Introduction to Deep Learning

Álvaro Barbero Jiménez
alvaro.barbero.jimenez@gmail.com

Afi Escuela

Some Deep Learning achievements

The New York Times

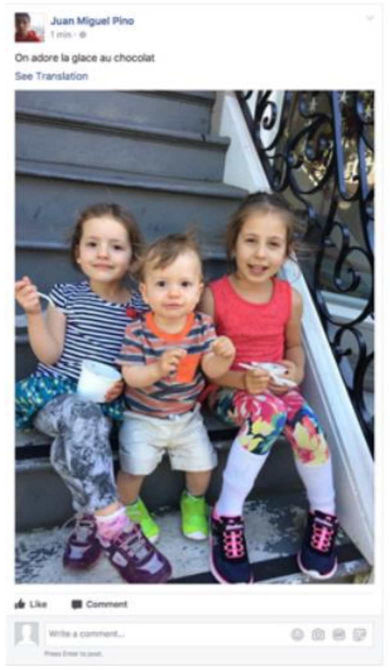
Google's AlphaGo Defeats Chinese Go Master in Win for A.I.

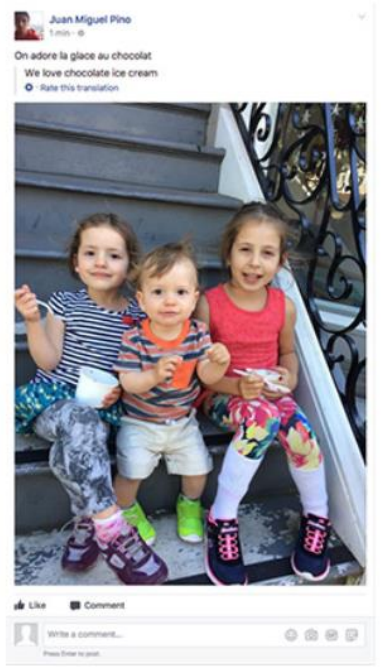
HONG KONG — It isn't looking good for humanity.

Code

Transitioning entirely to neural machine translation

Language translation is one of the ways we can give people the power to build community and bring the world closer together. It can help people connect with family members who live overseas, or better understand the perspective of someone who speaks a different language. We use machine translation to translate text in posts and comments automatically, in order to break language barriers and allow people around the world to communicate with each other.


Juan Miguel Pino
1 min ·
On adore la glace au chocolat
See Translation


Juan Miguel Pino
1 min ·
On adore la glace au chocolat
We love chocolate ice cream
· Rate this translation

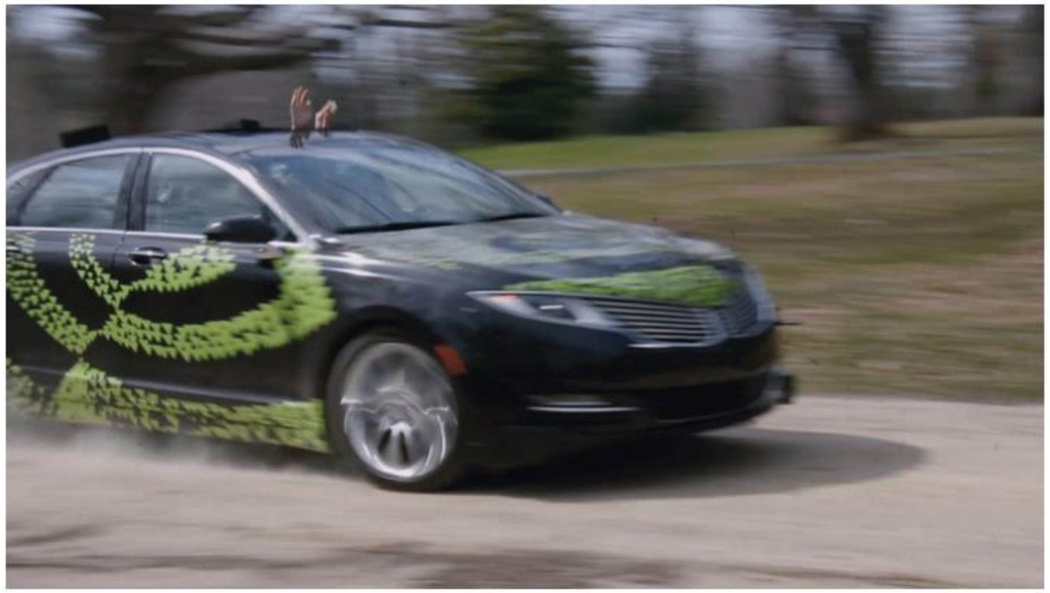
https://www.nytimes.com/2017/05/23/business/google-deepmind-alphago-go-champion-defeat.html?mcubz=0

https://code.facebook.com/posts/289921871474277/transitioning-entirely-to-neural-machine-translation/

Some Deep Learning achievements

End-to-End Deep Learning for Self-Driving Cars

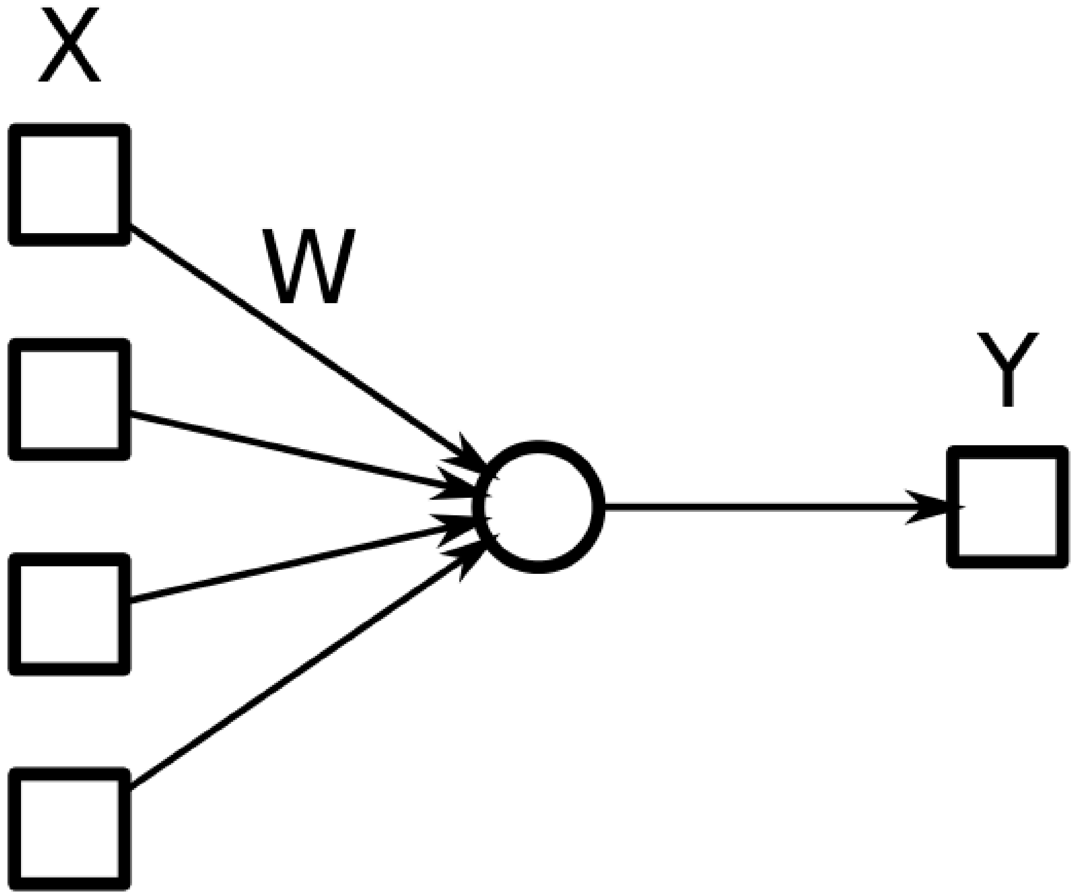
In a new automotive application, we have used convolutional neural networks (CNNs) to map the raw pixels from a front-facing camera to the steering commands for a self-driving car. This powerful end-to-end approach means that with minimum training data from humans, the system learns to steer, with or without lane markings, on both local roads and highways. The system can also operate in areas with unclear visual guidance such as parking lots or unpaved roads. [Editor's Note: be sure to check out the new post "Explaining How End-to-End Deep Learning Steers a Self-Driving Car"].



NVIDIA ACCELERATED COMPUTING

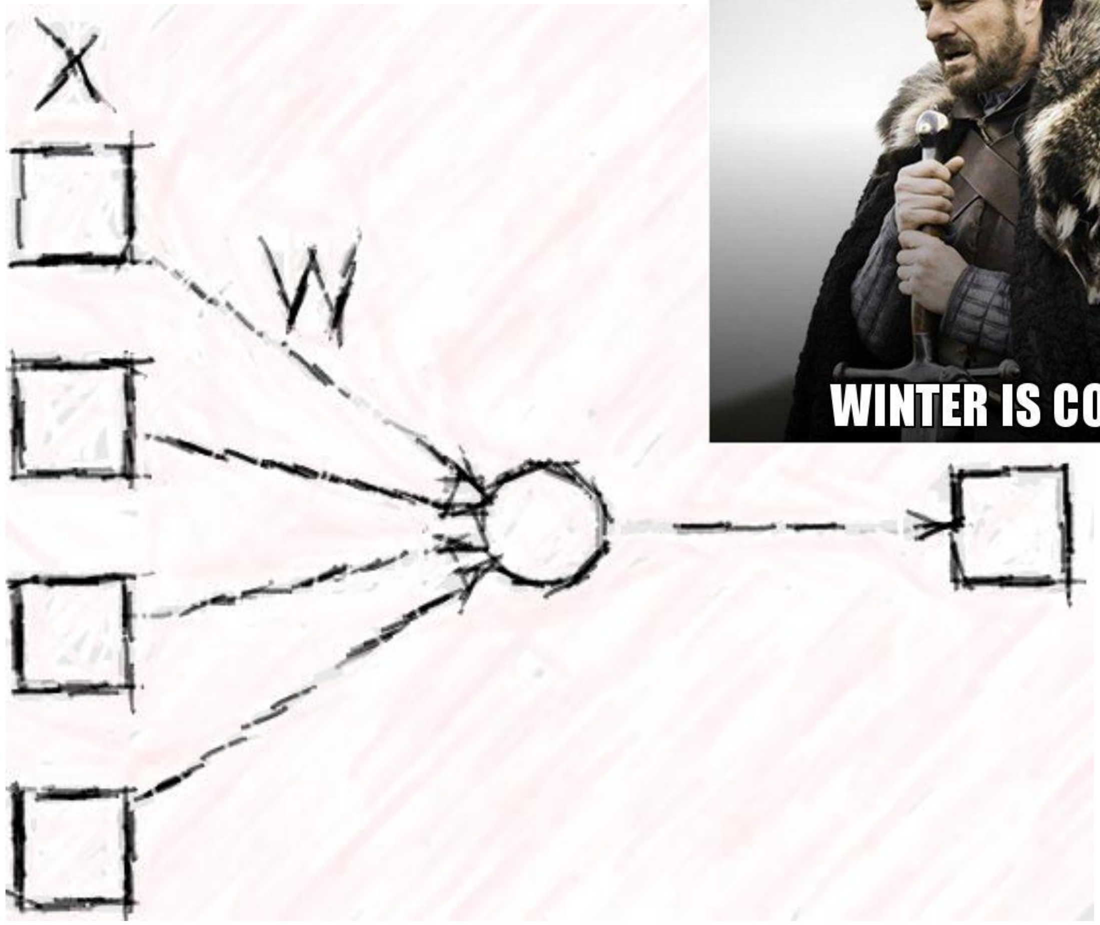
https://devblogs.nvidia.com/parallelforall/deep-learning-self-driving-cars/

First generation of neural networks



- Rosenblatt’s perceptron (1957)
- Simple learning algorithm
- Solves linear classification problems

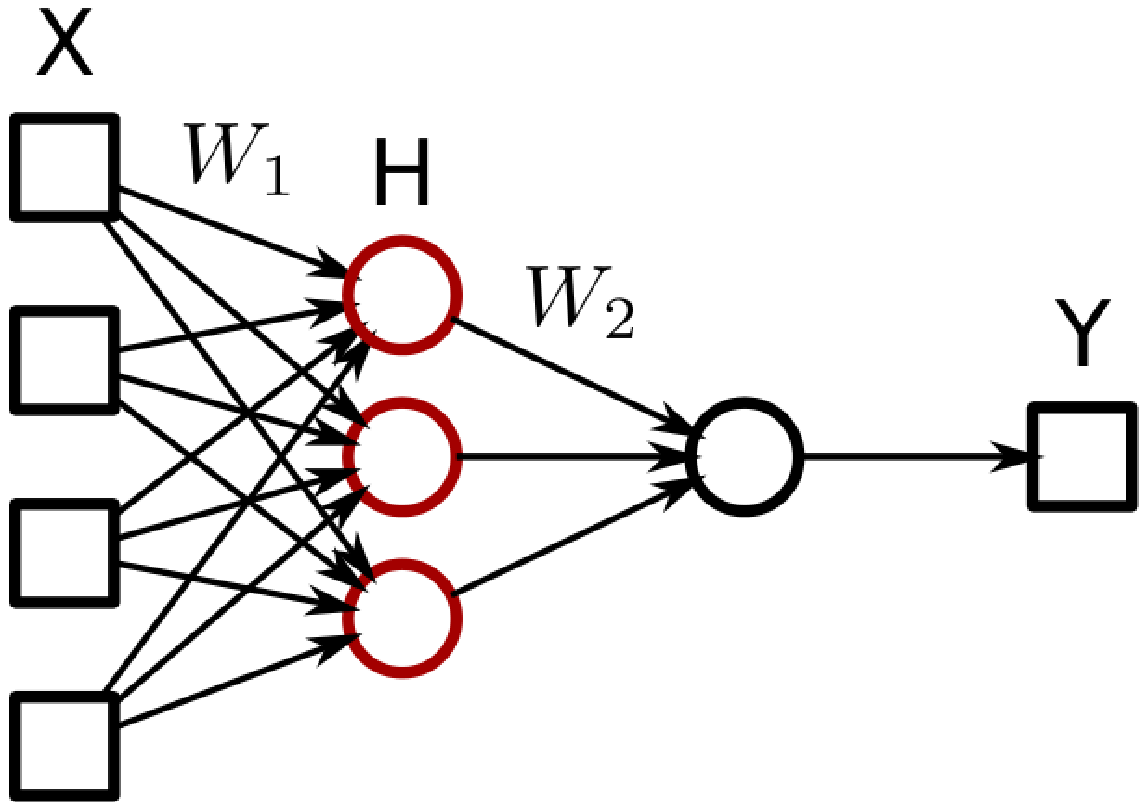
First AI Winter



Perceptrons cannot learn how to solve non-linear problems

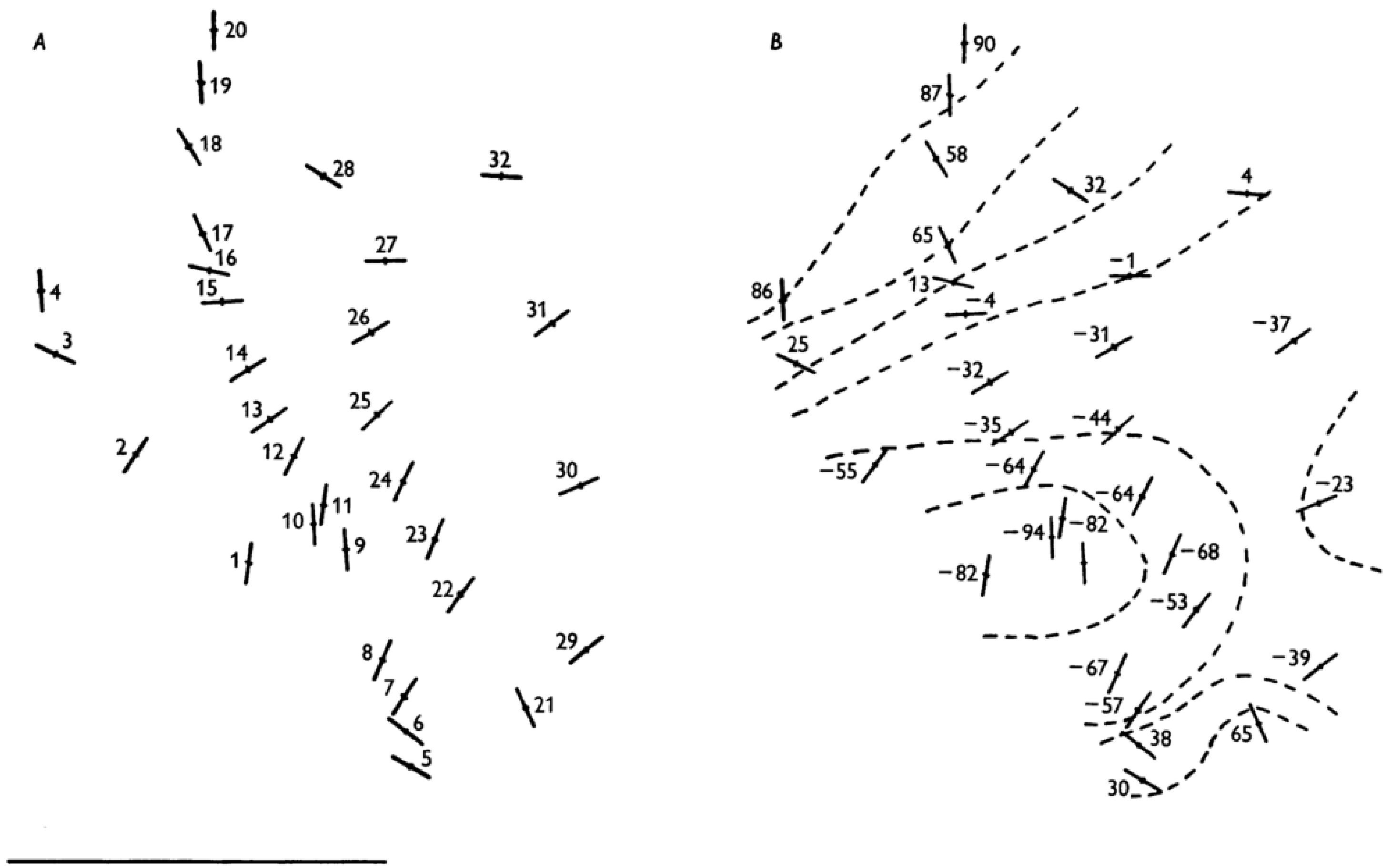
And many practical problems are non-linear!!

Second generation of neural networks



- Multilayer perceptron (1986)
- Add a layer of hidden units
- More involved learning algorithm (backpropagation)
- Solves non–linear classification problems

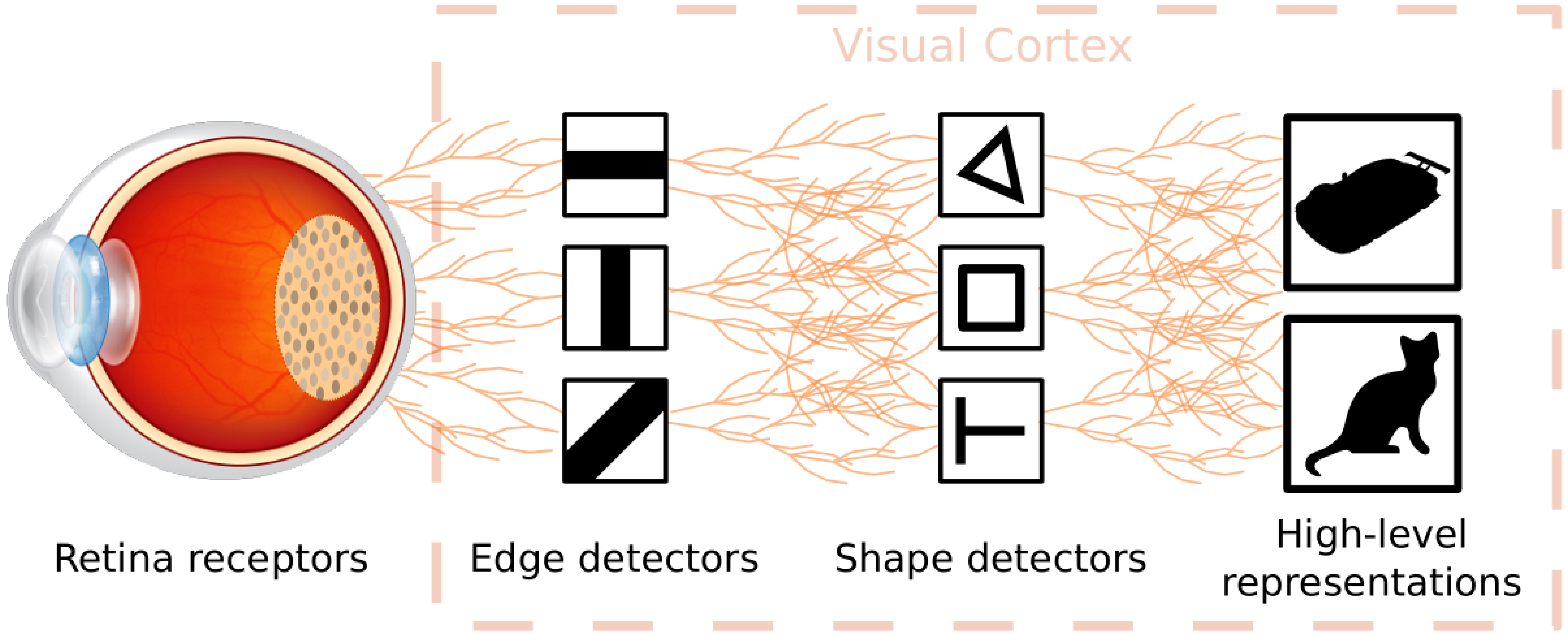
Motivation for more layers



Text-fig. 4. Surface map of a region in post-lateral gyrus. Receptive-field orientations are shown for 32 superficial penetrations. In *A* penetrations are numbered in sequence. In *B* receptive-field orientations are given in degrees of arc, zero corresponding to a horizontal orientation. Interrupted lines are drawn to separate regions of relatvely constant axis orientation. Scale, 1 mm. (See also Pl. 2.)

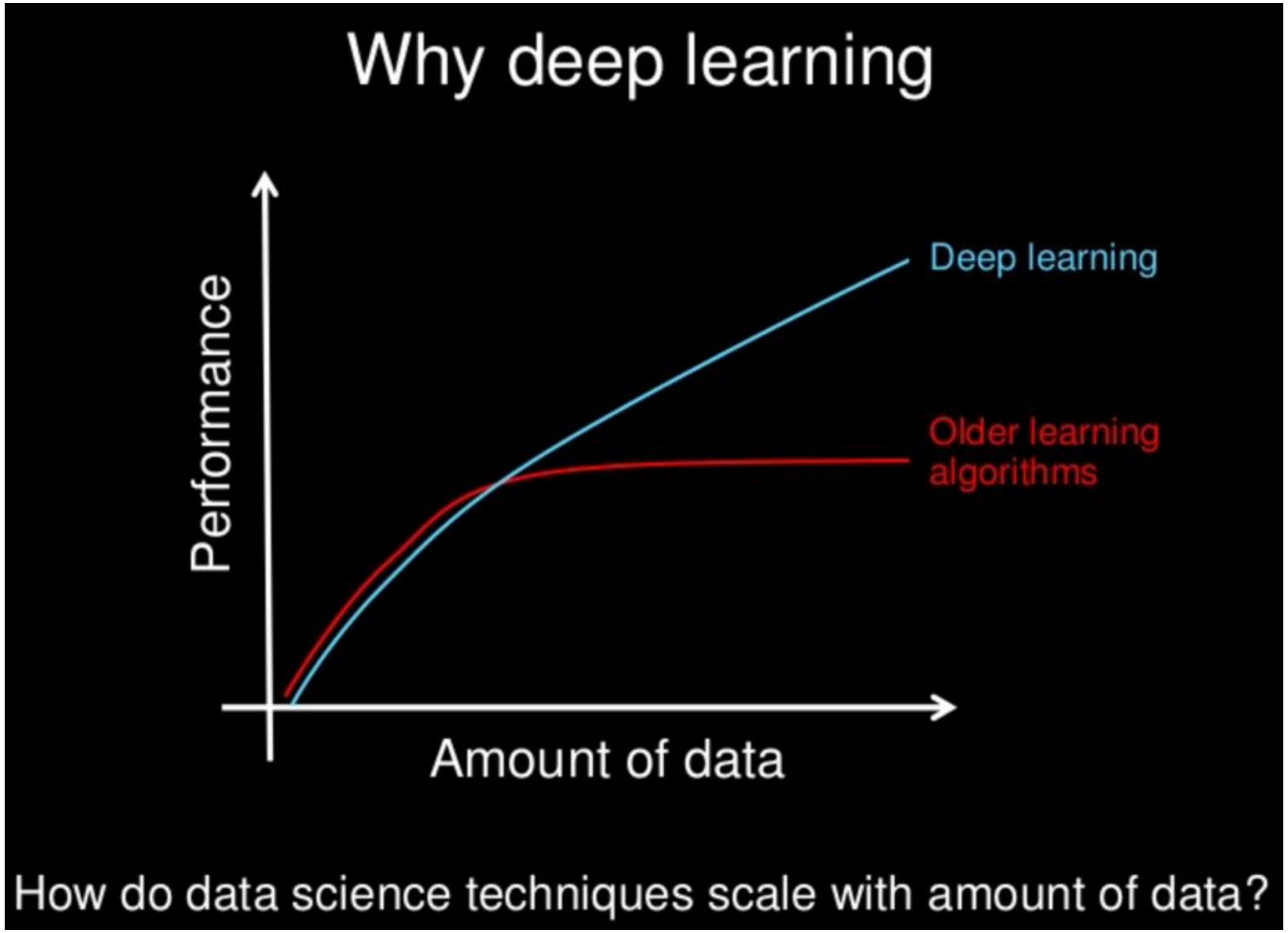
Hubel & Wiesel. Shape and arrangement of columns in cat's striate cortex

Motivation for more layers



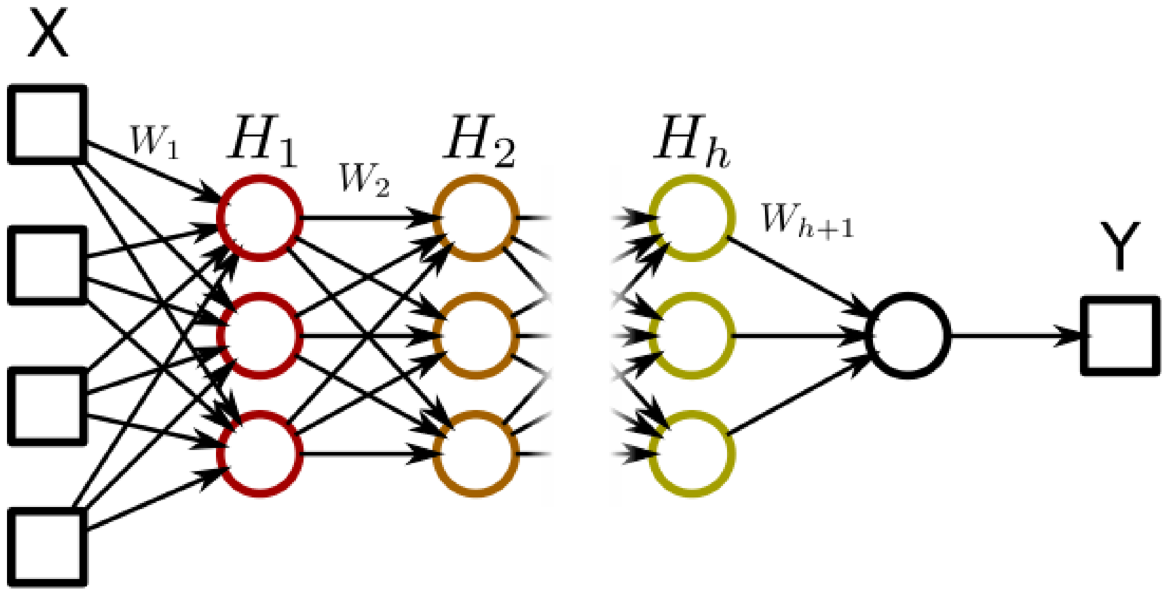
- Visual system as hierarchical structure: several layers of representation
- Each layer builds on top of the previous one to obtain more complex representations
- Mother Nature says: more than one hidden layer can be really useful

Motivation for more layers



Picture by Andrew Ng

Third generation of neural networks

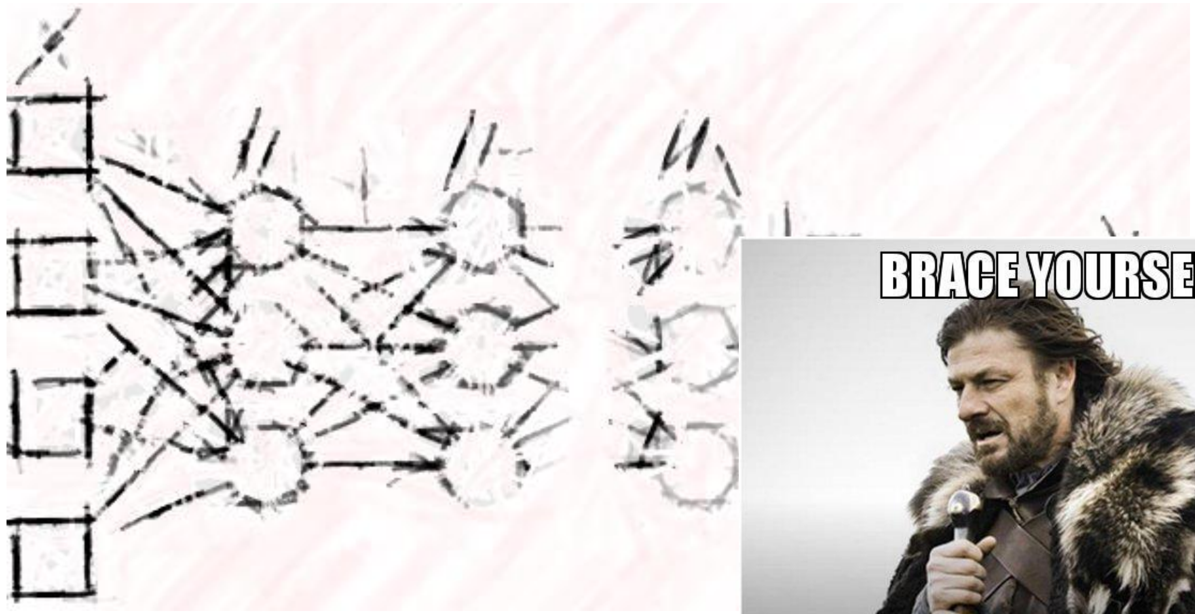


General idea: use many layers of hidden units.

→ Deep Learning

In theory, backpropagation can be used to train several hidden layers

Second AI Winter



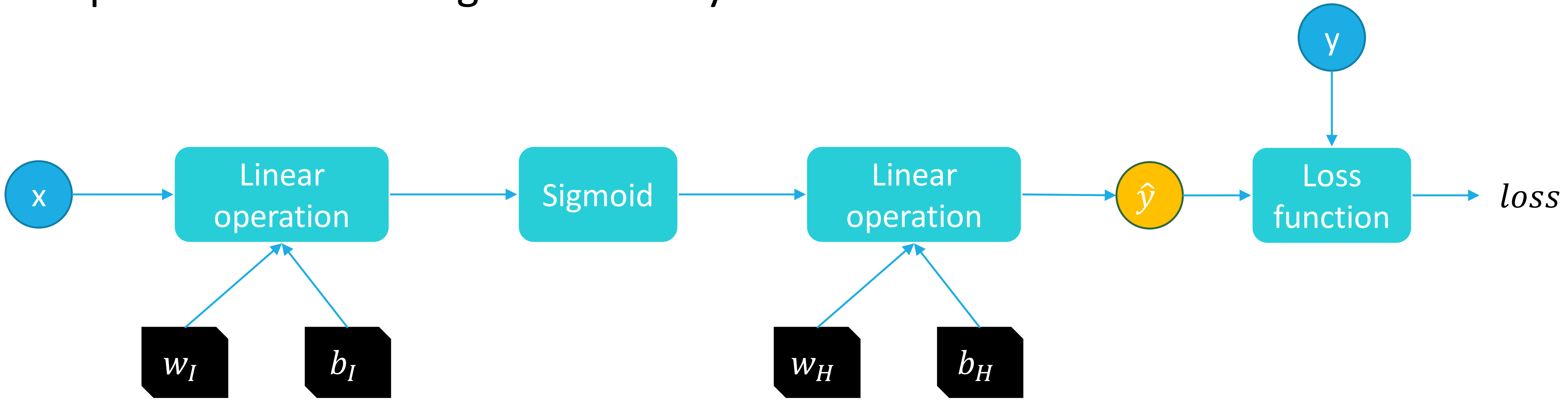
But it doesn't work!

Why??

What is really a neural network?

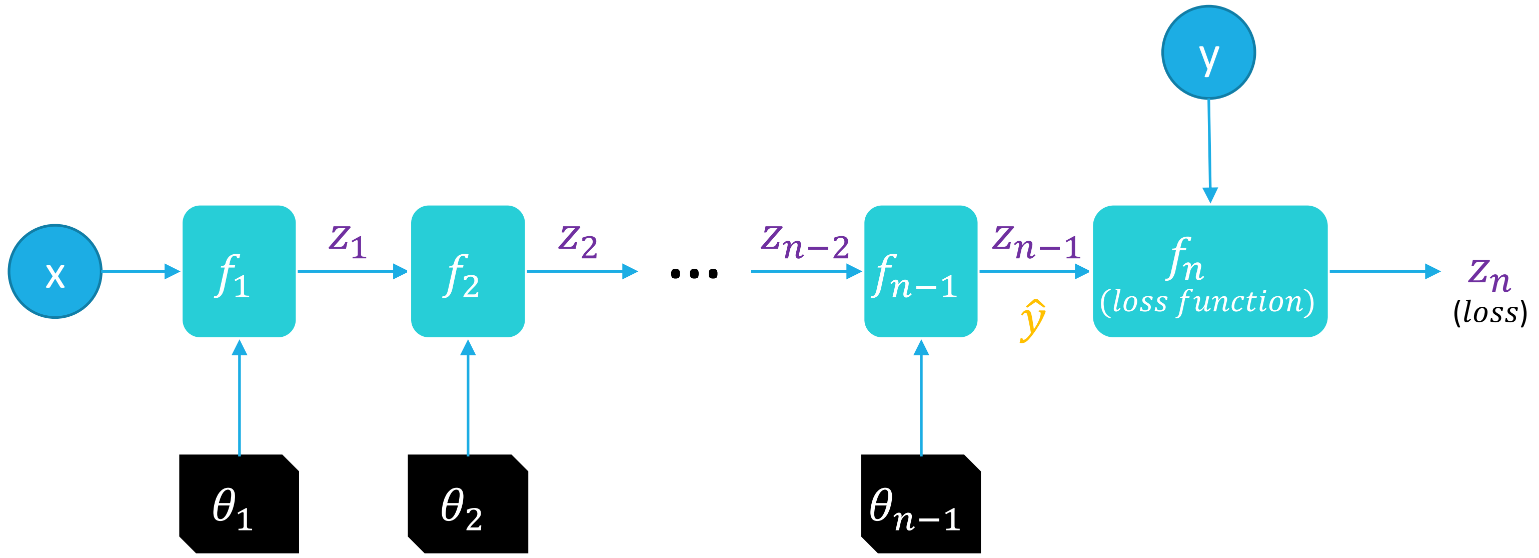
A neural network can be thought as an arbitrary directed acyclic graph (DAG) of operations, from an input vector x to an output vector $\hat{y}$. Each operation has some parameters to be tuned to minimize a loss function.

Example: MLP with a single hidden layer



What is backpropagation doing?

Take a general network F with layers f_i each with parameters θ_i as



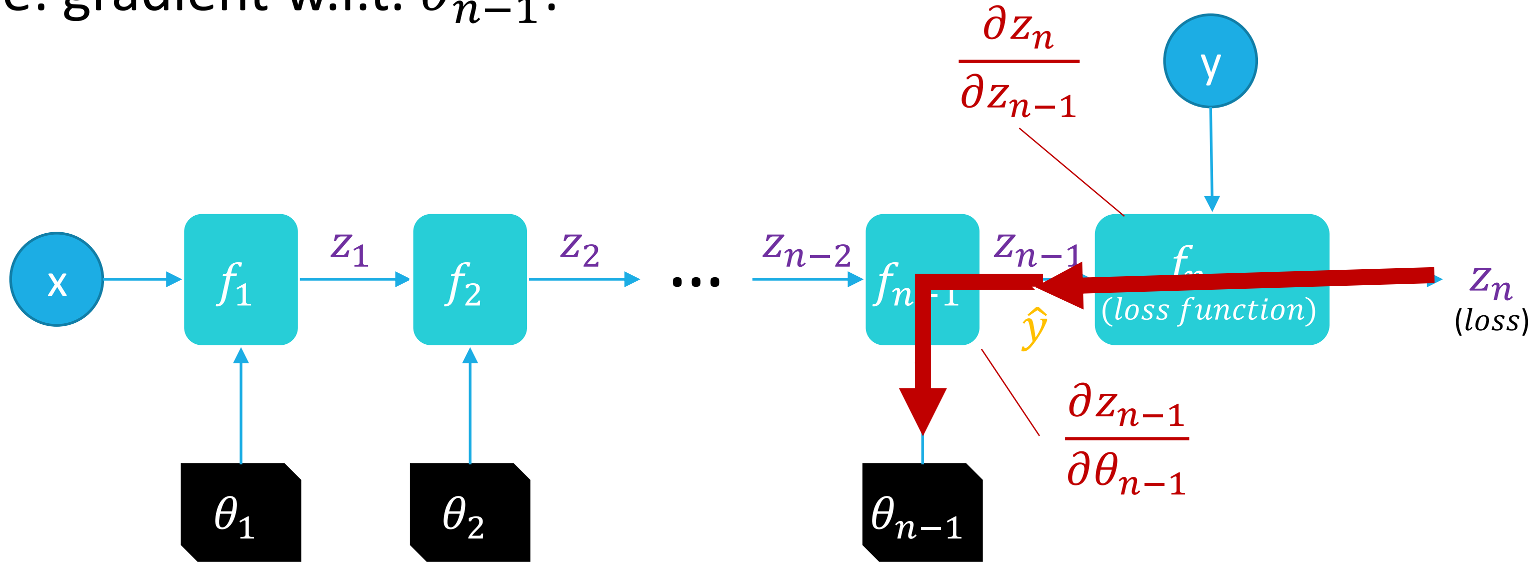
Backpropagation computes the gradients of the whole network for each network parameter. By following the chain rule we have

$$\frac{\partial F}{\partial \theta_i} = \frac{\partial (f_n \circ f_{n-1} \circ \cdots \circ f_1)}{\partial \theta_i} = \frac{\partial z_n}{\partial z_{n-1}} \frac{\partial z_{n-1}}{\partial z_{n-2}} \dots \frac{\partial z_{i+1}}{\partial z_i} \frac{\partial z_i}{\partial \theta_i} \quad \forall\, i$$

What is backpropagation doing?

Visually, backpropagation travels the network backwards from $loss$ to θ_i. Each function in the path adds a partial derivative of its output with respect to its input.

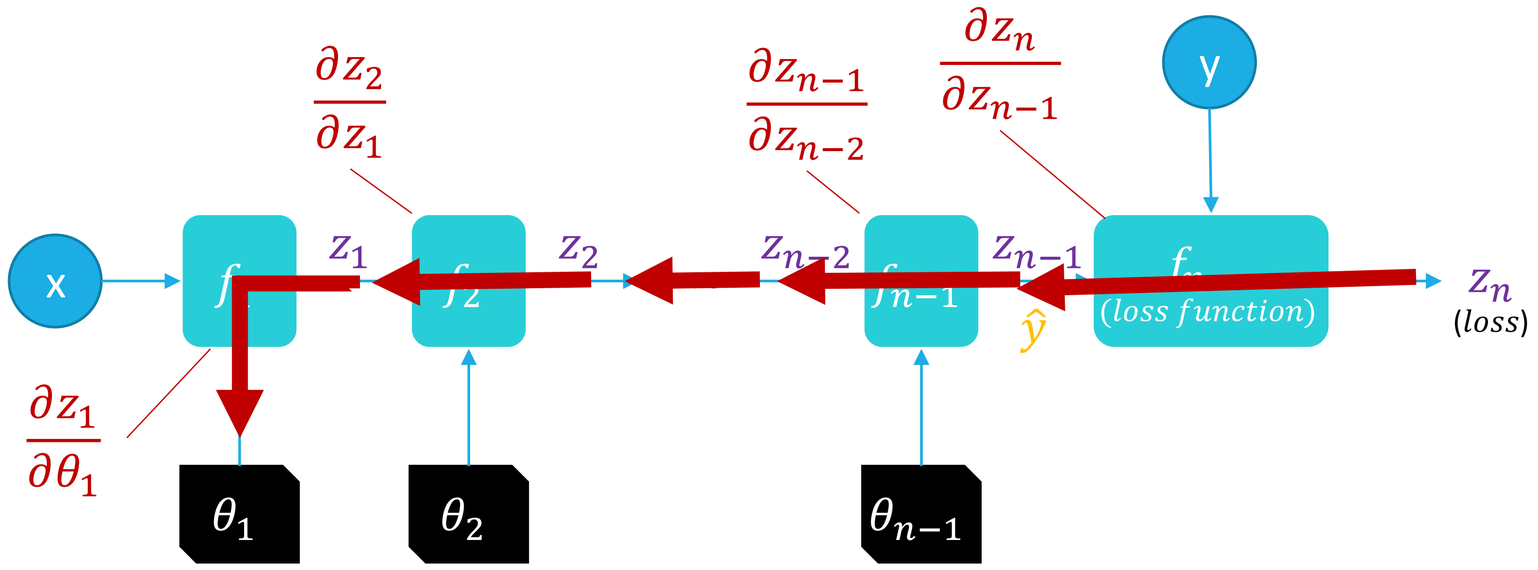
Example: gradient w.r.t. θ_{n-1}:



$$\frac{\partial F}{\partial \theta_{n-1}} = \frac{\partial z_n}{\partial z_{n-1}} \frac{\partial z_{n-1}}{\partial \theta_{n-1}}$$

What is backpropagation doing?

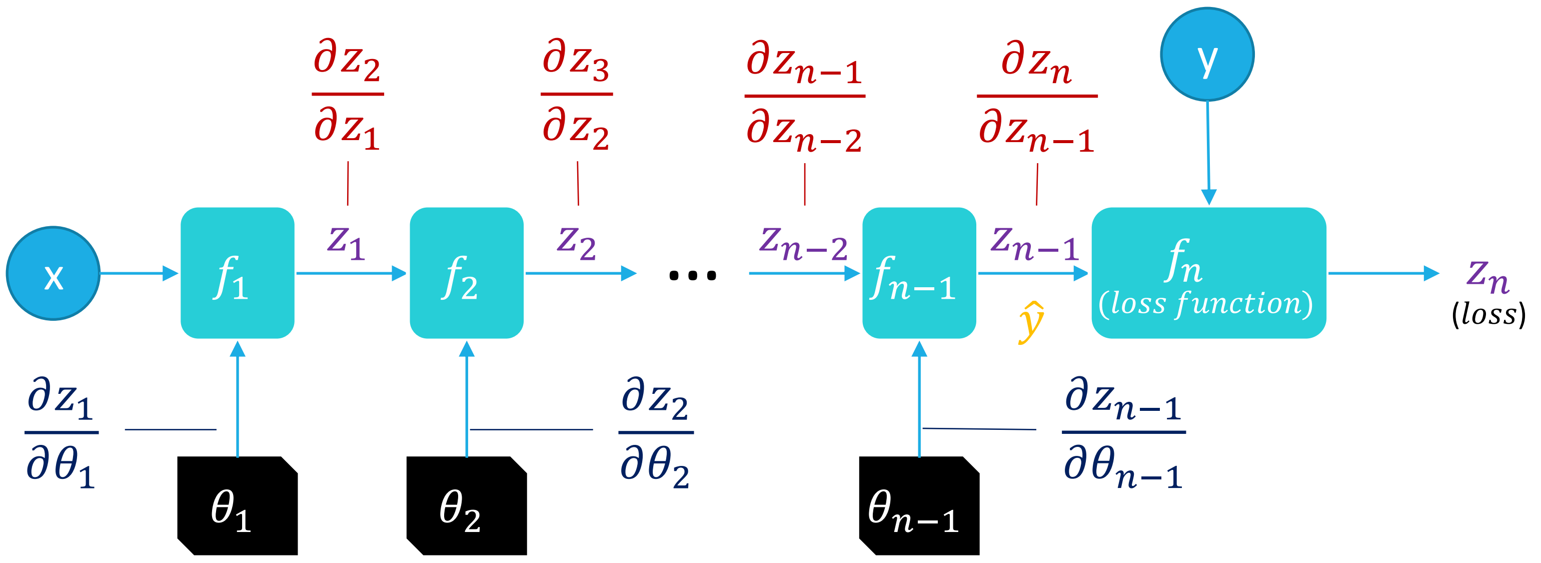
Another example: gradient w.r.t. θ_1:



x
f_1
z_1
f_2
z_2
z_{n-2}
f_{n-1}
z_{n-1}
f_n
(loss function)
z_n
(loss)
y
$\hat{y}$
$\frac{\partial z_2}{\partial z_1}$
$\frac{\partial z_{n-1}}{\partial z_{n-2}}$
$\frac{\partial z_n}{\partial z_{n-1}}$
$\frac{\partial z_1}{\partial \theta_1}$
θ_1
θ_2
θ_{n-1}

$$\frac{\partial F}{\partial \theta_1} = \frac{\partial z_n}{\partial z_{n-1}} \frac{\partial z_{n-1}}{\partial z_{n-2}} \cdots \frac{\partial z_2}{\partial z_1} \frac{\partial z_1}{\partial \theta_1}$$

Important observations about backpropagation

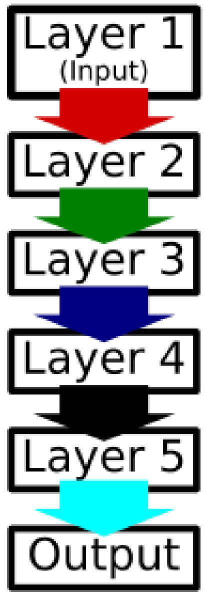


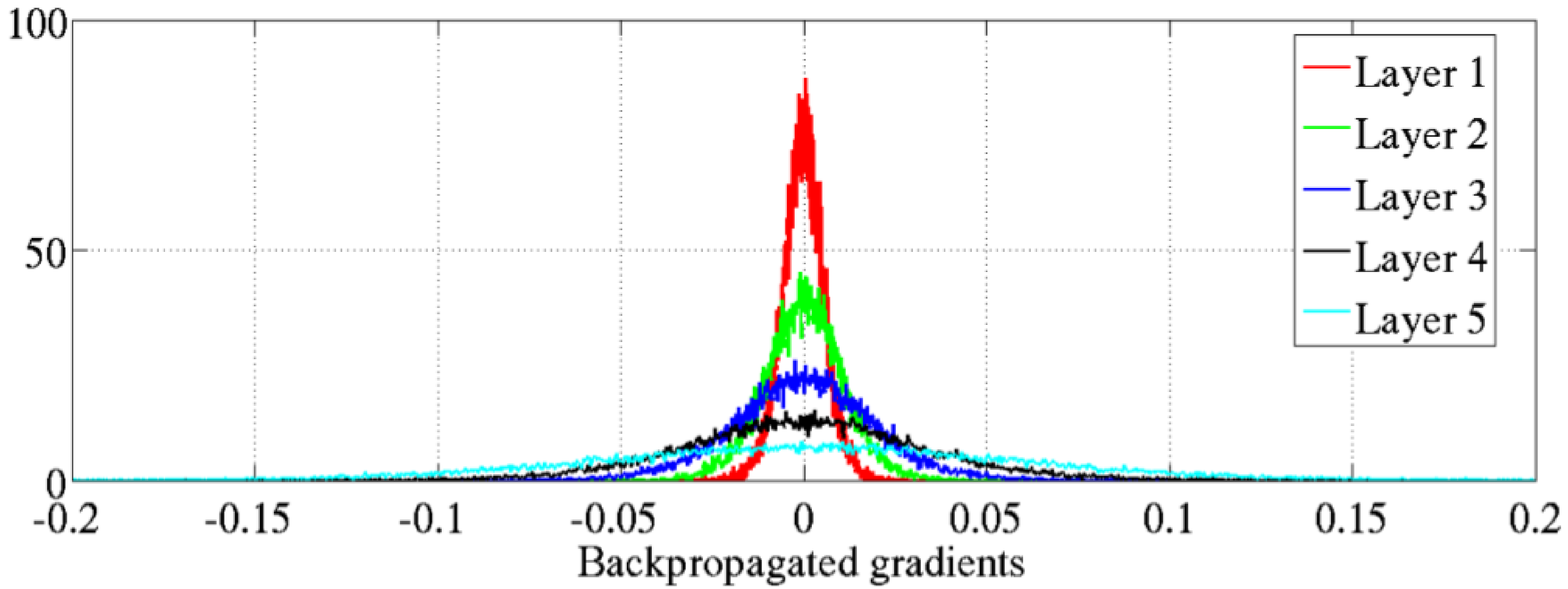
$\frac{\partial z_2}{\partial z_1}$
$\frac{\partial z_3}{\partial z_2}$
$\frac{\partial z_{n-1}}{\partial z_{n-2}}$
$\frac{\partial z_n}{\partial z_{n-1}}$
y
x
f_1
z_1
f_2
z_2
…
z_{n-2}
f_{n-1}
z_{n-1}
$\hat{y}$
f_n
(loss function)
z_n
(loss)
$\frac{\partial z_1}{\partial \theta_1}$
θ_1
θ_2
$\frac{\partial z_2}{\partial \theta_2}$
θ_{n-1}
$\frac{\partial z_{n-1}}{\partial \theta_{n-1}}$

$$\frac{\partial F}{\partial \theta_{n-1}} = \frac{\partial z_n}{\partial z_{n-1}} \frac{\partial z_{n-1}}{\partial \theta_{n-1}} \qquad \frac{\partial F}{\partial \theta_{n-2}} = \frac{\partial z_n}{\partial z_{n-1}} \frac{\partial z_{n-1}}{\partial z_{n-2}} \frac{\partial z_{n-2}}{\partial \theta_{n-2}} \qquad \frac{\partial F}{\partial \theta_{n-3}} = \frac{\partial z_n}{\partial z_{n-1}} \frac{\partial z_{n-1}}{\partial z_{n-2}} \frac{\partial z_{n-2}}{\partial z_{n-3}} \frac{\partial z_{n-3}}{\partial \theta_{n-3}}$$

- Terms in the form $\frac{\partial z_i}{\partial z_{i-1}}$ can be reused throughout $\frac{\partial F}{\partial \theta_i}$ computations $\forall\, i$ → efficient computation of derivatives for all network parameters.
- We can add any function f_i to a neural network, as long as we can compute every partial derivate of each output w.r.t. each input.
- Layers closer to the input → more terms in gradient calculation

The problem: vanishing gradients



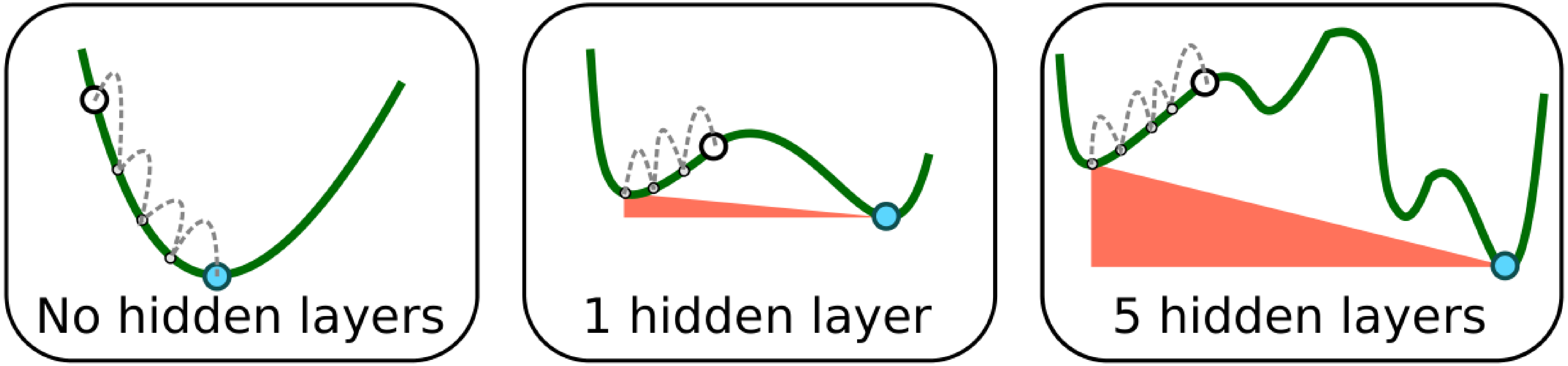


- In backpropagation, the gradient tends to zero in the layers nearest to the input
- Intuition: layers near to the output have larger influence in the error rate
- Training deep layers successfully is terribly slow
- Until deep layers learn some sensible weights, subsequent layers are trying to learn mostly from noise

Glorot & Bengio. Understanding the difficulty in training deep feedforward neural networks

Another problem: high non-linearity

- Backpropagation is a gradient descent method.
- Only guarantees finding a local minimum of the error function, which can be worse than the global minimum
- When introducing more non–linearities in the network (more layers), local minima multiply



- It is hard to find good parameter configurations for a deep network

How to learn several deep networks

Whole network learning

Use standard neural network methods, but avoid vanishing gradients and other problems by

- Specially designed activations
- Better regularization
- Better backpropagation methods

Present method of choice

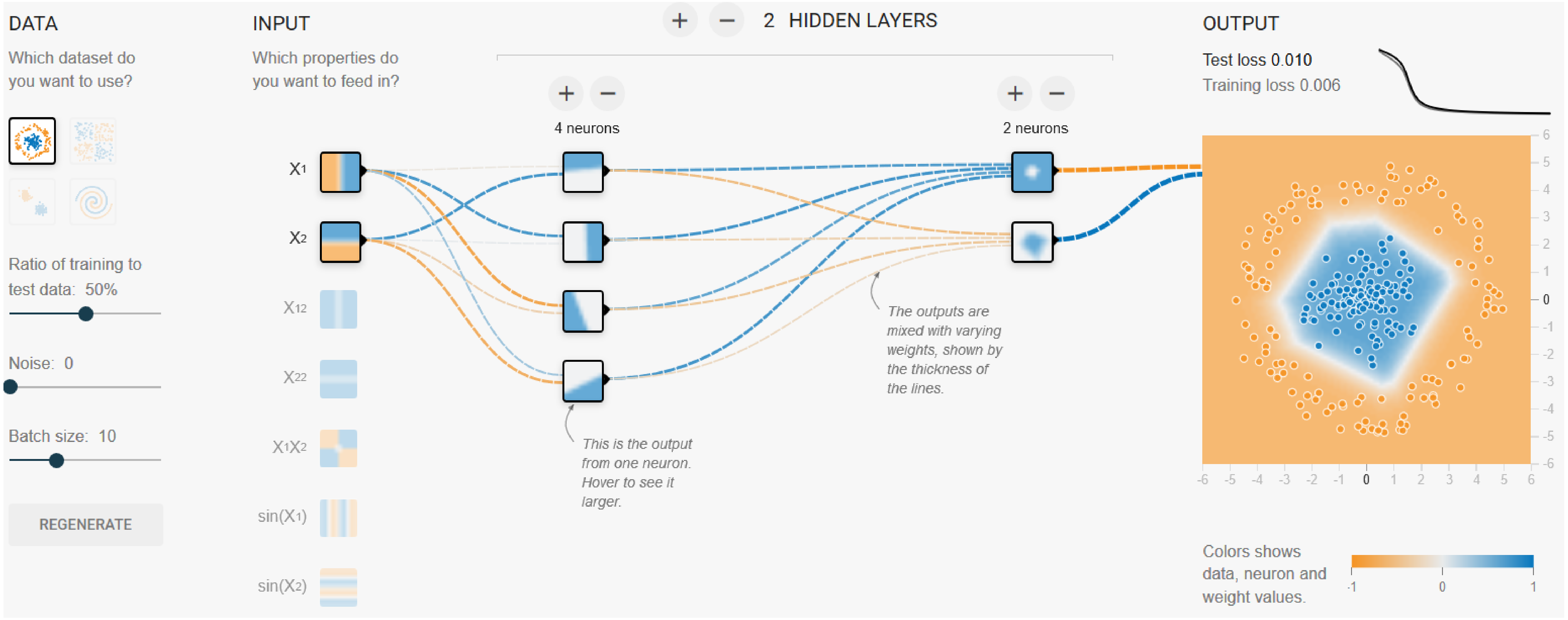
Incremental learning

Learn layers one by one, in an iterative fashion

- Deterministic approach: Autoencoders
- Probabilistic approach: Restricted Boltzmann Machines

Popular in the first years of Deep Learning, now less used

Working example



DATA

Which dataset do you want to use?

Ratio of training to test data: 50%

Noise: 0

Batch size: 10

REGENERATE

INPUT

Which properties do you want to feed in?

This is the output from one neuron. Hover to see it larger.

The outputs are mixed with varying weights, shown by the thickness of the lines.

OUTPUT

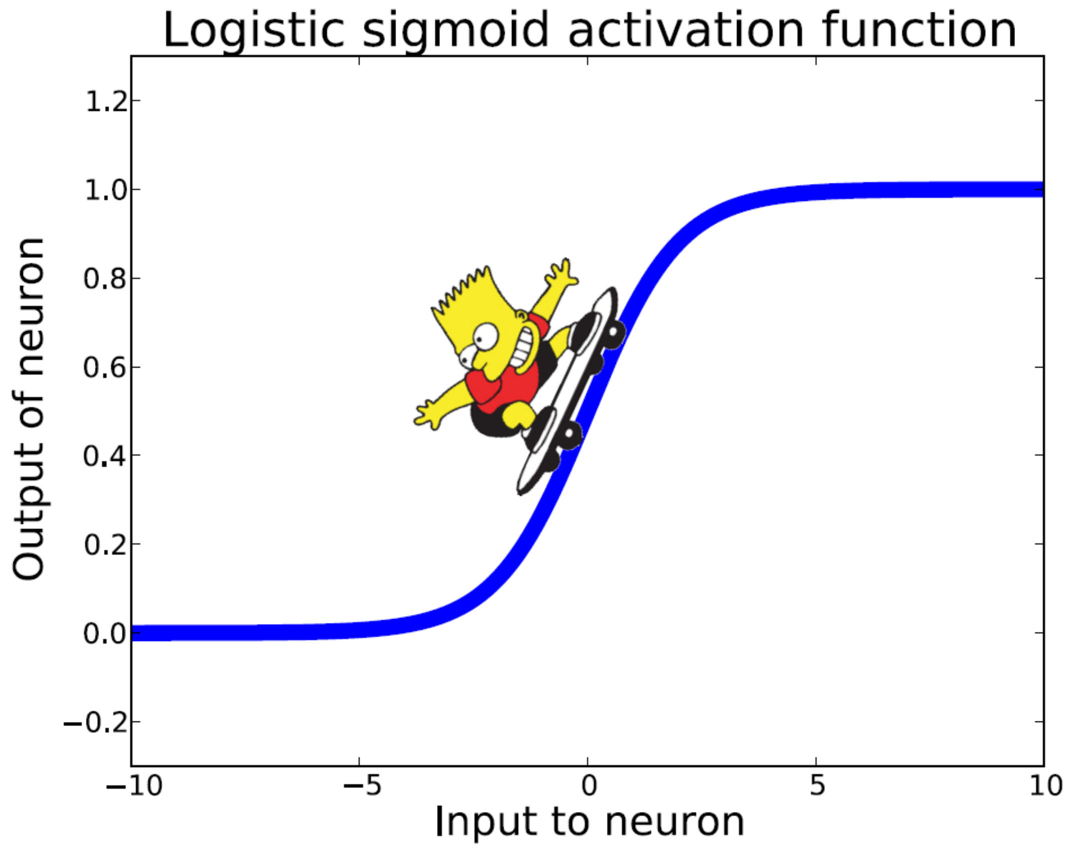
Test loss 0.010

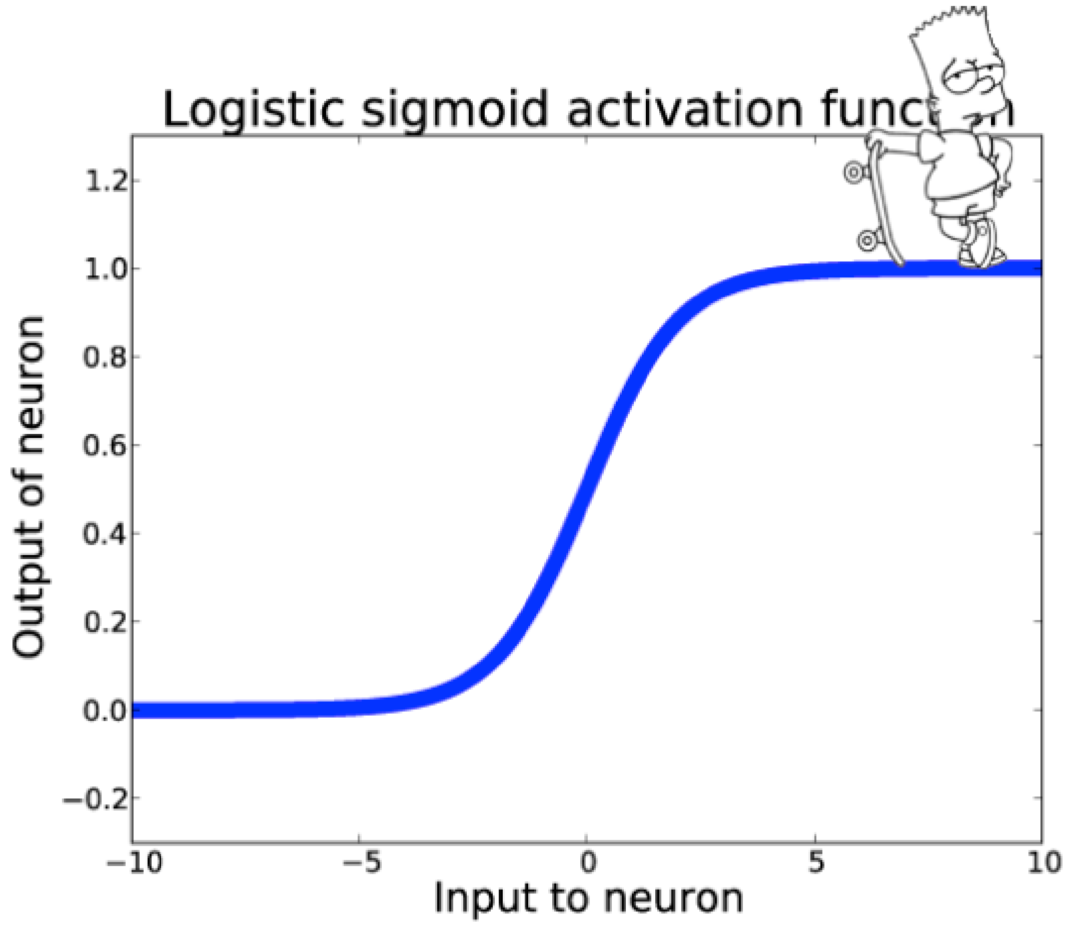
Training loss 0.006

Colors shows data, neuron and weight values.

A Neural Network playground - http://playground.tensorflow.org

The Bart Simpson problem



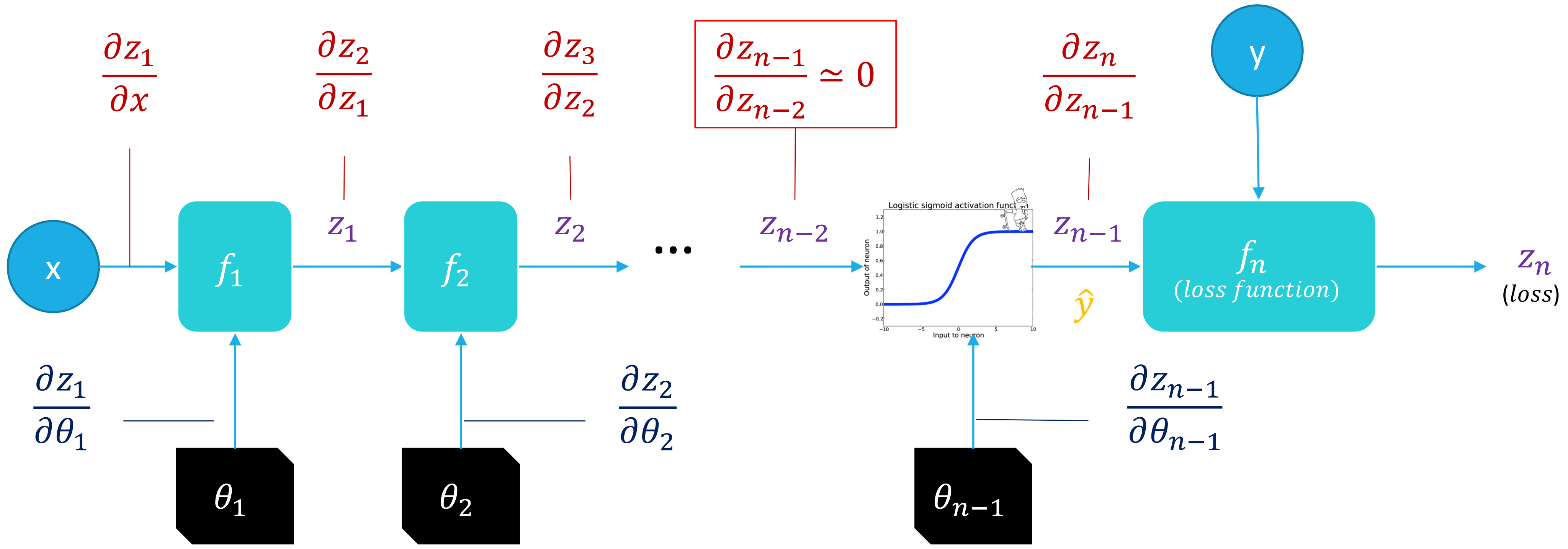


Goodfellow - Deep learning of representations and its application to computer vision

Serengil – An overview of the Vanishing Gradient Problem

The Simpsons by Matt Groening

The Bart Simpson problem in backpropagation terms

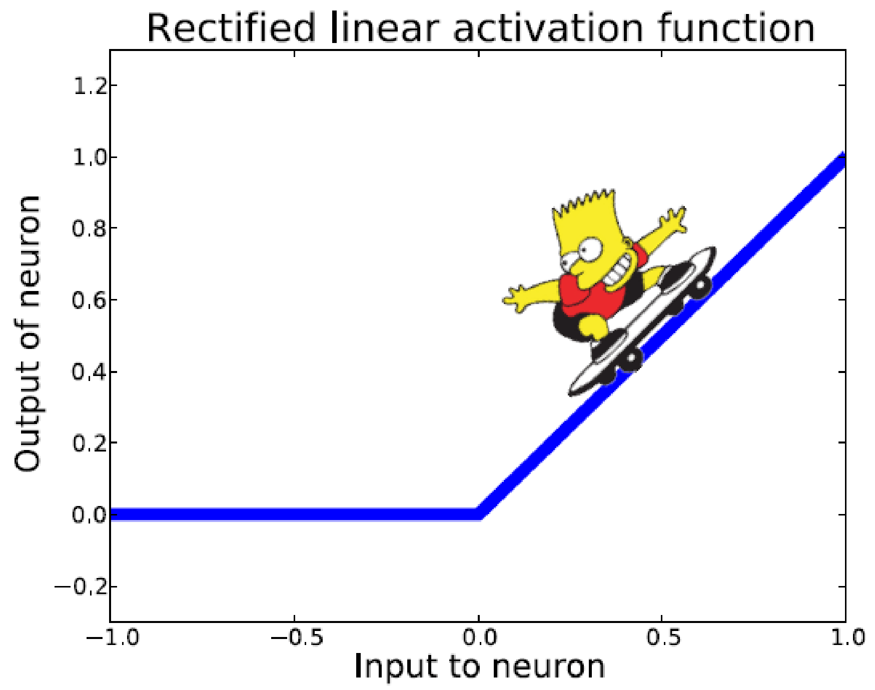

$\frac{\partial z_1}{\partial x}$
$\frac{\partial z_2}{\partial z_1}$
$\frac{\partial z_3}{\partial z_2}$
$\boxed{\frac{\partial z_{n-1}}{\partial z_{n-2}} \simeq 0}$
$\frac{\partial z_n}{\partial z_{n-1}}$
y
x
f_1
z_1
f_2
z_2
…
z_{n-2}
Logistic sigmoid activation function
Output of neuron
Input to neuron
z_{n-1}
$\hat{y}$
f_n (loss function)
z_n (loss)
$\frac{\partial z_1}{\partial \theta_1}$
$\frac{\partial z_2}{\partial \theta_2}$
$\frac{\partial z_{n-1}}{\partial \theta_{n-1}}$
θ_1
θ_2
θ_{n-1}

If the network includes a layer that can produce $\simeq 0$ terms in the partial derivative, all parameters in previous layers are likely to get very small gradients.

$$\frac{\partial F}{\partial \theta_i} = \frac{\partial z_n}{\partial z_{n-1}} \frac{\partial z_{n-1}}{\partial z_{n-2}} \dots \frac{\partial z_{i+1}}{\partial z_i} \frac{\partial z_i}{\partial \theta_i} = \frac{\partial z_n}{\partial z_{n-1}} (\simeq 0) \dots \frac{\partial z_{i+1}}{\partial z_i} \frac{\partial z_i}{\partial \theta_i} \simeq 0 \;\; \forall\, i$$

Rectified Linear Units

$$f(x) = \begin{cases} x & \text{if} \quad x > 0 \\ 0 & \text{if} \quad x \leq 0 \end{cases}$$



- A better choice for the activation function is to use Rectified Linear Units
- Very cheap to compute (no products or exponentials)
- Faster training
- Leads to sparser neuron activations

!!! Non–differentiable

- Backpropagation is performed by computing the derivative of f as

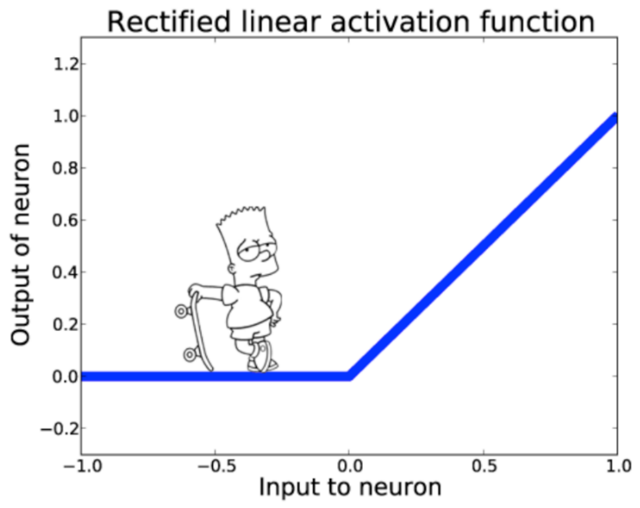
$$\frac{\partial f}{\partial x} \simeq \begin{cases} 1 & \text{if} \quad x > 0 \\ 0 & \text{if} \quad x \leq 0 \end{cases}$$

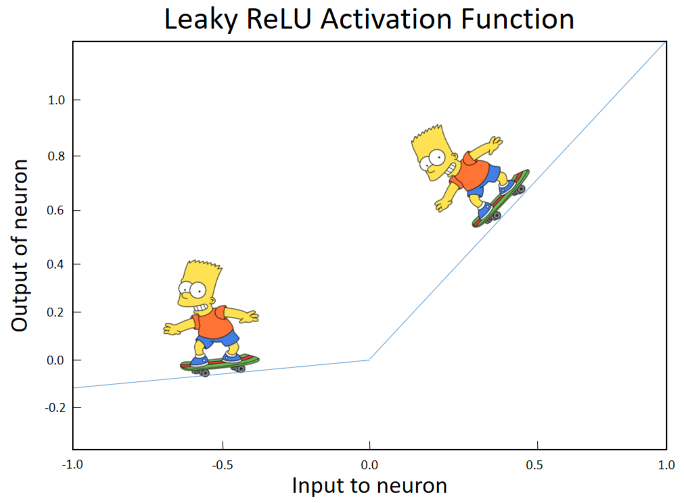
- This is a form of subgradient descent
- No small gradients in this formula

Zeiler et al. On Rectified Linear Units For Speech Processing.

Leaky and Parametric Rectified Linear Units

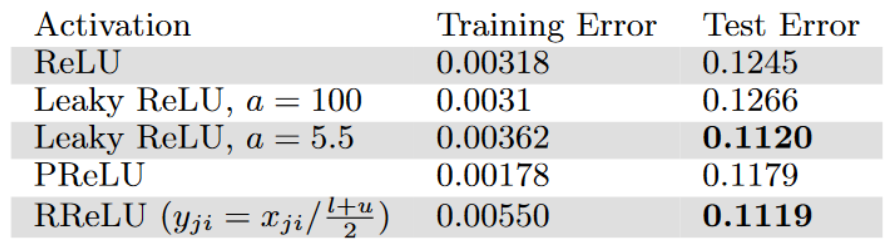
Some papers show that including a small slope in the negative side of ReLU units (Leaky ReLU) can improve model performance. This slope parameter a can also be learned by the network (PReLU).





$$f(x) = \begin{cases} x & \text{if} \quad x > 0 \\ ax & \text{if} \quad x \leq 0 \end{cases}$$

$$\frac{\partial f}{\partial x} \simeq \begin{cases} 1 & \text{if} \quad x > 0 \\ a & \text{if} \quad x \leq 0 \end{cases}$$



Activation	Training Error	Test Error
ReLU	0.00318	0.1245
Leaky ReLU, $a = 100$	0.0031	0.1266
Leaky ReLU, $a = 5.5$	0.00362	**0.1120**
PReLU	0.00178	0.1179
RReLU $(y_{ji} = x_{ji}/\frac{l+u}{2})$	0.00550	**0.1119**

Table 3. Error rate of CIFAR-10 Network in Network with different activation function

Maas et al. Rectified Nonlinearities Improve Neural Network Acoustic Models

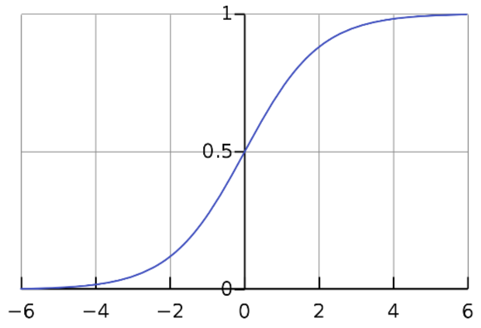
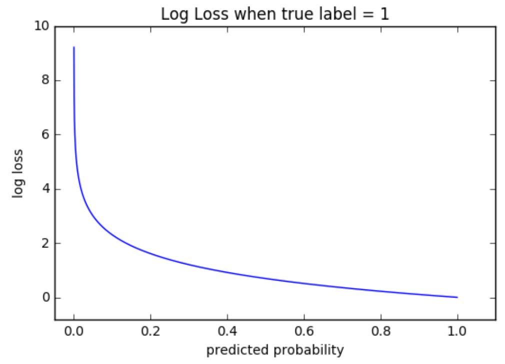
He et al. Delving Deeper into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification

Xu et al. Empirical Evaluation of Rectified Activations in Convolution Network

Serengil – An overview of the Vanishing Gradient Problem

The Simpsons by Matt Groening

Activations and losses in output layer

Problem class	Output layer activation	Loss function
Binary or multilabel classification	Sigmoid $f(x_i) = \frac{1}{1+e^{-x_i}}$ 	Binary cross entropy $l(\hat{y}, y) = -(y\ \log(\hat{y}) + (1-y)\log(1-\hat{y}))$ 
Multiclass classification	Softmax $f(x_i) = \frac{e^{x_i}}{\sum_d e^{x_d}}$	Categorical cross entropy $l(\hat{y}, y) = -\sum_c (y_c\ \log(\hat{y}_c))$ With y one-hot vector encoding true label, $\hat{y}$ output vector of class probabilities
Regression	Linear $f(x_i) = x_i$	Absolute error $l(\hat{y}, y) = \lvert\hat{y} - y\rvert$ Squared error $l(\hat{y}, y) = \lvert\lvert\hat{y} - y\rvert\rvert_2^2$

Regularization through Dropout

- For each pattern introduced in the network, randomly deactivate units with probability p
- Implicitly trains 2^n networks sharing the same weights, but with all the possible combinations of the activated/deactivated n neurons of the network
- Forces the network develop neurons robust to missing connections

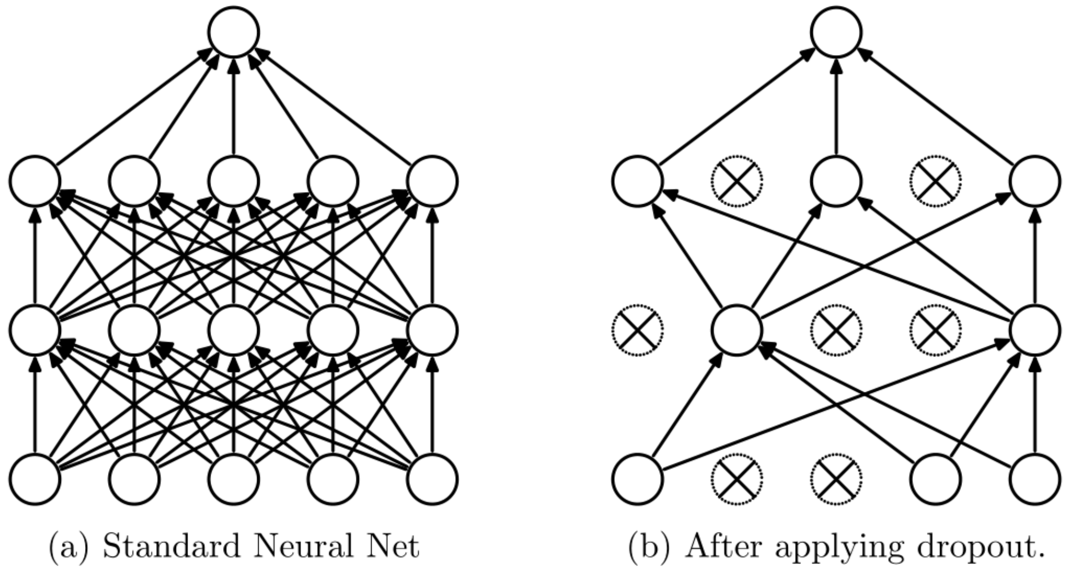
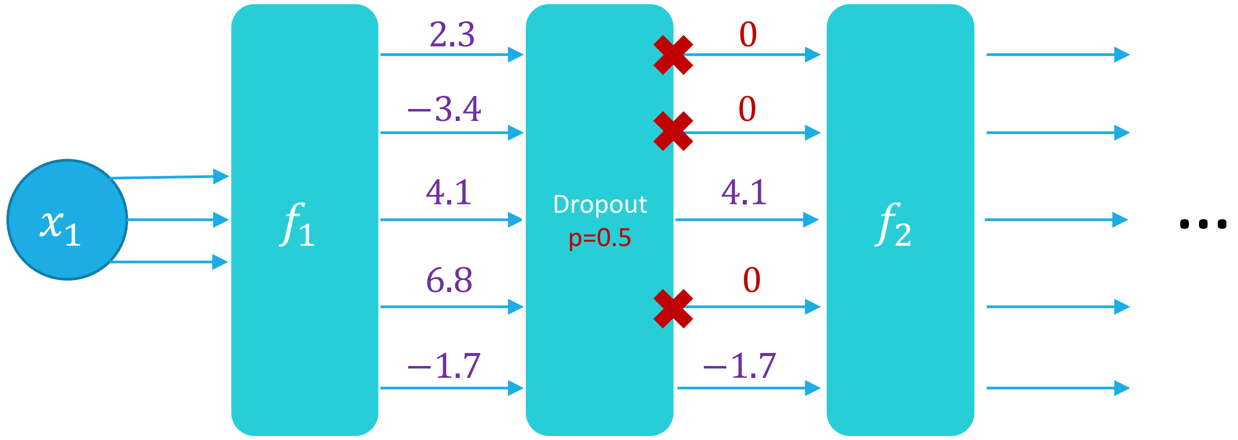


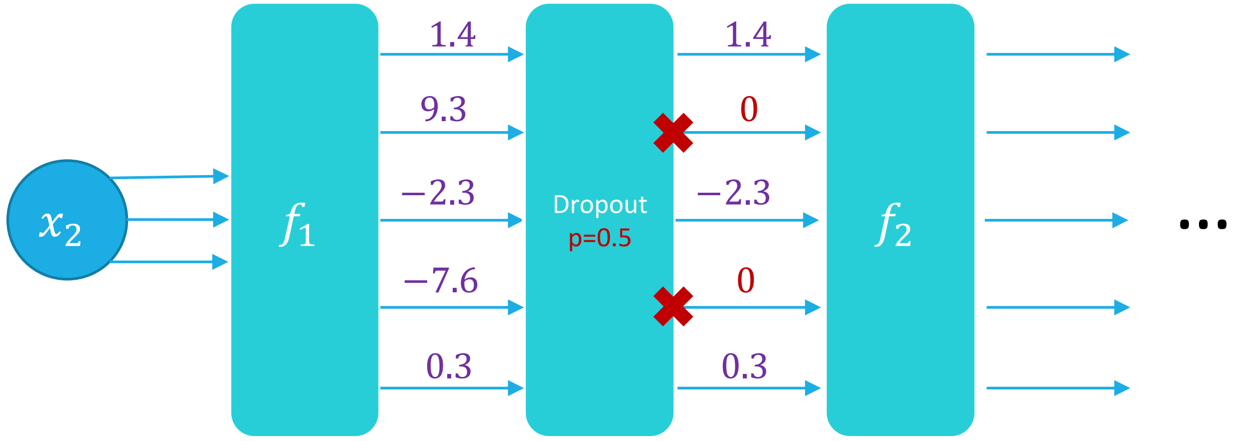
Figure 1: Dropout Neural Net Model. **Left**: A standard neural net with 2 hidden layers. **Right**: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

Srivastava. Improving Neural Networks with Dropout

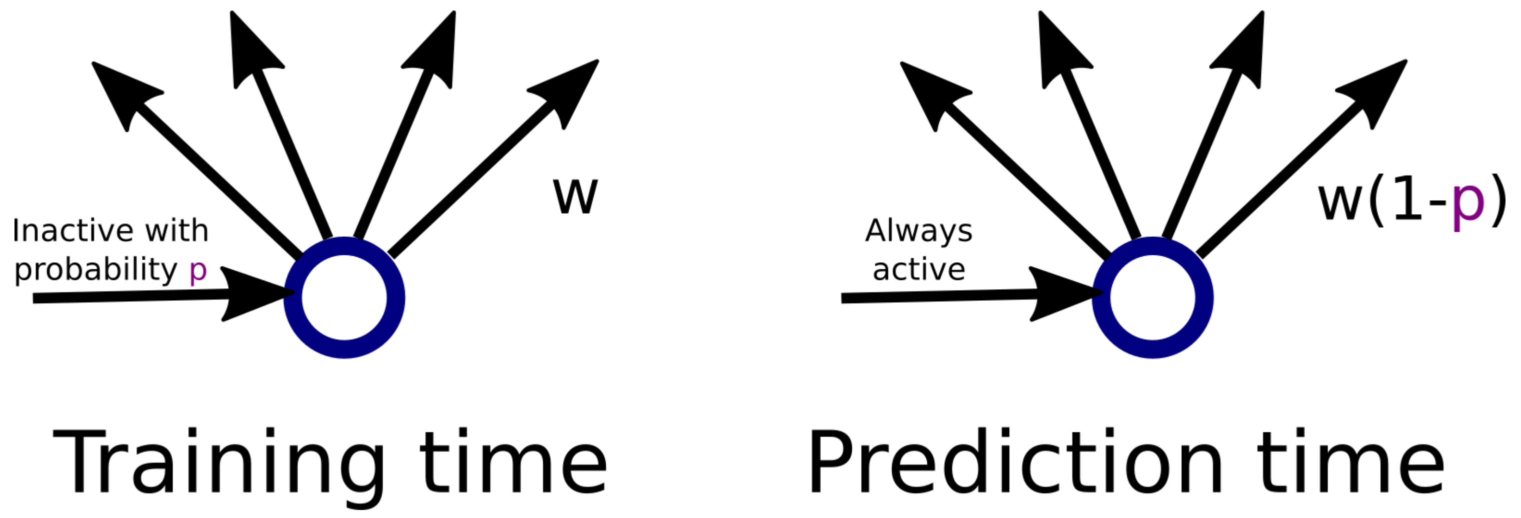
Srivastava et al. Dropout: A simple way to prevent neural networks from overfitting

Dropout as a deep network layer





Regularization through Dropout - Prediction

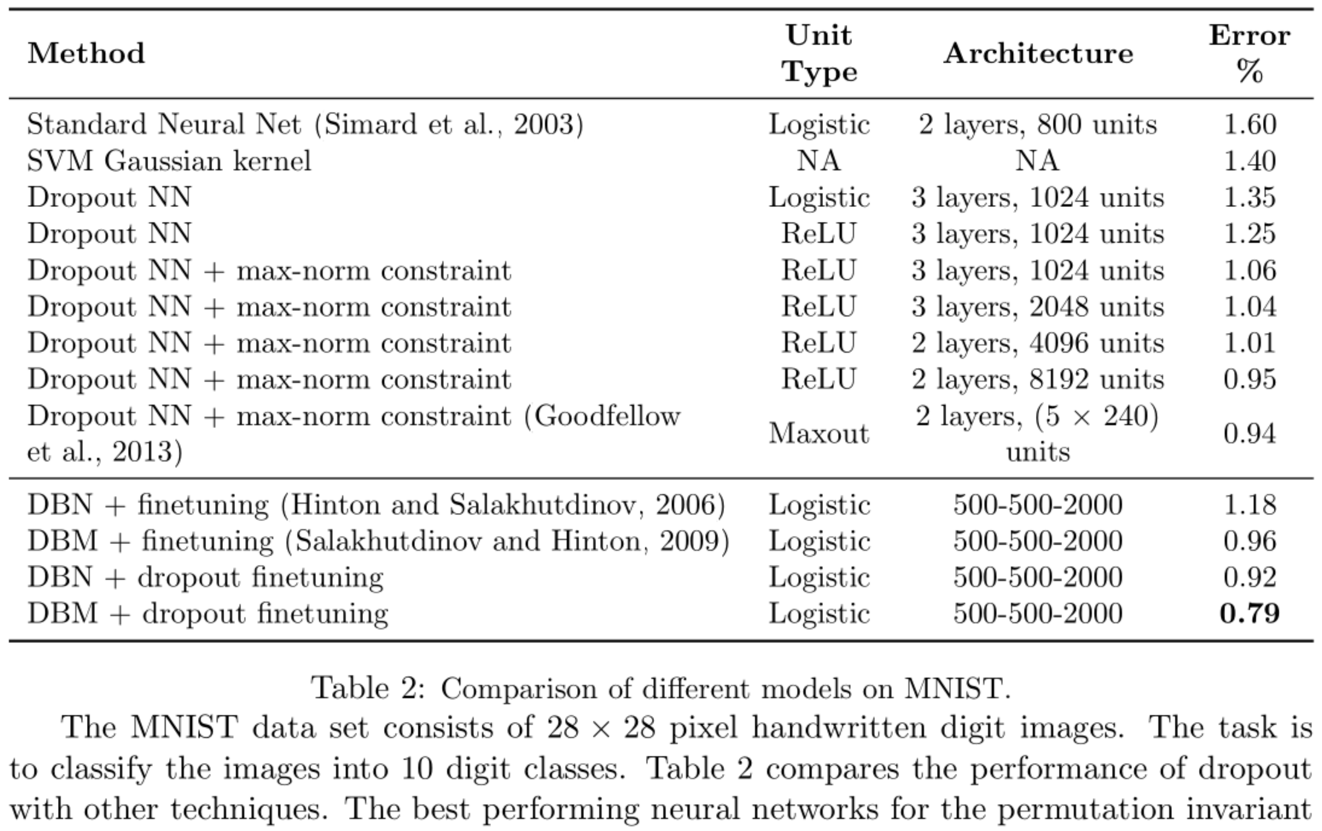


- At prediction time neurons are never dropped, but to compensate we need to scale each weight in the network by the inverse dropout probability 1-p
- With this scaling we approximate the average of the 2^n implicitly trained networks

Srivasta. Improving Neural Networks with Dropout

Srivastava et al. Dropout: A simple way to prevent neural networks from overfitting

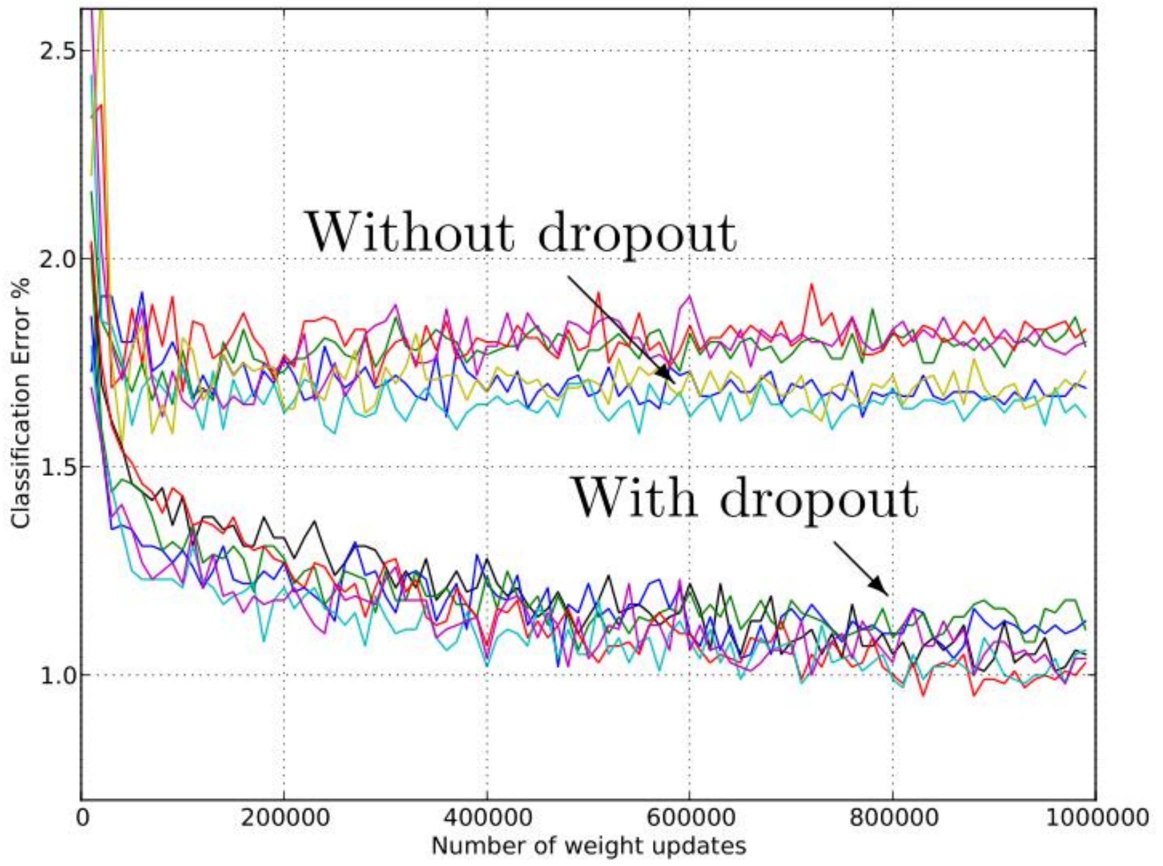
Regularization through Dropout - Performance



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**

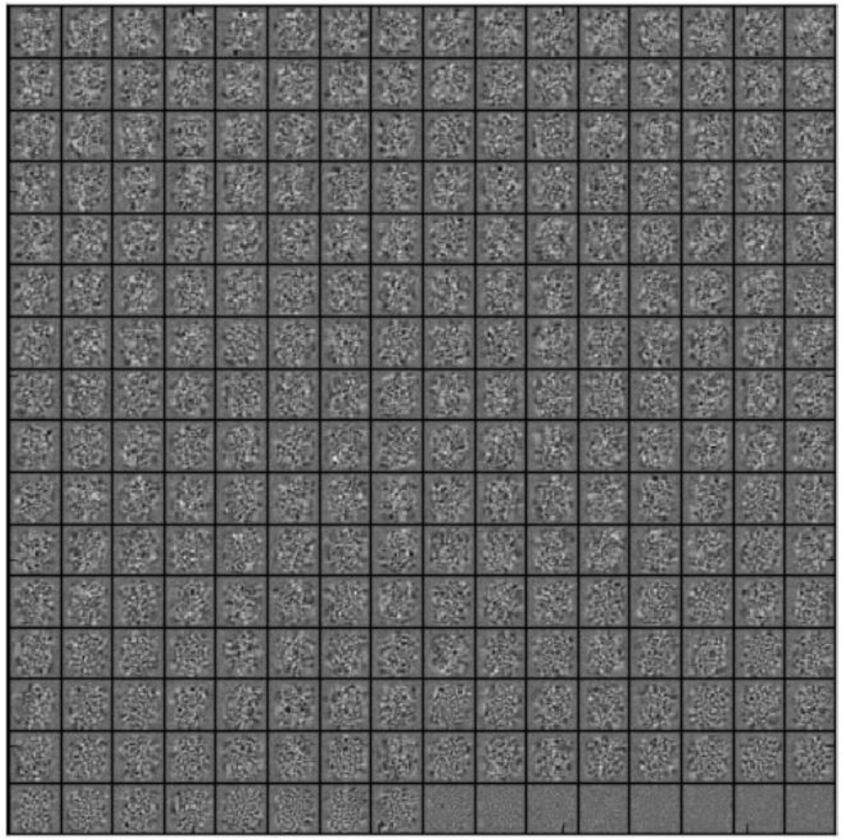
Table 2: Comparison of different models on MNIST.

The MNIST data set consists of 28×28 pixel handwritten digit images. The task is to classify the images into 10 digit classes. Table 2 compares the performance of dropout with other techniques. The best performing neural networks for the permutation invariant



Srivastava et al. Dropout: A simple way to prevent neural networks from overfitting

Regularization through Dropout – Learned units


(a) Without dropout

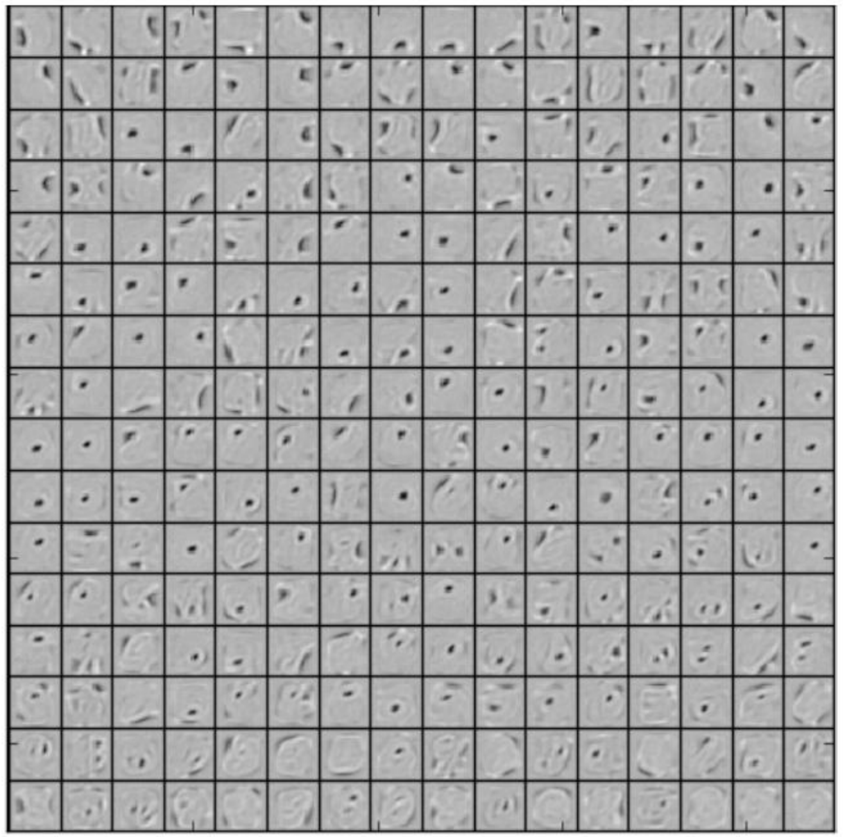
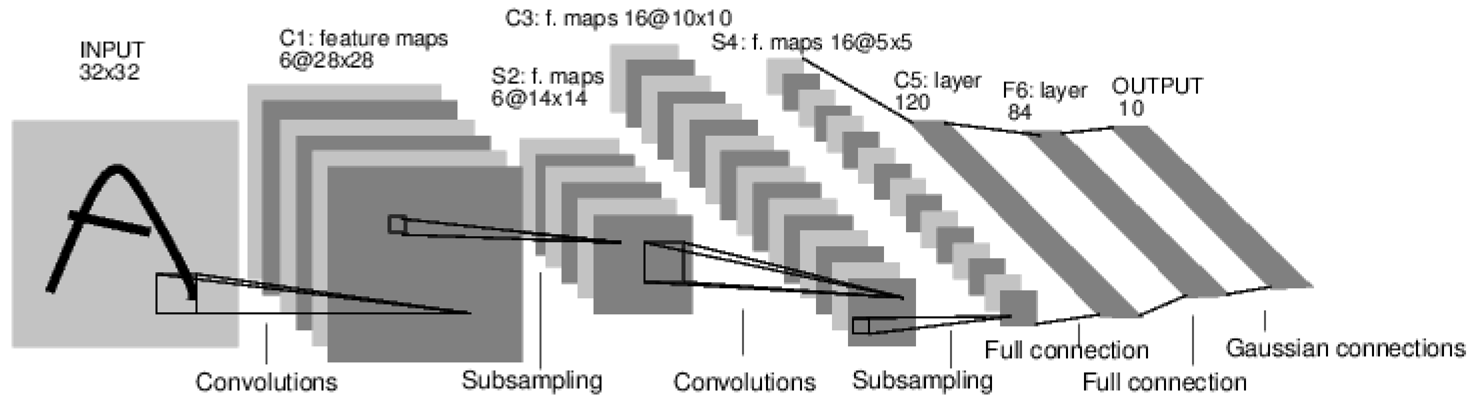

(b) Dropout with $p = 0.5$.

Figure 7: Features learned on MNIST with one hidden layer autoencoders having 256 rectified linear units.

Srivastava et al. Dropout: A simple way to prevent neural networks from overfitting

Designing Deep learning models

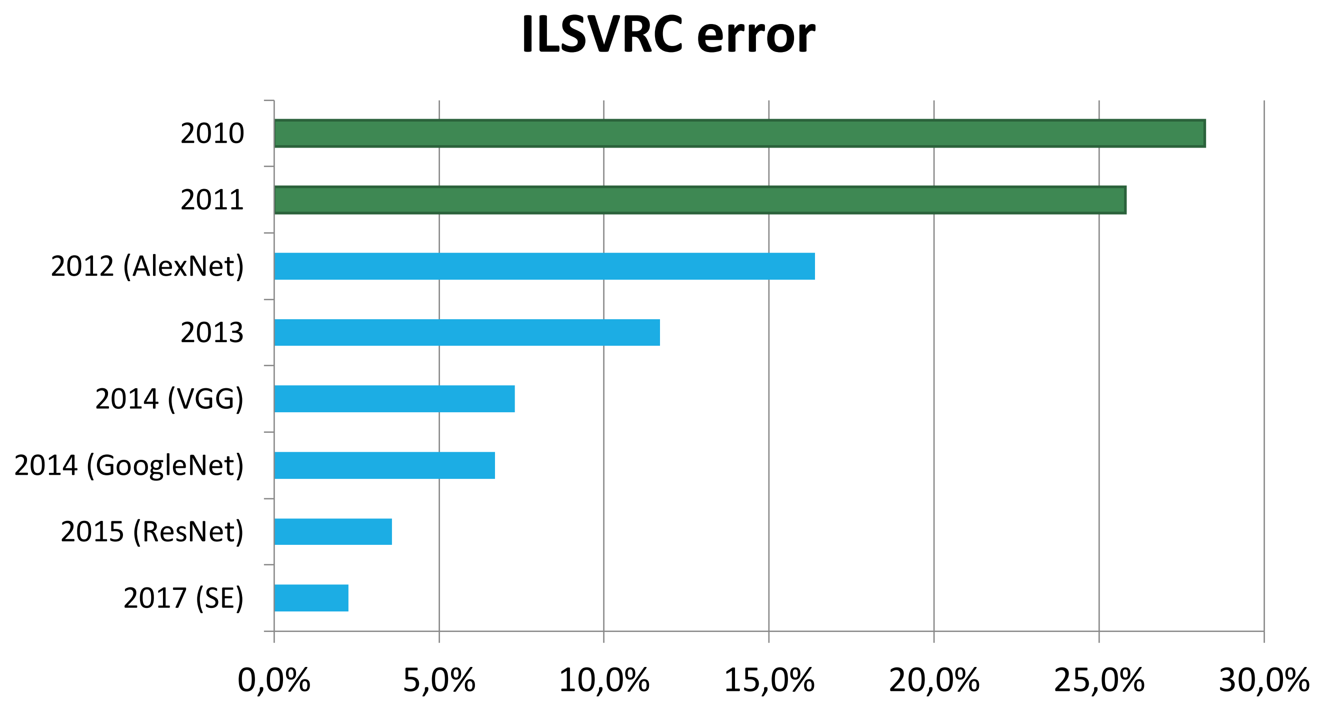


- Think in layers, not neurons
  - Each layer is stacked on top of others to achieve higher levels of abstraction
- Different kinds of layers are used
  - Data input layers
  - Output (loss) layers
  - Fully connected ("classic") layers
  - Convolutional layers for image processing
  - Recurrent layers for sequence processing
  - …

Stacking layers

Trend in the last years: leverage advances in optimization methods, faster hardware and network design to stack more and more layers of neurons.

ImageNet competition: deeper networks every year!



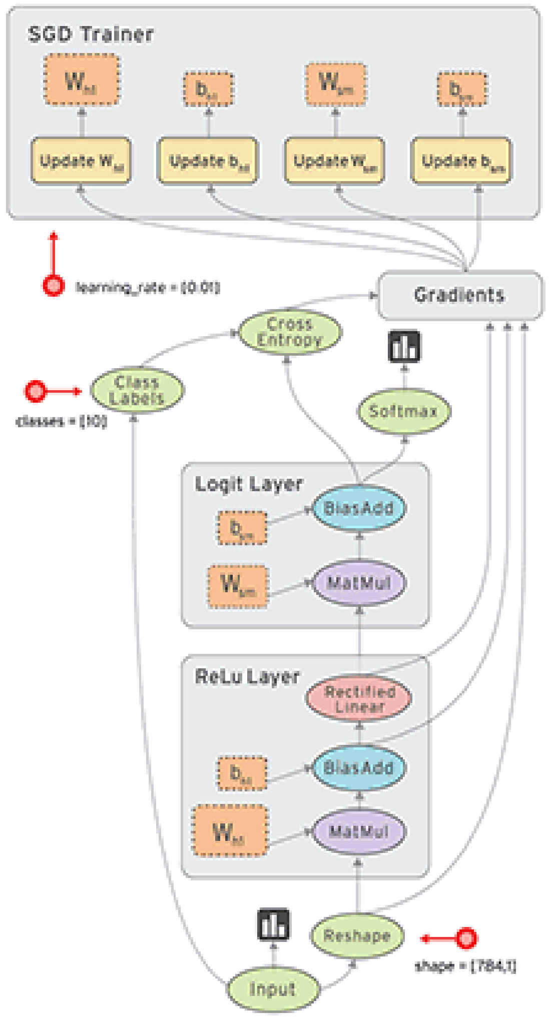
•2012: AlexNet, 8 layers (Krizhevsky et al. – ImageNet Classification with Deep Convolutional Neural Networks)
•2014: VGG, 19 layers (Simonyan & Zisserman – Very Deep Convolutional Networks for Large-Scale Image Recognition)
•2015: ResNet - 152 layers (He et al. – Deep Residual Learning for Image Recognition)
•2017: Squeeze-and-excitation – 152 layers (Hu et al. – Squeeze-and-Excitation Networks)

Modern approach to building deep networks

1. Define network structure: computational graph (structure), layer types (functions), loss

2. Use a automatic differentiation backend to infer backpropagation caculations
   - ➢ The backend automatically generates C/CUDA code

3. Train the network with an optimization method



TensorFlow - https://www.tensorflow.org/

Deep Learning frameworks

High-level
(network design)

keras

PYTORCH

torch

TensorFlow

Chainer

Low-level
(symbolic differentiation backend)

Caffe2

Microsoft CNTK

dmlc mxnet

theano
(deprecated)

Example: learning in TensorFlow

```
import tensorflow as tf
import numpy as np

# Create 100 phony x, y data points in NumPy, y = x * 0.1 + 0.3
x_data = np.random.rand(100).astype(np.float32)
y_data = x_data * 0.1 + 0.3

# Try to find values for W and b that compute y_data = W * x_data + b
# (We know that W should be 0.1 and b 0.3, but Tensorflow will
# figure that out for us.)
W = tf.Variable(tf.random_uniform([1], -1.0, 1.0))
b = tf.Variable(tf.zeros([1]))
y = W * x_data + b

# Minimize the mean squared errors.
loss = tf.reduce_mean(tf.square(y - y_data))
optimizer = tf.train.GradientDescentOptimizer(0.5)
train = optimizer.minimize(loss)

# Before starting, initialize the variables.  We will 'run' this first.
init = tf.initialize_all_variables()

# Launch the graph.
sess = tf.Session()
sess.run(init)

# Fit the line.
for step in xrange(201):
    sess.run(train)
    if step % 20 == 0:
        print(step, sess.run(W), sess.run(b))

# Learns best fit is W: [0.1], b: [0.3]
```

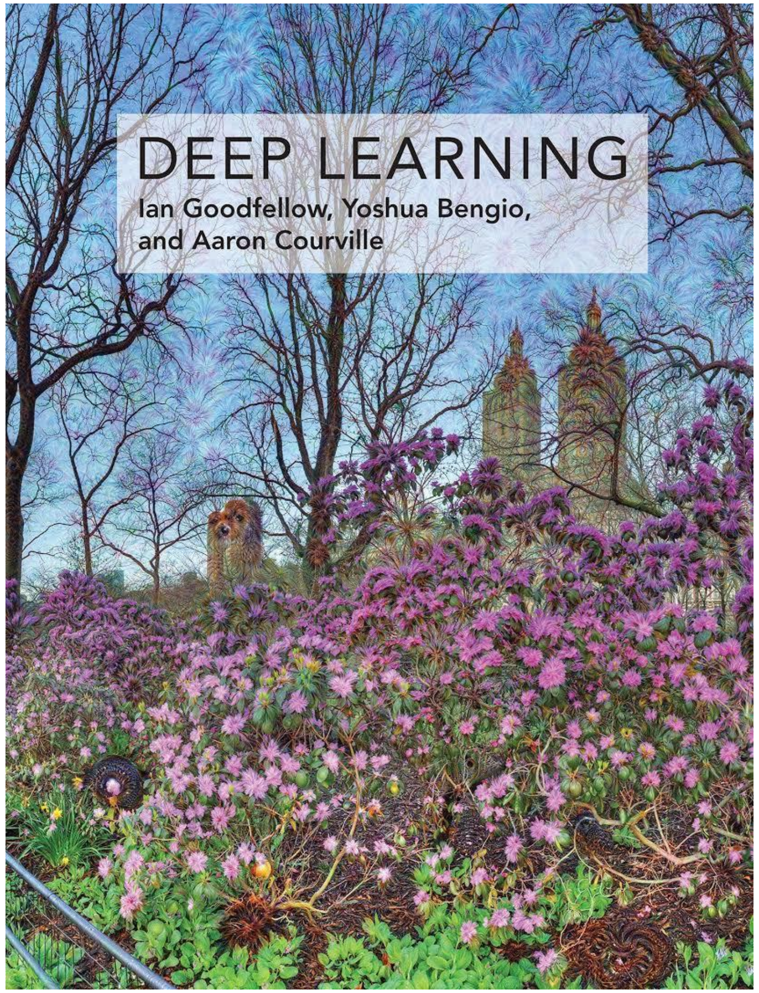
TensorFlow - https://www.tensorflow.org/

Example: learning in Keras

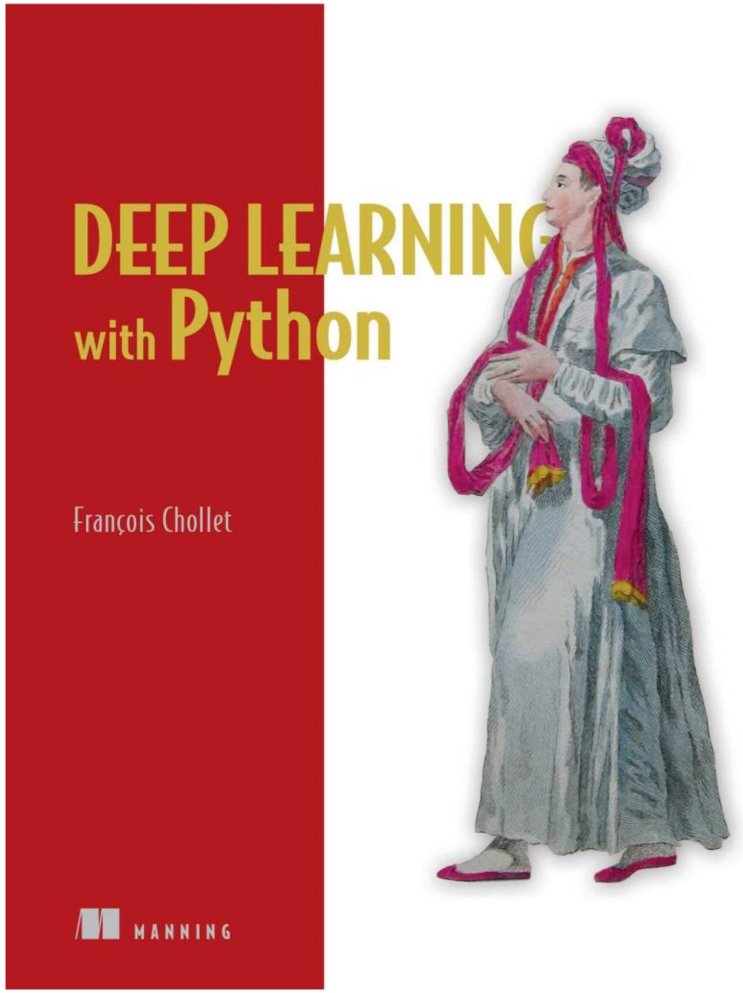
```python
# For a single-input model with 2 classes (binary classification):

model = Sequential()
model.add(Dense(32, activation='relu', input_dim=100))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])

# Generate dummy data
import numpy as np
data = np.random.random((1000, 100))
labels = np.random.randint(2, size=(1000, 1))

# Train the model, iterating on the data in batches of 32 samples
model.fit(data, labels, epochs=10, batch_size=32)
```

Keras - https://keras.io/

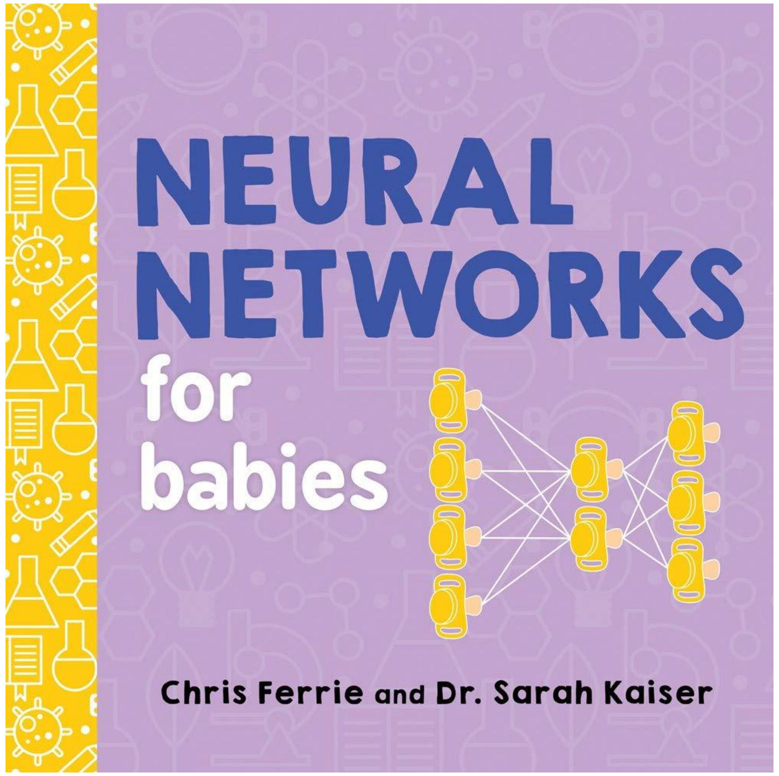
Physical reading material



https://www.deeplearningbook.org/



https://www.amazon.es/Deep-Learning-Python-Francois-Chollet/dp/1617294438



https://www.amazon.com/Neural-Networks-Babies-Baby-University/dp/1492671207

Afi Escuela