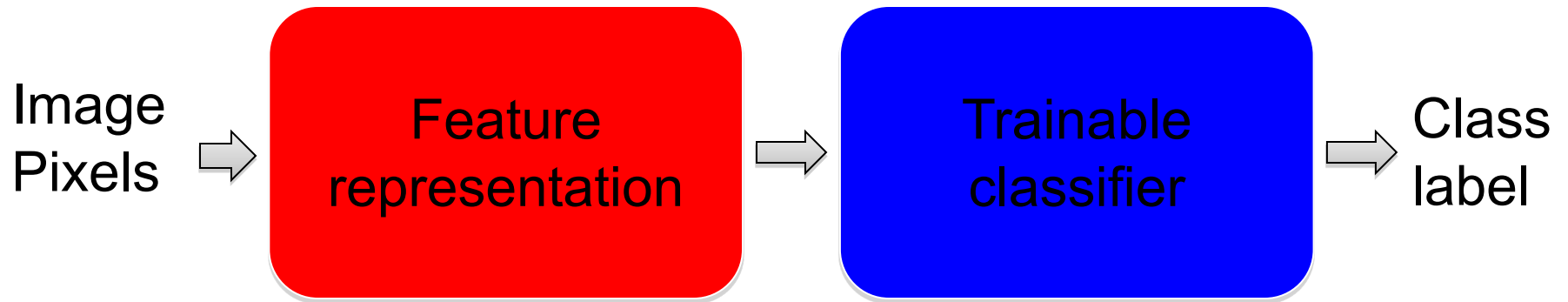


Introduction to Neural Networks

Outline

- Perceptrons
 - Perceptron update rule
- Multi-layer neural networks
 - Training method
- Best practices for training classifiers
- After that: convolutional neural networks

Recall: “Shallow” recognition pipeline



- Hand-crafted feature representation
- Off-the-shelf trainable classifier

“Deep” recognition pipeline



- Learn a *feature hierarchy* from pixels to classifier
- Each layer extracts features from the output of previous layer
- Train all layers jointly

Neural networks vs. SVMs (a.k.a. “deep” vs. “shallow” learning)

deep learning

Search term

support vector machine

Search term

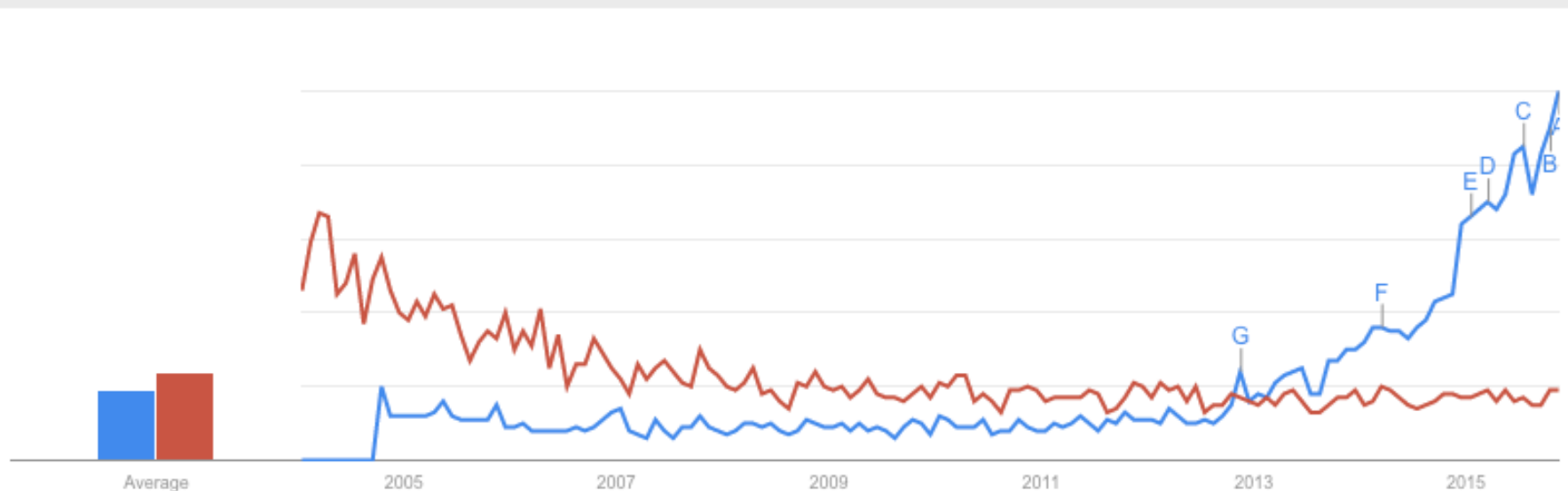
+ Add term



Google Trends

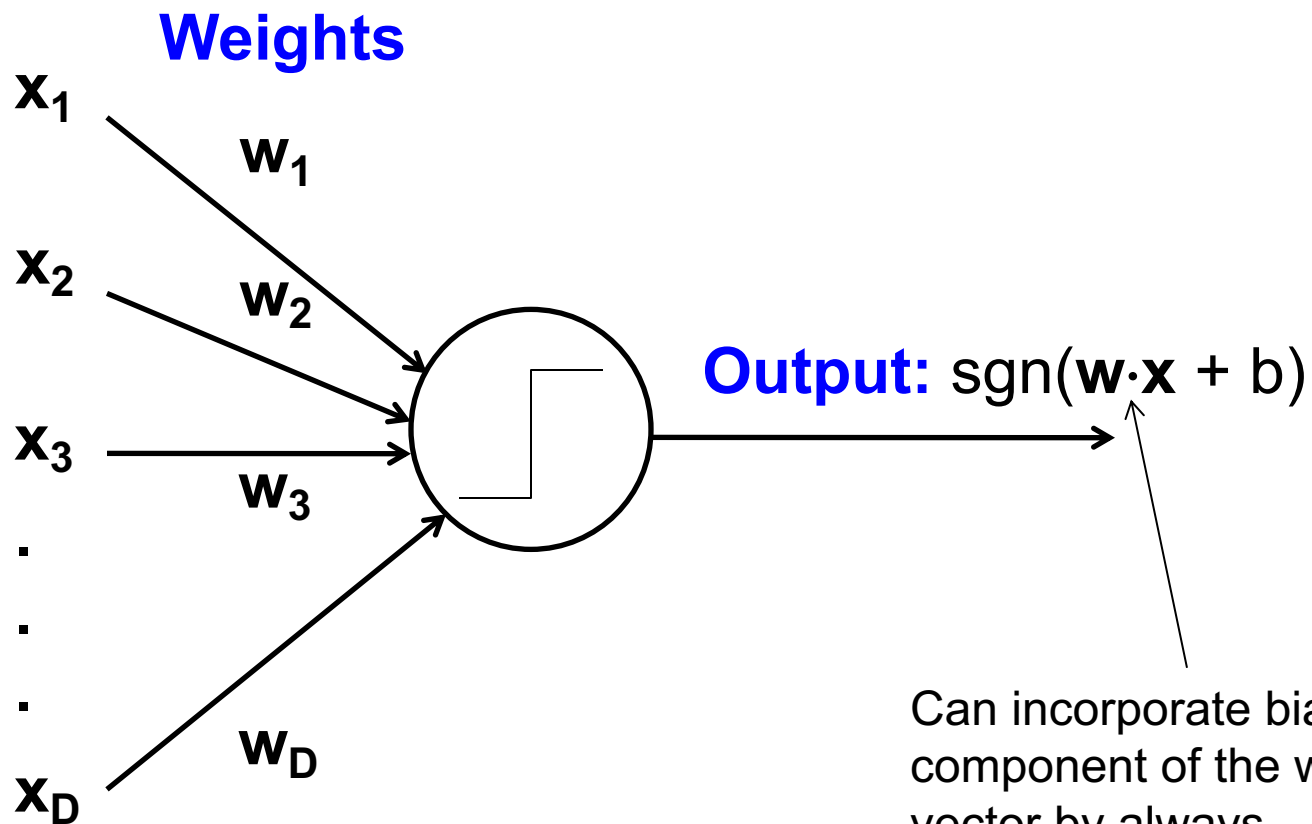
Interest over time ?

News headlines Forecast ?



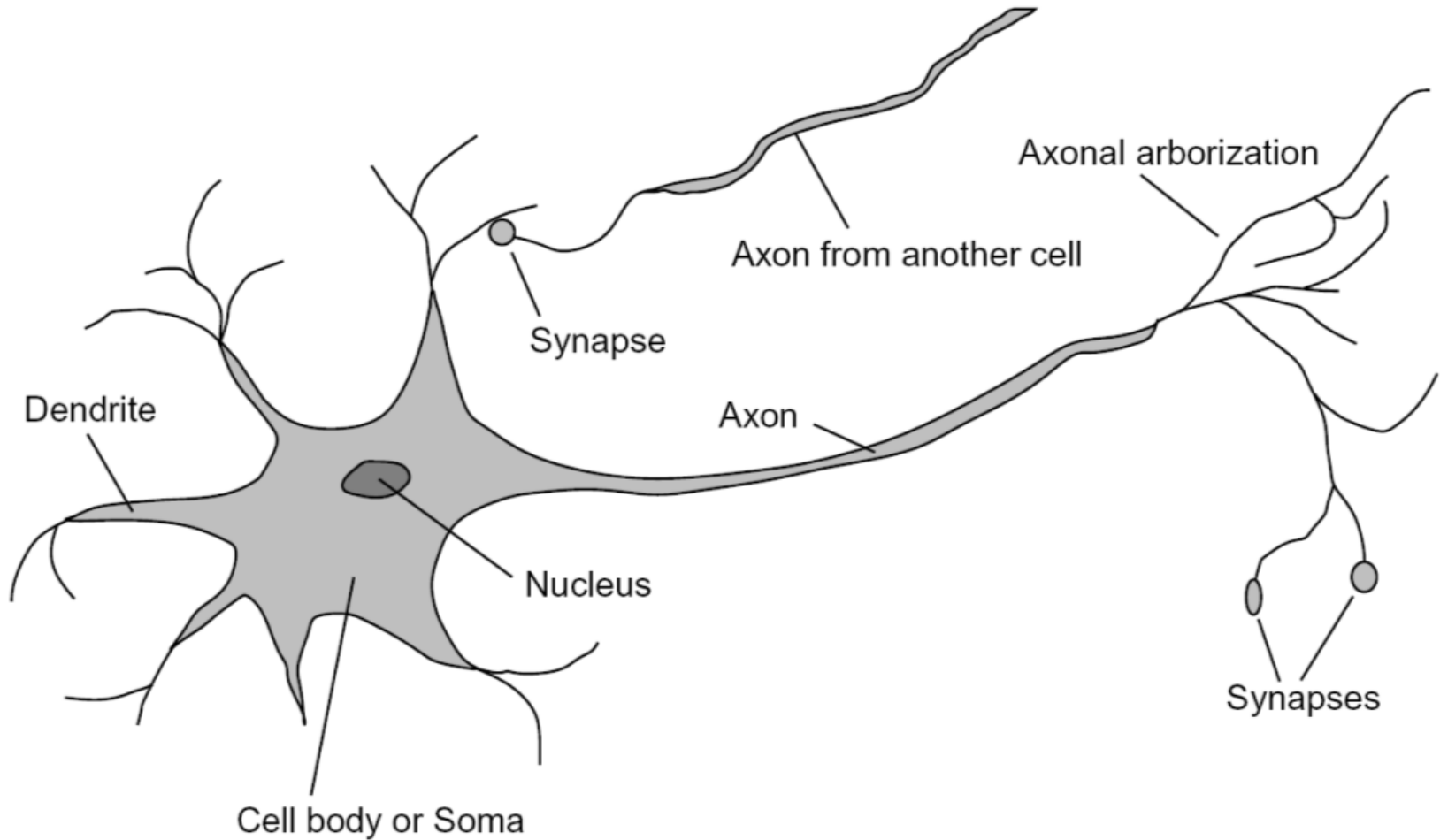
Linear classifiers revisited: Perceptron

Input



Can incorporate bias as component of the weight vector by always including a feature with value set to 1

Loose inspiration: Human neurons



NEW NAVY DEVICE LEARNS BY DOING

Psychologist Shows Embryo
of Computer Designed to
Read and Grow Wiser

WASHINGTON, July 7 (UPI)—The Navy revealed the embryo of an electronic computer today that it expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence.

The embryo—the Weather Bureau's \$2,000,000 "704" computer—learned to differentiate between right and left after fifty attempts in the Navy's demonstration for newsmen.

The service said it would use this principle to build the first of its Perceptron thinking machines that will be able to read and write. It is expected to be finished in about a year at a cost of \$100,000.

Dr. Frank Rosenblatt, designer of the Perceptron, conducted the demonstration. He said the machine would be the first device to think as the human brain. As do human be-

ings, Perceptron will make mistakes at first, but will grow wiser as it gains experience, he said.

Dr. Rosenblatt, a research psychologist at the Cornell Aeronautical Laboratory, Buffalo, said Perceptrons might be fired to the planets as mechanical space explorers.

Without Human Controls

The Navy said the perceptron would be the first non-living mechanism "capable of receiving, recognizing and identifying its surroundings without any human training or control."

The "brain" is designed to remember images and information it has perceived itself. Ordinary computers remember only what is fed into them on punch cards or magnetic tape.

Later Perceptrons will be able to recognize people and call out their names and instantly translate speech in one language to speech or writing in another language, it was predicted.

Mr. Rosenblatt said in principle it would be possible to build brains that could reproduce themselves on an assembly line and which would be conscious of their existence.

1958 New York Times...

In today's demonstration, the "704" was fed two cards, one with squares marked on the left side and the other with squares on the right side.

Learns by Doing

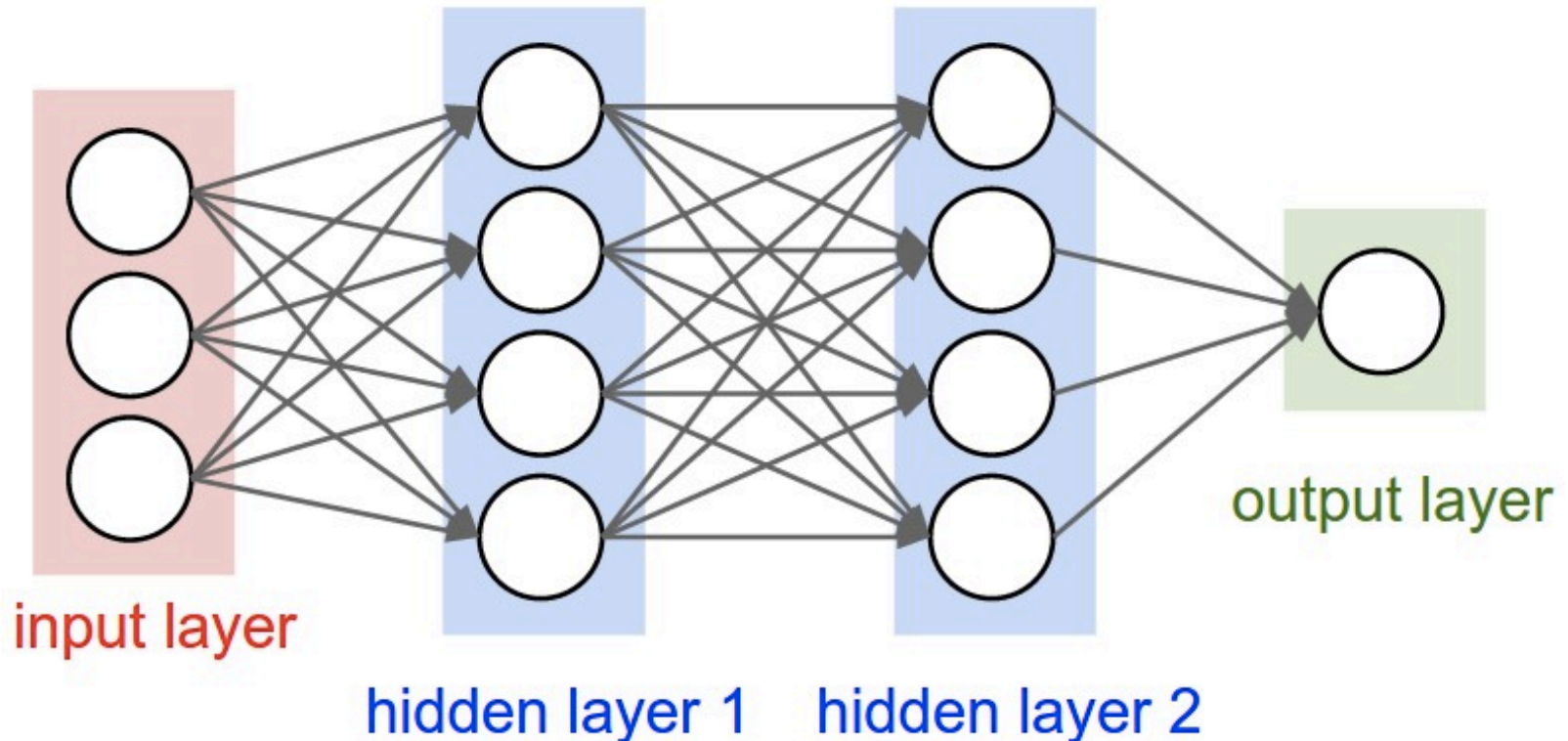
In the first fifty trials, the machine made no distinction between them. It then started registering a "Q" for the left squares and "O" for the right squares.

Dr. Rosenblatt said he could explain why the machine learned only in highly technical terms. But he said the computer had undergone a "self-induced change in the wiring diagram."

The first Perceptron will have about 1,000 electronic "association cells" receiving electrical impulses from an eye-like scanning device with 400 photo-cells. The human brain has 10,000,000,000 responsive cells, including 100,000,000 connections with the eyes.

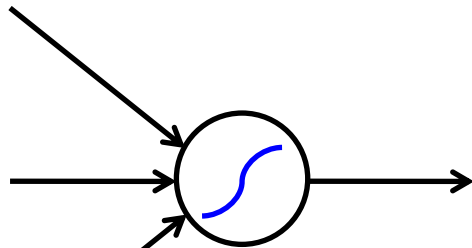
Multi-layer perceptrons

- To make nonlinear classifiers out of perceptrons, build a multi-layer neural network!
 - This requires each perceptron to have a nonlinearity

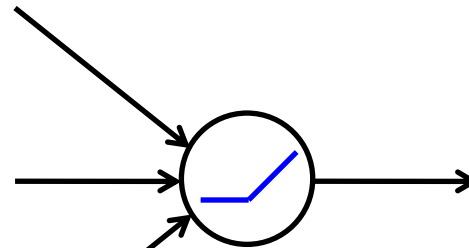


Multi-layer perceptrons

- To make nonlinear classifiers out of perceptrons, build a multi-layer neural network!
 - This requires each perceptron to have a nonlinearity
 - To be trainable, the nonlinearity should be *differentiable*



Sigmoid: $g(t) = \frac{1}{1 + e^{-t}}$



Rectified linear unit (ReLU): $g(t) = \max(0, t)$

Training of multi-layer networks

- Find network weights to minimize the prediction loss between true and estimated labels of training examples:

$$E(\mathbf{w}) = \sum_i l(\mathbf{x}_i, y_i; \mathbf{w})$$

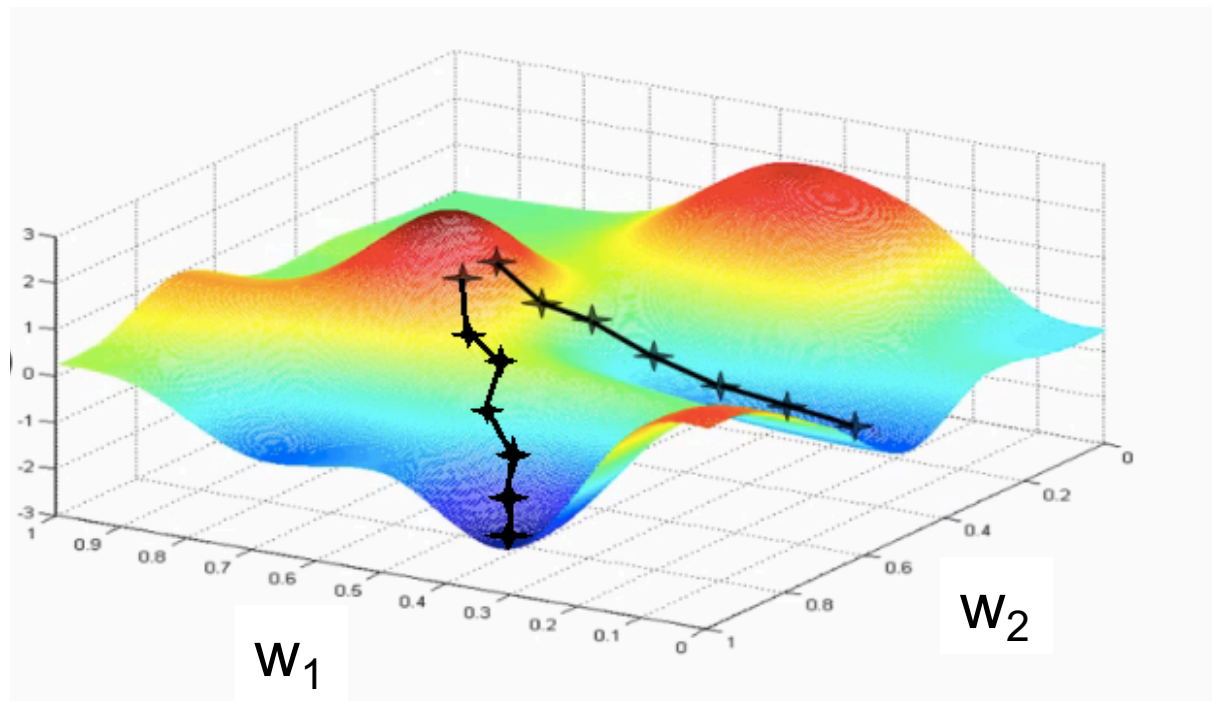
- Possible losses (for binary problems):
 - Quadratic loss: $l(\mathbf{x}_i, y_i; \mathbf{w}) = (f_{\mathbf{w}}(\mathbf{x}_i) - y_i)^2$
 - Log likelihood loss: $l(\mathbf{x}_i, y_i; \mathbf{w}) = -\log P_{\mathbf{w}}(y_i | \mathbf{x}_i)$
 - Hinge loss: $l(\mathbf{x}_i, y_i; \mathbf{w}) = \max(0, 1 - y_i f_{\mathbf{w}}(\mathbf{x}_i))$

Training of multi-layer networks

- Find network weights to minimize the prediction loss between true and estimated labels of training examples:

$$E(\mathbf{w}) = \sum_i l(\mathbf{x}_i, y_i; \mathbf{w})$$

- Update weights by **gradient descent**: $\mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{\partial E}{\partial \mathbf{w}}$



Training of multi-layer networks


- Find network weights to minimize the prediction loss between true and estimated labels of training examples:

$$E(\mathbf{w}) = \sum_i l(\mathbf{x}_i, y_i; \mathbf{w})$$

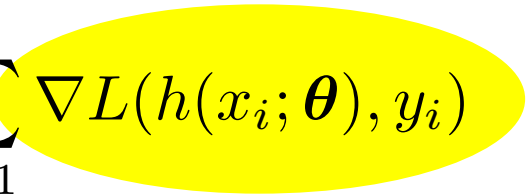
- Update weights by **gradient descent**: $\mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{\partial E}{\partial \mathbf{w}}$
- **Back-propagation**: gradients are computed in the direction from output to input layers and combined using chain rule
- **Stochastic gradient descent**: compute the weight update w.r.t. one training example (or a small batch of examples) at a time, cycle through training examples in random order in multiple epochs

Back-propagation

Back-propagation

$$\min_{\boldsymbol{\theta}} \frac{1}{N} \sum_{i=1}^N L(h(x_i; \boldsymbol{\theta}), y_i)$$


Convolutional network

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \lambda \frac{1}{N} \sum_{i=1}^N \nabla L(h(x_i; \boldsymbol{\theta}), y_i)$$


Gradient descent update

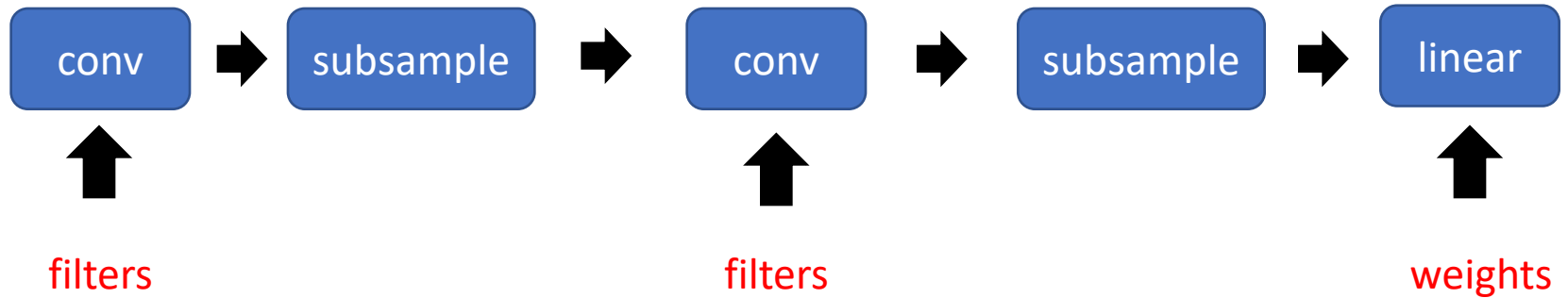
Computing the gradient of the loss

$$\nabla L(h(x; \boldsymbol{\theta}), y)$$

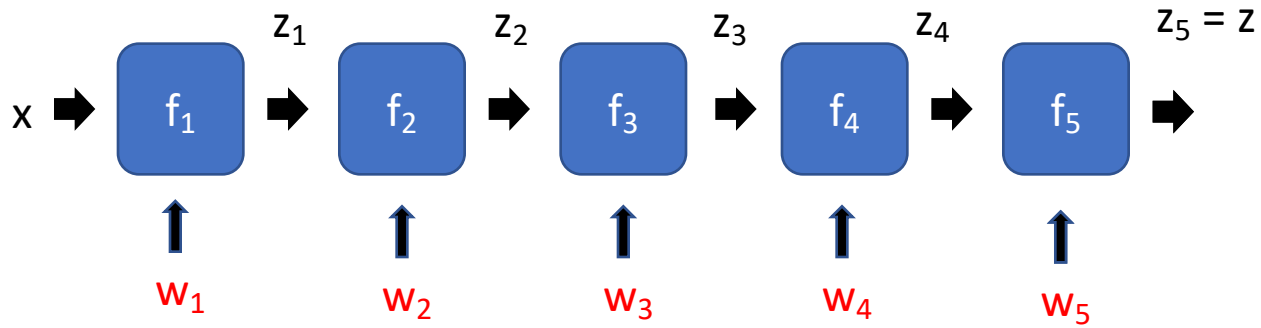
$$z = h(x; \boldsymbol{\theta})$$

$$\nabla_{\boldsymbol{\theta}} L(z, y) = \frac{\partial L(z, y)}{\partial z} \frac{\partial z}{\partial \boldsymbol{\theta}}$$

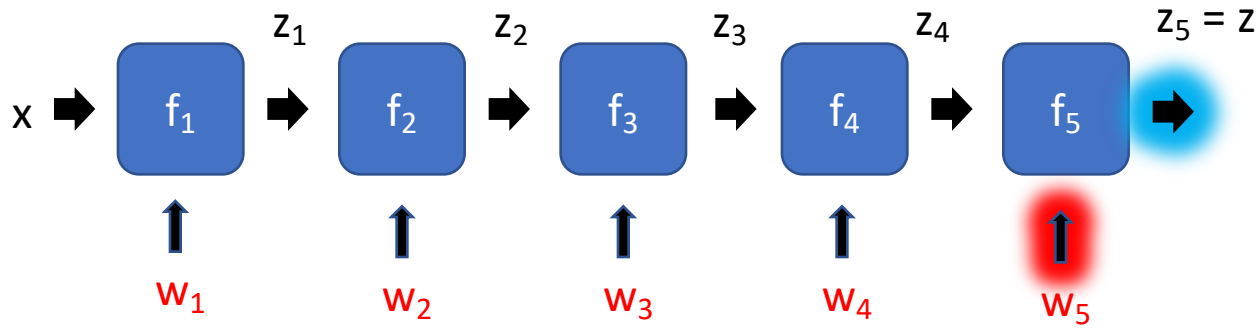
Convolutional networks



Gradient Computation

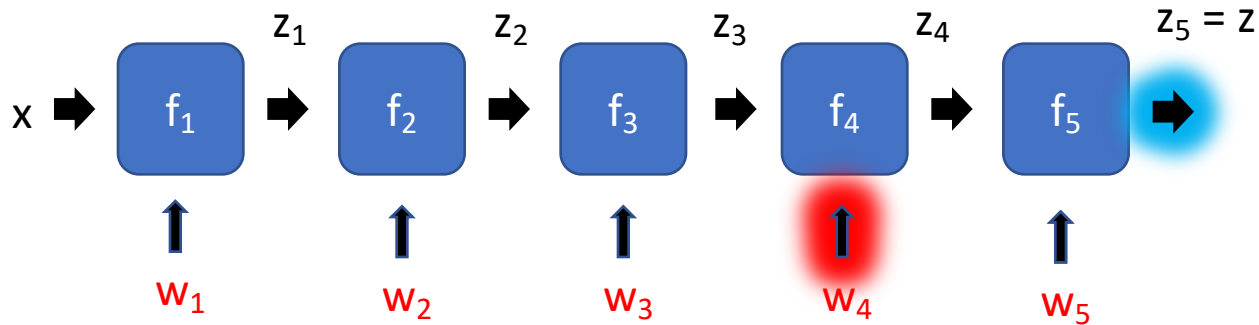


Gradient Computation



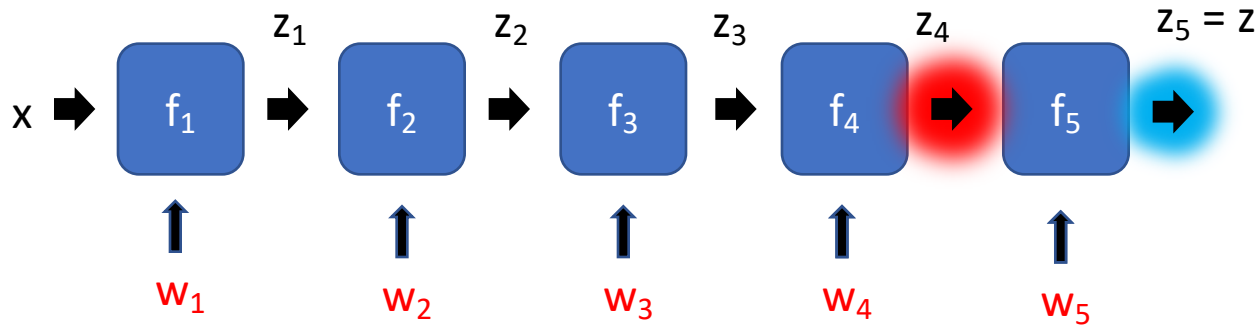
$$\frac{\partial z}{\partial w_5}$$

Gradient Computation



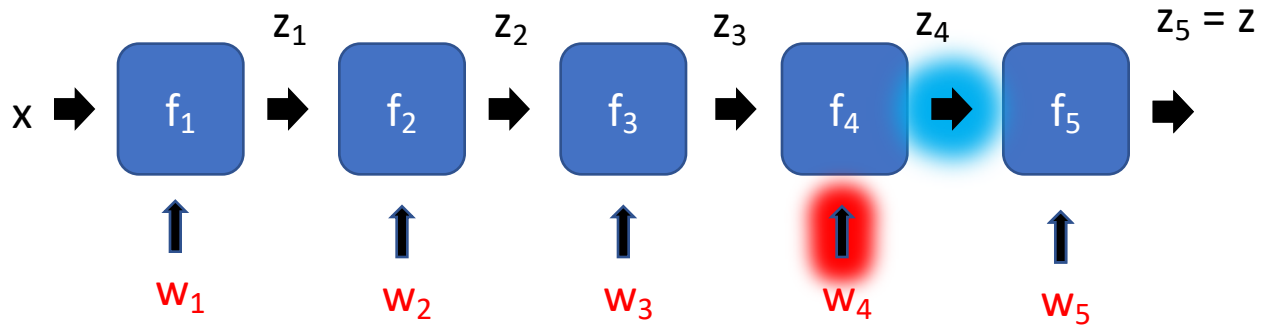
$$\frac{\partial z}{\partial w_4}$$

Gradient Computation



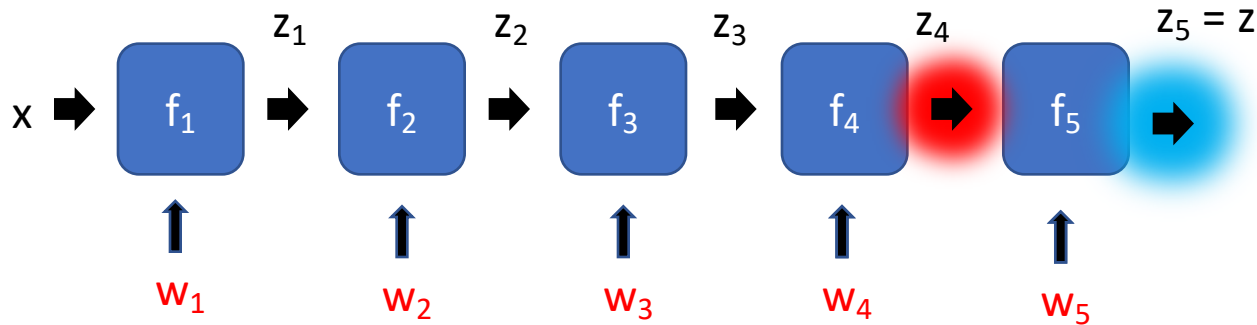
$$\frac{\partial z}{\partial w_4} = \frac{\partial z}{\partial z_4} \frac{\partial z_4}{\partial w_4} :$$

Gradient Computation



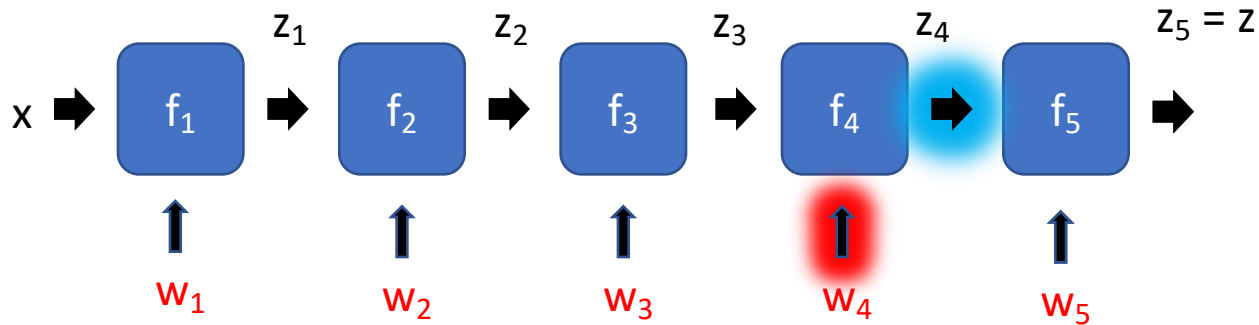
$$\frac{\partial z}{\partial w_4} = \frac{\partial z}{\partial z_4} \frac{\partial z_4}{\partial w_4} :$$

Gradient Computation



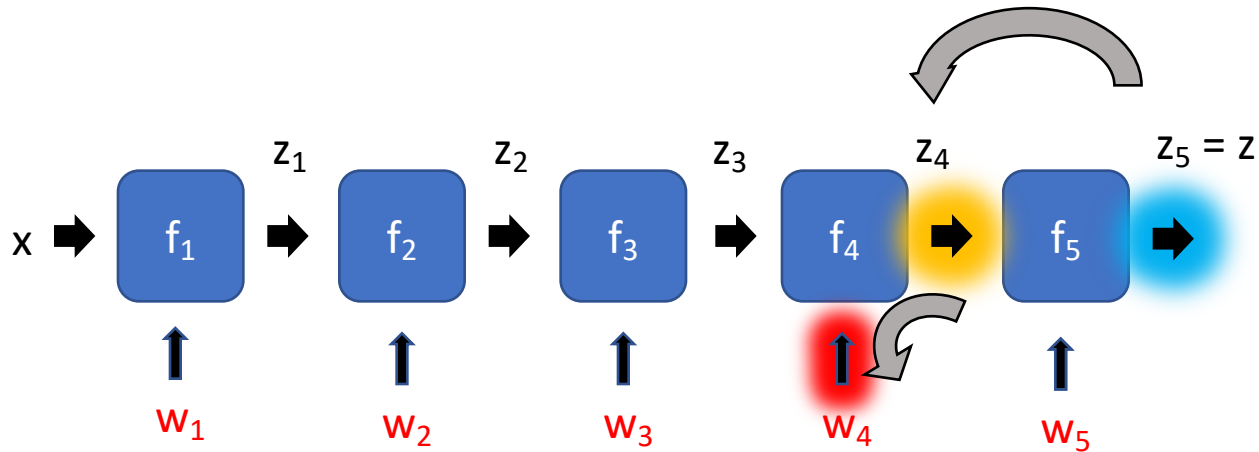
$$\frac{\partial z}{\partial w_4} = \frac{\partial z}{\partial z_4} \frac{\partial z_4}{\partial w_4} = \frac{\partial f_5(z_4, w_5)}{\partial z_4} \frac{\partial f_4(z_3, w_4)}{\partial w_4}$$

Gradient Computation



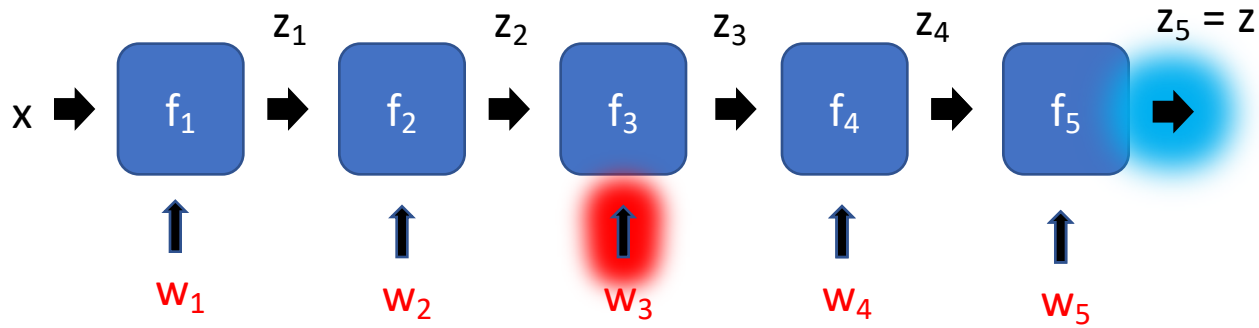
$$\frac{\partial z}{\partial w_4} = \frac{\partial z}{\partial z_4} \frac{\partial z_4}{\partial w_4} = \frac{\partial f_5(z_4, w_5)}{\partial z_4} \frac{\partial f_4(z_3, w_4)}{\partial w_4}$$

Gradient Computation



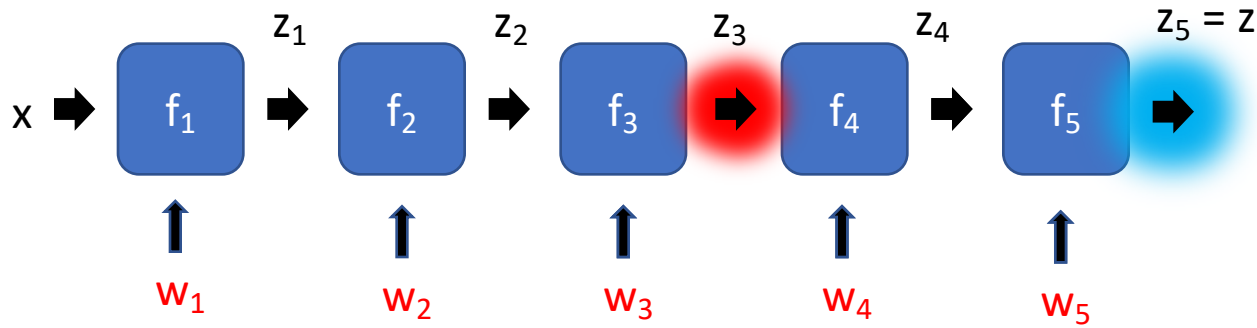
$$\frac{\partial z}{\partial w_4} = \frac{\partial z}{\partial z_4} \frac{\partial z_4}{\partial w_4} = \frac{\partial f_5(z_4, w_5)}{\partial z_4} \frac{\partial f_4(z_3, w_4)}{\partial w_4}$$

Gradient Computation



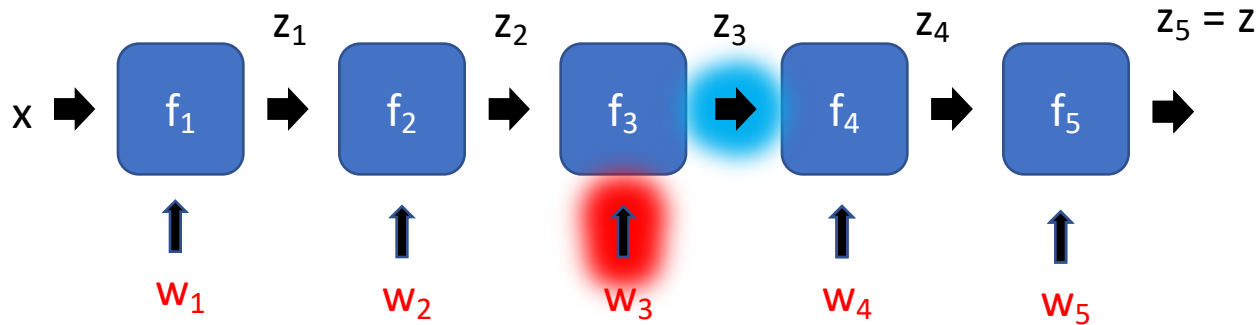
$$\frac{\partial z}{\partial w_3}$$

Gradient Computation



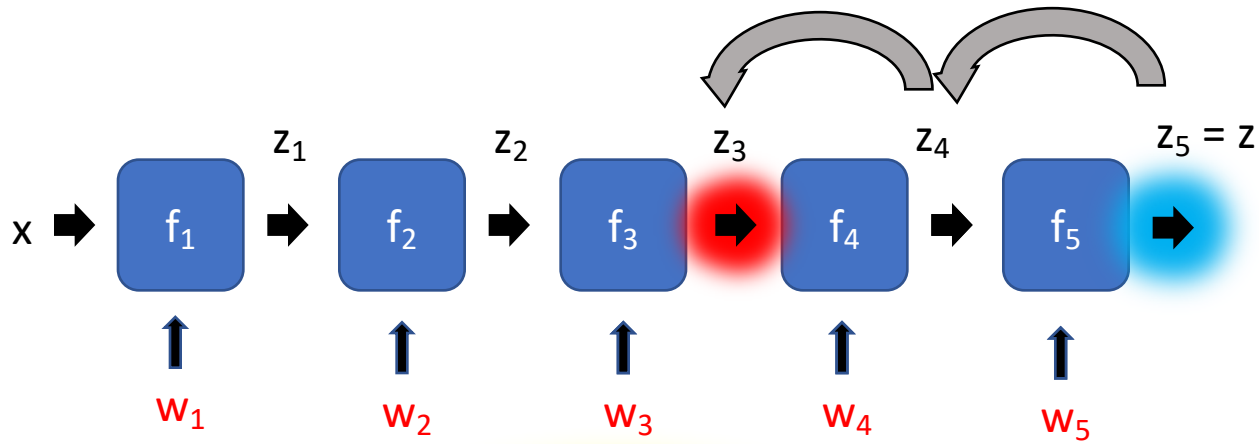
$$\frac{\partial z}{\partial w_3} = \frac{\partial z}{\partial z_3} \frac{\partial z_3}{\partial w_3}$$

Gradient Computation



$$\frac{\partial z}{\partial w_3} = \frac{\partial z}{\partial z_3} \frac{\partial z_3}{\partial w_3}$$

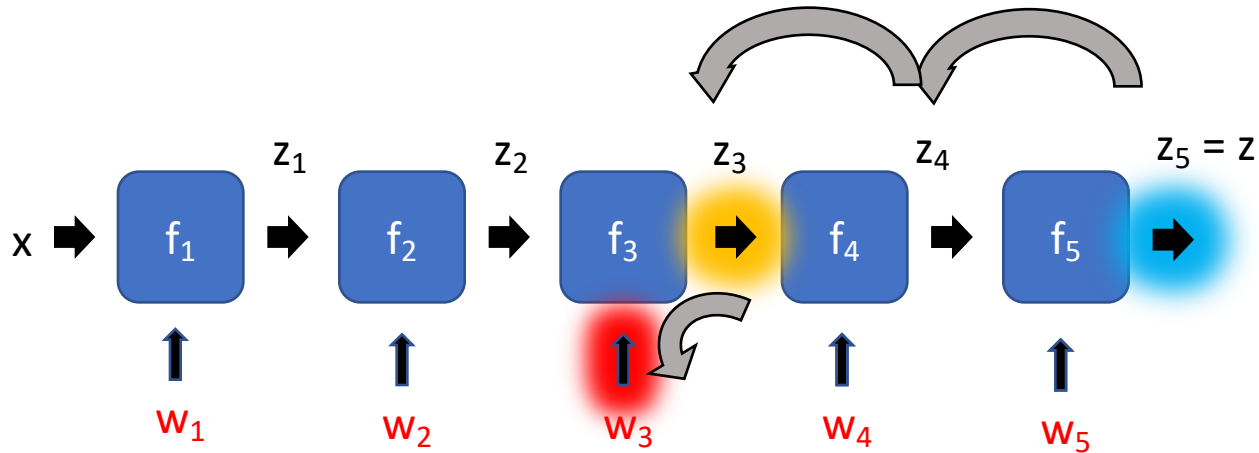
Gradient Computation



$$\frac{\partial z}{\partial z_3} = \frac{\partial z}{\partial z_4} \frac{\partial z_4}{\partial z_3}$$

$$\frac{\partial z}{\partial w_3} = \frac{\partial z}{\partial z_3} \frac{\partial z_3}{\partial w_3}$$

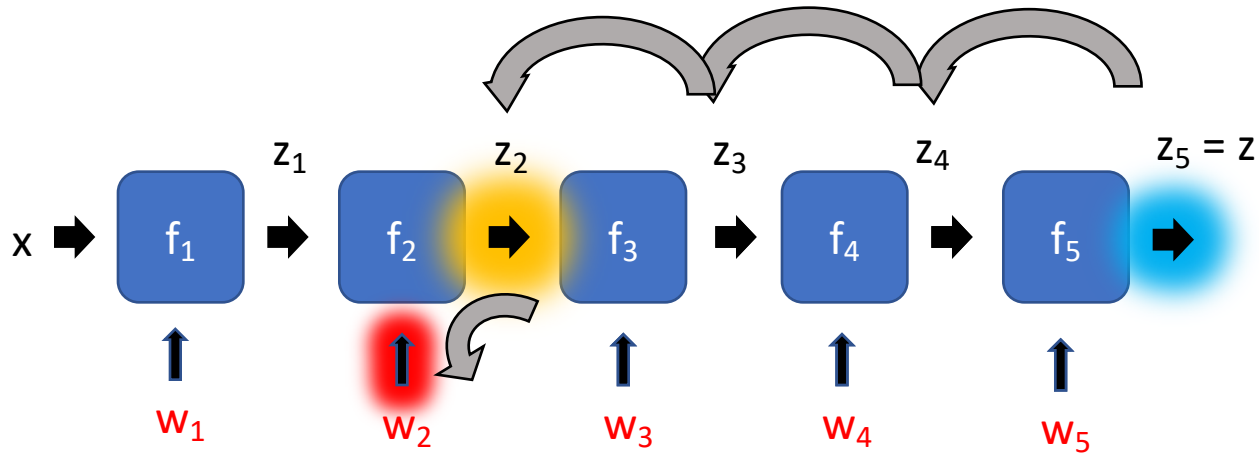
Gradient Computation



$$\frac{\partial z}{\partial z_3} = \frac{\partial z}{\partial z_4} \frac{\partial z_4}{\partial z_3}$$

$$\frac{\partial z}{\partial w_3} = \frac{\partial z}{\partial z_3} \frac{\partial z_3}{\partial w_3}$$

Gradient Computation

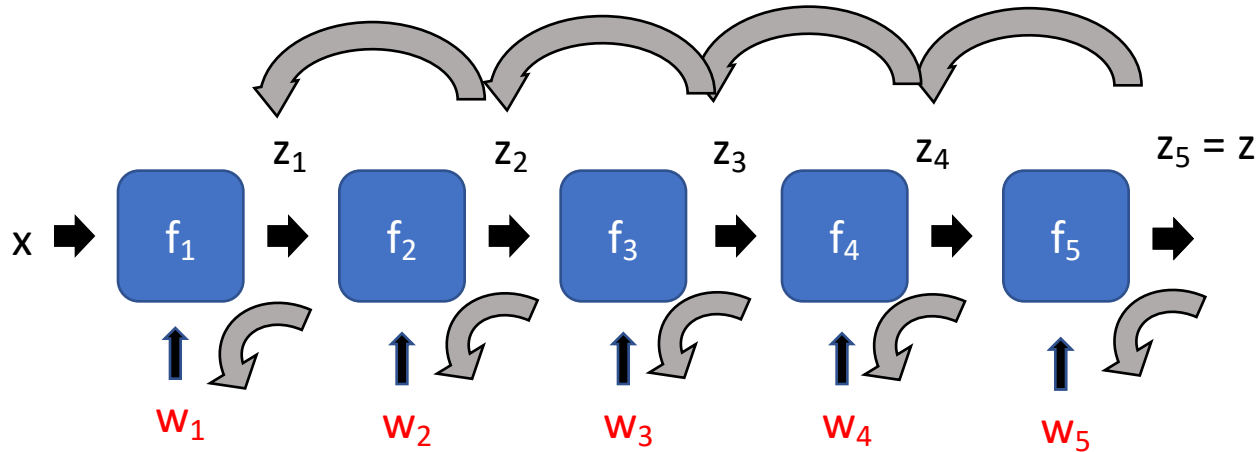


$$\frac{\partial z}{\partial z_2} = \frac{\partial z}{\partial z_3} \frac{\partial z_3}{\partial z_2}$$

$$\frac{\partial z}{\partial w_2} = \frac{\partial z}{\partial z_2} \frac{\partial z_2}{\partial w_2}$$

Recurrence going backward!!

Gradient Computation



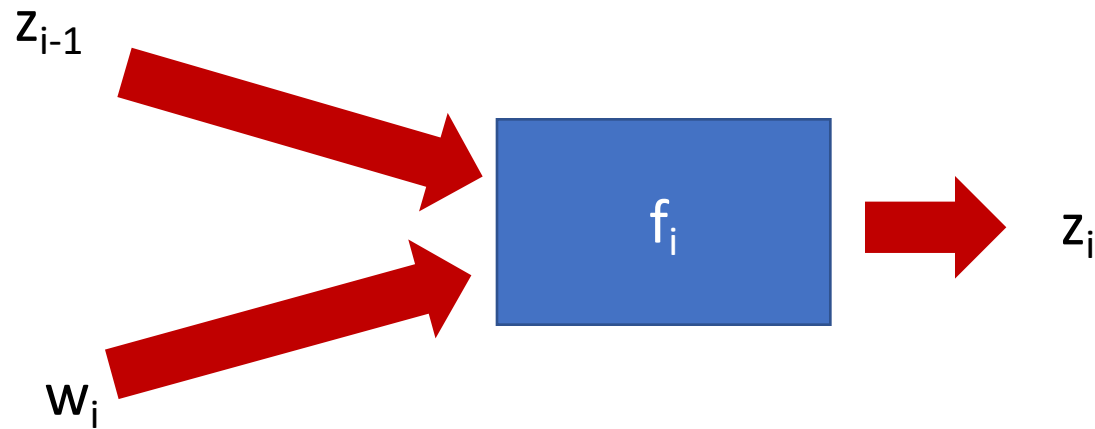
Back-Propagation

Backpropagation for a sequence of functions

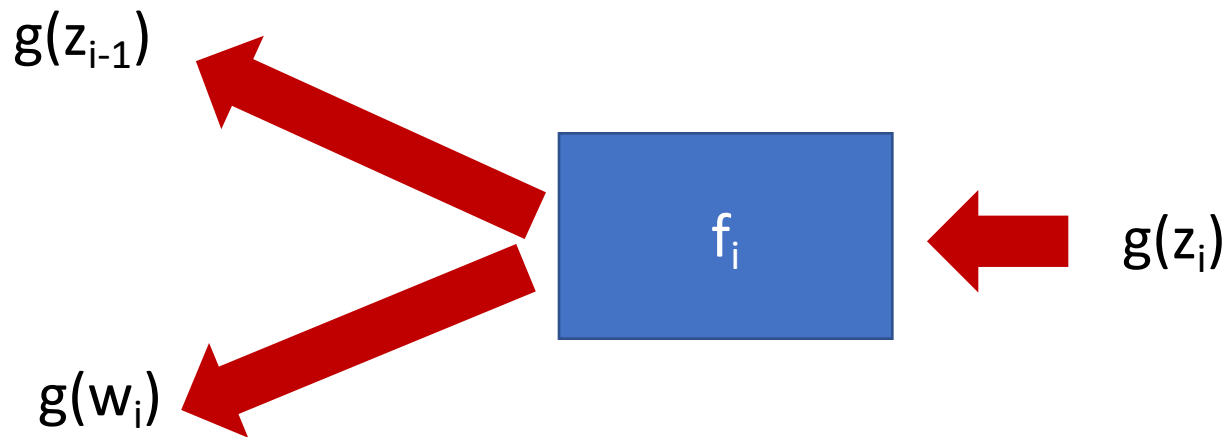
- Each “function” has a “forward” and “backward” module
- Forward module for f_i
 - takes z_{i-1} and weight w_i as input
 - produces z_i as output
- Backward module for f_i
 - takes $g(z_i)$ as input
 - produces $g(z_{i-1})$ and $g(w_i)$ as output

$$g(z_{i-1}) = g(z_i) \frac{\partial z_i}{\partial z_{i-1}} \quad g(w_i) = g(z_i) \frac{\partial z_i}{\partial w_i}$$

Backpropagation for a sequence of functions



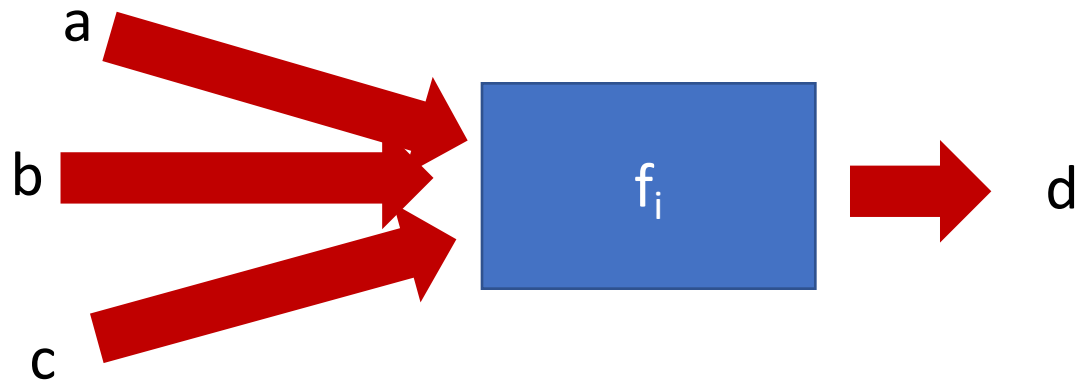
Backpropagation for a sequence of functions



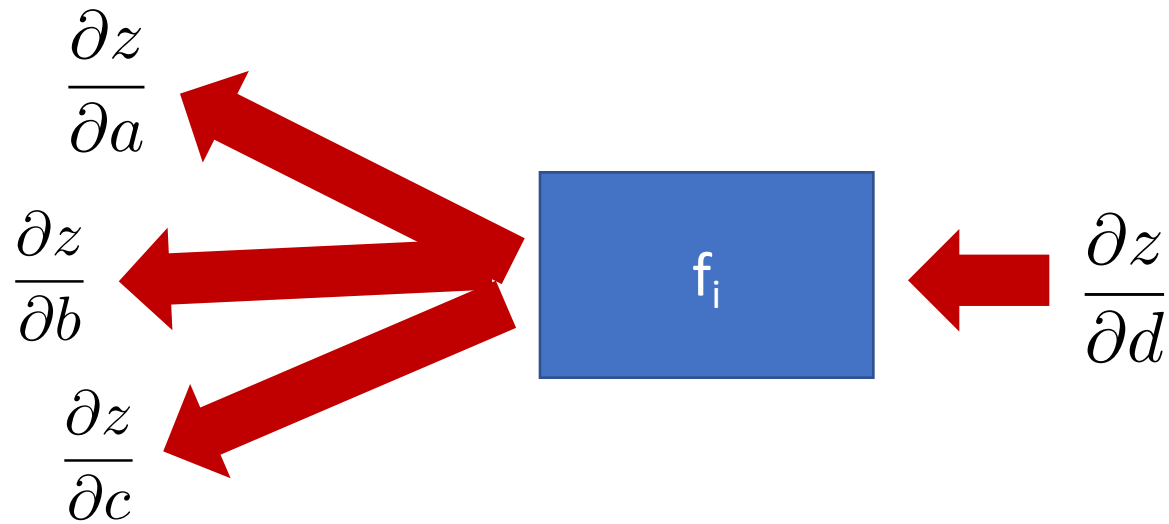
Computation graph - Functions

- Each node implements two functions
 - A “forward”
 - Computes output given input
 - A “backward”
 - Computes derivative of z w.r.t input, given derivative of z w.r.t output

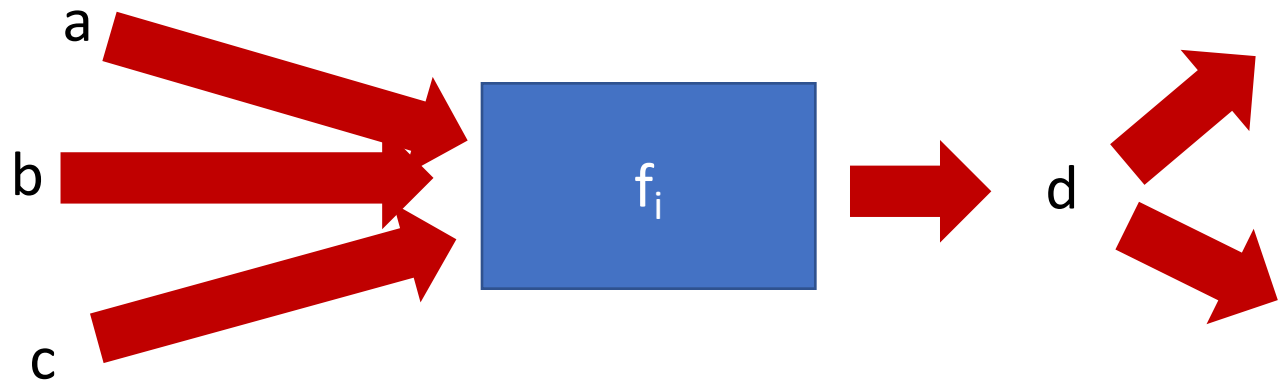
Computation graphs



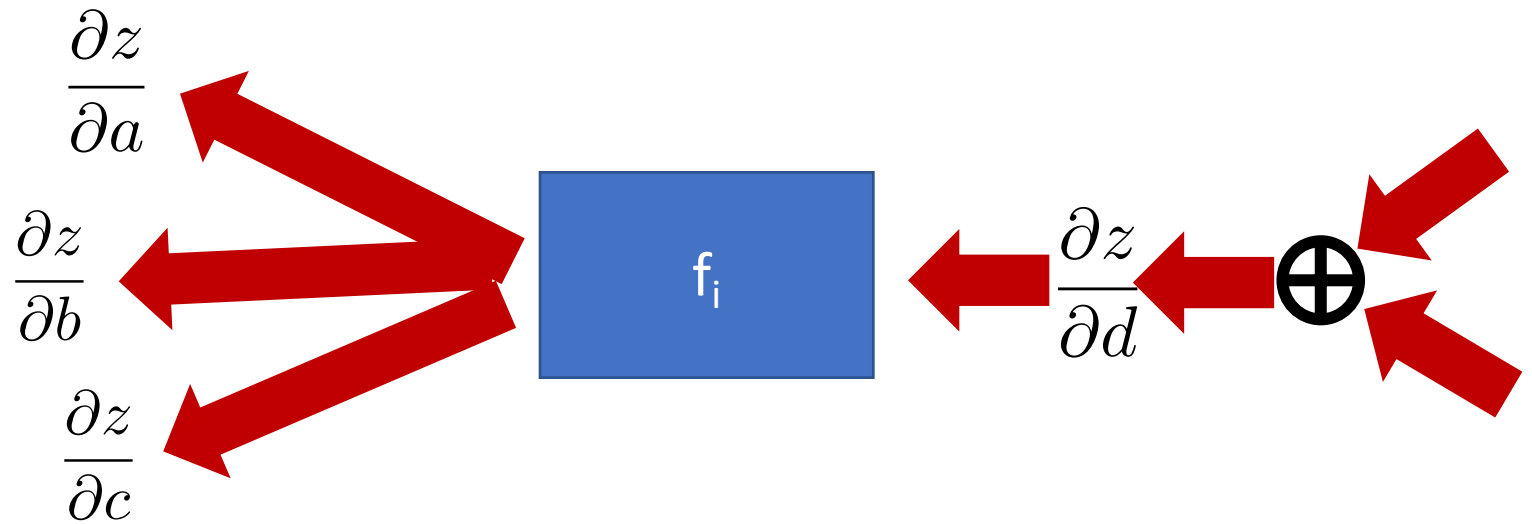
Computation graphs



Computation graphs



Computation graphs

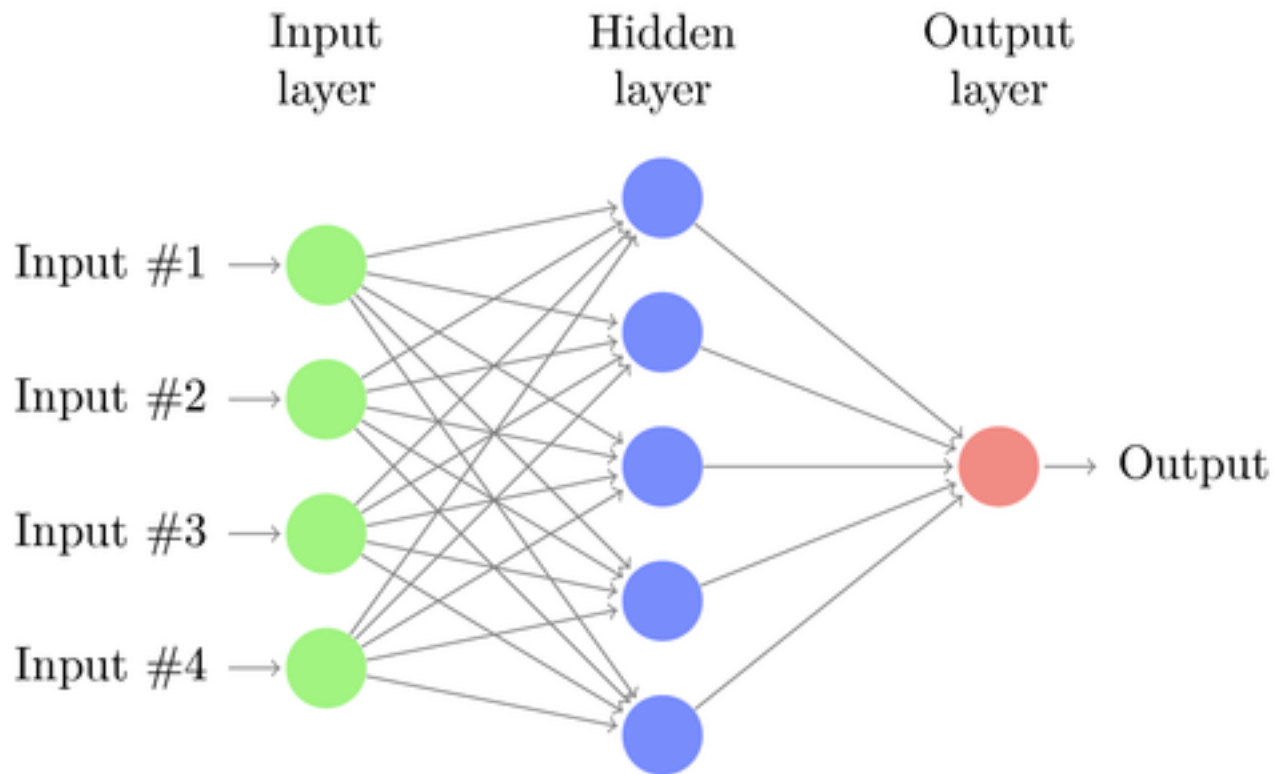


Neural network frameworks



Network with a single hidden layer

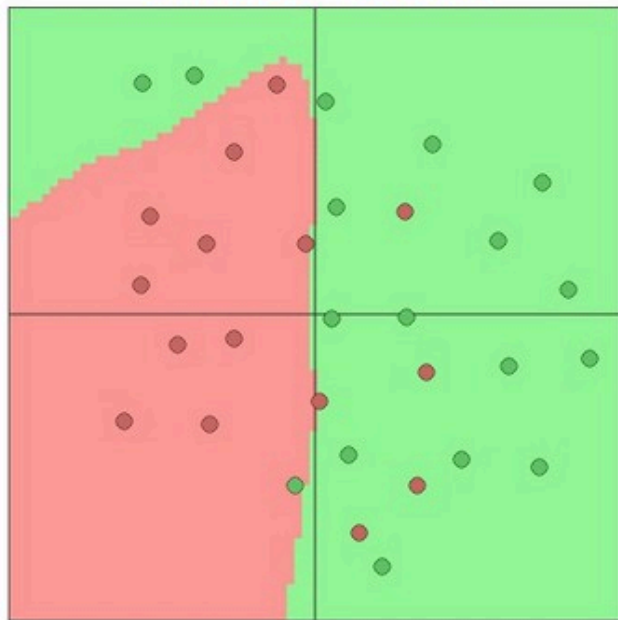
- Neural networks with at least one hidden layer are *universal function approximators*



Network with a single hidden layer

- Hidden layer size and *network capacity*:

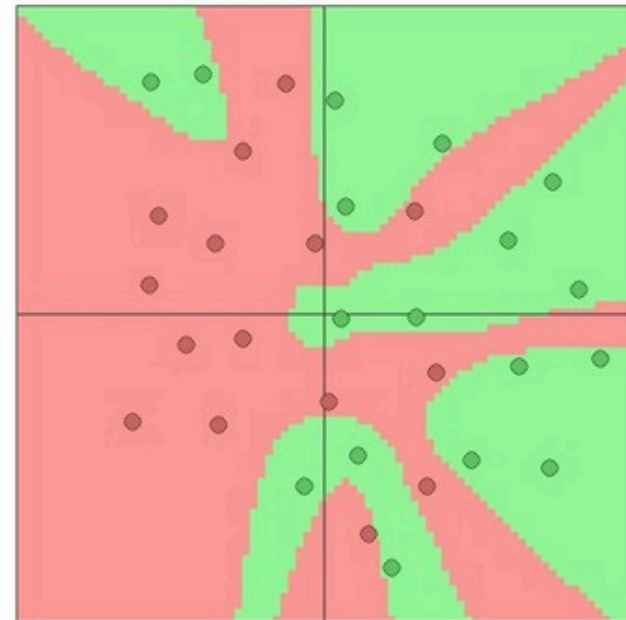
3 hidden neurons



6 hidden neurons



20 hidden neurons

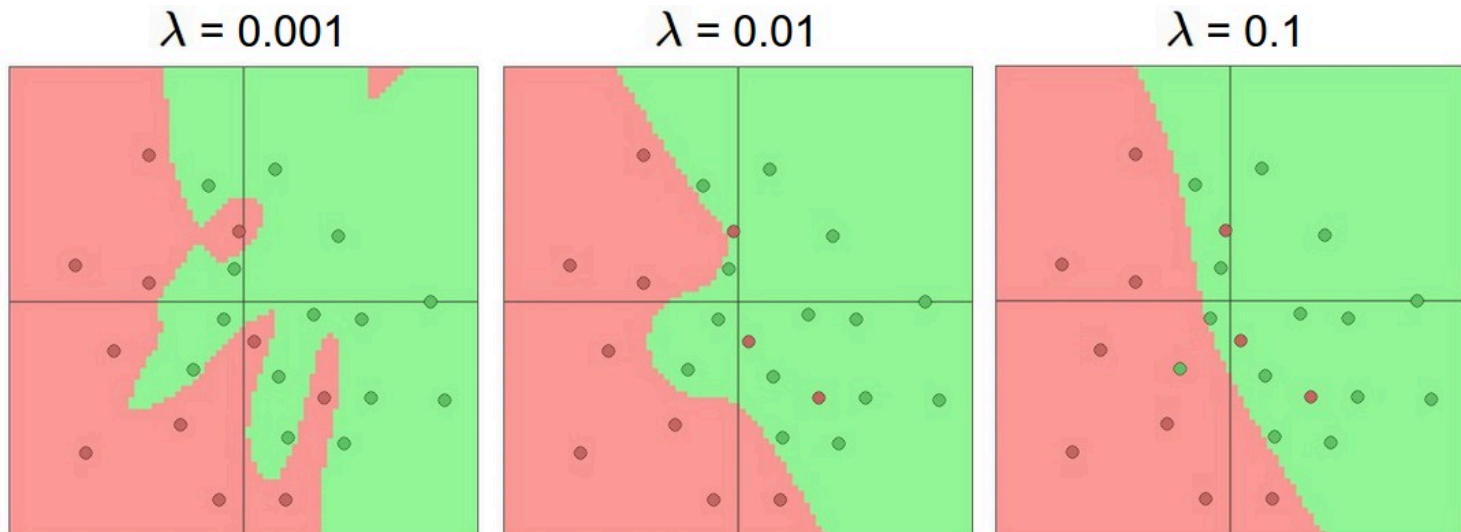


Regularization

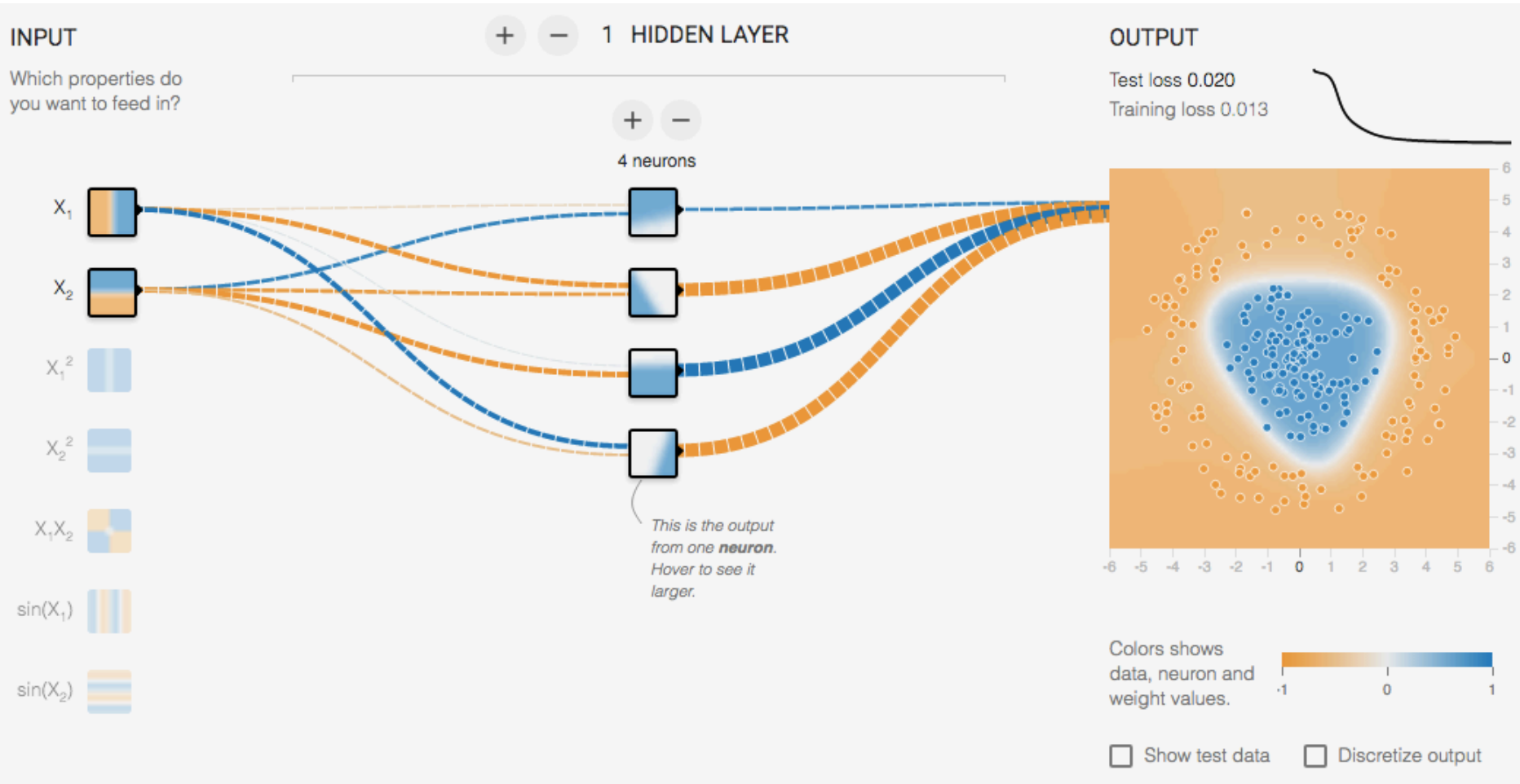
- It is common to add a penalty (e.g., quadratic) on weight magnitudes to the objective function:

$$E(\mathbf{w}) = \sum_i l(\mathbf{x}_i, y_i; \mathbf{w}) + \lambda \|\mathbf{w}\|^2$$

- Quadratic penalty encourages network to use all of its inputs “a little” rather than a few inputs “a lot”



Multi-Layer Network Demo



<http://playground.tensorflow.org/>

Dealing with multiple classes

- If we need to classify inputs into C different classes, we put C units in the last layer to produce C *one-vs.-others* scores f_1, f_2, \dots, f_C
- Apply *softmax* function to convert these scores to probabilities:

$$\text{softmax}(f_1, \dots, f_C) = \left(\frac{\exp(f_1)}{\sum_j \exp(f_j)}, \dots, \frac{\exp(f_C)}{\sum_j \exp(f_j)} \right)$$

- If one of the inputs is much larger than the others, then the corresponding softmax value will be close to 1 and others will be close to 0
- Use log likelihood (*cross-entropy*) loss:
$$l(\mathbf{x}_i, y_i; \mathbf{w}) = -\log P_{\mathbf{w}}(y_i | \mathbf{x}_i)$$

Neural networks: Pros and cons

- **Pros**

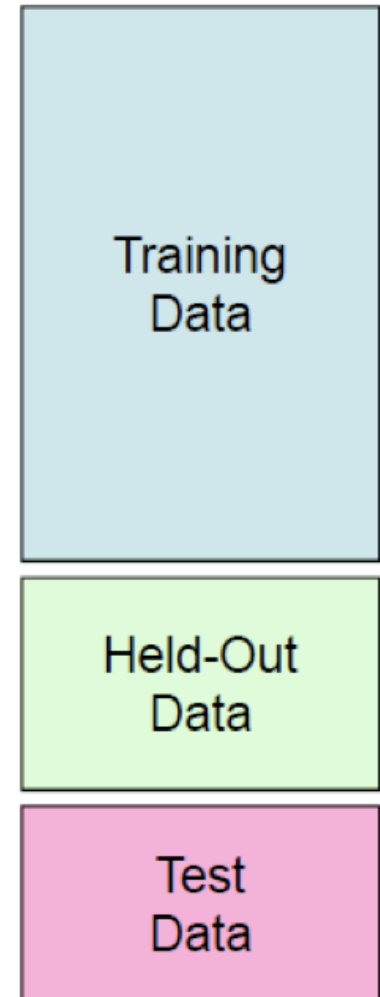
- Flexible and general function approximation framework
- Can build extremely powerful models by adding more layers

- **Cons**

- Hard to analyze theoretically (e.g., training is prone to local optima)
- Huge amount of training data, computing power may be required to get good performance
- The space of implementation choices is huge (network architectures, parameters)

Best practices for training classifiers

- Goal: obtain a classifier with **good generalization** or performance on never before seen data
1. Learn *parameters* on the **training set**
 2. Tune *hyperparameters* (implementation choices) on the *held out validation set*
 3. Evaluate performance on the **test set**
 - Crucial: do not peek at the test set when iterating steps 1 and 2!




What's the big deal?

Baidu admits cheating in international supercomputer competition



Baidu recently apologised for violating the rules of an international supercomputer test in May, when the Chinese search engine giant claimed to beat both Google and Microsoft on the ImageNet image-recognition test.

 By [Cyrus Lee](#) | June 10, 2015 -- 00:15 GMT (17:15 PDT) | Topic: [China](#)

TECHNOLOGY

The New York Times

Computer Scientists Are Astir After Baidu Team Is Barred From A.I. Competition

By [JOHN MARKOFF](#) JUNE 3, 2015



Baidu caught gaming recent supercomputer performance test

 by [Andrew Tarantola](#) | [@terrortola](#) | June 3rd 2015 At 11:09pm



IMAGENET Large Scale Visual Recognition Challenge (ILSVRC)

Date: June 2, 2015

Dear ILSVRC community,

This is a follow up to the announcement on [May 19, 2015](#) with some more details and the status of the test server.

During the period of November 28th, 2014 to May 13th, 2015, there were at least 30 accounts used by a team from Baidu to submit to the test server at least 200 times, far exceeding the specified limit of two submissions per week. This includes short periods of very high usage, for example with more than 40 submissions over 5 days from March 15th, 2015 to March 19th, 2015. Figure A below shows submissions from ImageNet accounts known to be associated with the team in question. Figure B shows a comparison to the activity from all other accounts.

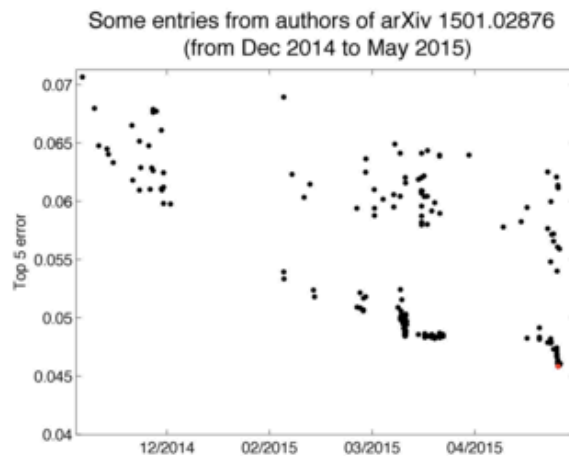


Figure A

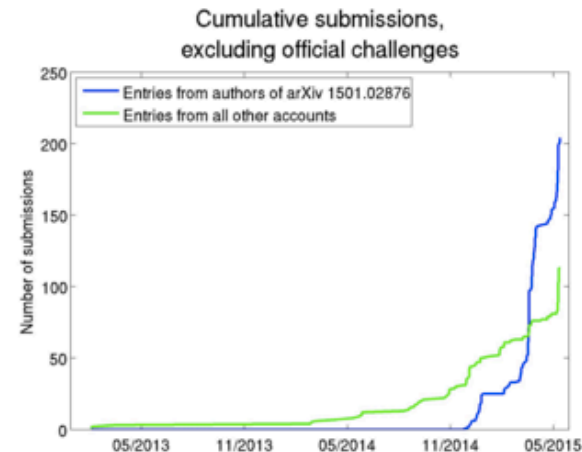


Figure B

The results obtained during this period are reported in a [recent arXiv paper](#). Because of the violation of the regulations of the test server, these results may not be directly comparable to results obtained and reported by other teams. To make this clear, by exploiting the ability to test many slightly different solutions on the test server it is possible to 1) select the best out of a set of very similar solutions based on test performance and achieve a small but potentially significant advantage and 2) choose methods for further research and development based directly on the test data instead of using only the training and validation data for such choices.

<http://www.image-net.org/challenges/LSVRC/announcement-June-2-2015>

Important Considerations

1. Inductive Bias
2. Data
3. Supervision
4. Loss functions
5. Optimization / Initialization

Development Process

1. Collect lots of labeled data
2. Setup network architecture
3. Setup loss function
4. Sanity checks
 1. Is your data correct?
 2. Can you overfit to a small set?
5. Hyperparameters
 1. Learning hyperparameters: batch size, learning rates, how much to train, regularization, optimizer.
 2. Architectural hyper-parameters: Non-linearities, #layers, #neurons, loss functions.
6. Hacking
 1. Reducing iteration time
 2. Maximizing GPU utilization

Optimizers

1. Stochastic gradient descent
2. Stochastic gradient descent with momentum
3. Adam