

Introduction to Deep Learning


A. G. Schwing & S. Fidler

University of Toronto, 2014

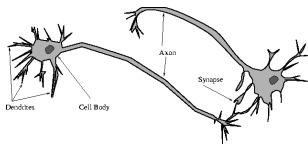
Outline

- 1 Universality of Neural Networks
- 2 Learning Neural Networks
- 3 Deep Learning
- 4 Applications
- 5 References

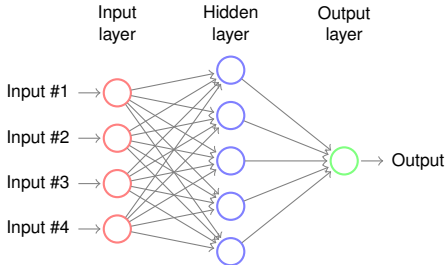
What are neural networks?

Let's ask 

- Biological



- Computational



What are neural networks?

...**Neural networks** (NNs) are computational models inspired by biological neural networks [...] and **are used to** estimate or **approximate functions...** [\[Wikipedia\]](#)

What are neural networks? The people behind

Origins:

- Traced back to threshold logic [W. McCulloch and W. Pitts, 1943]
- Perceptron [F. Rosenblatt, 1958]

More recently:

- G. Hinton (UofT)
- Y. LeCun (NYU)
- A. Ng (Stanford)
- Y. Bengio (University of Montreal)
- J. Schmidhuber (IDSIA, Switzerland)
- R. Salakhutdinov (UofT)
- H. Lee (University of Michigan)
- R. Fergus (NYU)
- ... (and many more having made significant contributions; apologies for not being able to mention everyone and all the students behind)

What are neural networks? Use cases

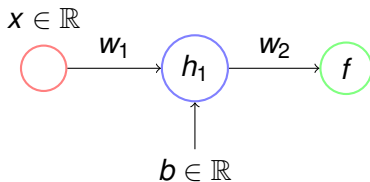
- Classification
- Playing video games
- Captcha
- Neural Turing Machine (e.g., learn how to sort) Alex Graves

<http://www.technologyreview.com/view/532156/googles-secretive-deepmind-startup-unveils-a-neural-turing-machine/>

What are neural networks?

Example:

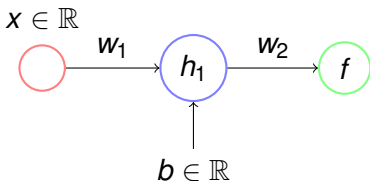
- input x
- parameters w_1, w_2, b



How to compute the function?

Forward propagation/pass, inference, prediction:

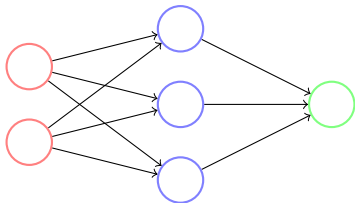
- Given input x and parameters w, b
- Compute latent variables/intermediate results in a feed-forward manner
- Until we obtain output function f



How to compute the function?

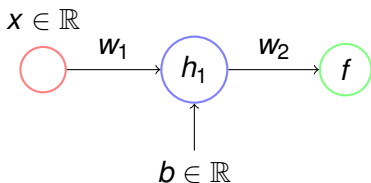
Forward propagation/pass, inference, prediction:

- Given input x and parameters w, b
- Compute latent variables/intermediate results in a feed-forward manner
- Until we obtain output function f



How to compute the function?

Example: input x , parameters w_1, w_2, b



$$h_1 = \sigma(w_1 \cdot x + b)$$
$$f = w_2 \cdot h_1$$

Sigmoid function:

$$\sigma(z) = 1/(1 + \exp(-z))$$

$x = \ln 2, b = \ln 3, w_1 = 2, w_2 = 2$

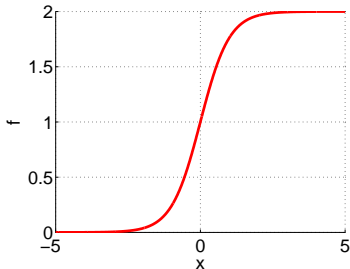
$h_1 = ?$

$f = ?$

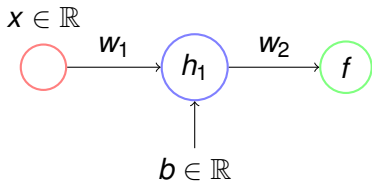
How to compute the function?

Given parameters, what is f for $x = 0$, $x = 1$, $x = 2$, ...

$$f = w_2 \sigma(w_1 \cdot x + b)$$

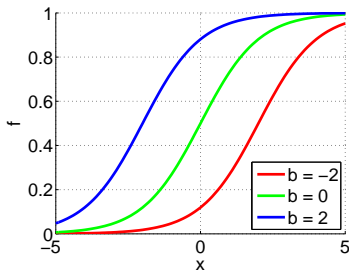


Let's mess with parameters:

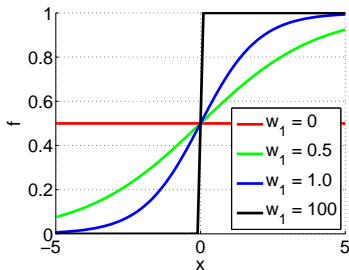


$$h_1 = \sigma(w_1 \cdot x + b)$$
$$f = w_2 \cdot h_1$$
$$\sigma(z) = 1 / (1 + \exp(-z))$$

$w_1 = 1.0$



$b = 0$

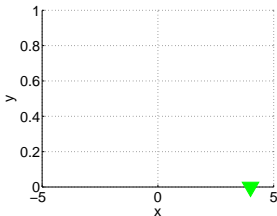


Keep in mind the step function.

How to use Neural Networks for binary classification?

Feature/Measurement: x

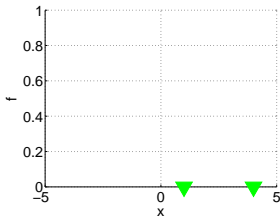
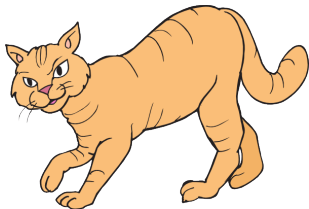
Output: How likely is the input to be a cat?



How to use Neural Networks for binary classification?

Feature/Measurement: x

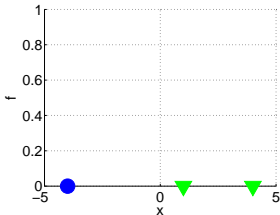
Output: How likely is the input to be a cat?



How to use Neural Networks for binary classification?

Feature/Measurement: x

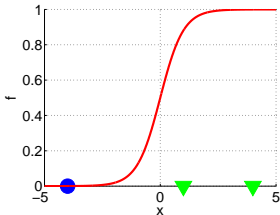
Output: How likely is the input to be a cat?



How to use Neural Networks for binary classification?

Feature/Measurement: x

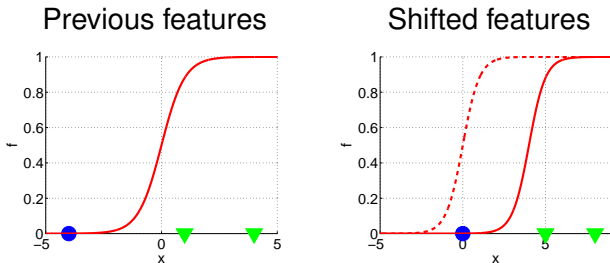
Output: How likely is the input to be a cat?



How to use Neural Networks for binary classification?

Shifted feature/measurement: x

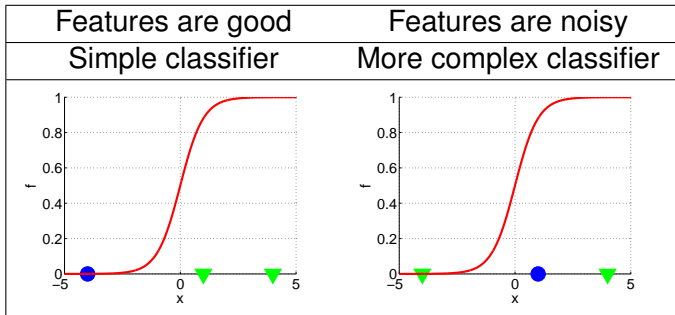
Output: How likely is the input to be a cat?



Learning/Training means finding the right parameters.

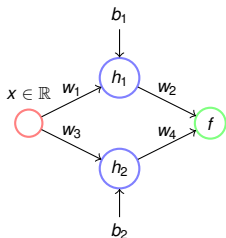
So far we are able to scale and translate sigmoids.

- How well can we approximate an arbitrary function?
- With the simple model we are obviously not going very far.



- How can we generalize?

Let's use more hidden variables:

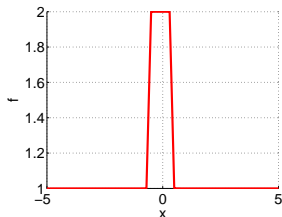


$$h_1 = \sigma(w_1 \cdot x + b_1)$$

$$h_2 = \sigma(w_3 \cdot x + b_2)$$

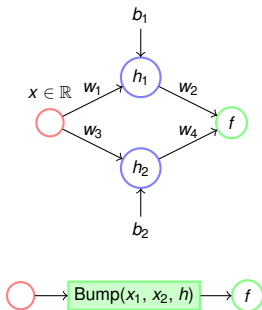
$$f = w_2 \cdot h_1 + w_4 \cdot h_2$$

Combining two step functions gives a bump.



$$w_1 = -100, b_1 = 40, w_3 = 100, b_2 = 60, w_2 = 1, w_4 = 1$$

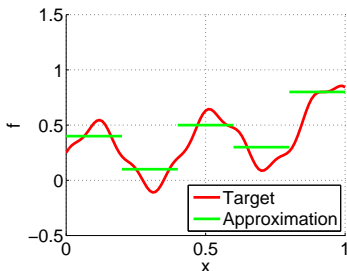
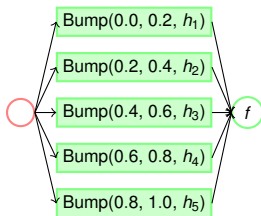
So let's simplify:



We simplify a pair of hidden nodes to a “bump” function:

- Starts at x_1
- Ends at x_2
- Has height h

Now we can represent “bumps” very well. How can we generalize?



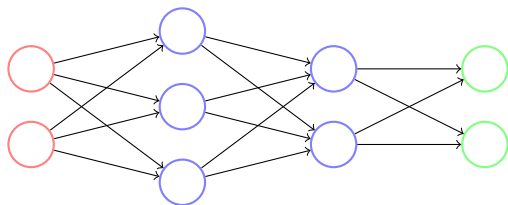
More bumps gives more accurate approximation.
Corresponds to a single layer network.

- Universality: theoretically we can approximate an arbitrary function
- So we can learn a really complex cat classifier
- Where is the catch?

- Universality: theoretically we can approximate an arbitrary function
- So we can learn a really complex cat classifier
- Where is the catch?
- Complexity, we might need quite a few hidden units
- Overfitting, memorize the training data

Generalizations are possible to

- include more input dimensions
- capture more output dimensions
- employ multiple layers for more efficient representations

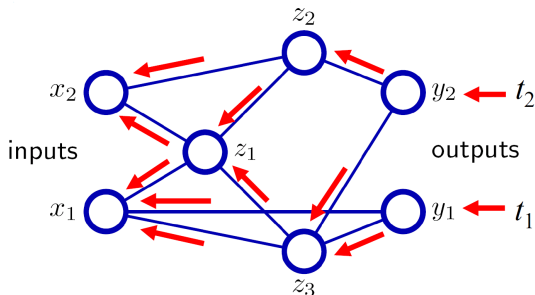


See '<http://neuralnetworksanddeeplearning.com/chap4.html>' for a great read!

How do we find the parameters to obtain a good approximation? How do we tell a computer to do that?

Intuitive explanation:

- Compute approximation error at the output
- Propagate error back by computing individual contributions of parameters to error



[Fig. from H. Lee]

Intuitive example:

- Target function: $5x^2$
- Approximation: $f_w(x)$
- Domain of interest: $x \in [0, 1]$
- Error:

$$e(w) = \int_0^1 (5x^2 - f_w(x))^2 dx$$

- Program of interest:

$$\min_w e(w) = \min_w \int_0^1 (5x^2 - f_w(x))^2 dx$$

Chain rule is important: $w_1, w_2, x \in \mathbb{R}$

Assume

$$e(w_1, w_2) = f(w_2, h(w_1, x))$$

Derivatives are:

$$\frac{\partial e(w_1, w_2)}{\partial w_2} = \frac{\partial f(w_2, h(w_1, x))}{\partial w_2}$$

$$\begin{aligned} \frac{\partial e(w_1, w_2)}{\partial w_1} &= \frac{\partial f(w_2, h(w_1, x))}{\partial w_1} \\ &= \frac{\partial f}{\partial h} \cdot \frac{\partial h}{\partial w_1} \quad \text{Chain rule} \end{aligned}$$

Back propagation does not work well for deep networks:

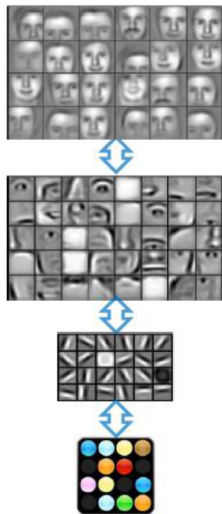
- Diffusion of gradient signal (multiplication of many small numbers)
- Attractivity of many local minima (random initialization is very far from good points)
- Requires a lot of training samples
- Need for significant computational power

Solution: 2 step approach

- Greedy layerwise pre-training
- Perform full fine tuning at the end

Why go deep?

- Representation efficiency (fewer computational units for the same function)
- Hierarchical representation (non-local generalization)
- Combinatorial sharing (re-use of earlier computation)
- Works very well

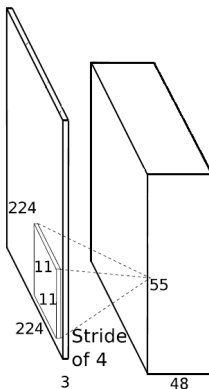


[Fig. from H. Lee]

To obtain more flexibility/non-linearity we use additional function prototypes:

- Sigmoid
- Rectified linear unit (ReLU)
- Pooling
- Dropout
- Convolutions

Convolutions

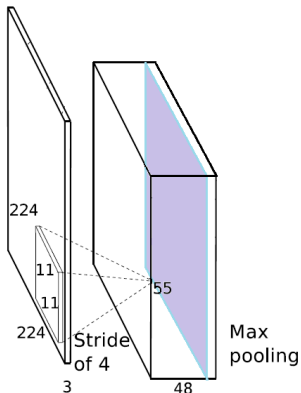


What do the numbers mean?

See Sanja's lecture 14 for the answers...

[Fig. adapted from A. Krizhevsky]

Max Pooling



What is happening here?

[Fig. adapted from A. Krizhevsky]

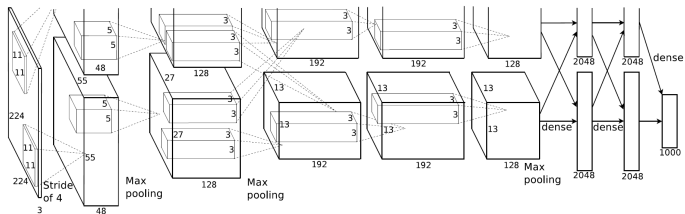
Rectified Linear Unit (ReLU)

- Drop information if smaller than zero
- Fixes the problem of vanishing gradients to some degree

Dropout

- Drop information at random
- Kind of a regularization, enforcing redundancy

A famous deep learning network called “AlexNet.”



- The network won the ImageNet competition in 2012.
- How many parameters?
- Given an image, what is happening?
- Inference Time: about 2ms per image when processing many images in parallel
- Training Time: forever (maybe 2-3 weeks)

[Fig. adapted from A. Krizhevsky]

Demo

Neural networks have been used for many applications:

- Classification and Recognition in Computer Vision
- Text Parsing in Natural Language Processing
- Playing Video Games
- Stock Market Prediction
- Captcha

Demos:

- Russ website
- Antonio Places website

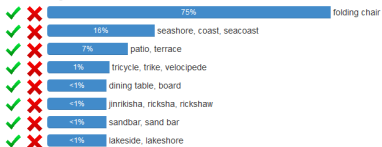
Classification in Computer Vision: ImageNet Challenge

<http://deeplearning.cs.toronto.edu/>

Since it's the end of the semester, let's find the beach...



Possible tags:



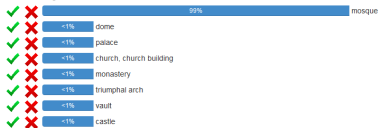
Classification in Computer Vision: ImageNet Challenge

<http://deeplearning.cs.toronto.edu/>

A place to maybe prepare for exams...



Possible tags:



Links:

- Tutorials: <http://deeplearning.net/tutorial/deeplearning.pdf>
- Toronto Demo by Russ and students:
<http://deeplearning.cs.toronto.edu/>
- MIT Demo by Antonio and students:
<http://places.csail.mit.edu/demo.html>
- Honglak Lee:
<http://deeplearningworkshopnips2010.files.wordpress.com/2010/09/r-workshop-tutorial-final.pdf>
- Yann LeCun:
<http://www.cs.nyu.edu/~yann/talks/lecun-ranzato-icml2013.pdf>
- Richard Socher: <http://lxmls.it.pt/2014/socher-lxmls.pdf>

Videos:

- Video games: <https://www.youtube.com/watch?v=mARt-xPabIE>
- Captcha: <http://singularityhub.com/2013/10/29/tiny-ai-startup-vicarious-says-its-solved-captcha/>
- <https://www.youtube.com/watch?v=lge-dl2JUAM#t=27>
- Stock exchange:
<http://cs.stanford.edu/people/eroberts/courses/soco/projects/neural-networks/Applications/stocks.html>