Introduction to Deep Learning

A. G. Schwing & S. Fidler

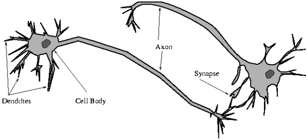
University of Toronto, 2015

Outline

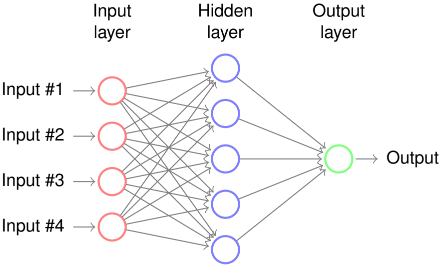
What are neural networks?

Let's ask Google

- 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?

Origins:

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

What are neural networks? Use cases

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

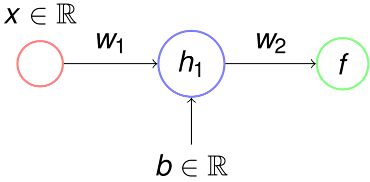
  http://www.technologyreview.com/view/532156/googles-secretive-deepmind-startup-unveils-a-neural-turing-machine/

What are neural networks?
Example:

- input x
- parameters w_1, w_2, b

What are neural networks?
Example:

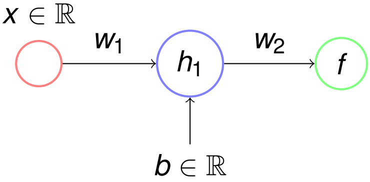
- input x
- parameters w_1, w_2, b



How to compute the function?

Forward propagation/pass, inference, prediction:

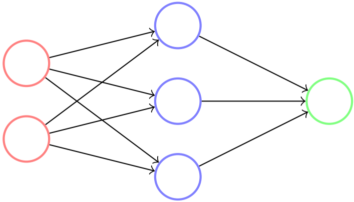
- Given input x and parameters w, b
- Compute (latent variables/) intermediate results in a feed-forward manner
- Until we obtain output function f



How to compute the function?

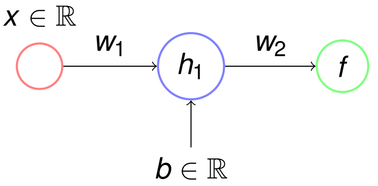
Forward propagation/pass, inference, prediction:

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



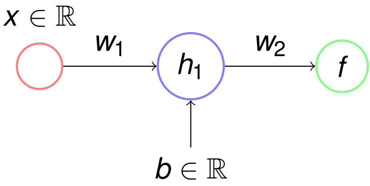
How to compute the function?

Example: input x, parameters w_1, w_2, b



How to compute the function?

Example: input x, parameters w_1, w_2, b



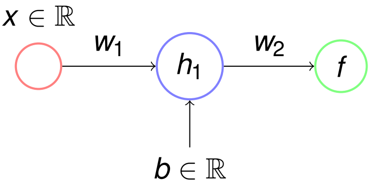
$$
\begin{aligned}
h_1 &= \sigma(w_1 \cdot x + b) \\
f &= w_2 \cdot h_1
\end{aligned}
$$

Sigmoid function:

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

How to compute the function?

Example: input x, parameters w_1, w_2, b



$$\begin{aligned} h_1 &= \sigma(w_1 \cdot x + b) \\ f &= w_2 \cdot h_1 \end{aligned}$$

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?

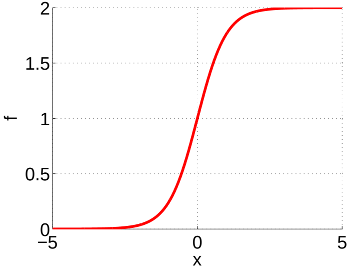
Given parameters, what is f for $x = 0$, $x = 1$, $x = 2$, ...

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

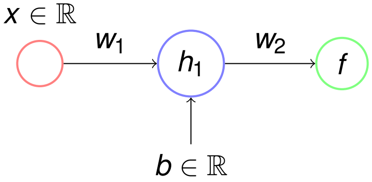
How to compute the function?

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

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

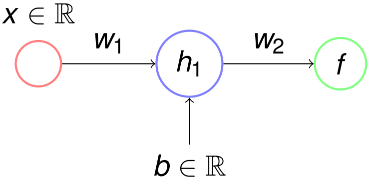


Let's mess with parameters:



$$\begin{aligned} h_1 &= \sigma(w_1 \cdot x + b) \\ f &= w_2 \cdot h_1 \\ \sigma(z) &= 1/(1 + \exp(-z)) \end{aligned}$$

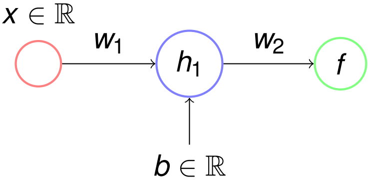
Let's mess with parameters:



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

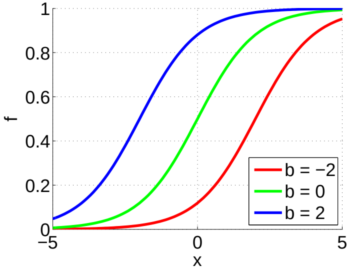
$w_1 = 1.0, b$ changes

Let's mess with parameters:

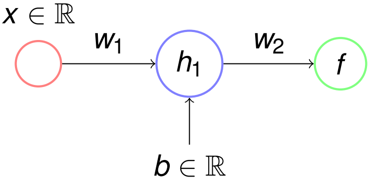


$$
\begin{aligned}
h_1 &= \sigma(w_1 \cdot x + b) \\
f &= w_2 \cdot h_1 \\
\sigma(z) &= 1/(1 + \exp(-z))
\end{aligned}
$$

$w_1 = 1.0$, b changes

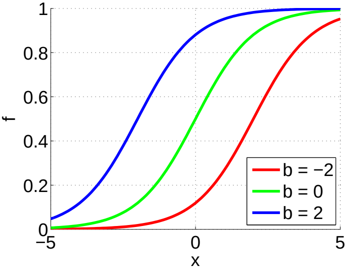


Let's mess with parameters:



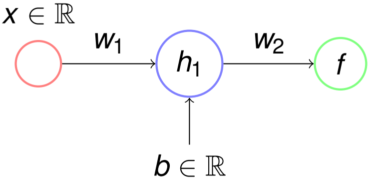
$$
\begin{aligned}
h_1 &= \sigma(w_1 \cdot x + b) \\
f &= w_2 \cdot h_1 \\
\sigma(z) &= 1/(1 + \exp(-z))
\end{aligned}
$$

$w_1 = 1.0$, b changes



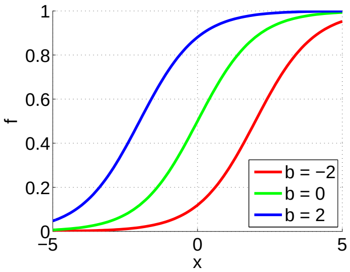
$b = 0$, w_1 changes

Let's mess with parameters:

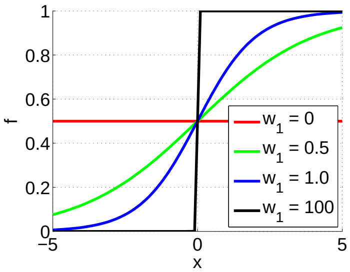


$$
\begin{aligned}
h_1 &= \sigma(w_1 \cdot x + b) \\
f &= w_2 \cdot h_1 \\
\sigma(z) &= 1/(1 + \exp(-z))
\end{aligned}
$$

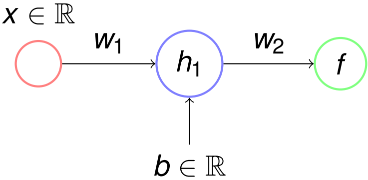
$w_1 = 1.0$, b changes



$b = 0$, w_1 changes

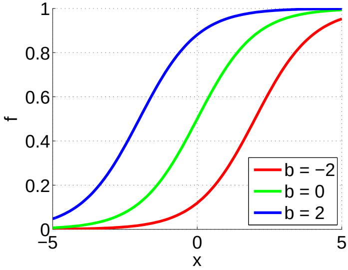


Let's mess with parameters:

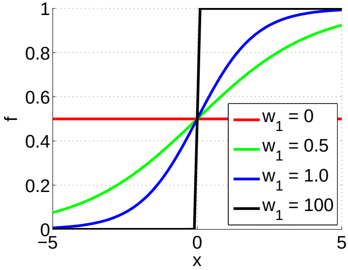


$$h_1 = \sigma(w_1 \cdot x + b)$$
$$f = w_2 \cdot h_1$$
$$\sigma(z) = 1/(1 + \exp(-z))$$

$w_1 = 1.0$, b changes



$b = 0$, w_1 changes

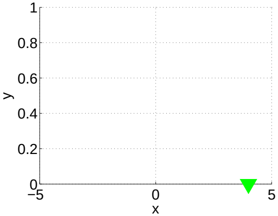


Keep in mind the step function.

How to use Neural Networks for binary classification?

Feature/Measurement: x

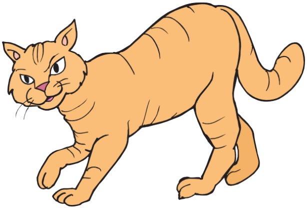
Output: How likely is the input to be a cat?

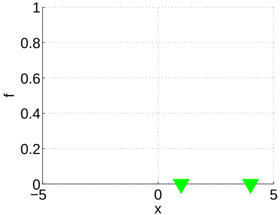


How to use Neural Networks for binary classification?

Feature/Measurement: x

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

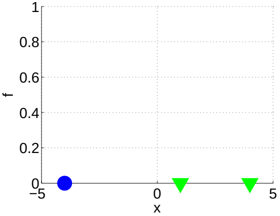




How to use Neural Networks for binary classification?

Feature/Measurement: x

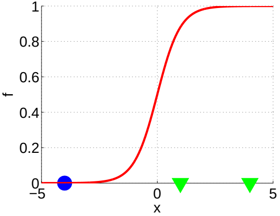
Output: How likely is the input to be a cat?



How to use Neural Networks for binary classification?

Feature/Measurement: x

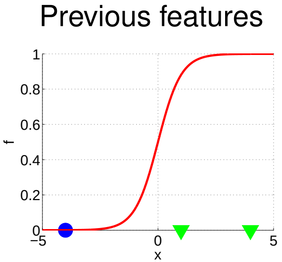
Output: How likely is the input to be a cat?



How to use Neural Networks for binary classification?

Shifted feature/measurement: x

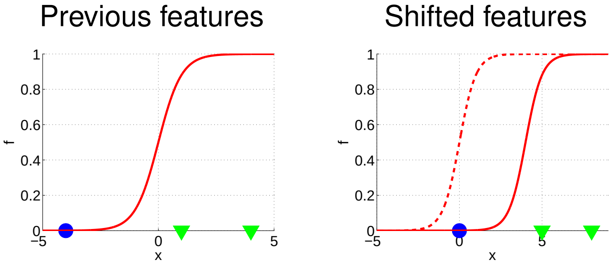
Output: How likely is the input to be a cat?



How to use Neural Networks for binary classification?

Shifted feature/measurement: x

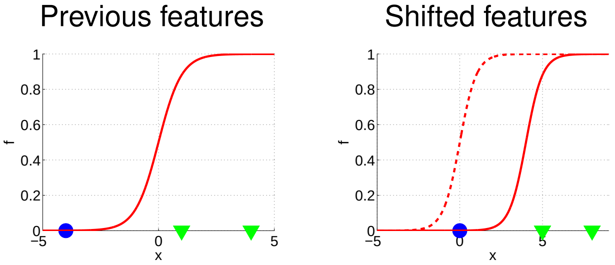
Output: How likely is the input to be a cat?



How to use Neural Networks for binary classification?

Shifted feature/measurement: x

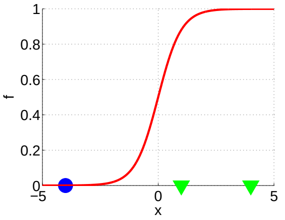
Output: How likely is the input to be a cat?



Learning/Training means finding the right parameters.

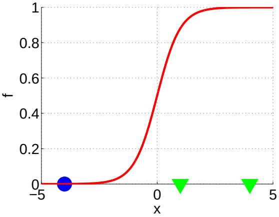
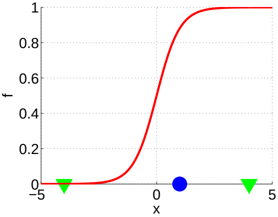
So far we are able to scale and translate sigmoids.

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

Features are good
Simple classifier


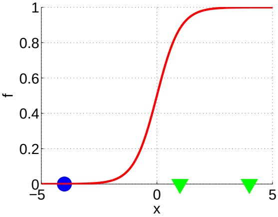
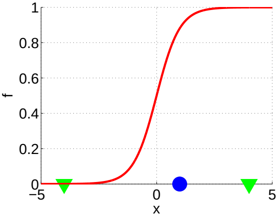
So far we are able to scale and translate sigmoids.

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

Features are good	Features are noisy
Simple classifier	More complex classifier
	

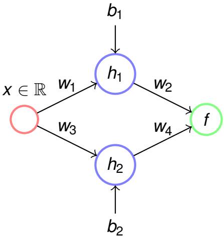
So far we are able to scale and translate sigmoids.

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

Features are good	Features are noisy
Simple classifier	More complex classifier
	

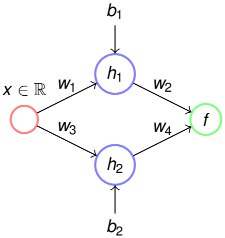
- How can we generalize?

Let's use more hidden variables:



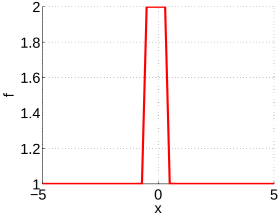
$$
\begin{aligned}
h_1 &= \sigma(w_1 \cdot x + b_1) \\
h_2 &= \sigma(w_3 \cdot x + b_2) \\
f &= w_2 \cdot h_1 + w_4 \cdot h_2
\end{aligned}
$$

Let's use more hidden variables:



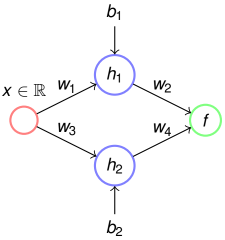
$$\begin{aligned} h_1 &= \sigma(w_1 \cdot x + b_1) \\ h_2 &= \sigma(w_3 \cdot x + b_2) \\ f &= w_2 \cdot h_1 + w_4 \cdot h_2 \end{aligned}$$

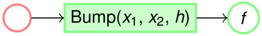
Combining two step functions gives a bump.



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

So let's simplify:



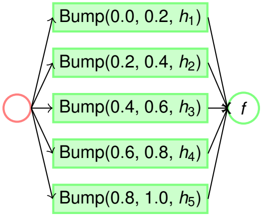


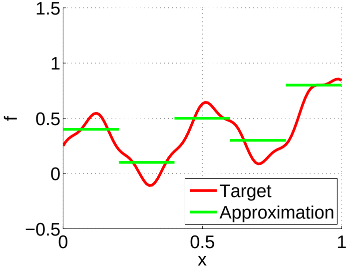
We simplify a pair of hidden nodes to a "bump" function:

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

Now we can represent "bumps" very well. How can we generalize?

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?

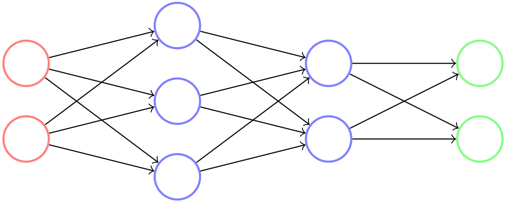
- Universality: theoretically we can approximate an arbitrary function
- So we can learn a really complex cat classifier
- Where is the catch?

- Complexity, we might need quite a few hidden units
- Overfitting, memorize the training data

Generalizations are possible to

Generalizations are possible to

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



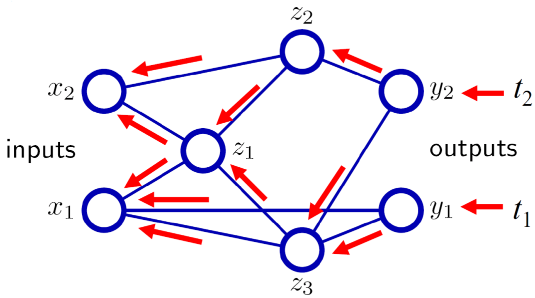
See ‘http://neuralnetworksanddeeplearning.com/chap4.html’ for a great read!

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

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]

Example for backpropagation of error:

- Target function: $5x^2$
- Approximation: $f(x, w)$
- Domain of interest: $x \in \{0, 1, 2, 3\}$
- Error:

$$e(w) = \sum_{x \in \{0,1,2,3\}} (5x^2 - f(x, w))^2$$

Example for backpropagation of error:

- Target function: $5x^2$
- Approximation: $f(x, w)$
- Domain of interest: $x \in \{0, 1, 2, 3\}$
- Error:

$$e(w) = \sum_{x \in \{0,1,2,3\}} (5x^2 - f(x, w))^2$$

- Program of interest:

$$\min_w e(w) = \min_w \sum_{x \in \{0,1,2,3\}} \underbrace{(5x^2 - f(x, w))^2}_{\ell(x,w)}$$

How to optimize?

Example for backpropagation of error:

- Target function: $5x^2$
- Approximation: $f(x, w)$
- Domain of interest: $x \in \{0, 1, 2, 3\}$
- Error:

$$e(w) = \sum_{x \in \{0,1,2,3\}} (5x^2 - f(x, w))^2$$

- Program of interest:

$$\min_w e(w) = \min_w \sum_{x \in \{0,1,2,3\}} \underbrace{(5x^2 - f(x, w))^2}_{\ell(x,w)}$$

How to optimize? **Gradient descent**

Gradient descent

$$\min_w e(w)$$

Gradient descent

$$\min_{w} e(w)$$

Algorithm: start with w_0, $t = 0$

1. Compute gradient $g_t = \frac{\partial e}{\partial w}\Big|_{w=w_t}$
2. Update $w_{t+1} = w_t - \eta g_t$
3. Set $t \leftarrow t + 1$

Chain rule is important to compute gradients:

$$\min_w e(w) = \min_w \sum_{x\in\{0,1,2,3\}} \underbrace{(5x^2 - f(x, w))^2}_{\ell(x,w)}$$

Chain rule is important to compute gradients:

$$\min_w e(w) = \min_w \sum_{x \in \{0,1,2,3\}} \underbrace{(5x^2 - f(x, w))^2}_{\ell(x,w)}$$

Loss function: $\ell(x, w)$

Chain rule is important to compute gradients:

$$\min_w e(w) = \min_w \sum_{x \in \{0,1,2,3\}} \underbrace{(5x^2 - f(x, w))^2}_{\ell(x,w)}$$

Loss function: $\ell(x, w)$

- Squared loss
- Log loss
- Hinge loss

Chain rule is important to compute gradients:

$$\min_w e(w) = \min_w \sum_{x \in \{0,1,2,3\}} \underbrace{(5x^2 - f(x,w))^2}_{\ell(x,w)}$$

Loss function: $\ell(x, w)$

- Squared loss
- Log loss
- Hinge loss

Derivatives:

$$\begin{aligned} \frac{\partial e(w)}{w} &= \sum_{x \in \{0,1,2,3\}} \frac{\partial \ell(x,w)}{\partial w} \\ &= \end{aligned}$$

Chain rule is important to compute gradients:

$$\min_w e(w) = \min_w \sum_{x\in\{0,1,2,3\}} \underbrace{(5x^2 - f(x,w))^2}_{\ell(x,w)}$$

Loss function: $\ell(x, w)$

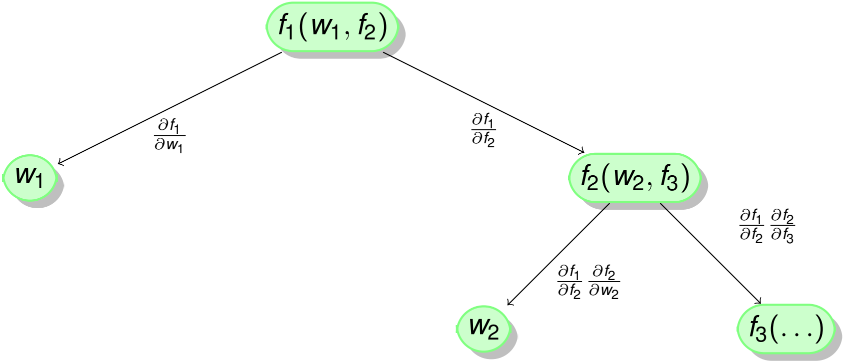
- Squared loss
- Log loss
- Hinge loss

Derivatives:

$$\begin{aligned} \frac{\partial e(w)}{w} &= \sum_{x\in\{0,1,2,3\}} \frac{\partial \ell(x,w)}{\partial w} \\ &= \sum_{x\in\{0,1,2,3\}} \frac{\partial \ell(x,w)}{\partial f} \frac{\partial f(x,w)}{\partial w} \end{aligned}$$

Slightly more complex example:
Composite function represented as a directed a-cyclic graph

$$\ell(x, w) = f_1(w_1, f_2(w_2, f_3(\ldots)))$$



Repeated application of chain rule for efficient computation of all gradients

Back propagation doesn't work well for deep sigmoid 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

Back propagation doesn't work well for deep sigmoid networks:

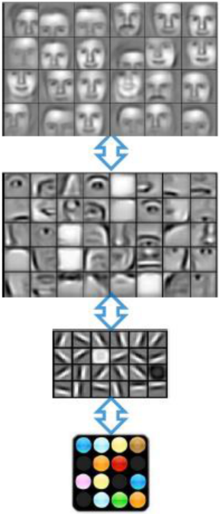
- Diffusion of gradient signal (multiplication of many small numbers)
- Attractivity of many local minima (random initialization is very far from good points)
- Requires a lot of training samples
- Need for significant computational power

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



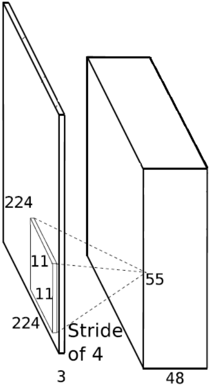
[Fig. from H. Lee]

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

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

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

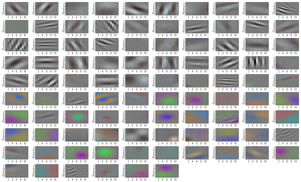
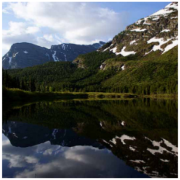
Convolutions

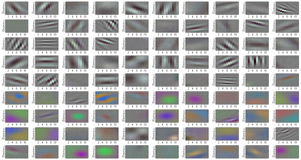
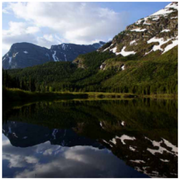


What do the numbers mean?

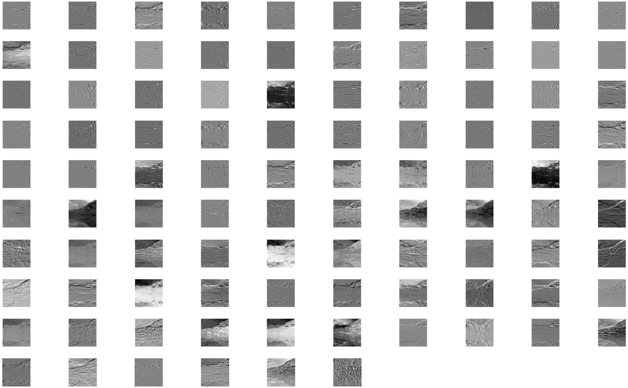
See Sanja's lecture 14 for the answers...

[Fig. adapted from A. Krizhevsky]

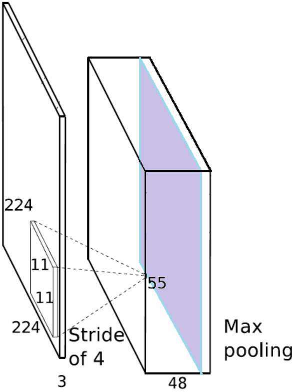
$f_{\text{conv}}($  ,  $)$

$f_{\text{conv}}($  $,$  $)$

$=$

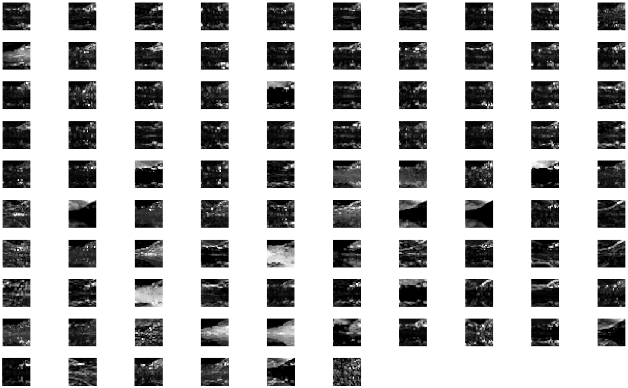


Max Pooling



What is happening here?

[Fig. adapted from A. Krizhevsky]



Rectified Linear Unit (ReLU)

Rectified Linear Unit (ReLU)

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

Rectified Linear Unit (ReLU)

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

Dropout

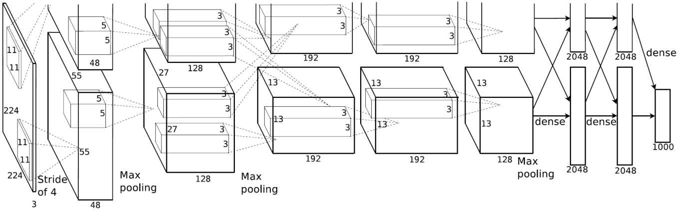
Rectified Linear Unit (ReLU)

- 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 on the GPU
- Training Time: a few days given a single recent GPU

[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

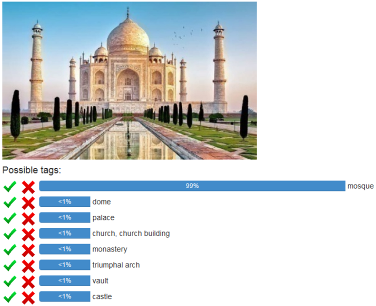
http://deeplearning.cs.toronto.edu/

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

Classification in Computer Vision: ImageNet Challenge

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

A place to maybe prepare for exams...



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-xPablE
- 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