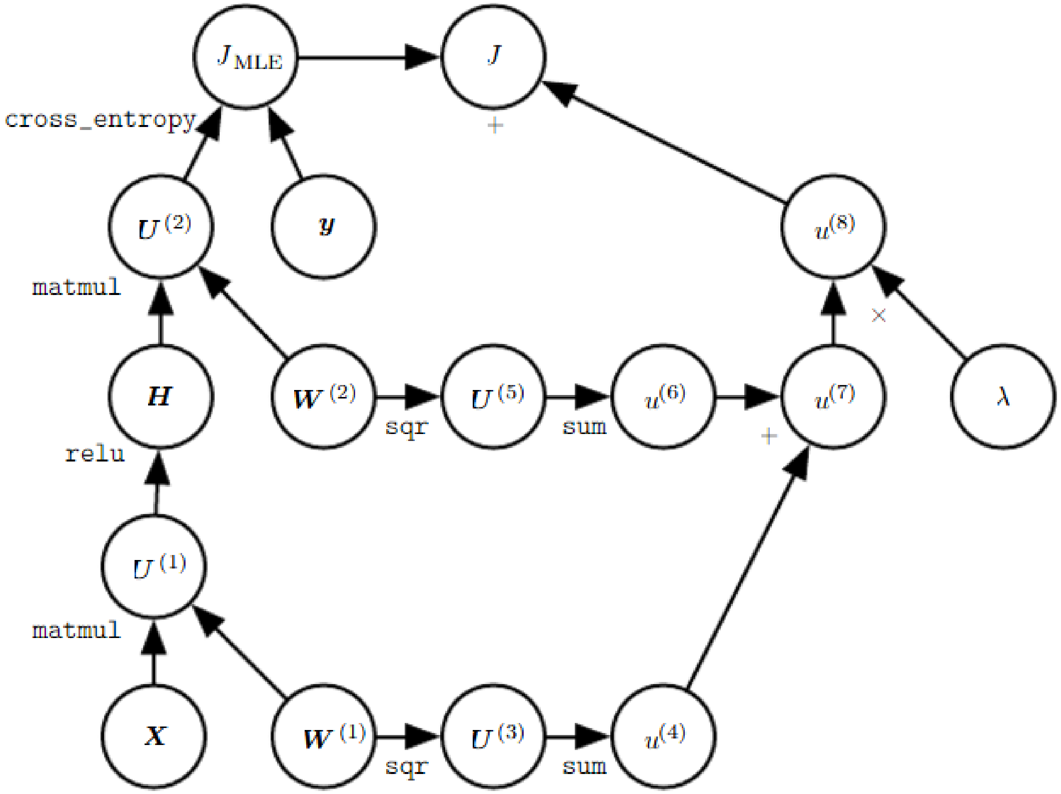
http://colah.github.io/posts/2015-08-Backprop/

The magic of computational graphs

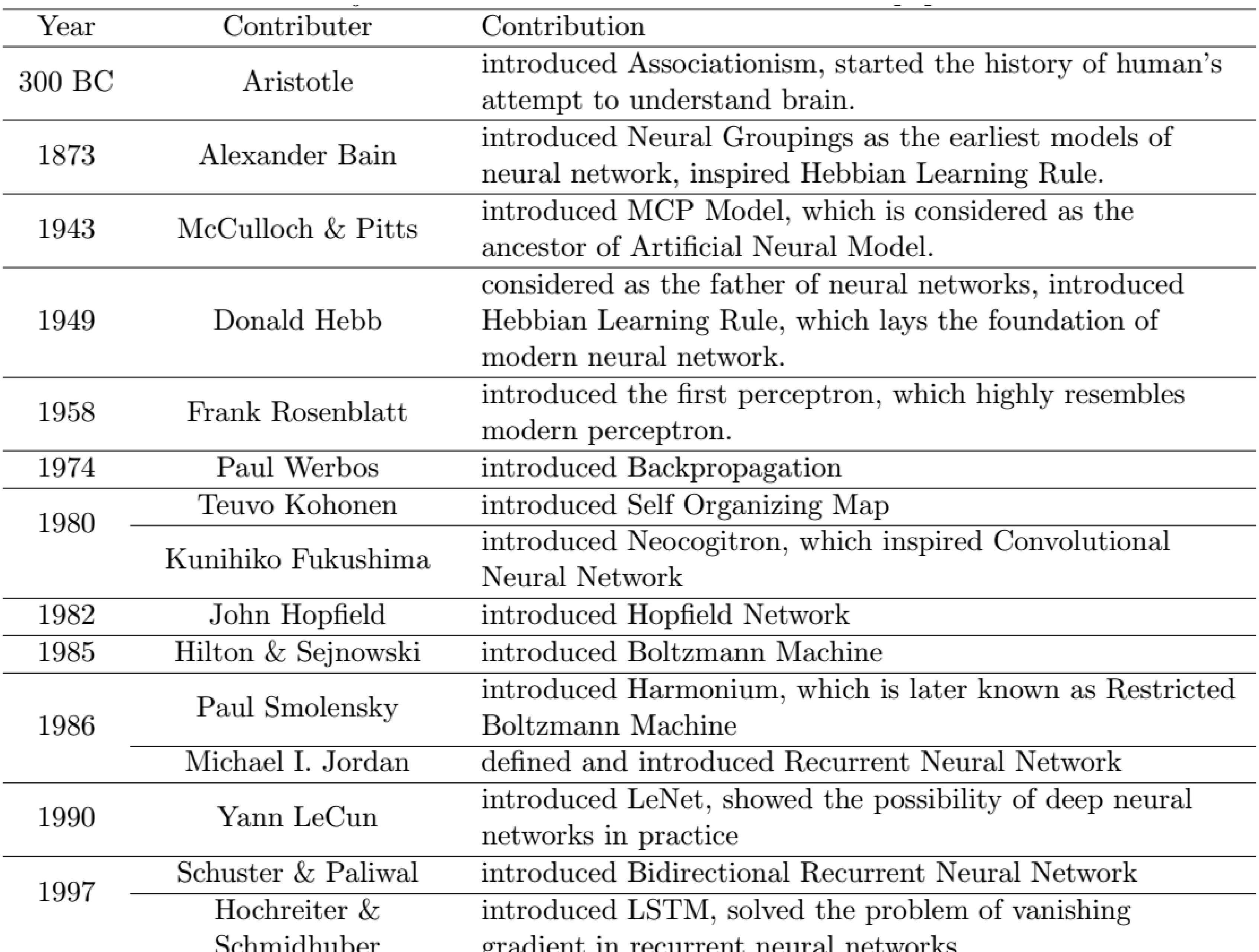
- Forward-mode differentiation gave us the derivative of our output with respect to a single input, but reverse-mode differentiation gives us all of them.
- For this graph, that’s only a factor of two speed up, but imagine a function with a million inputs and one output.
- Forward-mode differentiation would require us to go through the graph a million times to get the derivatives.
- Reverse-mode differentiation can get them all in one fell swoop!
- A speed up of a factor of a million is pretty nice!

Practical Example

$$J = J_{\text{MLE}} + \lambda \left(\sum_{i,j} \left(W_{i,j}^{(1)}\right)^2 + \sum_{i,j} \left(W_{i,j}^{(2)}\right)^2 \right)$$

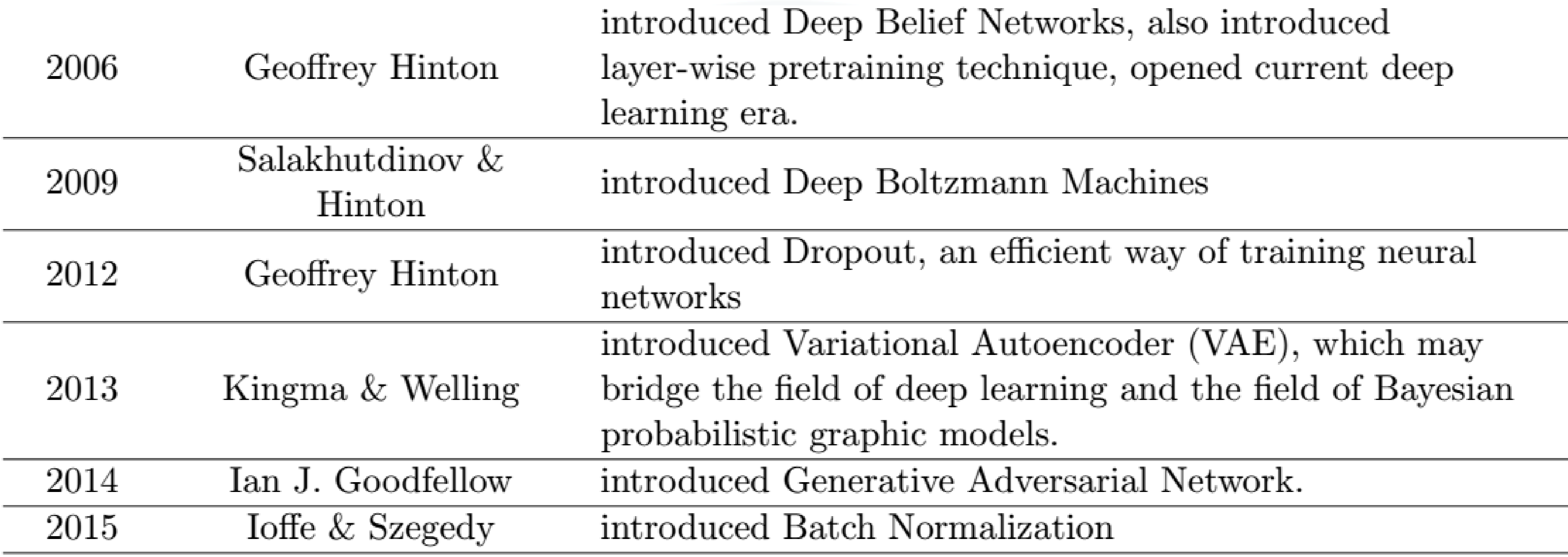


Origins of Deep Learning



Year	Contributer	Contribution
300 BC	Aristotle	introduced Associationism, started the history of human's attempt to understand brain.
1873	Alexander Bain	introduced Neural Groupings as the earliest models of neural network, inspired Hebbian Learning Rule.
1943	McCulloch & Pitts	introduced MCP Model, which is considered as the ancestor of Artificial Neural Model.
1949	Donald Hebb	considered as the father of neural networks, introduced Hebbian Learning Rule, which lays the foundation of modern neural network.
1958	Frank Rosenblatt	introduced the first perceptron, which highly resembles modern perceptron.
1974	Paul Werbos	introduced Backpropagation
1980	Teuvo Kohonen	introduced Self Organizing Map
	Kunihiko Fukushima	introduced Neocogitron, which inspired Convolutional Neural Network
1982	John Hopfield	introduced Hopfield Network
1985	Hilton & Sejnowski	introduced Boltzmann Machine
1986	Paul Smolensky	introduced Harmonium, which is later known as Restricted Boltzmann Machine
	Michael I. Jordan	defined and introduced Recurrent Neural Network
1990	Yann LeCun	introduced LeNet, showed the possibility of deep neural networks in practice
1997	Schuster & Paliwal	introduced Bidirectional Recurrent Neural Network
	Hochreiter & Schmidhuber	introduced LSTM, solved the problem of vanishing gradient in recurrent neural networks

Origins of Deep Learning



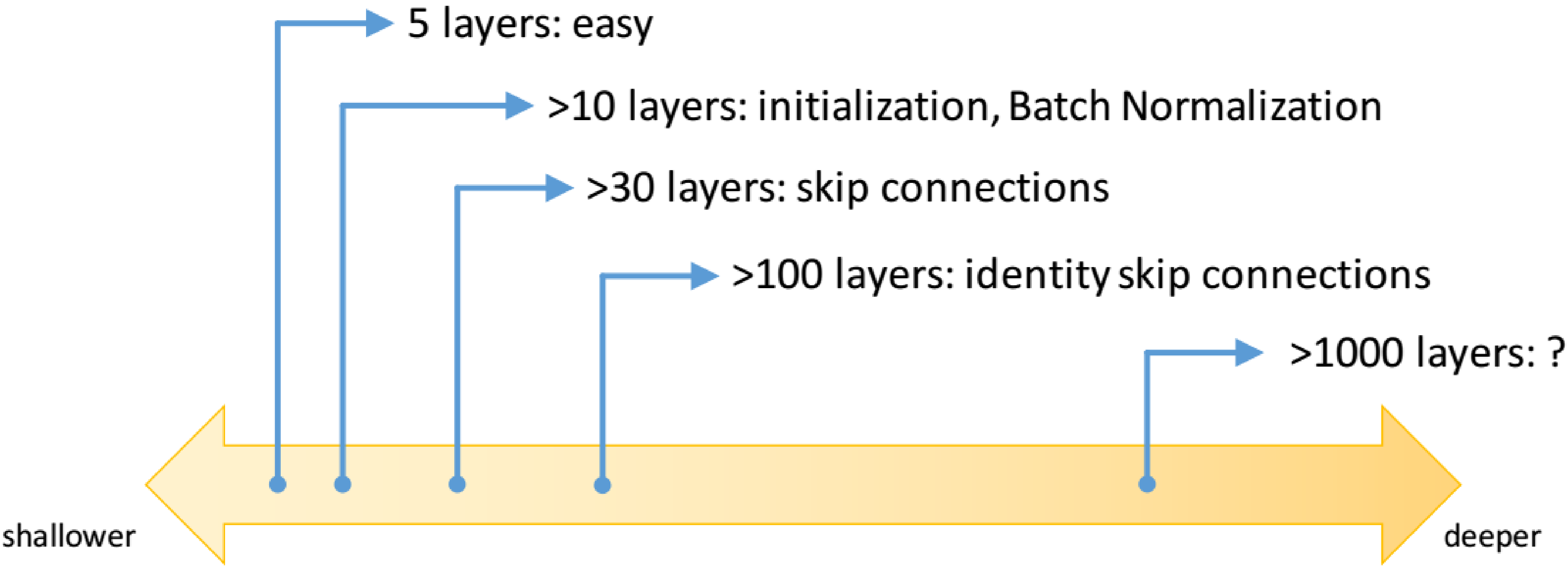
2006	Geoffrey Hinton	introduced Deep Belief Networks, also introduced layer-wise pretraining technique, opened current deep learning era.
2009	Salakhutdinov & Hinton	introduced Deep Boltzmann Machines
2012	Geoffrey Hinton	introduced Dropout, an efficient way of training neural networks
2013	Kingma & Welling	introduced Variational Autoencoder (VAE), which may bridge the field of deep learning and the field of Bayesian probabilistic graphic models.
2014	Ian J. Goodfellow	introduced Generative Adversarial Network.
2015	Ioffe & Szegedy	introduced Batch Normalization

- Wang, Haohan, and Bhiksha Raj. “On the Origin of Deep Learning.” *arXiv:1702.07800 [Cs, Stat]*, February 24, 2017. http://arxiv.org/abs/1702.07800.

Modern Practices

- Deep Convolutional Neural Networks
- Residual Networks
- Generative Models
  - Auto-encoders: VAE, NAE
  - Generative Adversarial Networks
  - Recurrent Neural Networks
- Recurrent Models
  - RNN
  - LSTM
- Transfer Learning
- Zero and One-shot learning

Spectrum of Depth

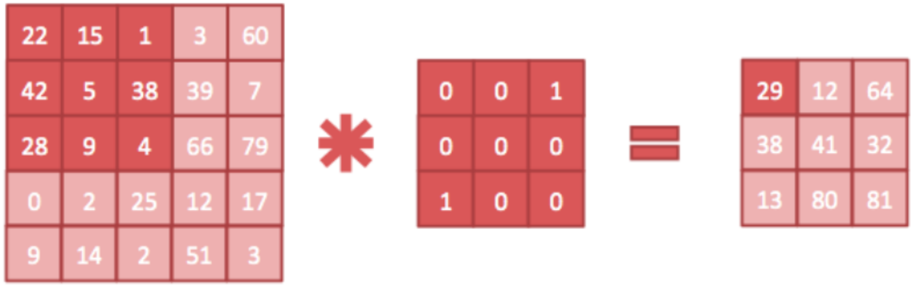

5 layers: easy
>10 layers: initialization, Batch Normalization
>30 layers: skip connections
>100 layers: identity skip connections
>1000 layers: ?
shallower
deeper

DCN: Deep Convolutional Networks

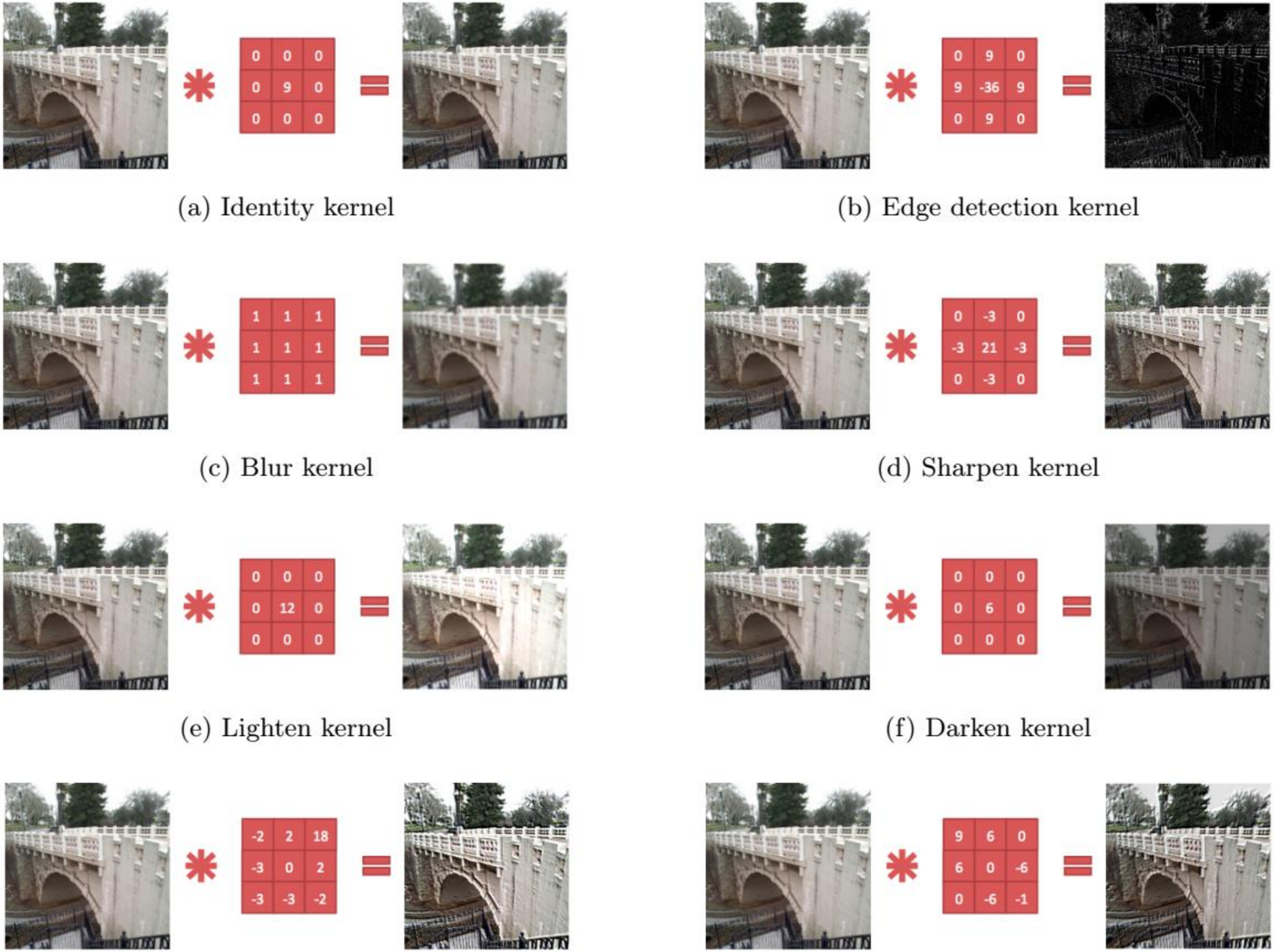
- A feed-forward network inspired from visual cortex
- Used for image recognition
- Objective
  - Find a set of filters which, when convolved with image, lead to the solution of the desired image recognition task
    - Invariant
    - Hierarchical
      - Increasing feature complexity
      - Increasing "Globality"

Basics

- The convolution operation



$$\begin{bmatrix} 22 & 15 & 1 & 3 & 60 \\ 42 & 5 & 38 & 39 & 7 \\ 28 & 9 & 4 & 66 & 79 \\ 0 & 2 & 25 & 12 & 17 \\ 9 & 14 & 2 & 51 & 3 \end{bmatrix} * \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 29 & 12 & 64 \\ 38 & 41 & 32 \\ 13 & 80 & 81 \end{bmatrix}$$



(a) Identity kernel

0	0	0
0	9	0
0	0	0

(b) Edge detection kernel

0	9	0
9	-36	9
0	9	0

(c) Blur kernel

1	1	1
1	1	1
1	1	1

(d) Sharpen kernel

0	-3	0
-3	21	-3
0	-3	0

(e) Lighten kernel

0	0	0
0	12	0
0	0	0

(f) Darken kernel

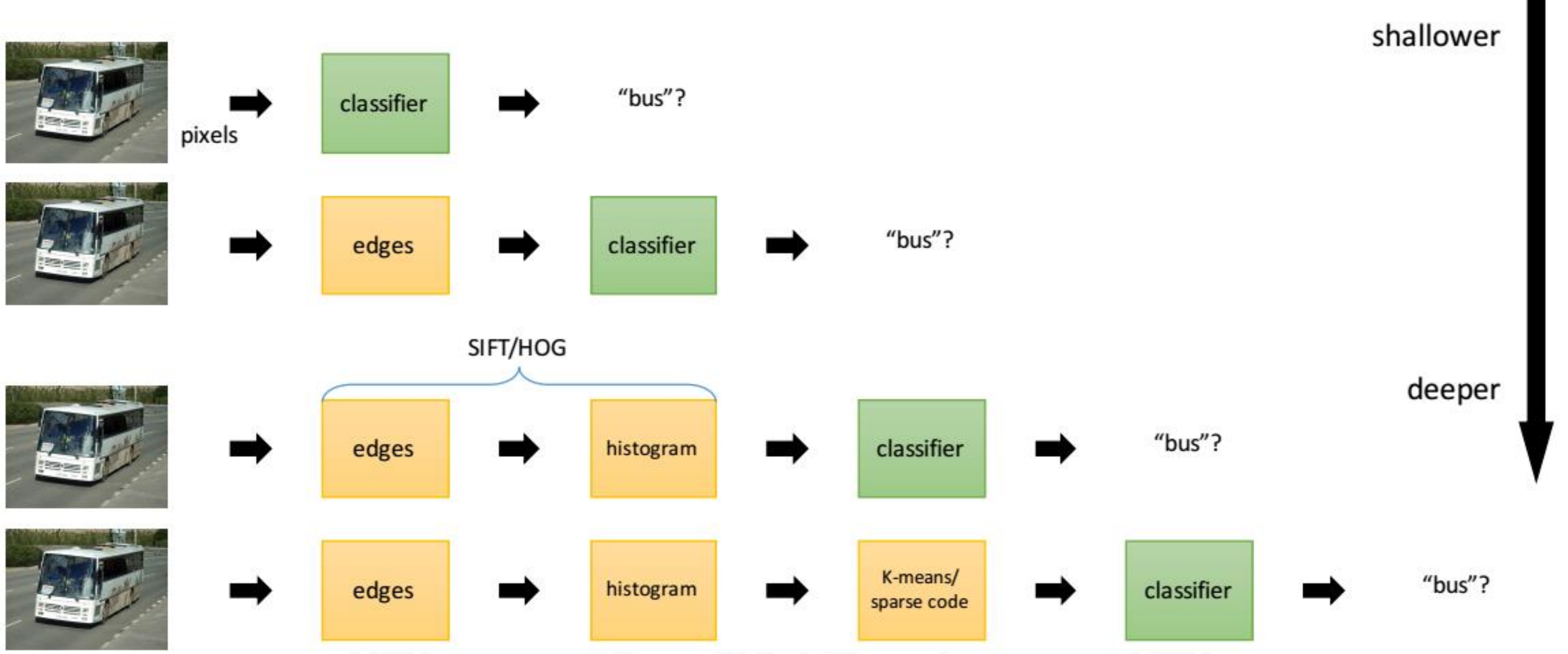
0	0	0
0	6	0
0	0	0

(g) Random kernel 1

-2	2	18
-3	0	2
-3	-3	-2

(h) Random kernel 2

9	6	0
6	0	-6
0	-6	-1



shallower

pixels → classifier → "bus"?

→ edges → classifier → "bus"?

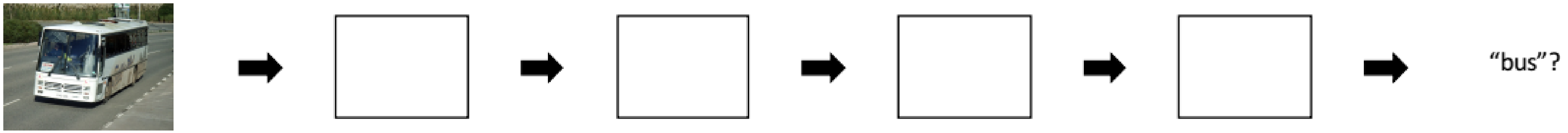
SIFT/HOG

→ edges → histogram → classifier → "bus"?

deeper

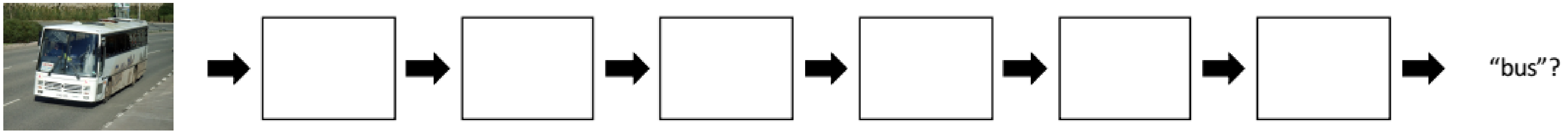
→ edges → histogram → K-means/ sparse code → classifier → "bus"?

Generic components ("layers"), less domain knowledge



"bus"?

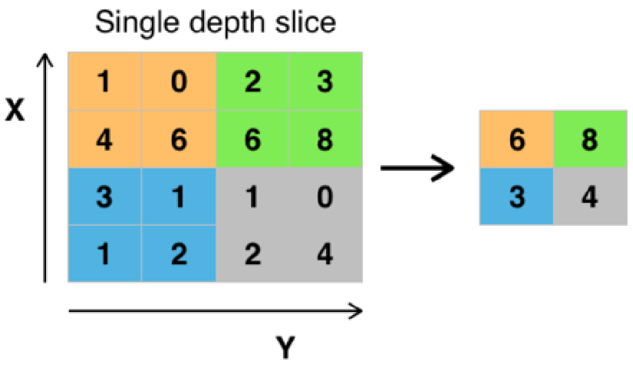
Repeat elementary layers => Going deeper



"bus"?

Structure

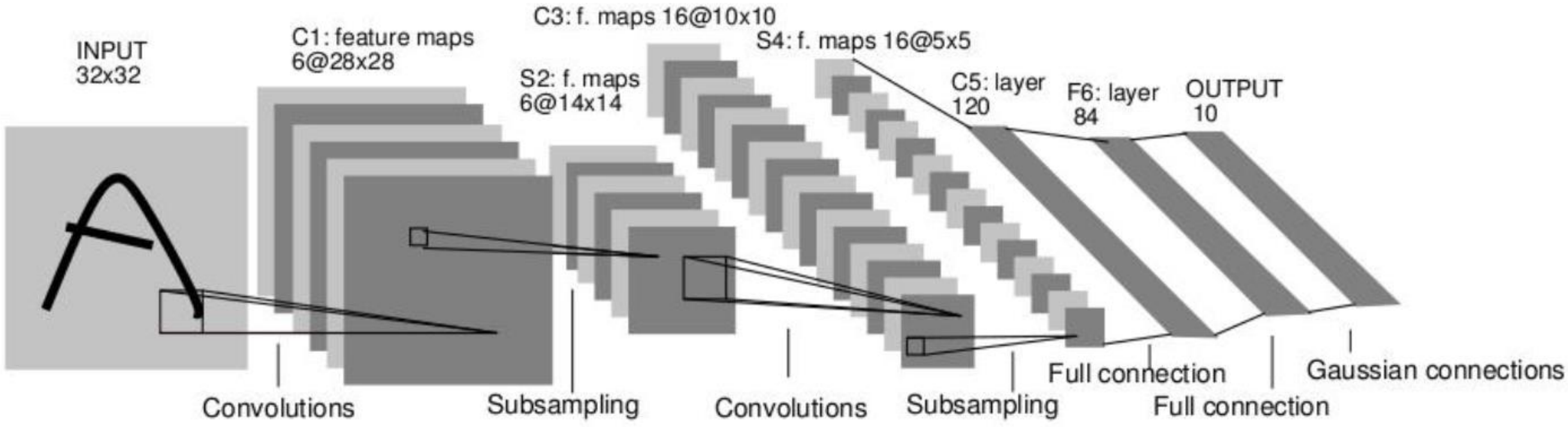
- Increasing “globality”
  - Input → Convolution → Non-linearity → Sub-sampling … → Fully Connected Layer (for classification)



1	0	2	3
4	6	6	8
3	1	1	0
1	2	2	4

→

6	8
3	4



```
import numpy
from keras.datasets import cifar10
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import Flatten
from keras.constraints import maxnorm
from keras.optimizers import SGD
from keras.layers.convolutional import Conv2D
from keras.layers.convolutional import MaxPooling2D
from keras.utils import np_utils
from keras import backend as K
K.set_image_dim_ordering('th')
# fix random seed for reproducibility
seed = 7
numpy.random.seed(seed)

# load data and preprocess it
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train = X_train / 255.0
X_test = X_test / 255.0
y_train = np_utils.to_categorical(y_train)
y_test = np_utils.to_categorical(y_test)
num_classes = y_test.shape[1]
```

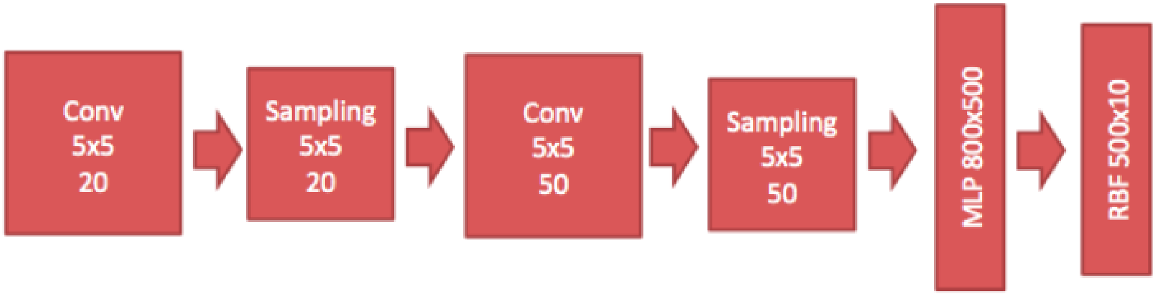
http://machinelearningmastery.com/object-recognition-convolutional-neural-networks-keras-deep-learning-library/

```
# Create the model : Feature Extraction
model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=(3, 32, 32), padding='same', activation='relu',
kernel_constraint=maxnorm(3)))
model.add(Dropout(0.2))
model.add(Conv2D(32, (3, 3), activation='relu', padding='same', kernel_constraint=maxnorm(3)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())

# Create the model : MLP Classification
model.add(Dense(512, activation='relu', kernel_constraint=maxnorm(3)))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

# Compile model
epochs = 25
lrate = 0.01
decay = lrate/epochs
sgd = SGD(lr=lrate, momentum=0.9, decay=decay, nesterov=False)
model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])
print(model.summary())

# Fit the model
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=epochs, batch_size=32)
# Final evaluation of the model
scores = model.evaluate(X_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

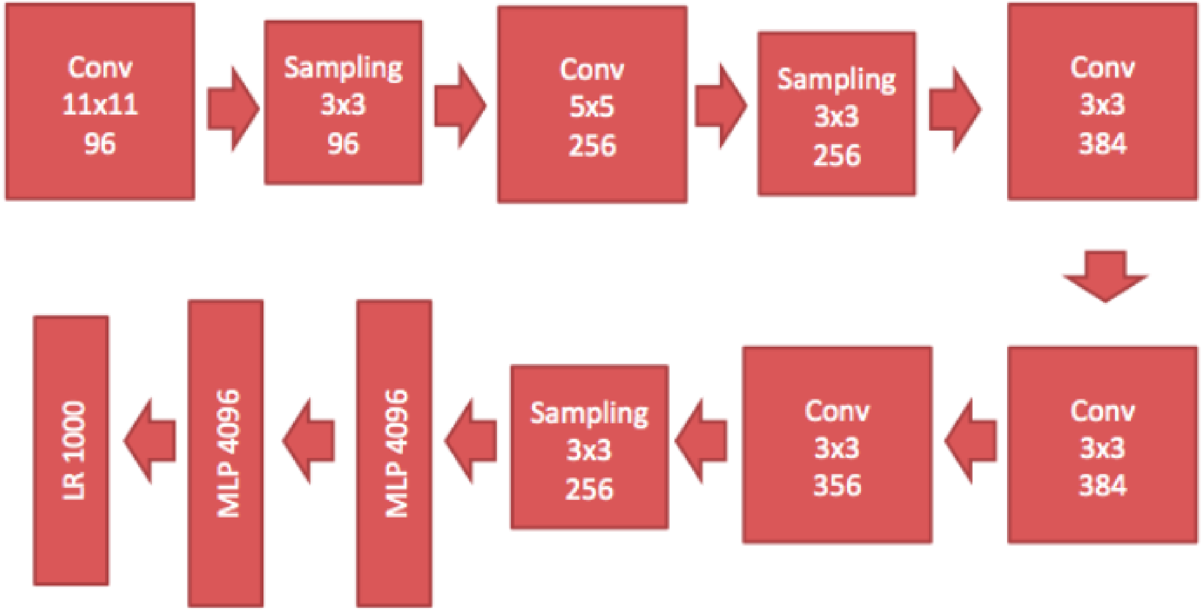
```
_________________________________________________________________
Layer (type)            Output Shape         Param #
=================================================================
conv2d_1 (Conv2D)            (None, 32, 32, 32)       896
_________________________________________________________________
dropout_1 (Dropout)          (None, 32, 32, 32)       0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 32, 32, 32)       9248
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 32, 16, 16)       0
_________________________________________________________________
flatten_1 (Flatten)          (None, 8192)             0
_________________________________________________________________
dense_1 (Dense)              (None, 512)              4194816
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)              0
_________________________________________________________________
dense_2 (Dense)              (None, 10)               5130
=================================================================
Total params: 4,210,090.0
Trainable params: 4,210,090.0
Non-trainable params: 0.0
_________________________________________________________________
```

Famous CNN

- LeNet (Le Cunn 1990, 1998)



- AlexNet



Famous CNN

- VGG19
- Inception
- XCeption

Important Concepts

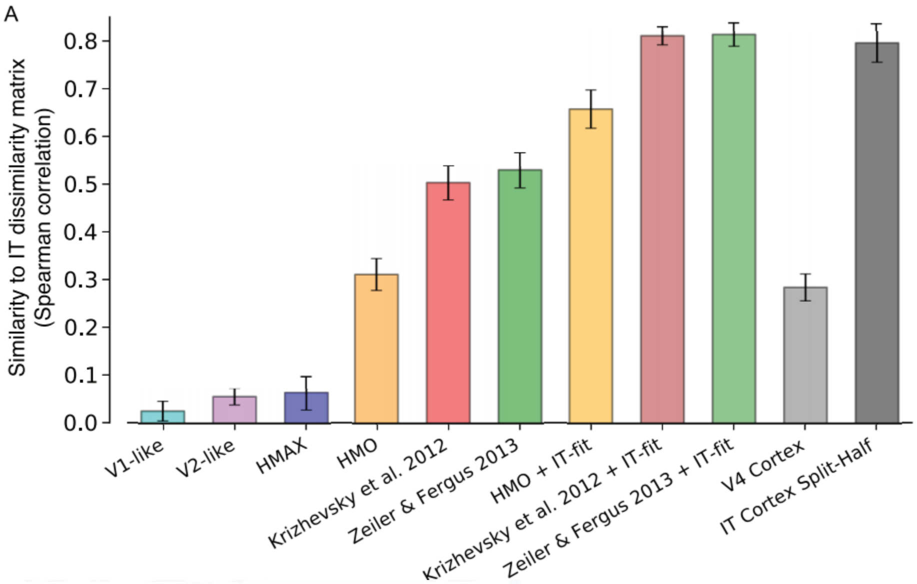
- Differences from fully connected nets
  - 3D volume of neurons
  - Local connectivity
  - Shared weights
- Hyper-parameter
  - Number of filters
  - Filter shape (receptive field)
  - Pooling type and shape
  - Regularization
    - Dropout
    - Data Augmentation
    - Early Stopping
    - Norm constraints
    - L1/L2 regularization

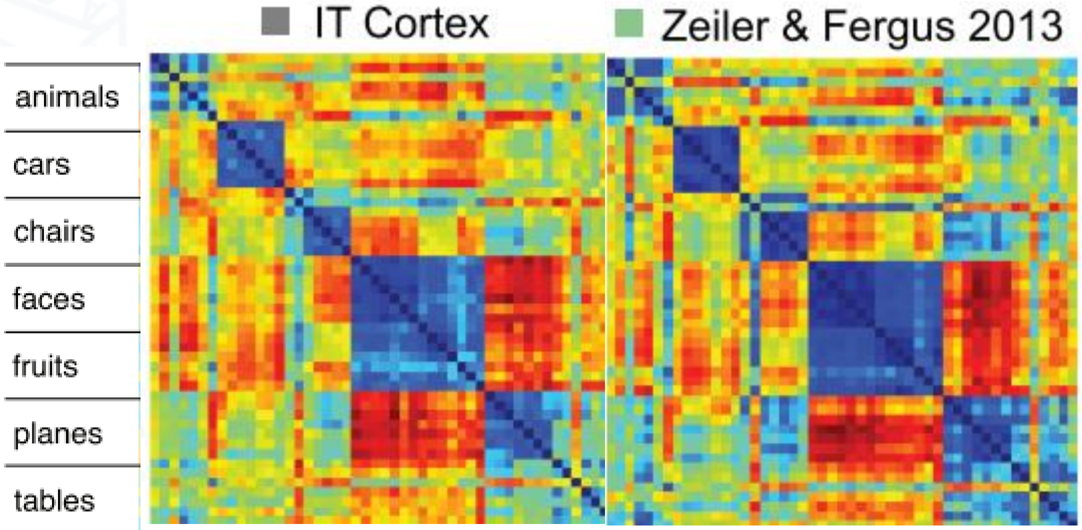
Reading

- Easy Reading: Machine Learning is Fun! Part 3: Deep Learning and Convolutional Neural Networks
  - https://medium.com/@ageitgey/machine-learning-is-fun-part-3-deep-learning-and-convolutional-neural-networks-f40359318721
- Required reading
  - ImageNet Classification with Deep Convolutional Neural Networks
- Results of various methods
  - http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html

Deep Nets vs. Monkey vs. Humans

- “latest DNNs rival the representational performance of IT cortex on this visual object recognition task”
  - Classification of 10 classes of objects
  - Implanted electrodes in the Inferior Temporal (IT) cortex and cortical visual areas (V1-V4) in primates with a linear classifier on the measurements
  - Human Classification
  - Convolutional Deep Neural Networks

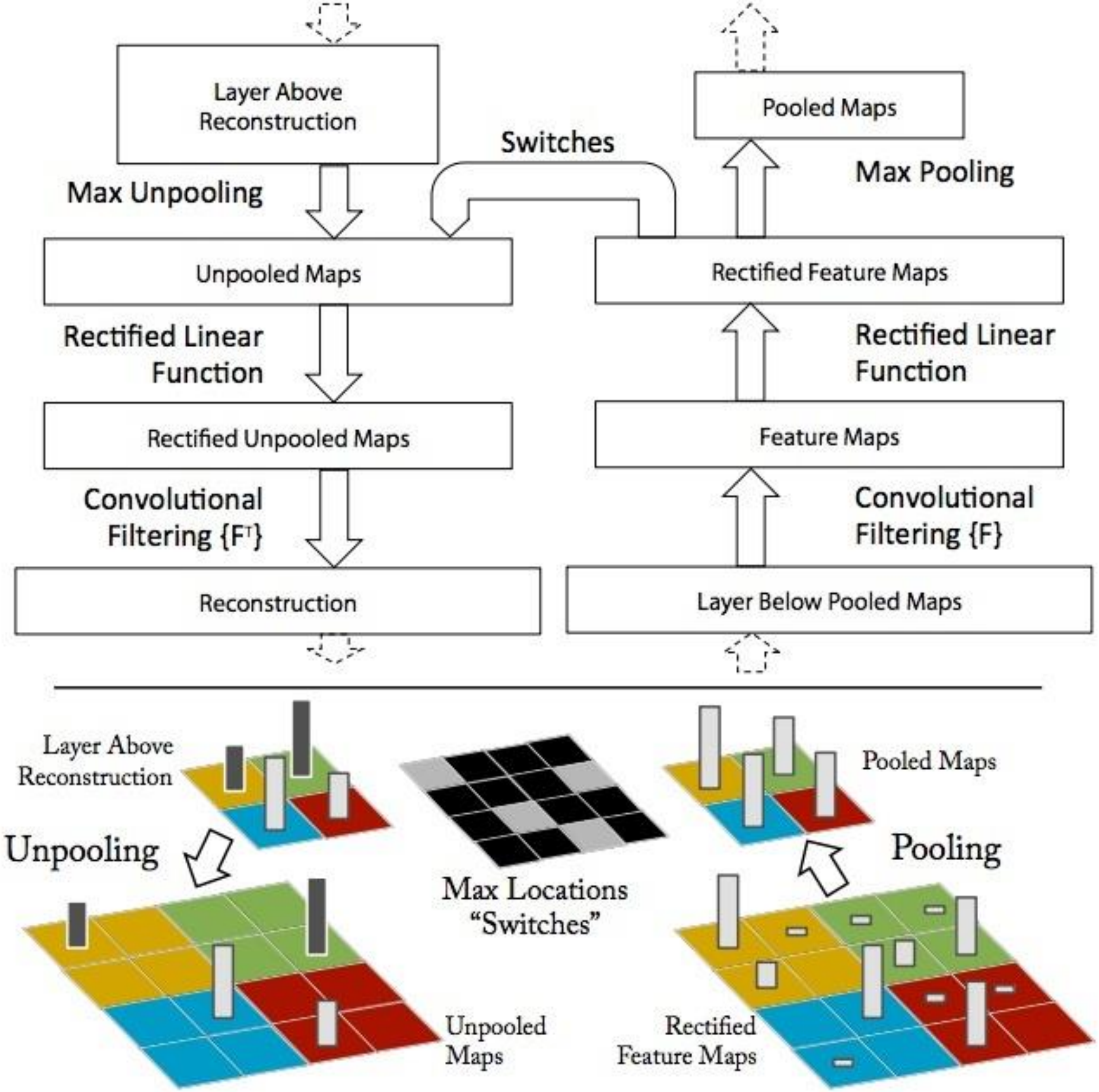




Cadieu CF, Hong H, Yamins DLK, Pinto N, Ardila D, et al. (2014) Deep Neural Networks Rival the Representation of Primate IT Cortex for Core Visual Object Recognition. PLoS Comput Biol 10(12): e1003963. doi:10.1371/journal.pcbi.1003963

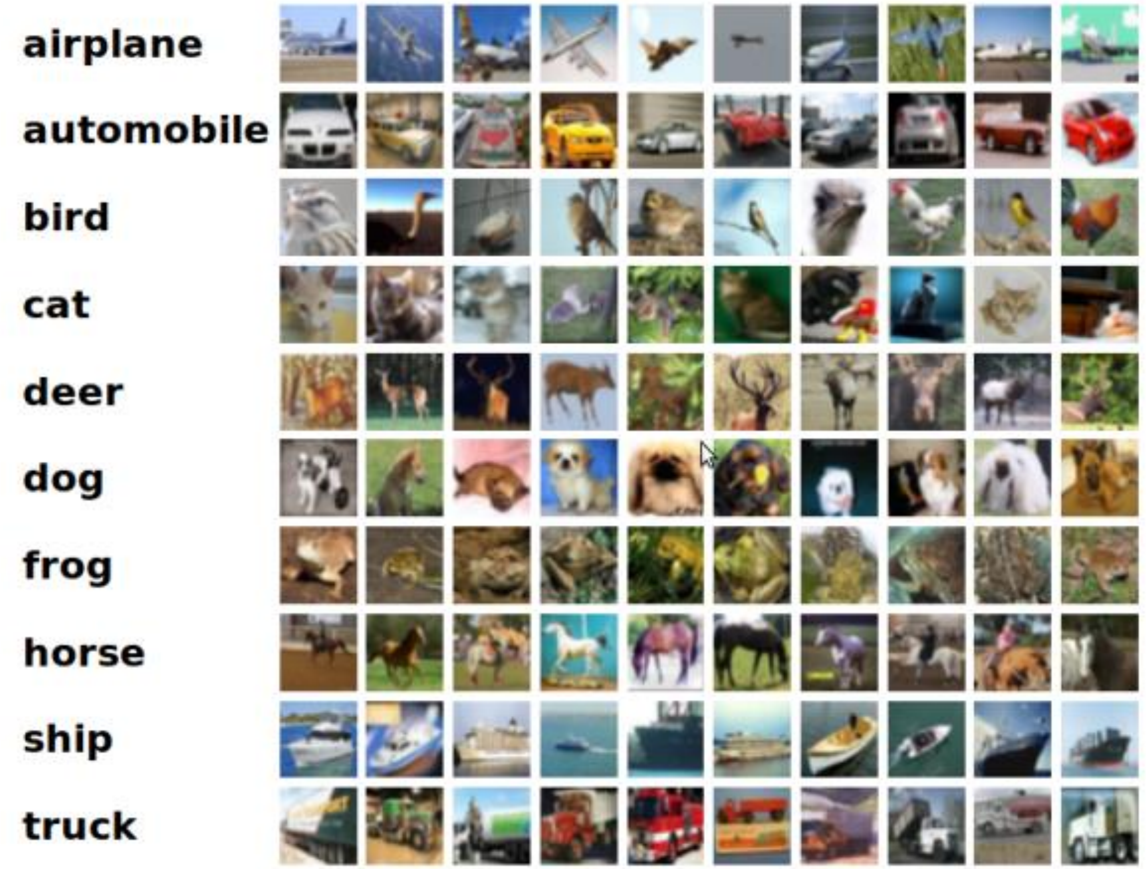
Deconvolution Network

- What is each layer learning?
  - Which filter activation you want to visualize
  - Pass the image forward
  - Zero out all filter activations in the last layer except the one you want to visualize
  - Now go back to the image space but through the deconv net
    - Unpool?
    - ReLU
    - Deconv (transpose the filter)
  - Follow these three steps till you reach the image layer
- Identify what images and what parts of the image activates a filter (or feature) strongly!

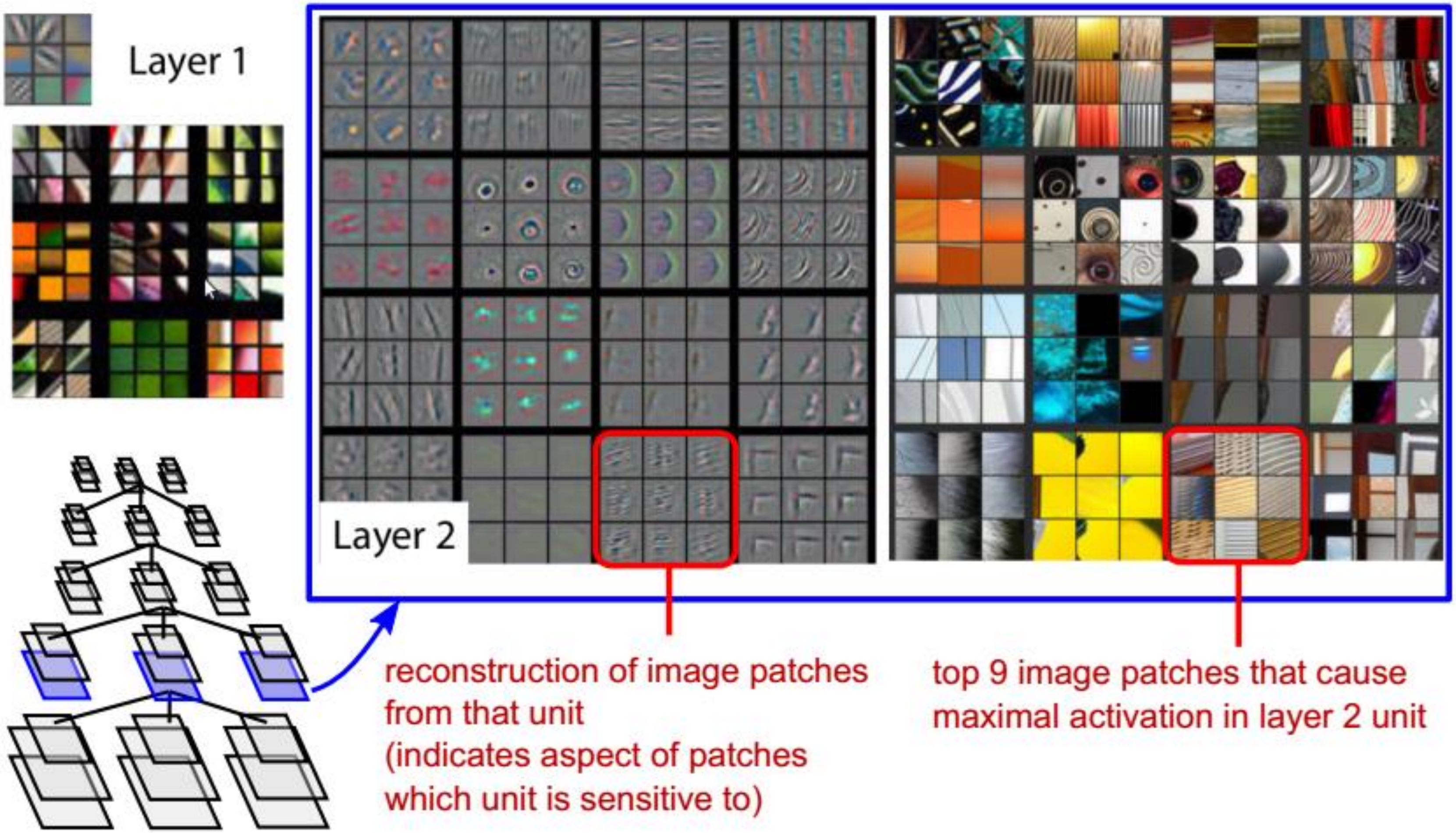


Visalization of a CNN

- **Visualizing and Understanding Convolutional Networks by Zeiler and Fergus, ECCV 2014, Part I, LNCS 8689, pp. 818–833, 2014.**

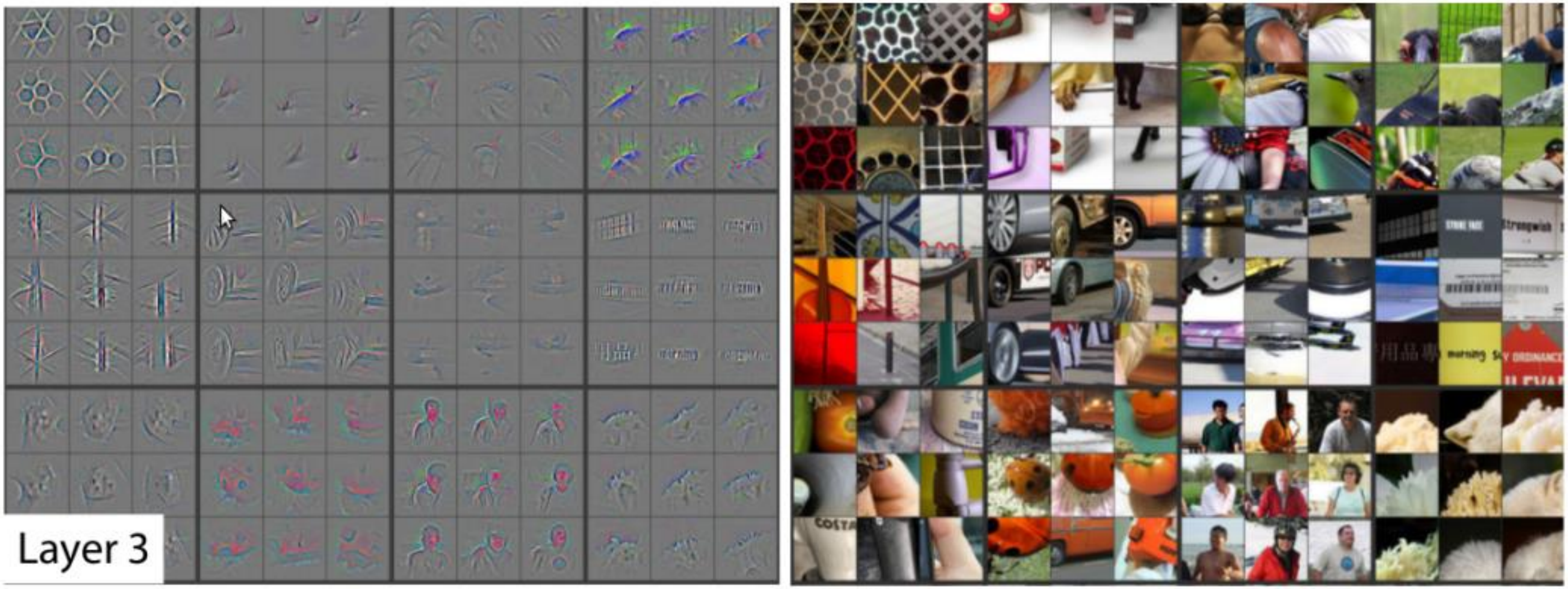


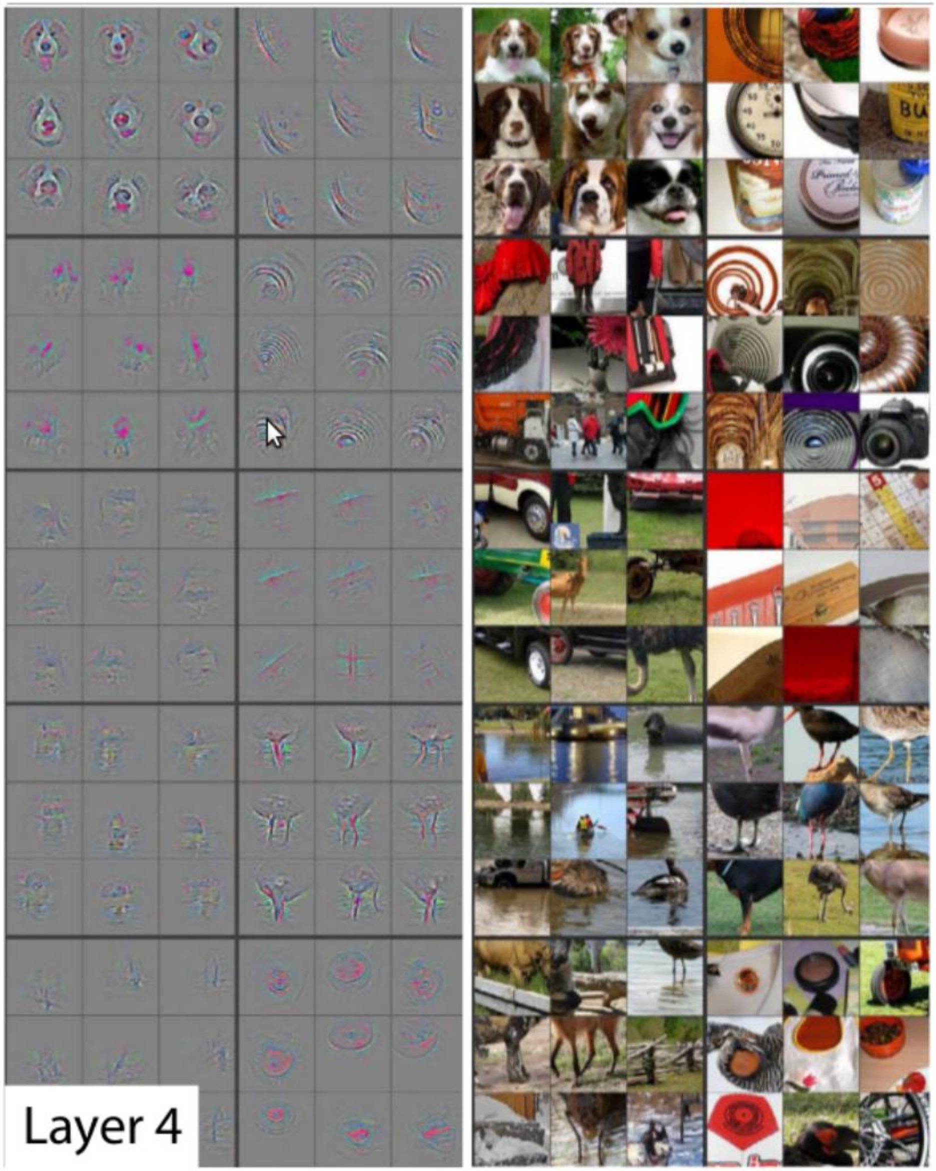
Visualization: https://youtu.be/ghEmQSxT6tw

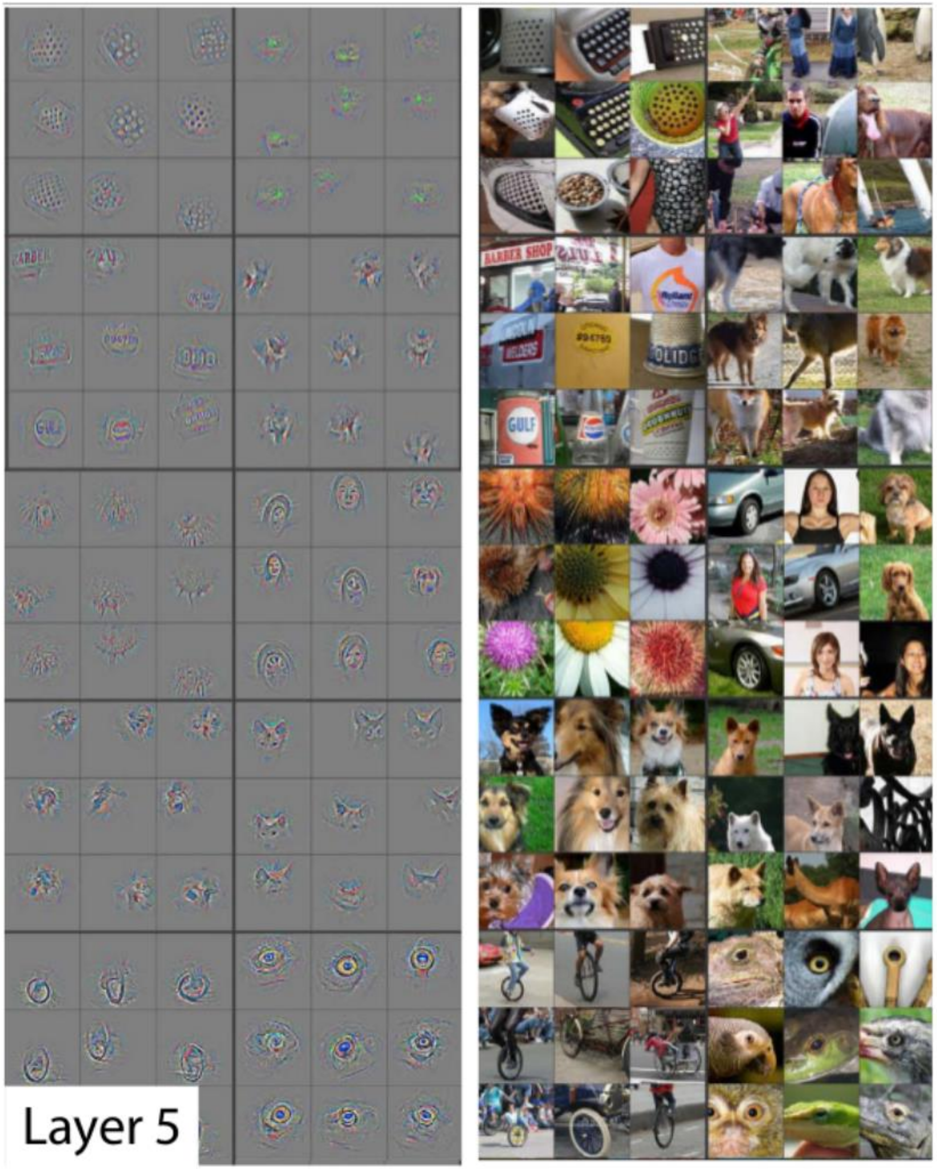


reconstruction of image patches from that unit (indicates aspect of patches which unit is sensitive to)

top 9 image patches that cause maximal activation in layer 2 unit


Layer 3


Layer 4


Layer 5

Fine Tuning/Transfer Learning

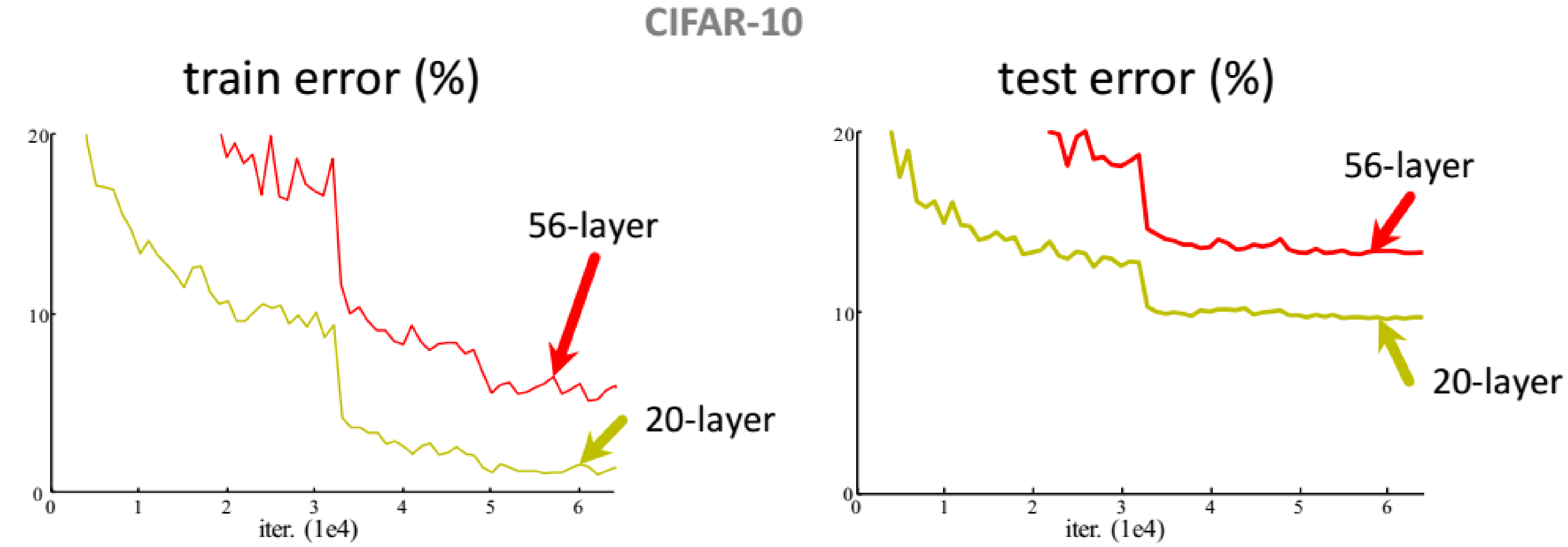
- A form of transfer learning
  - Use a pre-trained neural network for another classification task
    - Modify the weights of the final layers
- Trained image classification models for Keras
  - https://github.com/fchollet/deep-learning-models
- Transfer Learning: Recognition of traffic light
  - (https://medium.freecodecamp.com/recognizing-traffic-lights-with-deep-learning-23dae23287cc)
- Building powerful image classification models using very little data
  - https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html

What’s wrong with Convolutional Neural Networks?

- Watch the talk by Geoff Hinton on the subject:
  - https://www.youtube.com/watch?v=rTawFwUvnLE
- Also watch: The failures of deep learning
  - https://www.youtube.com/watch?v=jWVZnkTfB3c
- Beyond DCNN
  - Gabor Convolutional Networks
    - https://arxiv.org/abs/1705.01450v2
  - Convolutional Sequence to Sequence Learning
    - https://arxiv.org/abs/1705.03122v2
  - Do Deep Convolutional Nets Really Need to be Deep and Convolutional
    - https://arxiv.org/abs/1603.05691v4
  - Picasso: A Neural Network Visualizer
    - https://arxiv.org/abs/1705.05627v1

Increasing Depth (10-100 Layers)

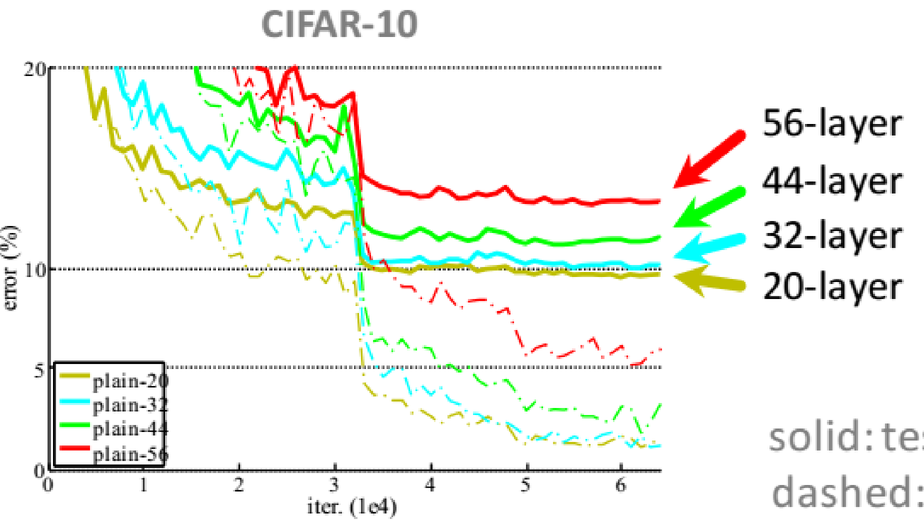
- What if we keep on stacking layers?
  - 56-layer net has **higher training error** and test error than 20-layer net



Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. “Deep Residual Learning for Image Recognition”. CVPR 2016

Simply Stacking Layers?

- "Overly deep" plain nets have **higher training error**
- A general phenomenon, observed in many datasets
- Reasons
  - Optimization failure


CIFAR-10

56-layer

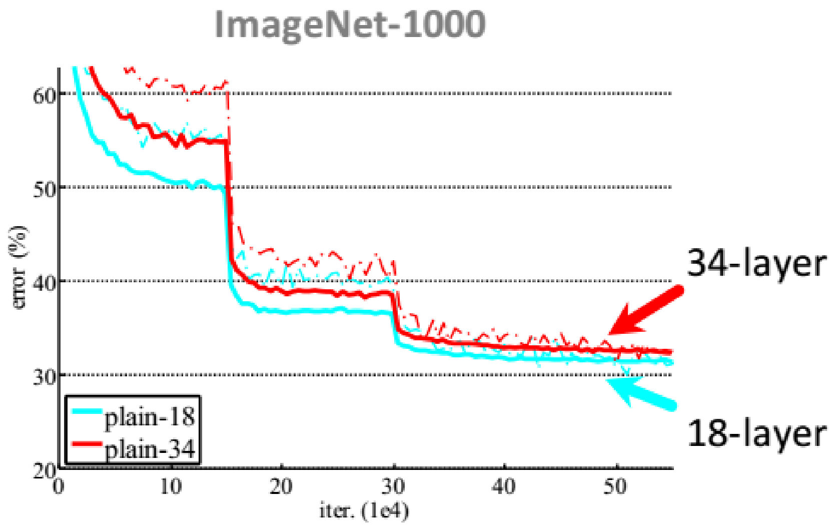
44-layer

32-layer

20-layer

solid: test/val

dashed: train

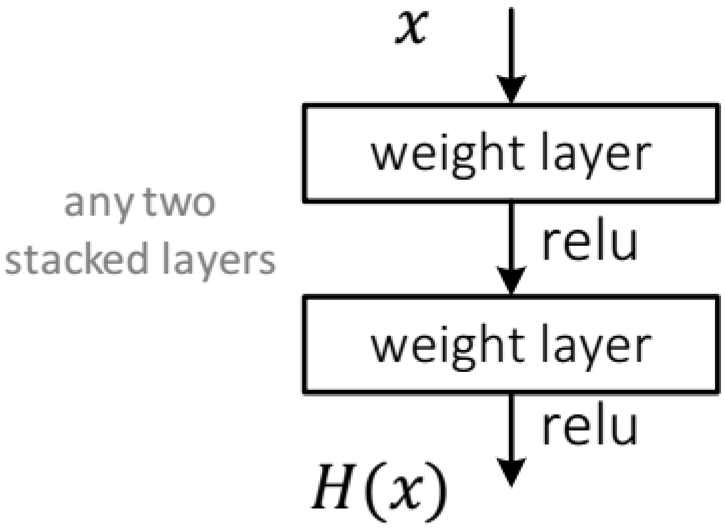

ImageNet-1000

34-layer

18-layer

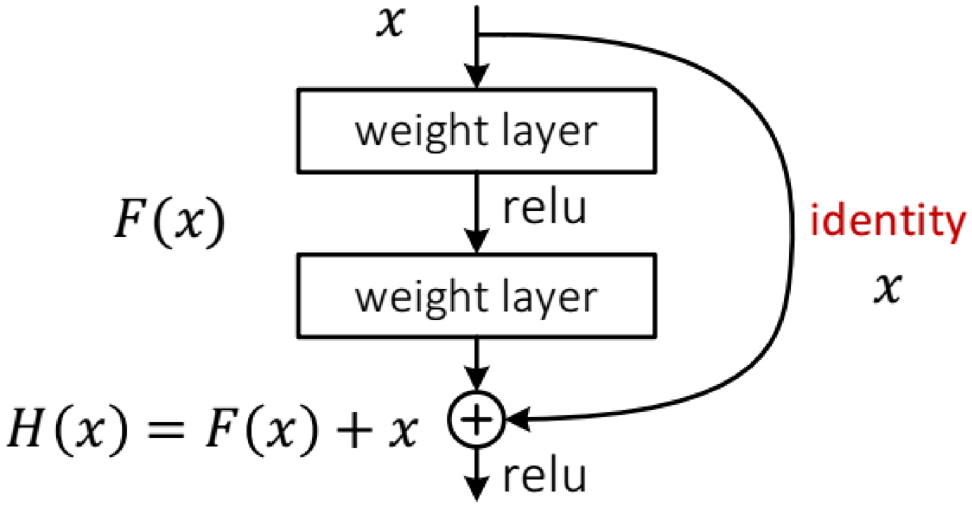
Residual Learning

Plain Network



H(x) is any desired mapping
Hope the 2 weight layers fit *H(x)*

Residual Network



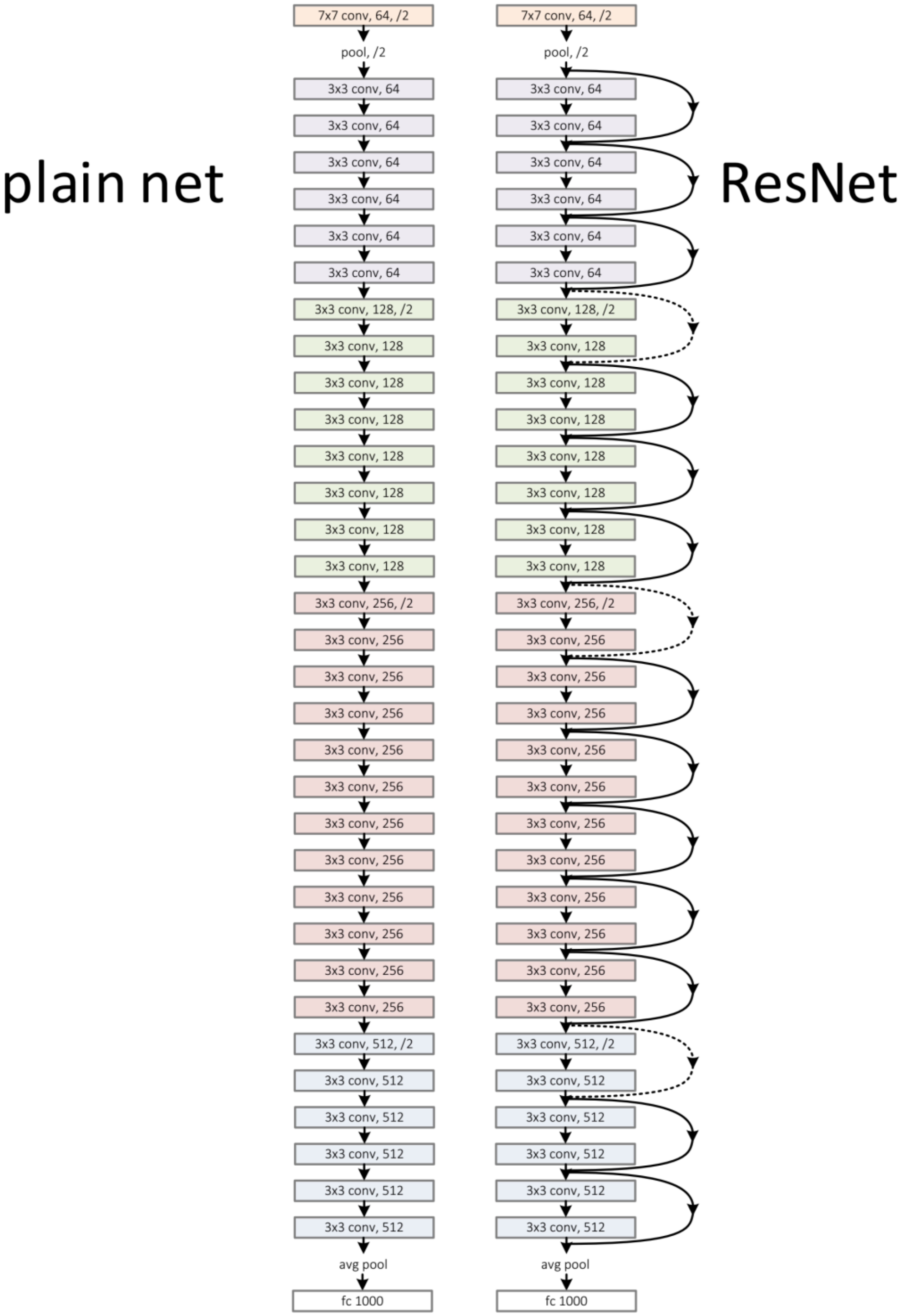
H(x) is any desired mapping
Hope the 2 weight layers fit *F(x)*

The network learns fluctuations *F(x)=H(x)-x*
Easier!

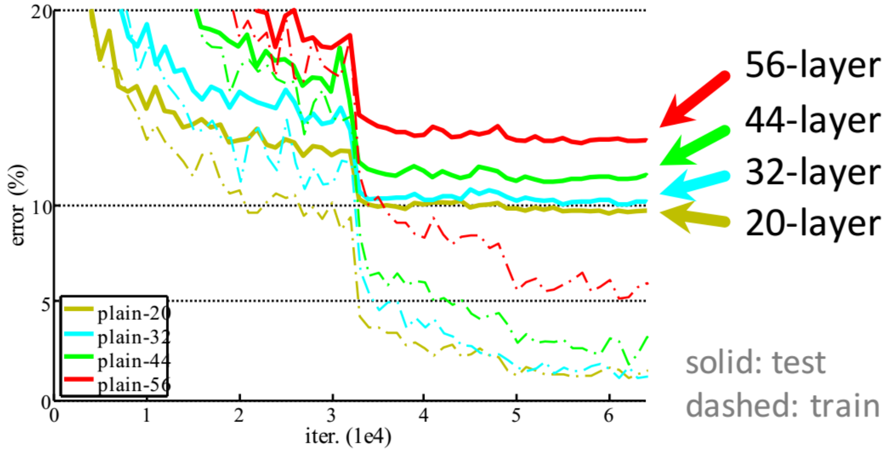
Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". CVPR 2016.

ResNet Models

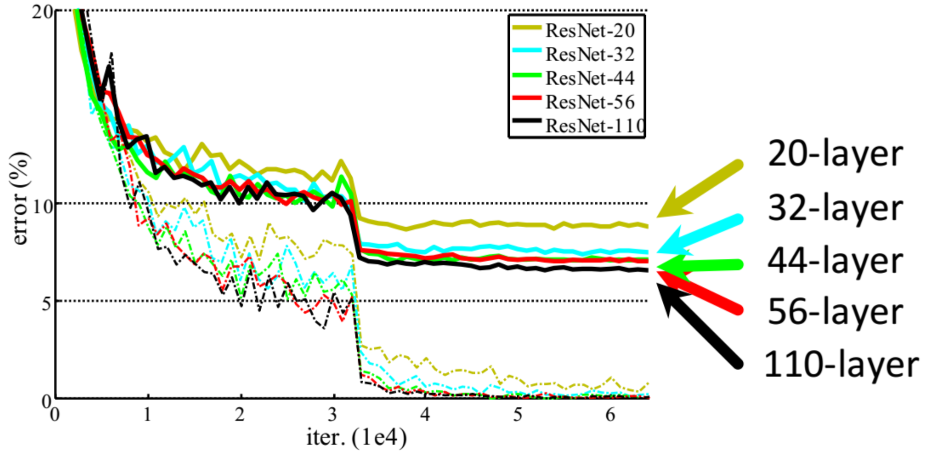
- No Dropout
- With Batch Normalization
- Use Data Augmentation



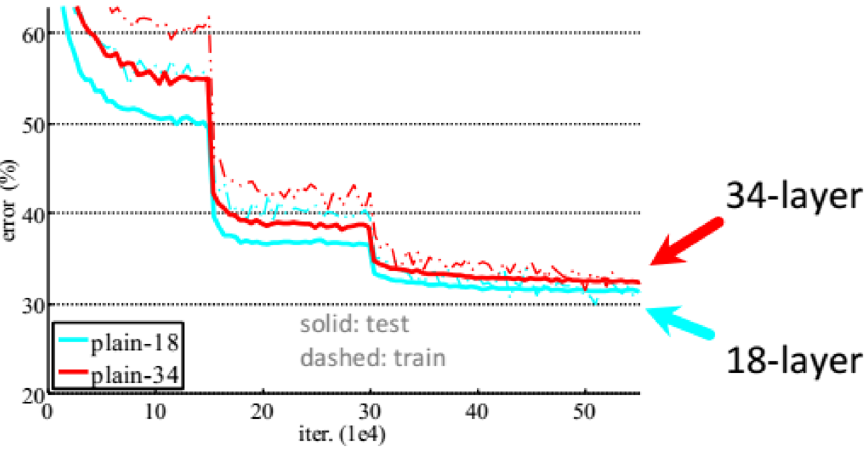
CIFAR-10 plain nets



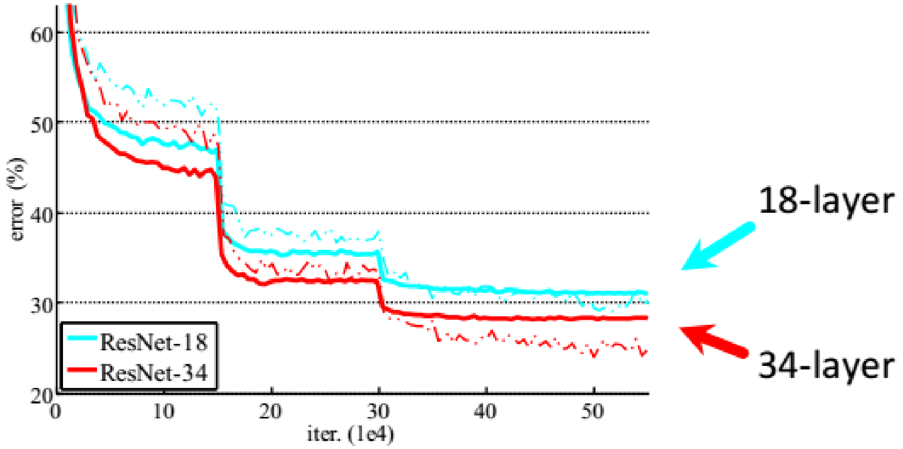
CIFAR-10 ResNets



ImageNet plain nets



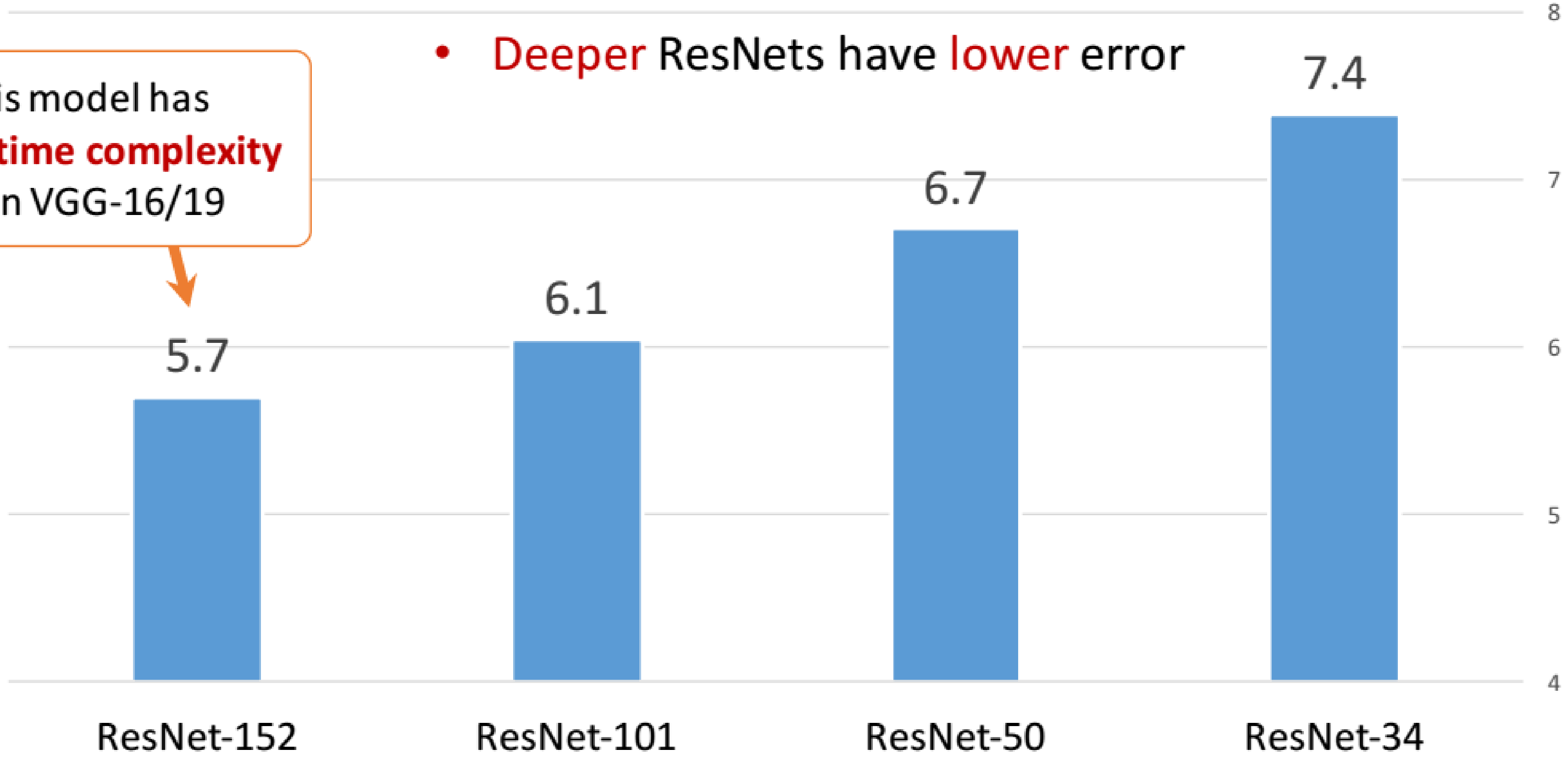
ImageNet ResNets



- Deep ResNets can be trained without difficulties
- Deeper ResNets have **lower training error**, and also lower test error

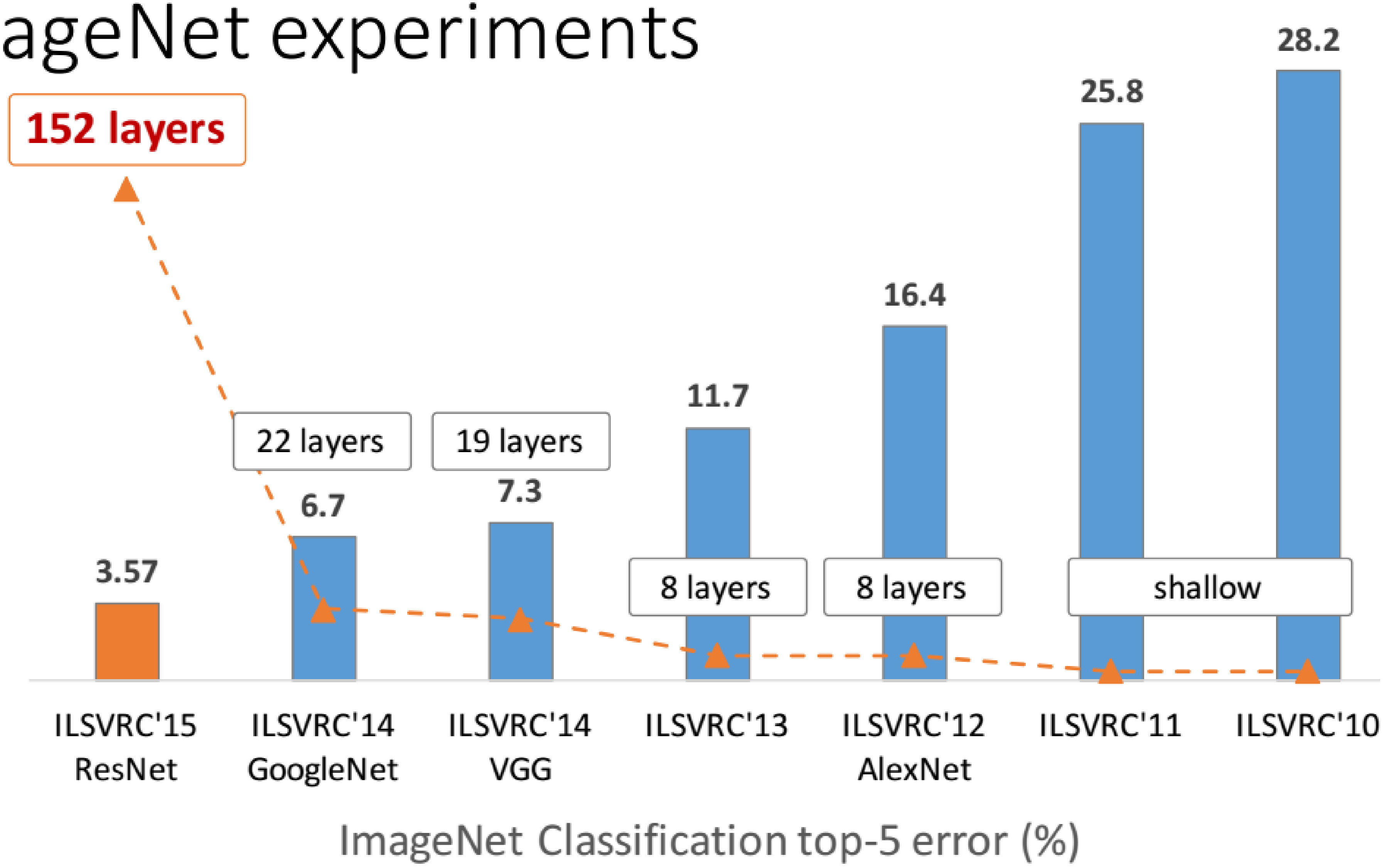
- Deeper ResNets have lower error

this model has
lower time complexity
than VGG-16/19



10-crop testing, top-5 val error (%)

ImageNet experiments



ResNet Results

- **1st places in all five main tracks**
  - ImageNet Classification: “*Ultra-deep*” 152-layer nets
  - ImageNet Detection: 16% better than 2nd
  - ImageNet Localization: 27% better than 2nd
  - COCO Detection: 11% better than 2nd
  - COCO Segmentation: 12% better than 2nd

Residual Networks

- Required Reading
  - Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". CVPR 2016.
- Many third-party implementations
  - list in https://github.com/KaimingHe/deep-residual-networks
  - Torch ResNet: https://github.com/facebook/fb.resnet.torch

Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". CVPR 2016.
Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Identity Mappings in Deep Residual Networks". arXiv 2016.

Beyond ResNets

- Residual Networks Behave Like Ensembles of Relatively Shallow Networks
  - https://arxiv.org/abs/1605.06431v2
- Fractal Networks
  - https://arxiv.org/abs/1605.07648
- Deep Stochastic Networks
  - https://arxiv.org/abs/1603.09382

Moving towards Generative Models

- Uptil now our models have been discriminatory
  - – Discriminate between classes
- Generative Models
  - – Models that can be used to generate examples!

Generative Models

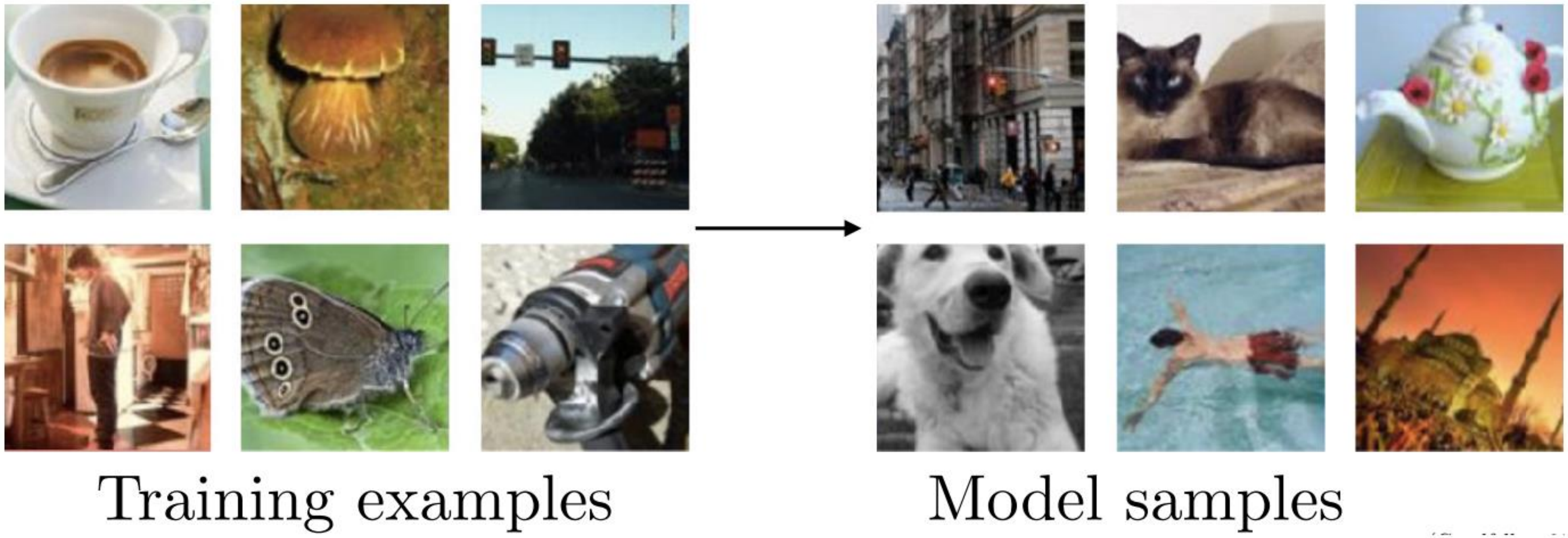
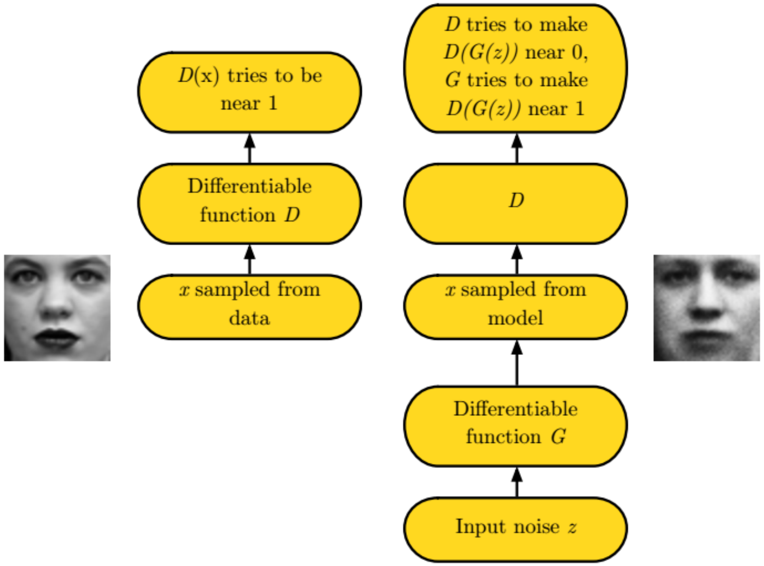


Figure 2: Some generative models are able to generate samples from the model distribution. In this illustration of the process, we show samples from the ImageNet (Deng et al., 2009, 2010; Russakovsky et al., 2014) dataset. An ideal generative model would be able to train on examples as shown on the left and then create more examples from the same distribution as shown on the right. At present, generative models are not yet advanced enough to do this correctly for ImageNet, so for demonstration purposes this figure uses actual ImageNet data to illustrate what an ideal generative model would produce.

NIPS 2016 Tutorial: Generative Adversarial Networks, Ian Goodfellow , https://arxiv.org/abs/1701.00160

Generative Adversarial Networks

- Also known as Turing Learning
- Unsupervised Learning for generating realistic examples
- Consists of two networks
  - Discriminator Network (D)
    - Given a data set
    - If an example has been chosen from the dataset, then D tries to output a probability value of 1.0
    - If the example is fake, then produce 0.0
  - Generator Network (G)
    - Input: Random noise
    - Output: An example resembling the examples in the dataset
    - The differentiator tries to produce a value of 0.0 for examples generated from the network
    - The objective of G is to produce images for which D produces high probabilities
    - Performs



$D(x)$ tries to be near 1
Differentiable function D
x sampled from data
D tries to make $D(G(z))$ near 0, G tries to make $D(G(z))$ near 1
D
x sampled from model
Differentiable function G
Input noise z

NIPS 2016 Tutorial: Generative Adversarial Networks by Ian Goodfellow, 2016
https://arxiv.org/abs/1701.00160

Generative Adversarial Networks
https://arxiv.org/abs/1406.2661

GANs Applications: Super-resolution Imaging

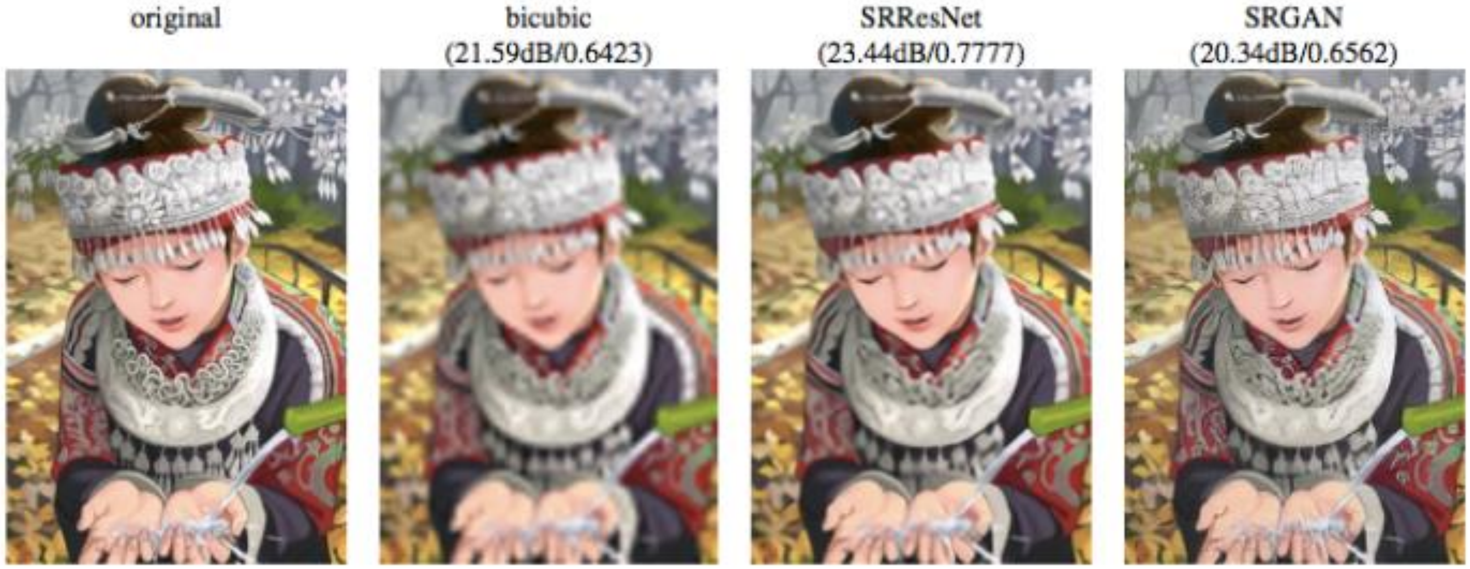


Figure 4: Ledig *et al.* (2016) demonstrate excellent single-image superresolution results that show the benefit of using a generative model trained to generate realistic samples from a multimodal distribution. The leftmost image is an original high-resolution image. It is then downsampled to make a low-resolution image, and different methods are used to attempt to recover the high-resolution image. The bicubic method is simply an interpolation method that does not use the statistics of the training set at all. SRResNet is a neural network trained with mean squared error. SRGAN is a GAN-based neural network that improves over SRGAN because it is able to understand that there are multiple correct answers, rather than averaging over many answers to impose a single best output.

GAN: Manipulation of images

- Interactive Image Generation, Modification and Warping
  - https://youtu.be/9c4z6YsBGQ0

Image to Image Translation

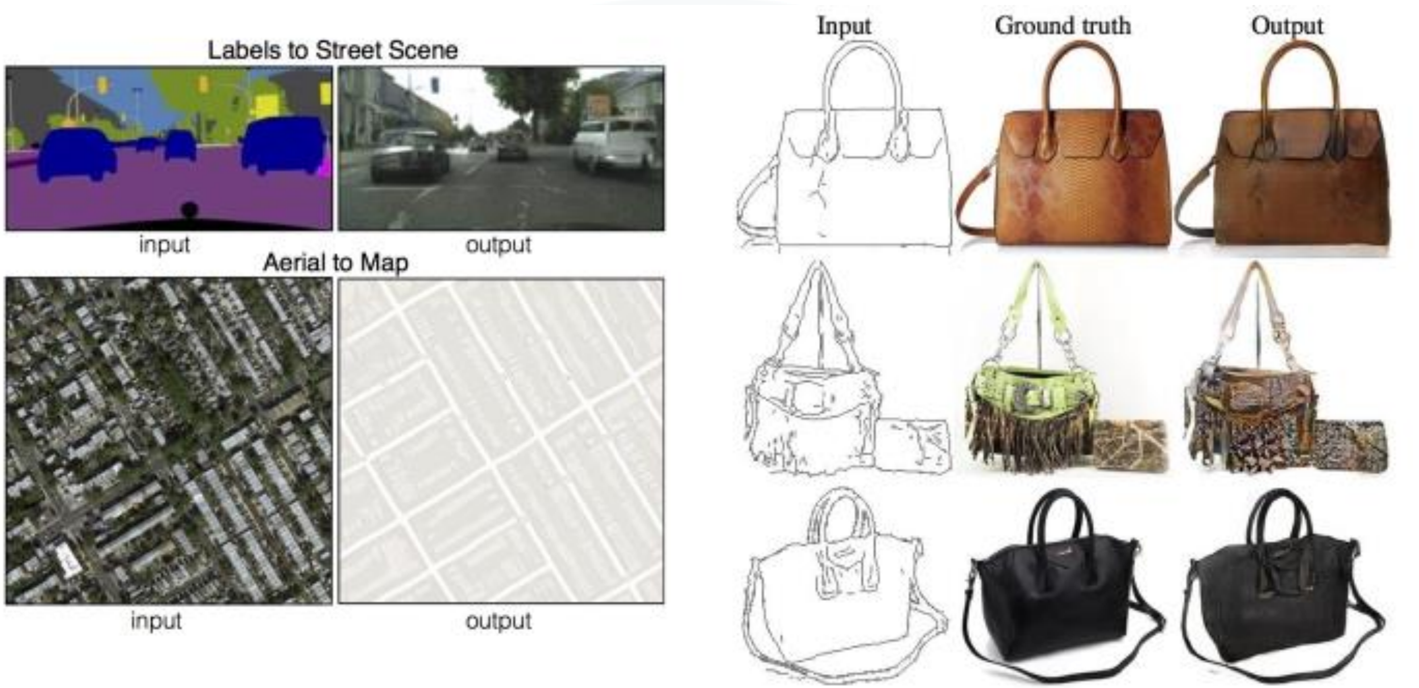
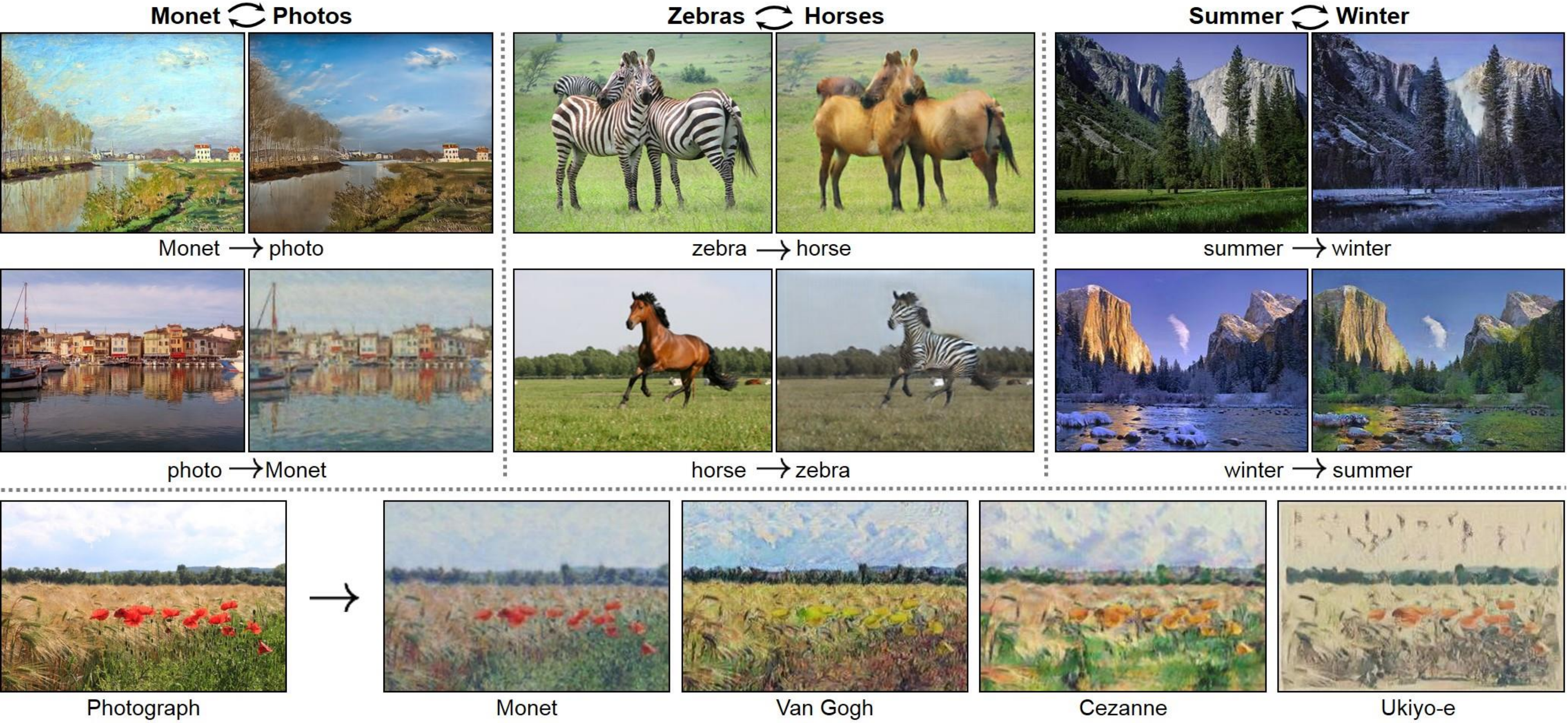


Figure 7: Isola *et al.* (2016) created a concept they called image to image translation, encompassing many kinds of transformations of an image: converting a satellite photo into a map, coverting a sketch into a photorealistic image, etc. Because many of these conversion processes have multiple correct outputs for each input, it is necessary to use generative modeling to train the model correctly. In particular, Isola *et al.* (2016) use a GAN. Image to image translation provides many examples of how a creative algorithm designer can find several unanticipated uses for generative models. In the future, presumably many more such creative uses will be found.

Neural Style Transfer

- Using CycleGAN
  - https://github.com/junyanz/CycleGAN

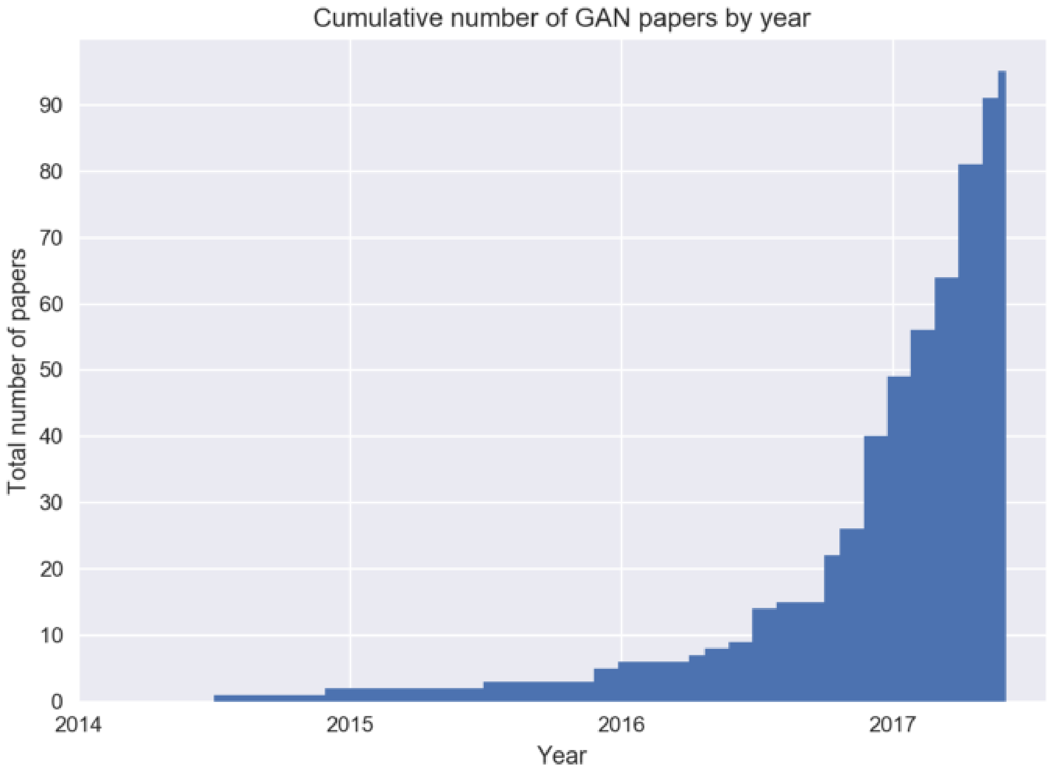


GAN Applications

- https://lyrebird.ai/demo
- Copy anyone’s voice!

GAN Zoo

- https://deephunt.in/the-gan-zoo-79597dc8c347

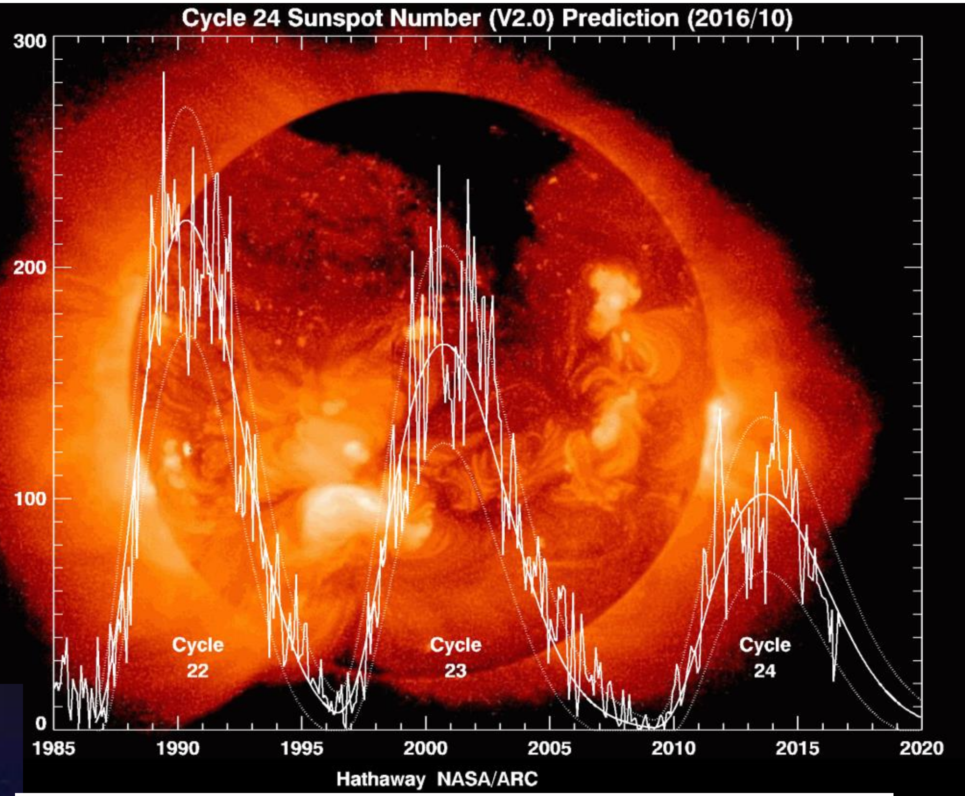


Applications

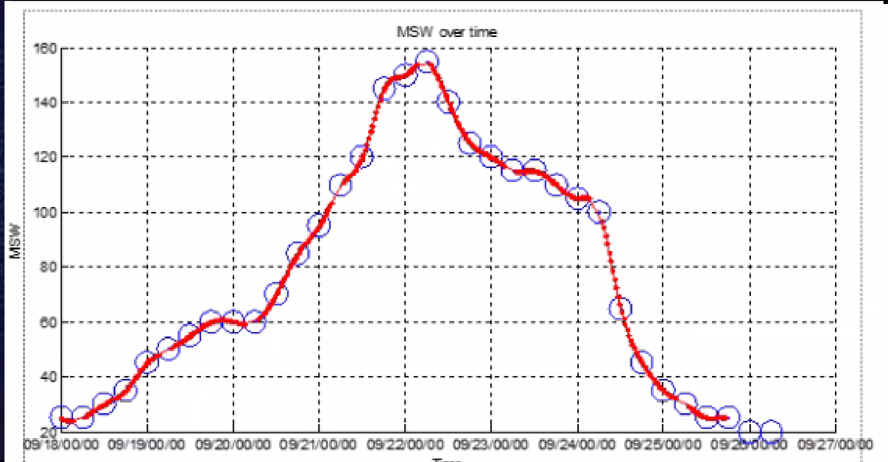
- 9 Cool Deep Learning Applications | Two Minute Papers
- https://youtu.be/Bui3DWs02h4?list=PLujxSBD-JXgnqDD1n-V30pKtp6Q886x7e
- https://www.youtube.com/watch?v=aKSILzbAqJs&index=65&list=PLujxSBD-JXgnqDD1n-V30pKtp6Q886x7e
- You said that? https://arxiv.org/abs/1705.02966v1

Predicting Temporal Data

- Predicting time series data
  - Number of sunspots
  - Hurricane intensity
- Mathematically,
- $h_t = f(\boldsymbol{x_1}, \boldsymbol{x_2}, \dots \boldsymbol{x_t}; h_1, h_2, \dots h_{t-1})$
  - f_t should approximate true values y_t for all times in the future

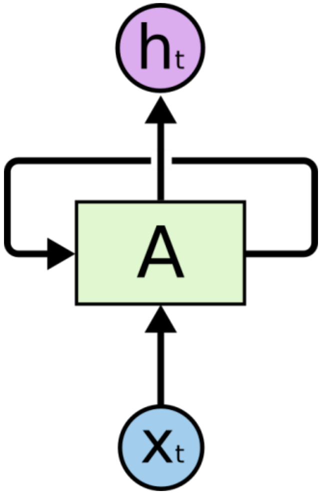


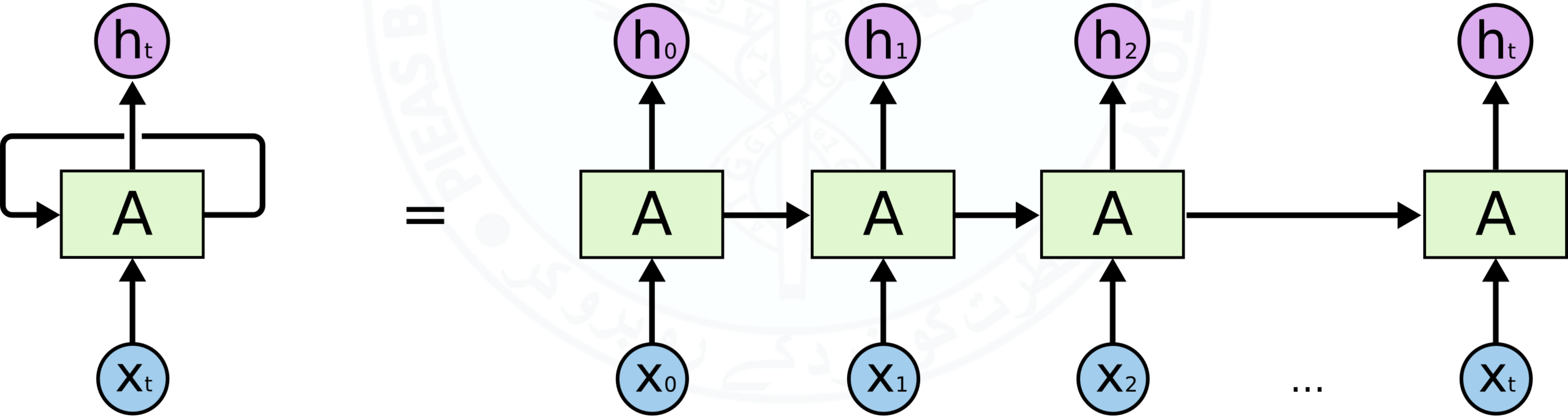
HURRICANE KATRINA
AUG 23-30, 2005
1,836 KILLED, 705 MISSING
DAMAGE: $90.1 BILLION (2010 USD)
WINDS TO 175 MPH
INTENSITY: 902 mb
HURRICANE
TROPICAL STORM
TROPICAL DEPRESSION
29th CAT 3 2nd Landfall
28th CAT 5
25th CAT 1 1st Landfall



Using Recurrent Networks

- Given the input $\mathbf{x}_t$ at time t and the previous outputs, predict the current output using a neural network A
- We can unroll the network and do "backpropagation through time"





http://colah.github.io/posts/2015-08-Understanding-LSTMs/

"The Unreasonable Effectiveness of Recurrent Neural Networks." http://karpathy.github.io/2015/05/21/rnn-effectiveness/.

Applications

- Spellings
- Grammar
- Learning to write text
- Poetry
- Write code
- Predicting time series

Issues with RNNs

- Fill in the blanks:
  - The car is on the ______.
  - The clouds are in the _____.
    - The gap between relevant information and the place where it is needed is small – easy for RNN to learn
  - Bismillah and Adiba are doing their projects with Dr. Fayyaz Minhas. The are classmates. They sit together in the lab. The project reports are due tomorrow and must be submitted to the supervisor for review prior to final submission. Bismillah and Adiba will submit their reports to ________.
    - Irrelevant information
    - Gap between relevant information and the place where it is used is larger
    - RNNs will have difficulty here.
- Yoshua Bengio, et al., Learning Long-Term Dependencies with Gradient Descent is Difficult, 1994.
- Long term dependencies are an issue
  - Solution: Long-Short Term Memories (LSTM)

Requirements for Recurrent Neural Network

- That the system be able to store information for an arbitrary duration.
- That the system be resistant to noise (i.e. fluctuations of the inputs that are random or irrelevant to predicting a correct output).
- That the system parameters be trainable (in reasonable time).

RNN to LSTM

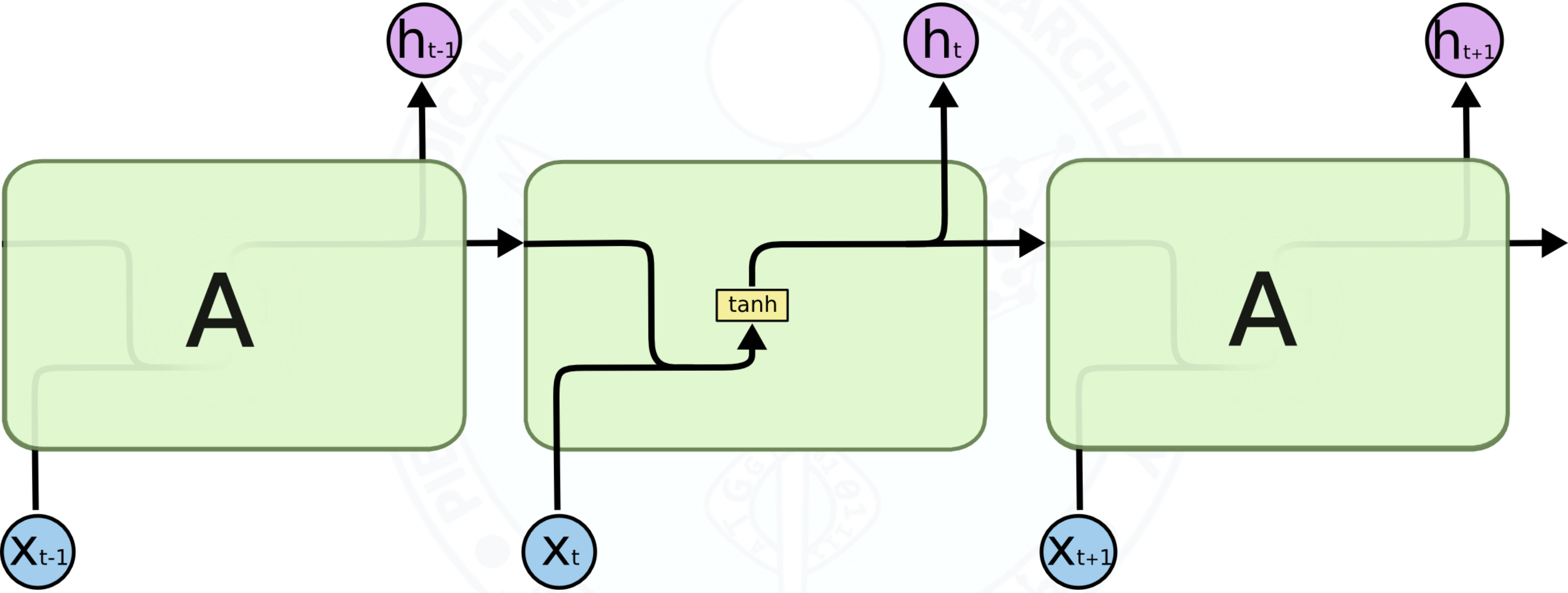
- “Hence standard RNNs fail to learn in the presence of time lags greater than 5 – 10 discrete time steps between relevant input events and target signals. The vanishing error problem casts doubt on whether standard RNNs can indeed exhibit significant practical advantages over time window-based feedforward networks. A recent model, “Long Short-Term Memory” (LSTM), is not affected by this problem. LSTM can learn to bridge minimal time lags in excess of 1000 discrete time steps by enforcing constant error ow through “constant error carrousels” (CECs) within special units, called cells”
  - Felix A. Gers, et al., Learning to Forget: Continual Prediction with LSTM, 2000

RNN to LSTM

- “Unfortunately, the range of contextual information that standard RNNs can access is in practice quite limited. The problem is that the influence of a given input on the hidden layer, and therefore on the network output, either decays or blows up exponentially as it cycles around the network’s recurrent connections. This shortcoming … referred to in the literature as the vanishing gradient problem … Long Short-Term Memory (LSTM) is an RNN architecture specifically designed to address the vanishing gradient problem.”
  - Alex Graves, et al., A Novel Connectionist System for Unconstrained Handwriting Recognition, 2009

LSTM (Hochreiter & Schmidhuber 1997)

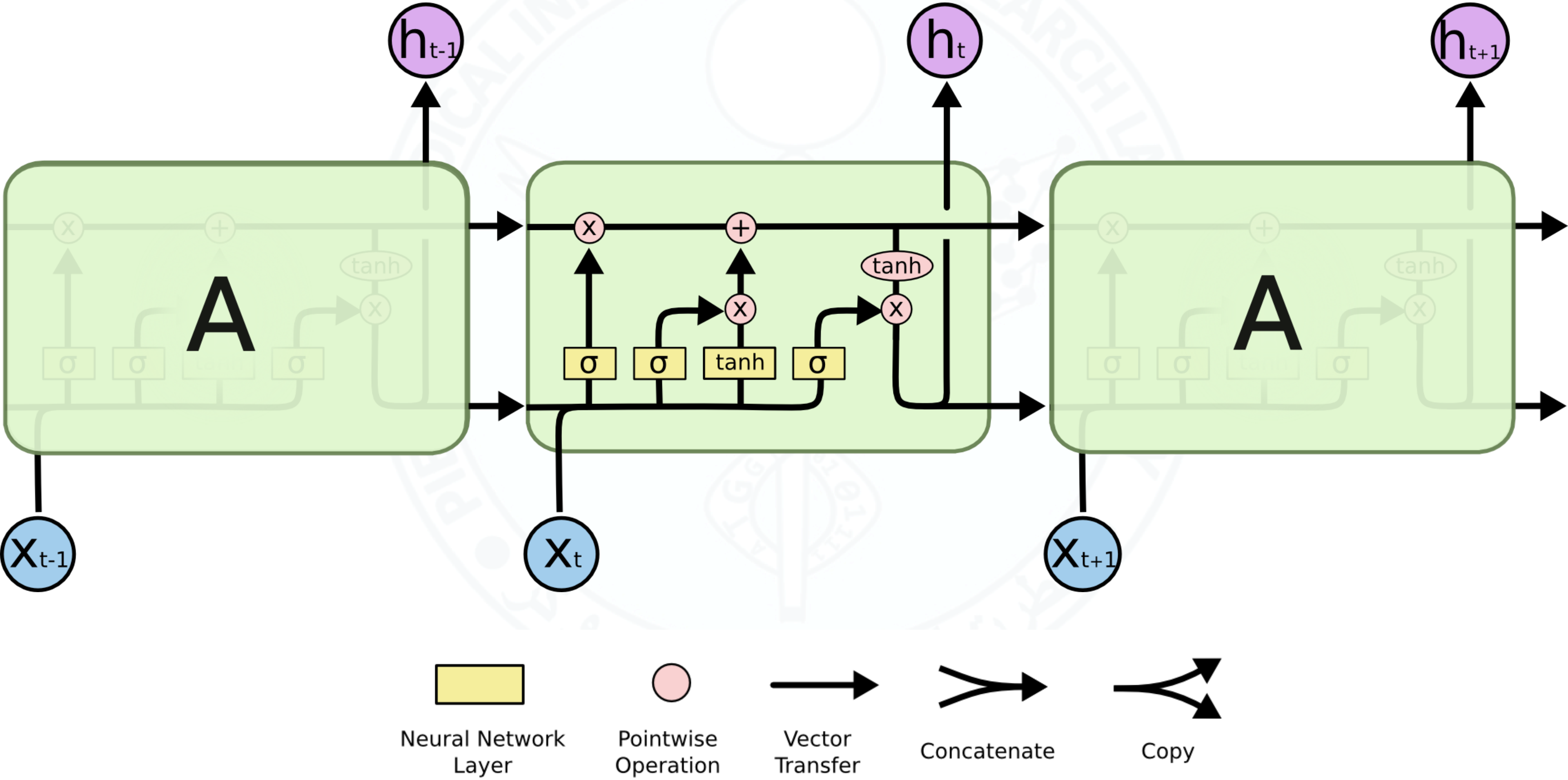
- RNNs can be represented as



The repeating module in a standard RNN contains a single layer.

$$h_t = tanh(\langle W, [h_{t-1}, x_t]\rangle)$$

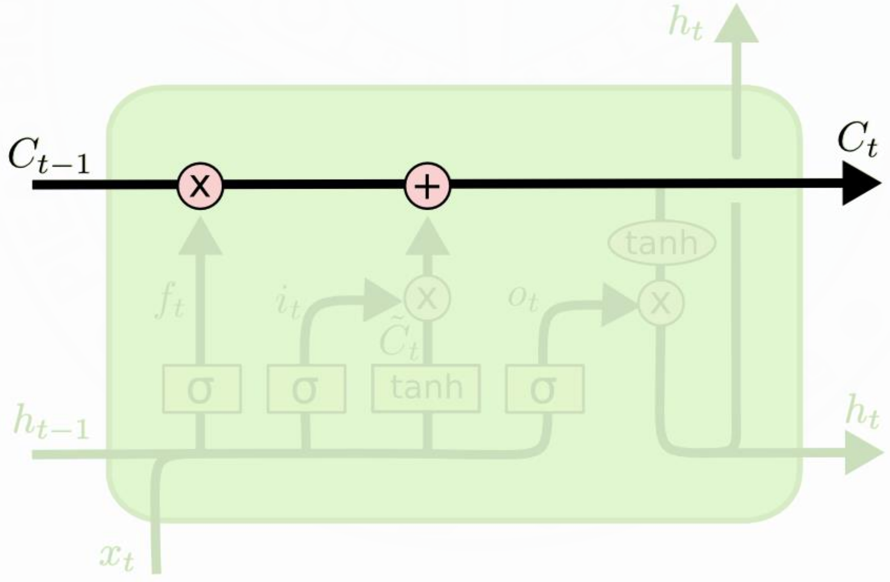
LSTM



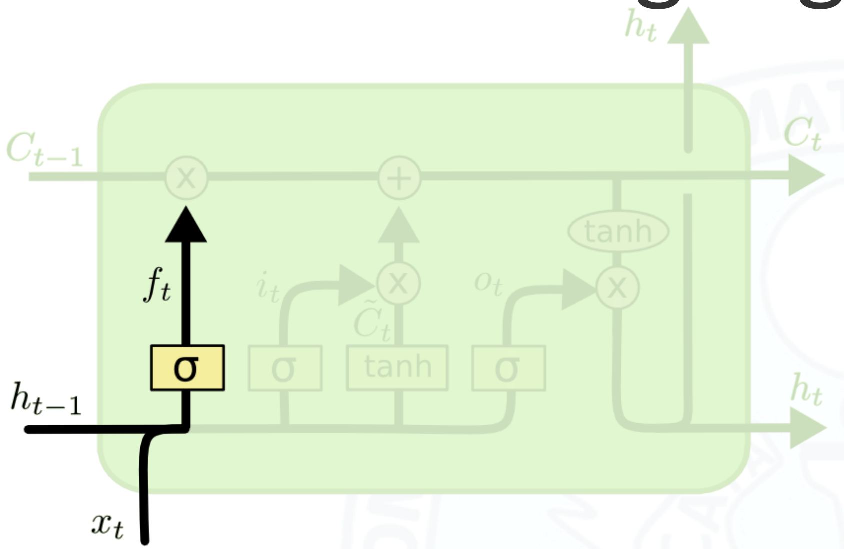
h_{t-1}
h_t
h_{t+1}
A
σ
σ
tanh
σ
x
+
x
x
tanh
A
X_{t-1}
X_t
X_{t+1}
Neural Network Layer
Pointwise Operation
Vector Transfer
Concatenate
Copy

LSTM: Cell State

- Each cell’s output is dependent on its cell state which is “gated”, i.e., information can be added or removed from the state



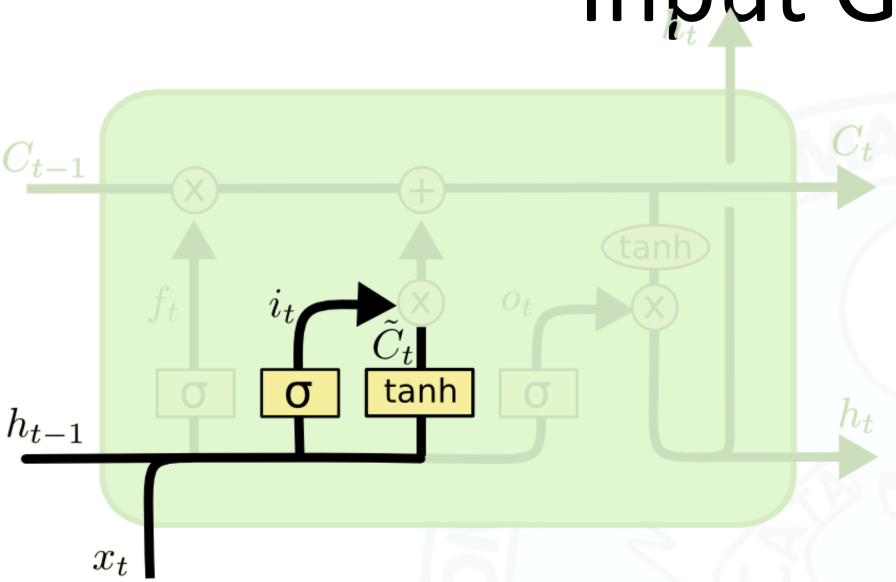
Forget gate layer



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

- It looks at ht-1 and xt and outputs a number between 0 and 1 which is multiplied with the cell state Ct-1 in an element-wise manner
  - In a language model trying to predict the next word based on all the previous ones. In such a problem, the cell state might include the gender of the present subject, so that the correct pronouns can be used. When we see a new subject, we want to forget the gender of the old subject.

Input Gate Layer

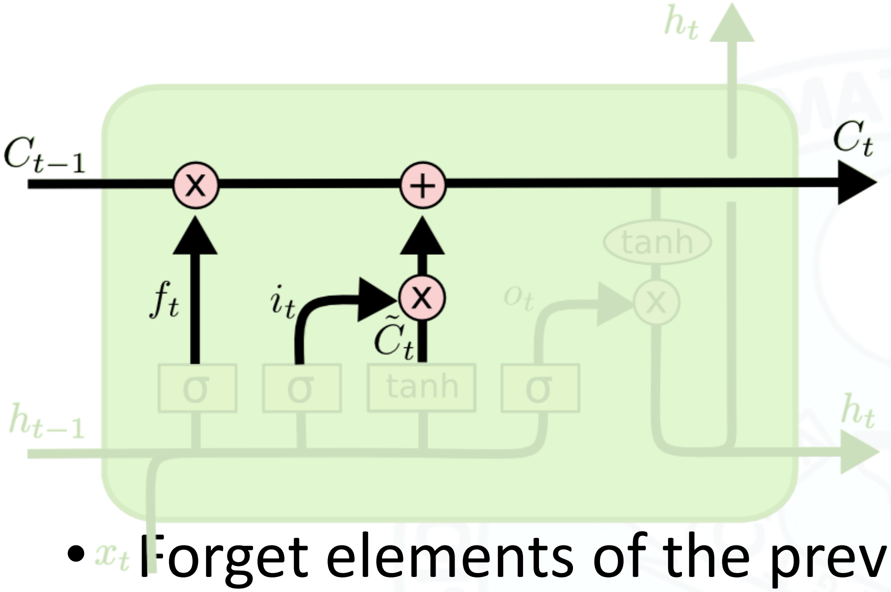


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

- The next step is to decide what new information we're going to store in the cell state. This has two parts.
  - First, a sigmoid layer called the "input gate layer" decides which values will be updated.
  - Next, a tanh layer creates a vector of new candidate values, $\widetilde{C}_t$, that could be added to the state. In the next step, we'll combine these two to create an update to the state.
- In the example of our language model, we'd want to add the gender of the new subject to the cell state, to replace the old one we're forgetting.

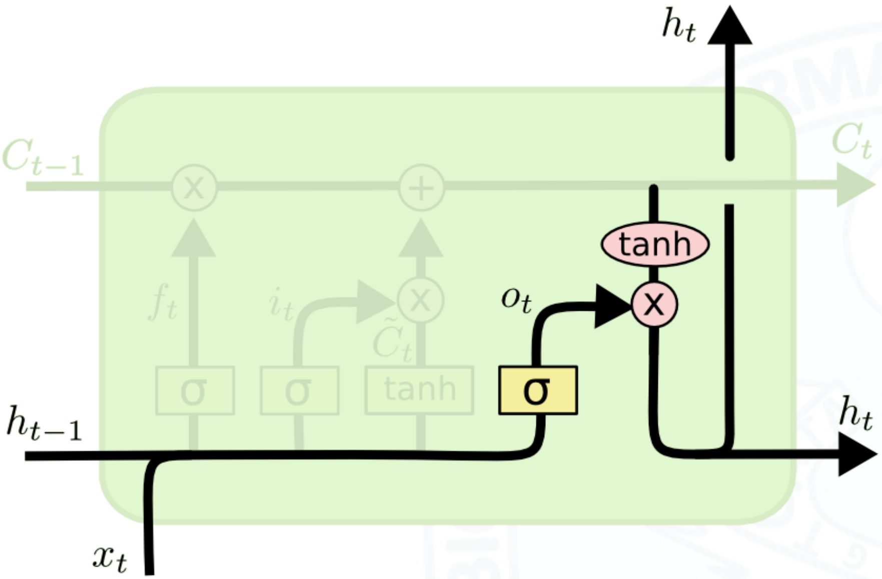
Cell State Update



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

- Forget elements of the previous cell state
- Create a tentative new cell state based on the current time cell, scale it by how much each element is to be updated and then add it to the gated previous cell state
- In the case of the language model, this is where we'd actually drop the information about the old subject's gender and add the new information, as we decided in the previous steps.

Generate Predictions



$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

- Output is filtered version of the cell state

Applications

- Time Series Prediction with LSTM Recurrent Neural Networks in Python with Keras
  - http://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras/
- Using MLP
  - http://machinelearningmastery.com/time-series-prediction-with-deep-learning-in-python-with-keras/
- Time Series Forecasting with the Long Short-Term Memory Network in Python
  - http://machinelearningmastery.com/time-series-forecasting-long-short-term-memory-network-python/

Applications

- **https://www.quora.com/What-are-the-various-applications-where-LSTM-networks-have-been-successfully-used**
- Language modeling (The tensorflow tutorial on PTB is a good place to start Recurrent Neural Networks) character and word level LSTM's are used
- Machine Translation also known as sequence to sequence learning (https://arxiv.org/pdf/1409.3215.pdf)
- Image captioning (with and without attention, https://arxiv.org/pdf/1411.4555v...)
- Hand writing generation (http://arxiv.org/pdf/1308.0850v5...)
- Image generation using attention models - my favorite (https://arxiv.org/pdf/1502.04623...)
- Question answering (http://www.aclweb.org/anthology/...)
- Video to text (https://arxiv.org/pdf/1505.00487...)

Transfer Learning

- http://sebastianruder.com/transfer-learning/
- Transfer Learning - Machine Learning's Next Frontier

DL Libraries

- pyTorch
  - Imperative Programming
    - Run, Run, Run…
  - Dynamic Computing Graphs
    - Graph built at run time
    - Build as you go
  - Good for research
- TensorFlow
  - Symbolic
    - Compile then run/fit
  - Static Computing Graphs
    - Build before you go
  - Good Documentation
  - Distributed Computing / Delivery
  - TensorFlow.js
- CAFFE
- Theano

Issues

- Deep Neural Networks are Easily Fooled
  - https://arxiv.org/abs/1412.1897v4
- Failures of deep learning
  - https://arxiv.org/abs/1703.07950
- Requires rethinking generalization
- Steps toward deep kernel methods from infinite neural networks
  - https://arxiv.org/abs/1508.05133
- Do Deep Neural Networks Really Need to be Deep?

The Future

- AutoML
  - DeepArchitect: Automatically Designing and Training Deep Architectures by Renato Negrinho, Geoff Gordon
    - https://github.com/negrinho/deep_architect
- Unsupervised Learning
  - GANs and GAN inspired models
  - Stopping GAN Violence with GUNs
    - https://arxiv.org/abs/1703.02528v1
  - Deep Stubborn Networks
    - http://www.kdnuggets.com/2017/04/deep-stubborn-networks-gan-refinement.html
  - Generative Ladder Networks
    - https://medium.com/towards-data-science/a-new-kind-of-deep-neural-networks-749bcde19108
- Applications of Deep Learning

End of Lecture

We want to make a machine that will be proud of us.

- Danny Hillis