CSC 2515: Introduction to Machine Learning

Lecture 6: Neural Networks

Amir-massoud Farahmand[1]

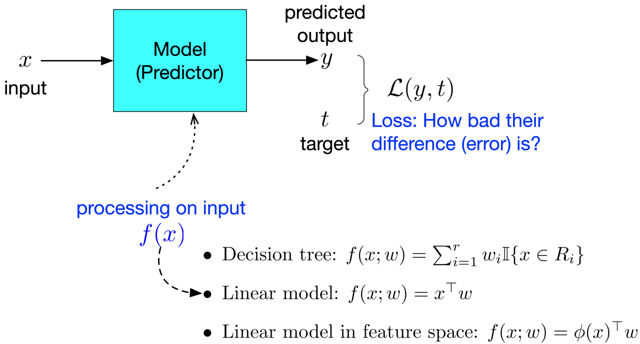
University of Toronto and Vector Institute

---

[1] Credit for slides goes to many members of the ML Group at the U of T, and beyond, including (recent past): Roger Grosse, Murat Erdogdu, Richard Zemel, Juan Felipe Carrasquilla, Emad Andrews, and myself.

Table of Contents

Today



x input

Model (Predictor)

predicted output y

t target

$\mathcal{L}(y,t)$

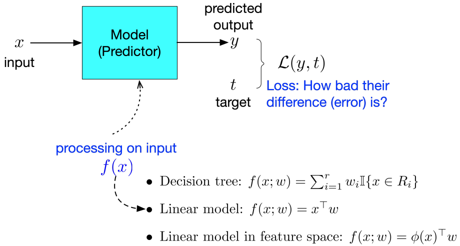
Loss: How bad their difference (error) is?

processing on input $f(x)$

- Decision tree: $f(x;w)=\sum_{i=1}^{r} w_i \mathbb{I}\{x \in R_i\}$
- Linear model: $f(x;w)=x^\top w$
- Linear model in feature space: $f(x;w)=\phi(x)^\top w$

- We have considered a modular framework to ML.
- We considered several loss functions for regression and classifications
- We have "mostly" focused on linear models.

Today



- Decision tree: $f(x;w) = \sum_{i=1}^{r} w_i \mathbb{I}\{x \in R_i\}$
- Linear model: $f(x;w) = x^\top w$
- Linear model in feature space: $f(x;w) = \phi(x)^\top w$

- Feature mapping can make linear models much more powerful.
- Coming up with feature mapping can be challenging.
- Kernel-based approach is a way to partially address it.
- (Artificial) Neural Networks is a general approach to represent more complex models.
- The predictor can be seen as a computer program that processes the input in order to generate the output. Some programs are simpler, some are more complex.

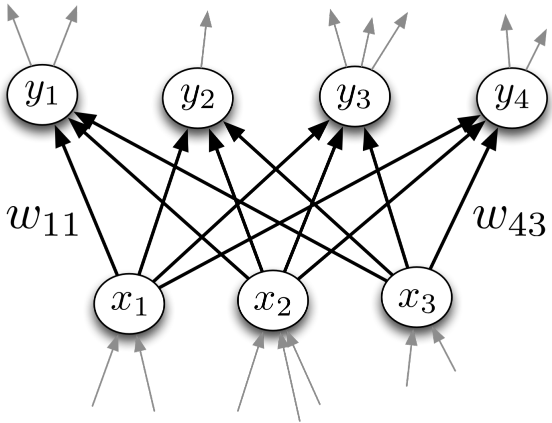
Today

Skills to Learn

- Multi-layer feedforward neural networks
- Backpropagation for training NN

Neural Networks

...



...

Inspiration: The Brain

- Our brain has $\sim 10^{11}$ neurons, each of which communicates (is connected) to $\sim 10^{4}$ other neurons

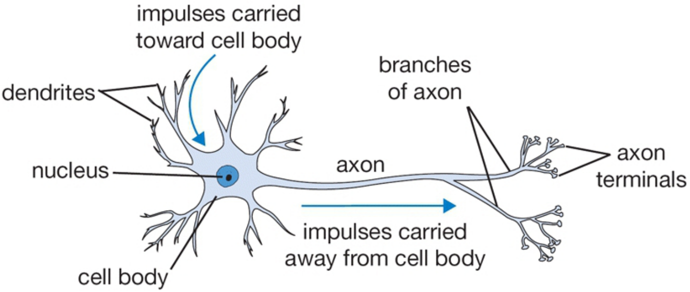
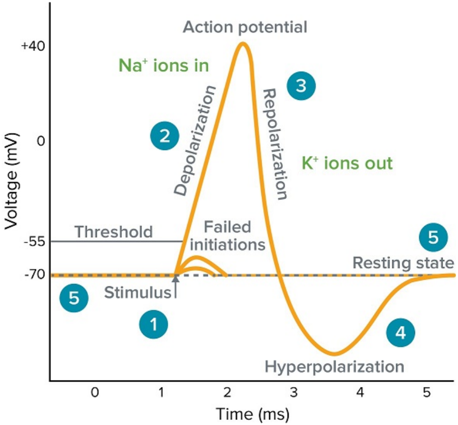


Figure: The basic computational unit of the brain: Neuron

[Pic credit: http://cs231n.github.io/neural-networks-1/]

Inspiration: The Brain

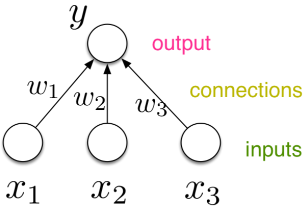
- Neurons receive input signals and accumulate voltage. After some threshold they will fire spiking responses.

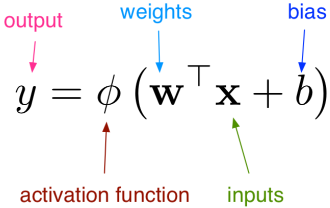


[Pic credit: www.moleculardevices.com]

Inspiration: The Brain

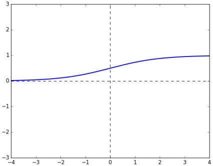
- For (artificial) neural nets, we use a much simpler model neuron, or unit:





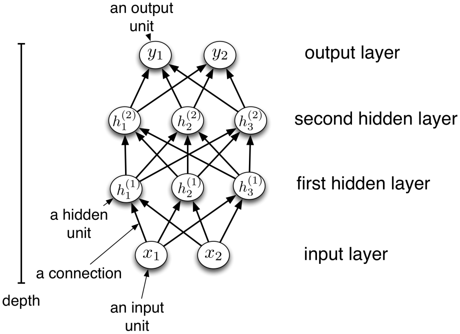
$$y = \phi\left(\mathbf{w}^\top \mathbf{x} + b\right)$$

- Compare with logistic activation function used in LR: $y = \sigma(\mathbf{w}^\top \mathbf{x} + b)$

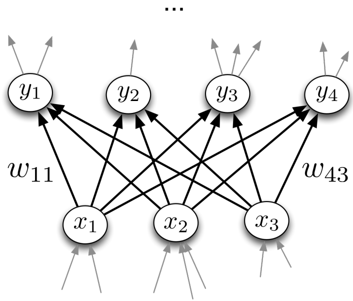


Multilayer Perceptrons (Feedforward Neural Networks)

- We can connect lots of units together into a directed acyclic graph.
- Typically, units are grouped together into layers.
- This gives a feed-forward neural network.
- That is in contrast to recurrent neural networks, which can have cycles.

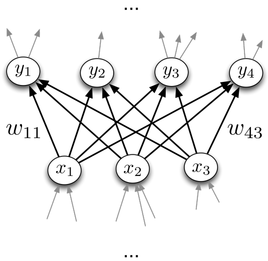


Multilayer Perceptrons (Feedforward Neural Networks)



- Each hidden layer i connects N_{i-1} input units to N_i output units.
- In the simplest case, all input units are connected to all output units. We call this a fully connected layer. We will consider other layer types later.
    - The inputs and outputs for a layer are distinct from the inputs and outputs to the network.

Multilayer Perceptrons (Feedforward Neural Networks)



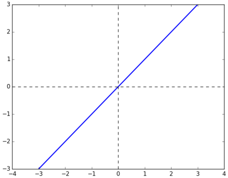
- If we need to compute $M[=N_i]$ outputs from $N=[N_{i-1}]$ inputs, we can do so in parallel using matrix multiplication. This means we will be using a $M \times N$ weight matrix.
- The output units are a function of the input units:

$$\mathbf{y} = f(\mathbf{x}) = \phi\left(\mathbf{W}\mathbf{x} + \mathbf{b}\right)$$

- A multilayer network consisting of fully connected layers is called a multilayer perceptron. Despite the name, it has nothing to do with the Perceptron algorithm.

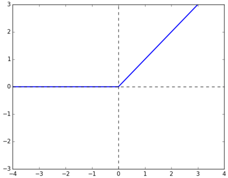
Activation Functions

Some activation functions:



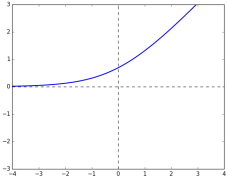
Identity

$$y = z$$



Rectified Linear Unit (ReLU)

$$y = \max(0, z)$$

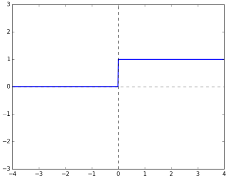


Soft ReLU

$$y = \log 1 + e^z$$

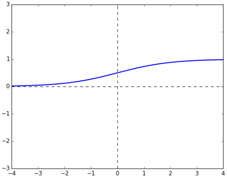
Activation Functions

Some activation functions:



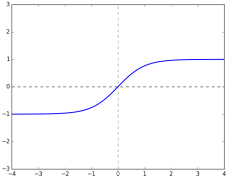
Hard Threshold

$$y = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{if } z \leq 0 \end{cases}$$



Logistic

$$y = \frac{1}{1 + e^{-z}}$$



Hyperbolic Tangent (tanh)

$$y = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

Multilayer Perceptrons (Feedforward Neural Networks)

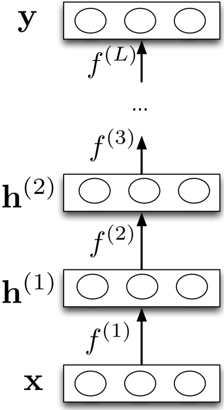
- Each layer computes a function, so the network computes a composition of functions:

$$\begin{aligned}
\mathbf{h}^{(1)} &= f^{(1)}(\mathbf{x}) = \phi(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}) \\
\mathbf{h}^{(2)} &= f^{(2)}(\mathbf{h}^{(1)}) = \phi(\mathbf{W}^{(2)}\mathbf{h}^{(1)} + \mathbf{b}^{(2)}) \\
&\vdots \\
\mathbf{y} &= f^{(L)}(\mathbf{h}^{(L-1)})
\end{aligned}$$

- Or more compactly:

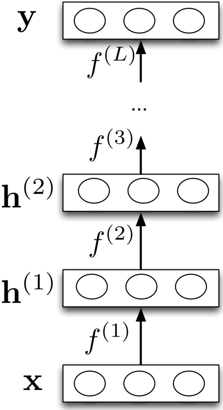
$$\mathbf{y} = f^{(L)} \circ \cdots \circ f^{(1)}(\mathbf{x}).$$

- Neural nets provide modularity: we can implement each layer's computations as a black box.



Multilayer Perceptrons (Feedforward Neural Networks)

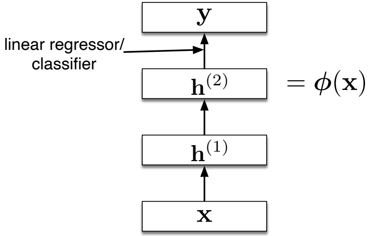
- Q: Write down the equations of a two layer NN (one hidden, one output), two hidden units, ϕ as the activation function of the hidden layer, and a linear one dimensional output layer.



Feature Learning

Last layer:

- If task is regression: choose
  $\mathbf{y} = f^{(L)}(\mathbf{h}^{(L-1)}) = (\mathbf{w}^{(L)})^T\mathbf{h}^{(L-1)} + b^{(L)}$
- If task is binary classification: choose
  $\mathbf{y} = f^{(L)}(\mathbf{h}^{(L-1)}) = \sigma((\mathbf{w}^{(L)})^T\mathbf{h}^{(L-1)} + b^{(L)})$
- Neural nets can be viewed as a way of learning features:



$\mathbf{y}$

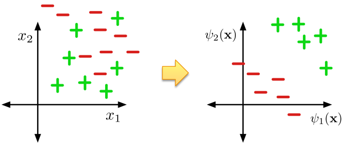
linear regressor/
classifier

$\mathbf{h}^{(2)} = \phi(\mathbf{x})$

$\mathbf{h}^{(1)}$

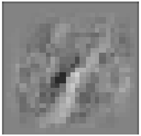
$\mathbf{x}$

- The goal:



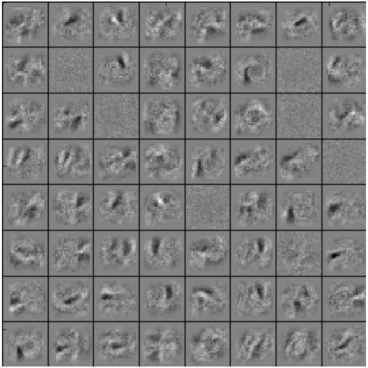
Feature Learning

- Suppose that we are trying to classify images of handwritten digits. Each image is represented as a vector of $28 \times 28 = 784$ pixel values.
- Each first-layer hidden unit computes $\phi(\mathbf{w}_i^T \mathbf{x})$. It acts as a feature detector.
- We can visualize $\mathbf{w}$ by reshaping it into an image. Here is an example that responds to a diagonal stroke.



Feature Learning

Here are some of the features learned by the first hidden layer of a handwritten digit classifier:



Expressive Power

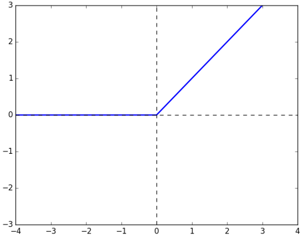
- We have seen that there are some functions that linear classifiers cannot represent. Are deep networks any better?
- Suppose a layer's activation function is the identity function, so the layer just computes an affine transformation of the input
  - We call this a linear layer
- Any sequence of *linear* layers can be equivalently represented with a single linear layer.

$$\mathbf{y} = \underbrace{\mathbf{W}^{(3)}\mathbf{W}^{(2)}\mathbf{W}^{(1)}}_{\triangleq \mathbf{W}'}\mathbf{x}$$

  - Deep linear networks are no more expressive than linear models.

Expressive Power

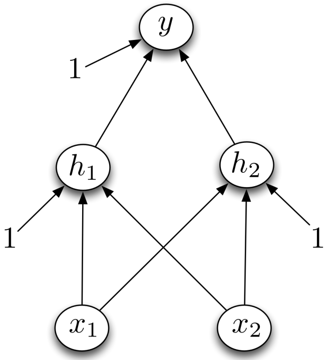
- Multilayer feed-forward neural nets with *nonlinear* activation functions are universal function approximators: they can approximate any function arbitrarily well.
- This has been shown for various activation functions (thresholds, logistic, ReLU, etc.)
  - Even though ReLU is "almost" linear, it is nonlinear enough.



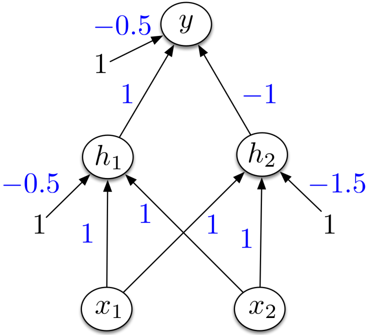
Multilayer Perceptrons

Designing a network to classify XOR:

Assume hard threshold activation function



Multilayer Perceptrons



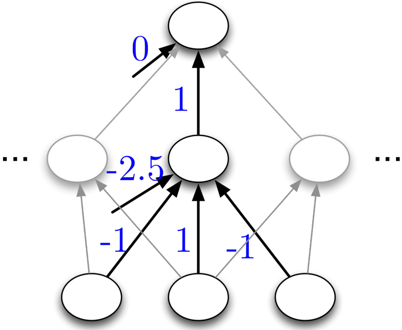
- h_1 computes $\mathbb{I}[x_1 + x_2 - 0.5 > 0]$
    - i.e. x_1 OR x_2
- h_2 computes $\mathbb{I}[x_1 + x_2 - 1.5 > 0]$
    - i.e. x_1 AND x_2
- y computes $\mathbb{I}[h_1 - h_2 - 0.5 > 0] \equiv \mathbb{I}[h_1 + (1 - h_2) - 1.5 > 0]$
    - i.e. h_1 AND (NOT h_2) $= x_1$ XOR x_2

Expressive Power

Universality for binary inputs and targets:

- Hard threshold hidden units, linear output
- Strategy: 2^D hidden units, each of which responds to one particular input configuration

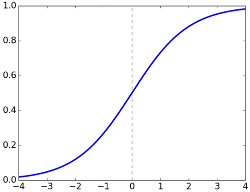
x_1	x_2	x_3	t
	⋮		⋮
-1	-1	1	-1
-1	1	-1	1
-1	1	1	1
	⋮		⋮



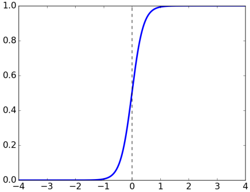
- Only requires one hidden layer, though it needs to be extremely wide.

Expressive Power

- What about the logistic activation function?
- You can approximate a hard threshold by scaling up the weights and biases:



$$y = \sigma(x)$$



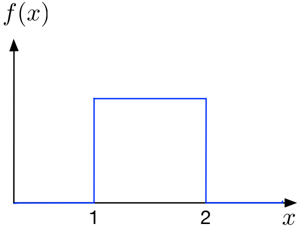
$$y = \sigma(5x)$$

- This is good: logistic units are differentiable, so we can train them with gradient descent.

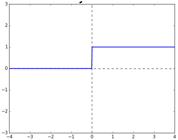
Expressive Power

Let us do some exercises ...

- Q: How can we represent the function that takes value of +1 in $x \in [1, 2]$ and 0 elsewhere using a simple NN with hard threshold activation function?



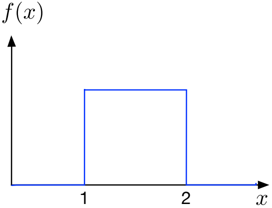
$$f(x) = w_1\phi(x - b_1) + w_2\phi(x - b_2)$$



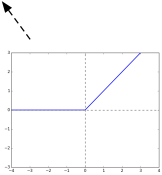
Expressive Power

Let us do some exercises ...

- Q: How can we approximately represent the function that takes value of +1 in $x \in [1, 2]$ and 0 elsewhere using a simple NN with ReLU activation function?



$$f(x) \approx w_1 \phi(v_1(x - b_1)) + w_2 \phi(v_2(x - b_2)) + ...$$



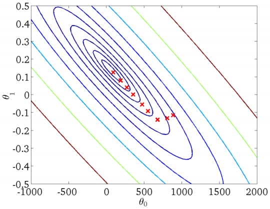
Expressive Power

- Limits of universality
    - You may need to represent an exponentially large network.
    - How can you find the appropriate weights to represent a given function?
    - If you can learn any function, you'll just overfit.
    - We desire a *compact* representation.

Training Neural Networks with Backpropagation

Recap: Gradient Descent

- **Recall:** gradient descent moves in the opposite of the gradient



- Weight space for a multilayer neural net: one coordinate for each weight or bias of the network, in *all* the layers
- Conceptually, not any different from what we have seen so far — just higher dimensional and harder to visualize!
- We want to define a loss $\mathcal{L}$ and compute the gradient of the cost $\mathrm{d}\mathcal{J}/\mathrm{d}\mathbf{w}$, which is the vector of partial derivatives.
  - This is the average of $\mathrm{d}\mathcal{L}/\mathrm{d}\mathbf{w}$ over all the training examples, so in this lecture we focus on computing $\mathrm{d}\mathcal{L}/\mathrm{d}\mathbf{w}$.

Univariate Chain Rule

- We have already been using the univariate Chain Rule.
- Recall: if $f(x)$ and $x(t)$ are univariate functions, then

$$\frac{\mathrm{d}}{\mathrm{d}t} f(x(t)) = \frac{\mathrm{d}f}{\mathrm{d}x} \frac{\mathrm{d}x}{\mathrm{d}t}.$$

Univariate Chain Rule

Recall: Univariate logistic least squares model

$$\begin{aligned} z &= wx + b \\ y &= \sigma(z) \\ \mathcal{L} &= \frac{1}{2}(y - t)^2 \end{aligned}$$

Let's compute the loss derivatives $\frac{\partial \mathcal{L}}{\partial w}, \frac{\partial \mathcal{L}}{\partial b}$.

Univariate Chain Rule

How you would have done it in calculus class:

$$\begin{aligned}
\mathcal{L} &= \frac{1}{2}(\sigma(wx+b)-t)^2 \\
\frac{\partial \mathcal{L}}{\partial w} &= \frac{\partial}{\partial w}\left[\frac{1}{2}(\sigma(wx+b)-t)^2\right] \\
&= \frac{1}{2}\frac{\partial}{\partial w}(\sigma(wx+b)-t)^2 \\
&= (\sigma(wx+b)-t)\frac{\partial}{\partial w}(\sigma(wx+b)-t) \\
&= (\sigma(wx+b)-t)\sigma'(wx+b)\frac{\partial}{\partial w}(wx+b) \\
&= (\sigma(wx+b)-t)\sigma'(wx+b)x
\end{aligned}$$

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial b} &= \frac{\partial}{\partial b}\left[\frac{1}{2}(\sigma(wx+b)-t)^2\right] \\
&= ? \qquad \text{(Exercise!)}
\end{aligned}$$

What are the disadvantages of this approach?

Univariate Chain Rule

A more structured way to do it:

Computing the loss:

$$\begin{aligned} z &= wx + b \\ y &= \sigma(z) \\ \mathcal{L} &= \frac{1}{2}(y-t)^2 \end{aligned}$$

Computing the derivatives:

$$\begin{aligned} \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}y} &= y - t \\ \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}z} &= \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}y}\frac{\mathrm{d}y}{\mathrm{d}z} = \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}y}\,\sigma'(z) \\ \frac{\partial\mathcal{L}}{\partial w} &= \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}z}\frac{\mathrm{d}z}{\mathrm{d}w} = \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}z}\,x \\ \frac{\partial\mathcal{L}}{\partial b} &= \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}z}\frac{\mathrm{d}z}{\mathrm{d}b} = \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}z} \end{aligned}$$

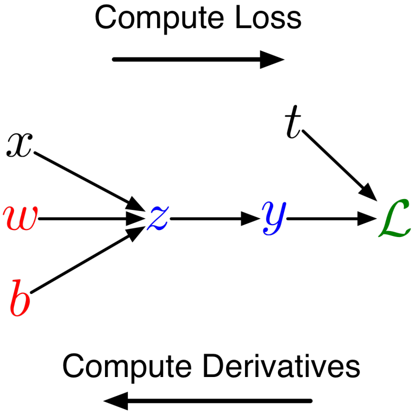
Remember: The goal is not to obtain closed-form solutions, but to be able to write a program that efficiently computes the derivatives.

Univariate Chain Rule

- We can diagram out the computations using a computation graph.
- The nodes represent all the inputs and computed quantities, and the edges represent which nodes are computed directly as a function of which other nodes.

Computing the loss:

$$\begin{aligned} z &= wx + b \\ y &= \sigma(z) \\ \mathcal{L} &= \frac{1}{2}(y - t)^2 \end{aligned}$$



Univariate Chain Rule

A slightly more convenient notation:

- Use $\overline{y}$ to denote the derivative of the loss w.r.t. y (i.e., $\mathrm{d}\mathcal{L}/\mathrm{d}y$), sometimes called the error signal.
- This emphasizes that the error signals are just values our program is computing (rather than a mathematical operation).

Computing the loss:

$$\begin{aligned} z &= wx + b \\ y &= \sigma(z) \\ \mathcal{L} &= \frac{1}{2}(y - t)^2 \end{aligned}$$

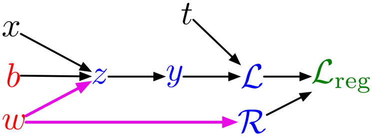
Computing the derivatives:

$$\begin{aligned} \overline{y} &= y - t \\ \overline{z} &= \overline{y}\,\sigma'(z) \\ \overline{w} &= \overline{z}\,x \\ \overline{b} &= \overline{z} \end{aligned}$$

Multivariate Chain Rule

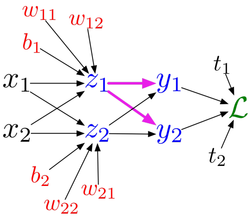
Problem: what if the computation graph has fan-out > 1?
This requires the Multivariate Chain Rule!

L_2-Regularized regression



$$z = wx + b$$
$$y = \sigma(z)$$
$$\mathcal{L} = \frac{1}{2}(y - t)^2$$
$$\mathcal{R} = \frac{1}{2}w^2$$
$$\mathcal{L}_{\mathrm{reg}} = \mathcal{L} + \lambda\mathcal{R}$$

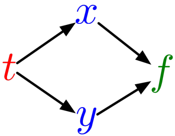
Softmax classifier with the cross-entropy loss



$$z_\ell = \sum_j w_{\ell j} x_j + b_\ell$$
$$y_k = \frac{e^{z_k}}{\sum_\ell e^{z_\ell}}$$
$$\mathcal{L} = -\sum_k t_k \log y_k$$

Multivariate Chain Rule

- Suppose that we have a function $f(x, y)$ and functions $x(t)$ and $y(t)$. (All the variables here are scalar-valued). Then



$$\frac{\mathrm{d}}{\mathrm{d}t} f(x(t), y(t)) = \frac{\partial f}{\partial x}\frac{\mathrm{d}x}{\mathrm{d}t} + \frac{\partial f}{\partial y}\frac{\mathrm{d}y}{\mathrm{d}t}$$

- Example:

$$\begin{aligned} f(x, y) &= y + e^{xy} \\ x(t) &= \cos t \\ y(t) &= t^2 \end{aligned}$$

- Plug in to Chain Rule:

$$\begin{aligned} \frac{\mathrm{d}f}{\mathrm{d}t} &= \frac{\partial f}{\partial x}\frac{\mathrm{d}x}{\mathrm{d}t} + \frac{\partial f}{\partial y}\frac{\mathrm{d}y}{\mathrm{d}t} \\ &= (ye^{xy}) \cdot (-\sin t) + (1 + xe^{xy}) \cdot 2t \end{aligned}$$

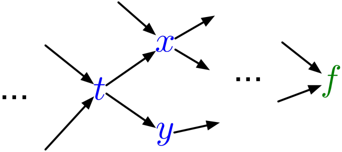
Multivariate Chain Rule

- In the context of backpropagation:

Mathematical expressions to be evaluated

$$\frac{\mathrm{d}f}{\mathrm{d}t} = \frac{\partial f}{\partial x}\frac{\mathrm{d}x}{\mathrm{d}t} + \frac{\partial f}{\partial y}\frac{\mathrm{d}y}{\mathrm{d}t}$$

Values already computed by our program



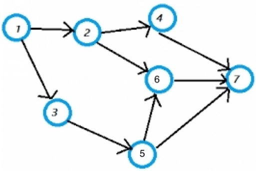
- In our notation:

$$\bar{t} = \overline{x}\,\frac{\mathrm{d}x}{\mathrm{d}t} + \overline{y}\,\frac{\mathrm{d}y}{\mathrm{d}t}$$

Backpropagation

Full backpropagation algorithm:

Let $v_1, \dots, v_N$ be a topological ordering of the computation graph (i.e. parents come before children.)

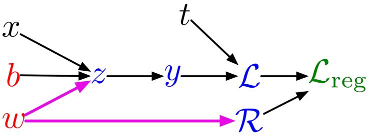


v_N denotes the variable we're trying to compute derivatives of (e.g. loss).

```
forward pass
    For $i = 1, \dots, N$
        Compute $v_i$ as a function of $\mathrm{Pa}(v_i)$
backward pass
    $\overline{v_N} = 1$
    For $i = N - 1, \dots, 1$
        $\overline{v_i} = \sum_{j \in \mathrm{Ch}(v_i)} \overline{v_j} \frac{\partial v_j}{\partial v_i}$
```

Backpropagation

Example: univariate logistic least squares regression



Forward pass:

$$\begin{aligned} z &= wx + b \\ y &= \sigma(z) \\ \mathcal{L} &= \frac{1}{2}(y-t)^2 \\ \mathcal{R} &= \frac{1}{2}w^2 \\ \mathcal{L}_{\text{reg}} &= \mathcal{L} + \lambda\mathcal{R} \end{aligned}$$

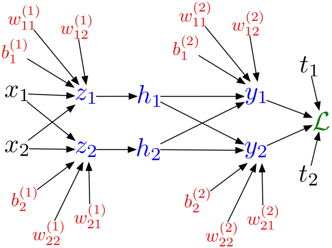
Backward pass:

$$\begin{aligned} \overline{\mathcal{L}_{\text{reg}}} &= 1 \\ \overline{\mathcal{R}} &= \overline{\mathcal{L}_{\text{reg}}}\,\frac{\mathrm{d}\mathcal{L}_{\text{reg}}}{\mathrm{d}\mathcal{R}} \\ &= \overline{\mathcal{L}_{\text{reg}}}\,\lambda \\ \overline{\mathcal{L}} &= \overline{\mathcal{L}_{\text{reg}}}\,\frac{\mathrm{d}\mathcal{L}_{\text{reg}}}{\mathrm{d}\mathcal{L}} \\ &= \overline{\mathcal{L}_{\text{reg}}} \\ \overline{y} &= \overline{\mathcal{L}}\,\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}y} \\ &= \overline{\mathcal{L}}\,(y-t) \end{aligned}$$

$$\begin{aligned} \overline{z} &= \overline{y}\,\frac{\mathrm{d}y}{\mathrm{d}z} \\ &= \overline{y}\,\sigma'(z) \\ \overline{w} &= \overline{z}\,\frac{\partial z}{\partial w} + \overline{\mathcal{R}}\frac{\mathrm{d}\mathcal{R}}{\mathrm{d}w} \\ &= \overline{z}\,x + \overline{\mathcal{R}}\,w \\ \overline{b} &= \overline{z}\,\frac{\partial z}{\partial b} \\ &= \overline{z} \end{aligned}$$

Backpropagation

Multilayer Perceptron (multiple outputs):



Forward pass:

$$z_i = \sum_j w_{ij}^{(1)} x_j + b_i^{(1)}$$

$$h_i = \sigma(z_i)$$

$$y_k = \sum_i w_{ki}^{(2)} h_i + b_k^{(2)}$$

$$\mathcal{L} = \frac{1}{2} \sum_k (y_k - t_k)^2$$

Backward pass:

$$\overline{\mathcal{L}} = 1$$

$$\overline{y_k} = \overline{\mathcal{L}}\,(y_k - t_k)$$

$$\overline{w_{ki}^{(2)}} = \overline{y_k}\,h_i$$

$$\overline{b_k^{(2)}} = \overline{y_k}$$

$$\overline{h_i} = \sum_k \overline{y_k} w_{ki}^{(2)}$$

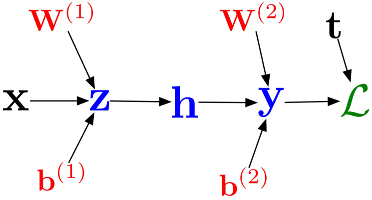
$$\overline{z_i} = \overline{h_i}\,\sigma'(z_i)$$

$$\overline{w_{ij}^{(1)}} = \overline{z_i}\,x_j$$

$$\overline{b_i^{(1)}} = \overline{z_i}$$

Backpropagation

In vectorized form:



Forward pass:

$$\begin{aligned}
\mathbf{z} &= \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)} \\
\mathbf{h} &= \sigma(\mathbf{z}) \\
\mathbf{y} &= \mathbf{W}^{(2)}\mathbf{h} + \mathbf{b}^{(2)} \\
\mathcal{L} &= \frac{1}{2}\|\mathbf{y} - \mathbf{t}\|^2
\end{aligned}$$

Backward pass:

$$\begin{aligned}
\overline{\mathcal{L}} &= 1 \\
\overline{\mathbf{y}} &= \overline{\mathcal{L}}\,(\mathbf{y} - \mathbf{t}) \\
\overline{\mathbf{W}^{(2)}} &= \overline{\mathbf{y}}\mathbf{h}^\top \\
\overline{\mathbf{b}^{(2)}} &= \overline{\mathbf{y}} \\
\overline{\mathbf{h}} &= \mathbf{W}^{(2)\top}\overline{\mathbf{y}} \\
\overline{\mathbf{z}} &= \overline{\mathbf{h}} \circ \sigma'(\mathbf{z}) \\
\overline{\mathbf{W}^{(1)}} &= \overline{\mathbf{z}}\mathbf{x}^\top \\
\overline{\mathbf{b}^{(1)}} &= \overline{\mathbf{z}}
\end{aligned}$$

Computational Cost

- Computational cost of forward pass: one add-multiply operation per weight

$$z_i = \sum_j w_{ij}^{(1)} x_j + b_i^{(1)}$$

- Computational cost of backward pass: two add-multiply operations per weight

$$\begin{aligned} \overline{w_{ki}^{(2)}} &= \overline{y_k}\, h_i \\ \overline{h_i} &= \sum_k \overline{y_k} w_{ki}^{(2)} \end{aligned}$$

- Rule of thumb: the backward pass is about as expensive as two forward passes.
- For a multilayer perceptron, this means the cost is linear in the number of layers, quadratic in the number of units per layer.

Backpropagation

- Backprop is used to train the overwhelming majority of neural nets today.
    - Even optimization algorithms much fancier than gradient descent (e.g. second-order methods) use backprop to compute the gradients.
- Despite its practical success, backprop is believed to be neurally implausible.

Conclusion

- Multi-layer feedforward NN addressed the feature learning problem
- Backpropagation as a method to learn NN

Convolutional Networks (Optional)

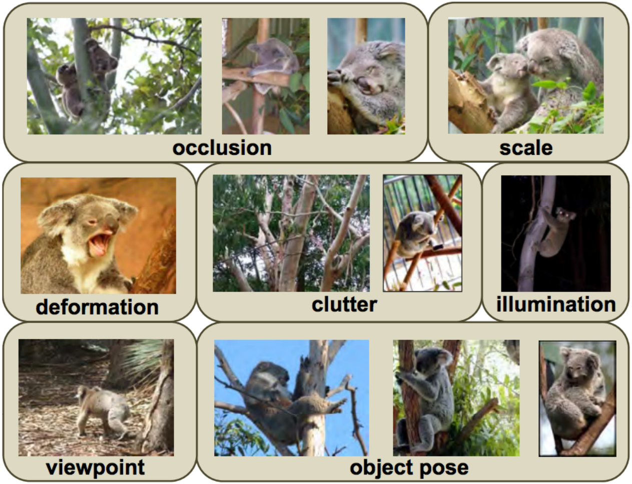
Neural Nets for Visual Object Recognition

- People are very good at recognizing shapes
  - Intrinsically difficult, computers are bad at it

- Why is it difficult?

Why is it a Problem?

- Difficult scene conditions



[From: Grauman & Leibe]

Why is it a Problem?

- Huge within-class variations. Recognition is mainly about modeling variation.

[Pi f S L b il]

Why is it a Problem?

- Tons of classes

[Biederman]

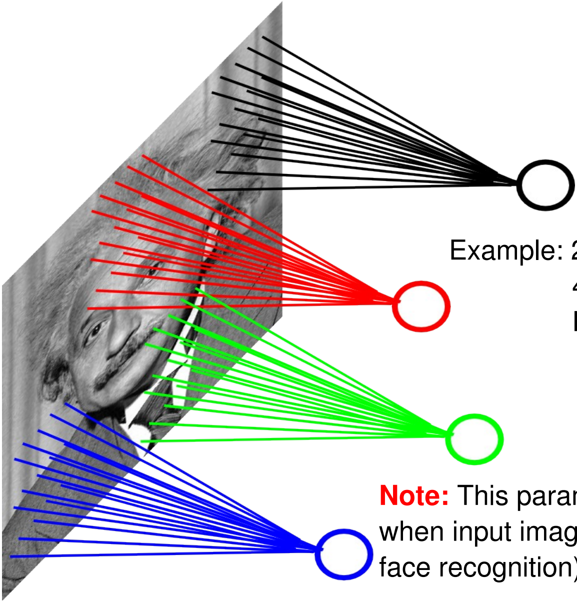
Neural Nets for Object Recognition

- People are very good at recognizing object
  - Intrinsically difficult, computers are bad at it
- Some reasons why it is difficult:
  - Segmentation: Real scenes are cluttered
  - Invariances: We are very good at ignoring all sorts of variations that do not affect class
  - Deformations: Natural object classes allow variations (faces, letters, chairs)
  - A huge amount of computation is required

How to Deal with Large Input Spaces

- How can we apply neural nets to images?
- Images can have millions of pixels, i.e., $\mathbf{x}$ is very high dimensional
- How many parameters do we have?
- Prohibitive to have fully-connected layers
- What can we do?
- We can use a locally connected layer

Locally Connected Layer



Example: 200x200 image
40K hidden units
Filter size: 10x10
4M parameters

Note: This parameterization is good when input image is registered (e.g., face recognition).

34

Ranzato

When Will this Work?

When Will this Work?

- This is good when the input is (roughly) registered

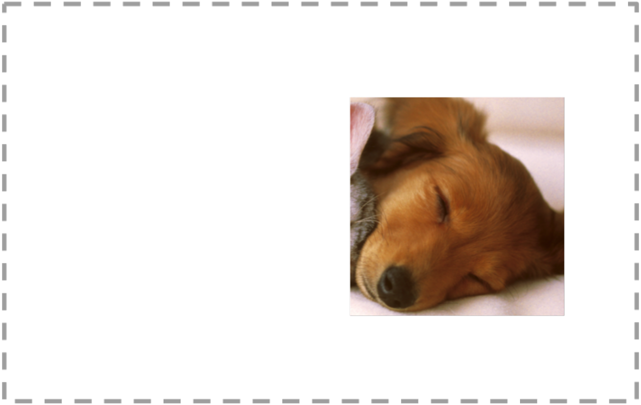
General Images

- The object can be anywhere

[Slide: Y. Zhu]

General Images

- The object can be anywhere



[Slide: Y. Zhu]

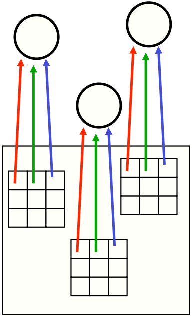
General Images

- The object can be anywhere

[Slide: Y. Zhu]

The replicated feature approach

The red connections all have the same weight.

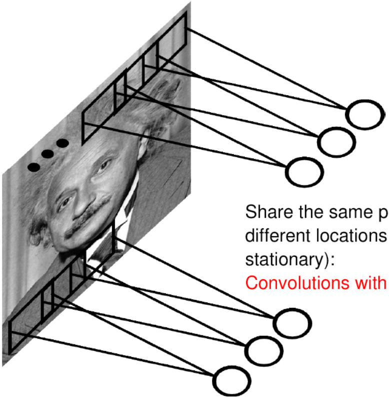


5

- Adopt approach apparently used in monkey visual systems
- Use many different copies of the same feature detector.
    - Copies have slightly different positions.
    - Could also replicate across scale and orientation.
        - Tricky and expensive
    - Replication **reduces the number of free parameters** to be learned.
- Use several **different feature types**, each with its own replicated pool of detectors.
    - Allows each patch of image to be represented in several ways.

Convolutional Neural Net

- Idea: Statistics are similar at different locations (Lecun 1998)
- Connect each hidden unit to a small input patch and share the weight across space
- This is called a convolution layer and the network is a convolutional network



Share the same parameters across different locations (assuming input is stationary):
Convolutions with learned kernels

36
Ranzato

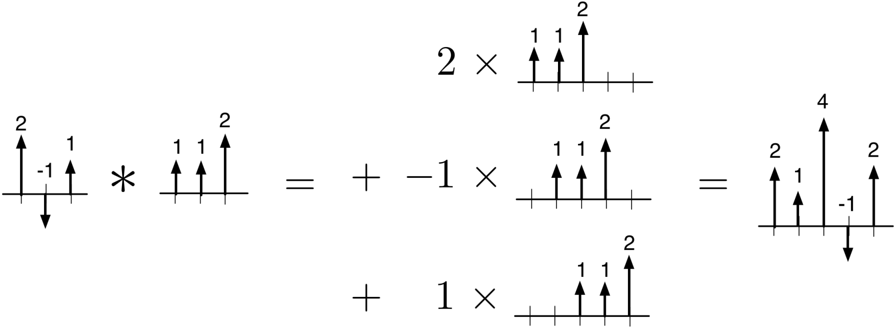
Convolution Operator

- Convolution layers are named after the convolution operator.
- If a and b are two arrays (or vector or signal), the convolution $a * b$ between them is defined as a new array (or vector or signal) with its t-th component being

$$(a * b)_t = \sum_{\tau} a_{\tau} b_{t-\tau}.$$

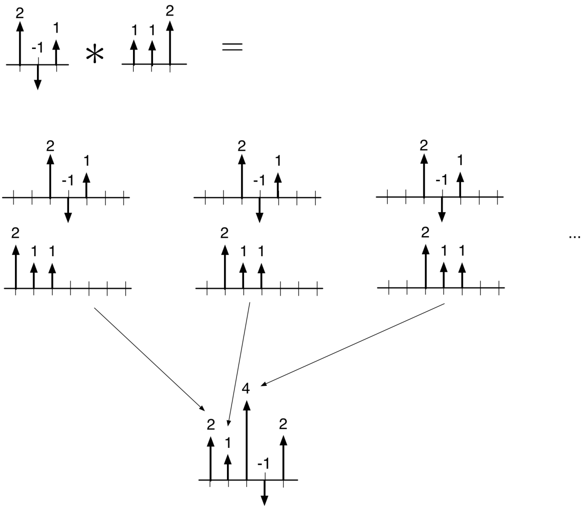
Convolution Operator

Method 1: translate-and-scale



Convolution Operator

Method 2: flip-and-filter



Convolution Operator

Convolution can also be viewed as matrix multiplication:

$$(2, -1, 1) * (1, 1, 2) = \begin{pmatrix} 1 & & \\ 1 & 1 & \\ 2 & 1 & 1 \\ & 2 & 1 \\ & & 2 \end{pmatrix} \begin{pmatrix} 2 \\ -1 \\ 1 \end{pmatrix}$$

Note: This is how convolution is typically implemented. It is more efficient than the fast Fourier transform (FFT) for modern conv nets on GPUs.

Convolution Operator

Some properties of convolution:

- Commutativity

$$a * b = b * a$$

- Distributivity

$$a * (\lambda_1 b + \lambda_2 c) = \lambda_1 a * b + \lambda_2 a * c$$

2-D Convolution Operator

2-D convolution is defined analogously to 1-D convolution.

If A and B are two 2-D arrays, then:

$$(A * B)_{ij} = \sum_s \sum_t A_{st} B_{i-s,j-t}.$$

2-D Convolution Operator

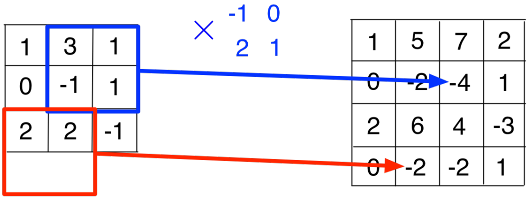
Method 1: Translate-and-Scale

$$\begin{array}{|c|c|c|}\hline 1&3&1\\\hline 0&-1&1\\\hline 2&2&-1\\\hline\end{array} * \begin{array}{|c|c|}\hline 1&2\\\hline 0&-1\\\hline\end{array} = \begin{array}{l} 1 \times \begin{array}{|c|c|c|c|}\hline 1&3&1&\\\hline 0&-1&1&\\\hline 2&2&-1&\\\hline &&&\\\hline\end{array} \\ + \; 2 \times \begin{array}{|c|c|c|c|}\hline &1&3&1\\\hline &0&-1&1\\\hline &2&2&-1\\\hline &&&\\\hline\end{array} \\ + -1 \times \begin{array}{|c|c|c|c|}\hline &&&\\\hline &1&3&1\\\hline &0&-1&1\\\hline &2&2&-1\\\hline\end{array} \end{array} = \begin{array}{|c|c|c|c|}\hline 1&5&7&2\\\hline 0&-2&-4&1\\\hline 2&6&4&-3\\\hline 0&-2&-2&1\\\hline\end{array}$$

2-D Convolution Operator

Method 2: Flip-and-Filter

$$\begin{array}{|c|c|c|}\hline 1 & 3 & 1 \\ \hline 0 & -1 & 1 \\ \hline 2 & 2 & -1 \\ \hline \end{array} \ast \begin{array}{|c|c|}\hline 1 & 2 \\ \hline 0 & -1 \\ \hline \end{array}$$



$$\begin{array}{|c|c|c|}\hline 1 & 3 & 1 \\ \hline 0 & -1 & 1 \\ \hline 2 & 2 & -1 \\ \hline \end{array} \quad \times \begin{array}{cc} -1 & 0 \\ 2 & 1 \end{array} \quad \begin{array}{|c|c|c|c|}\hline 1 & 5 & 7 & 2 \\ \hline 0 & -2 & -4 & 1 \\ \hline 2 & 6 & 4 & -3 \\ \hline 0 & -2 & -2 & 1 \\ \hline \end{array}$$

2-D Convolution Operator

We convolve an input by a kernel, or filter.

- The term Filter is used due to the original of convolutional in signal processing, in which the convolution operator is used to compute the effect of a linear filter on an input.
- Do not confuse this kernel with the kernels in an RKHS.

What does this filter do?

$*$

0	1	0
1	4	1
0	1	0

2-D Convolution Operator

What does this filter do?

$*$

0	-1	0
-1	8	-1
0	-1	0

2-D Convolution Operator

What does this filter do?

$*$

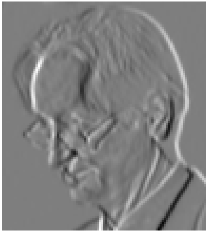
0	-1	0
-1	4	-1
0	-1	0

2-D Convolution Operator

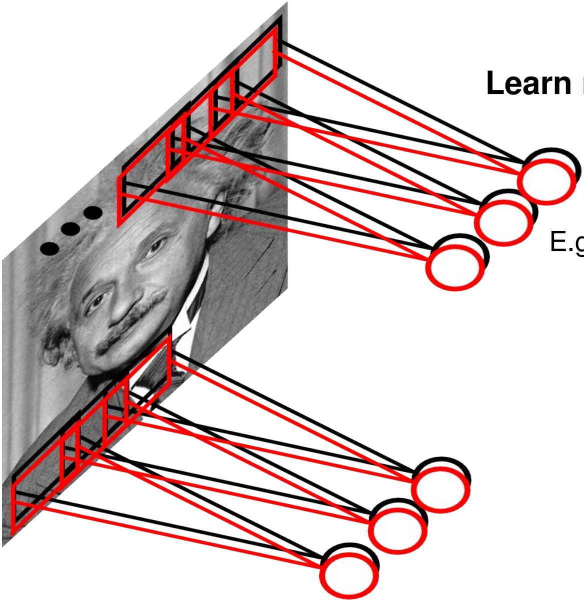
What does this filter do?

$*$

1	0	-1
2	0	-2
1	0	-1



Convolutional Layer



Learn multiple filters.

E.g.: 200x200 image

100 Filters

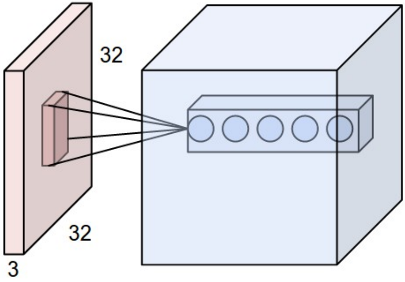
Filter size: 10x10

10K parameters

54

Ranzato

Convolutional Layer

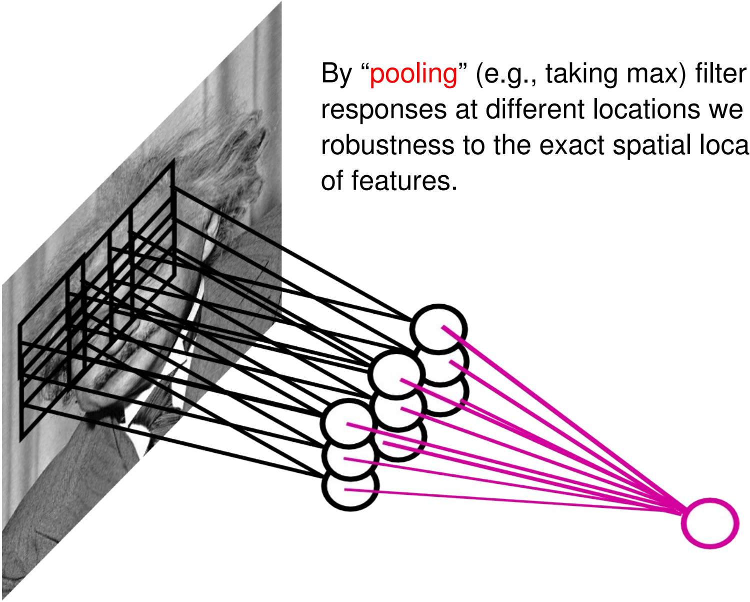


Hyperparameters of a convolutional layer:

- The number of filters (controls the **depth** of the output volume)
- The **stride**: how many units apart do we apply a filter spatially (this controls the spatial size of the output volume)
- The size $w \times h$ of the filters

[http://cs231n.github.io/convolutional-networks/]

Pooling Layer



By “pooling” (e.g., taking max) filter responses at different locations we gain robustness to the exact spatial location of features.

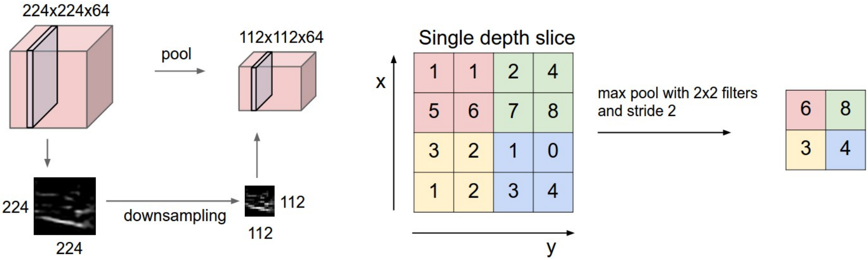
61

Ranzato

Pooling Options

- Max Pooling: return the maximal argument
- Average Pooling: return the average of the arguments
- Other types of pooling exist too

Pooling



1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

6	8
3	4

Figure: **Left:** Pooling, **right:** max pooling example

Hyperparameters of a pooling layer:

- The spatial extent F
- The stride

[http://cs231n.github.io/convolutional-networks/]

Backpropagation with Weight Constraints

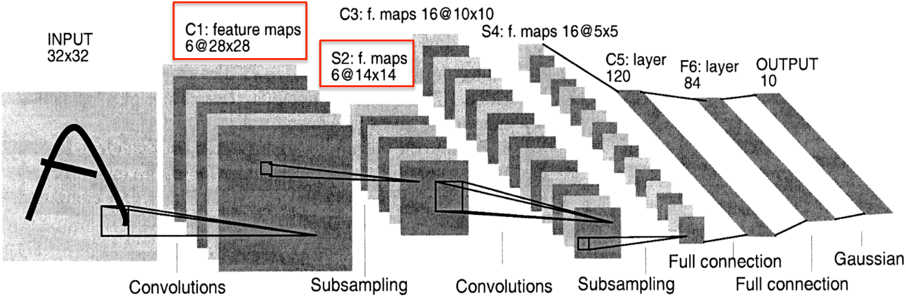
- The backpropagation algorithm from earlier can be applied directly to ConvNets
- This is covered in CSC2516.
- As a user, you do not need to worry about the details, since they are handled by automatic differentiation packages.

MNIST Dataset

- MNIST dataset of handwritten digits
  - **Categories:** 10 digit classes
  - **Source:** Scans of handwritten zip codes from envelopes
  - **Size:** 60,000 training images and 10,000 test images, grayscale, of size 28 × 28
  - **Normalization:** centered within in the image, scaled to a consistent size
    - The assumption is that the digit recognizer would be part of a larger pipeline that segments and normalizes images.
- In 1998, Yann LeCun and colleagues built a conv net called LeNet which was able to classify digits with 98.9% test accuracy.
  - It was good enough to be used in a system for automatically reading numbers on checks.

LeNet

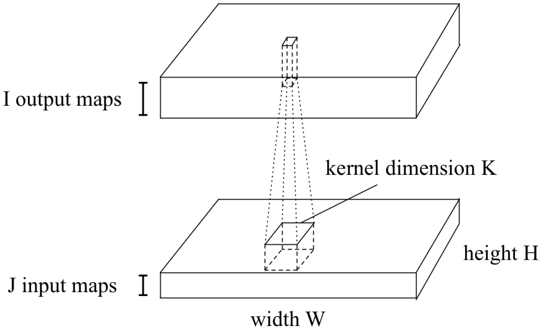
Here is the LeNet architecture, which was applied to handwritten digit recognition on MNIST in 1998:



Size of a Conv Net

- Ways to measure the size of a network:
  - **Number of units.** This is important because the activations need to be stored in memory during training (i.e. backprop).
  - **Number of weights.** This is important because the weights need to be stored in memory, and because the number of parameters affects the overfitting.
  - **Number of connections.** This is important because there are approximately 3 add-multiply operations per connection (1 for the forward pass, 2 for the backward pass).
- We saw that a fully connected layer with M input units and N output units has MN connections and MN weights.
- The story for conv nets is more complicated.

Size of a Conv Net



	fully connected layer	**convolution layer**
# output units	WHI	WHI
# weights	W^2H^2IJ	K^2IJ
# connections	W^2H^2IJ	WHK^2IJ

Size of a Conv Net

Sizes of layers in LeNet:

Layer	Type	# units	# connections	# weights
C1	convolution	4704	117,600	150
S2	pooling	1176	4704	0
C3	convolution	1600	240,000	2400
S4	pooling	400	1600	0
F5	fully connected	120	48,000	48,000
F6	fully connected	84	10,080	10,080
output	fully connected	10	840	840

Conclusions?

Size of a Conv Net

- Rules of thumb:
    - Most of the units and connections are in the convolution layers.
    - Most of the weights are in the fully connected layers.
- If you try to make layers larger, you'll run up against various resource limitations (i.e. computation time, memory)
- You'll repeat this exercise for AlexNet for homework.
    - Conv nets have gotten a LOT larger since 1998!

ImageNet

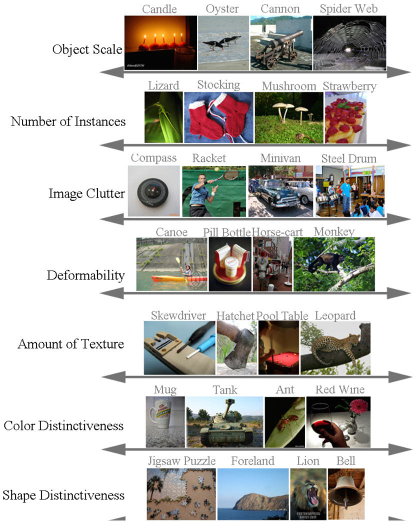
ImageNet is the modern object recognition benchmark dataset. It was introduced in 2009, and has led to amazing progress in object recognition since then.

ImageNet

- Used for the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), an annual benchmark competition for object recognition algorithms
- Design decisions
  - **Categories:** Taken from a lexical database called WordNet
    - WordNet consists of “synsets”, or sets of synonymous words
    - They tried to use as many of these as possible; almost 22,000 as of 2010
    - Of these, they chose the 1000 most common for the ILSVRC
    - The categories are really specific, e.g. hundreds of kinds of dogs
  - **Size:** 1.2 million full-sized images for the ILSVRC
  - **Source:** Results from image search engines, hand-labeled by Mechanical Turkers
    - Labeling such specific categories was challenging; annotators had to be given the WordNet hierarchy, Wikipedia, etc.
  - **Normalization:** none, although the contestants are free to do preprocessing

ImageNet

Images and object categories vary on a lot of dimensions



Russakovsky et al.

ImageNet

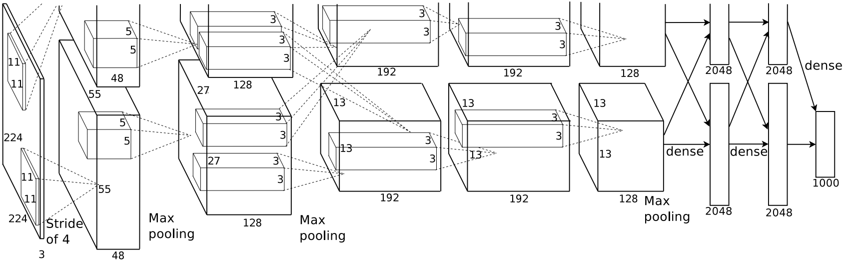
Size on disk:

MNIST
60 MB

ImageNet
50 GB

AlexNet

- AlexNet, 2012. 8 weight layers. 16.4% top-5 error (i.e. the network gets 5 tries to guess the right category).



(Krizhevsky et al., 2012)

- The two processing pathways correspond to 2 GPUs. (At the time, the network couldn't fit on one GPU.)
- AlexNet's stunning performance on the ILSVRC is what set off the deep learning boom of the last 8-9 years.

Inception

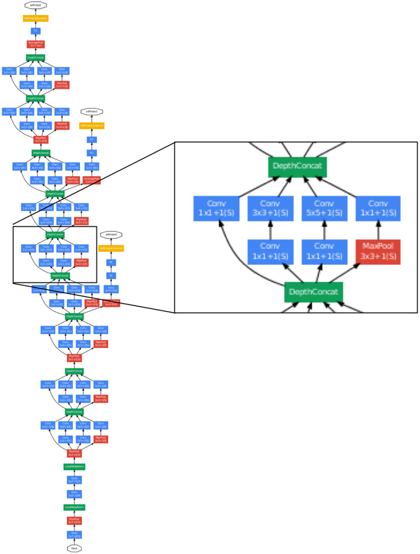
Inception, 2014. ("We need to go deeper!")

22 weight layers

Fully convolutional (no fully connected layers)

Convolutions are broken down into a bunch of smaller convolutions

6.6% test error on ImageNet



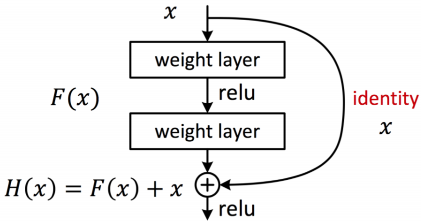
(Szegedy et al., 2014)

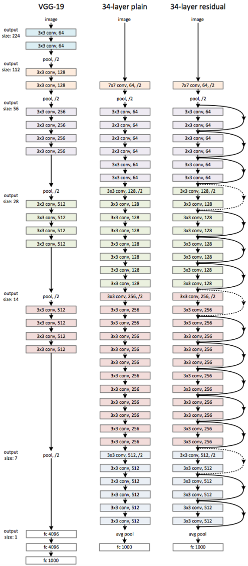
Inception

- They were really aggressive about cutting the number of parameters.
  - Motivation: train the network on a large cluster, run it on a cell phone
    - Memory at test time is the big constraint.
    - Having lots of units is OK, since the activations only need to be stored at training time (for backpropagation).
    - Parameters need to be stored both at training and test time, so these are the memory bottleneck.
  - How they did it
    - No fully connected layers (remember, these have most of the weights)
    - Break down convolutions into multiple smaller convolutions (since this requires fewer parameters total)
  - Inception has "only" 2 million parameters, compared with 60 million for AlexNet
  - This turned out to improve generalization as well. (Overfitting can still be a problem, even with over a million images!)

150 Layers!

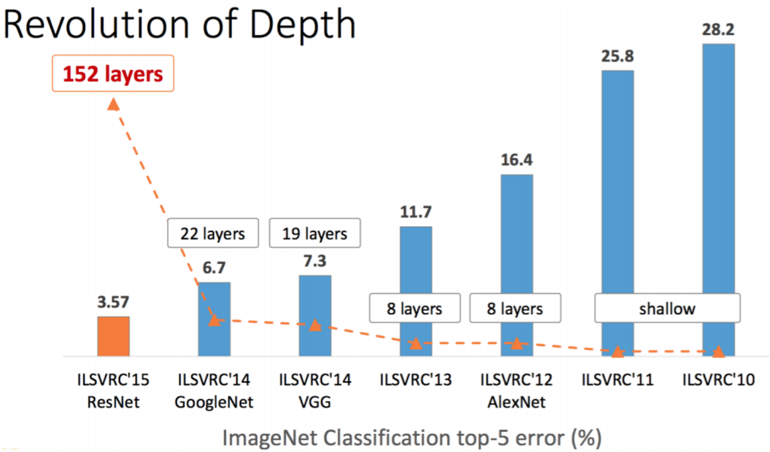
- Networks are now at 150 layers
- They use a skip connections with special form
- In fact, they don't fit on this screen
- Amazing performance!
- A lot of "mistakes" are due to wrong ground-truth





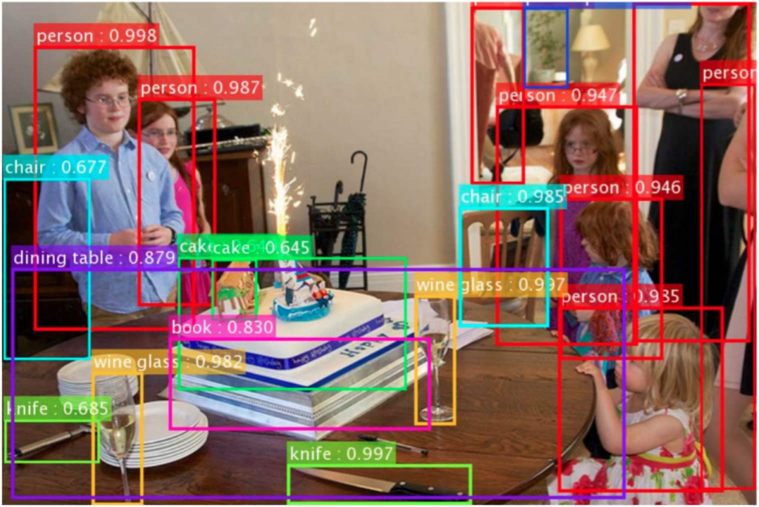
[He, K., Zhang, X., Ren, S. and Sun, J., 2015. Deep Residual Learning for Image Recognition. arXiv:1512.03385, 2016]

Results: Object Classification



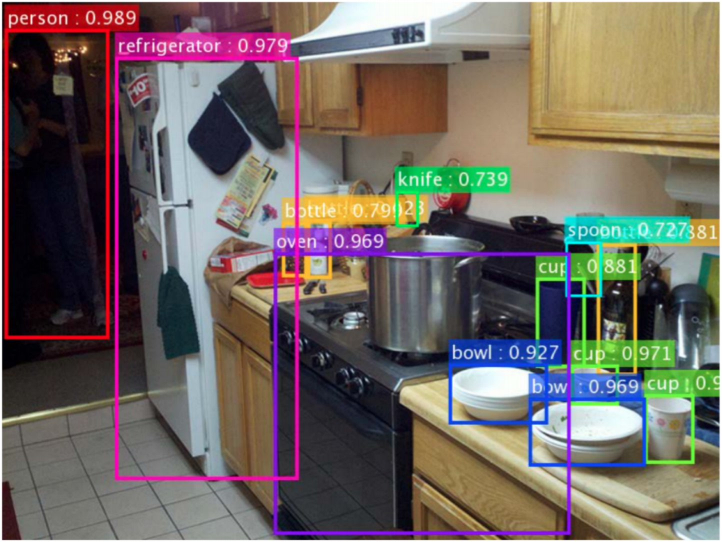
Slide: R. Liao, Paper: [He, K., Zhang, X., Ren, S. and Sun, J., 2015. Deep Residual Learning for Image Recognition. arXiv:1512.03385, 2016]

Results: Object Detection



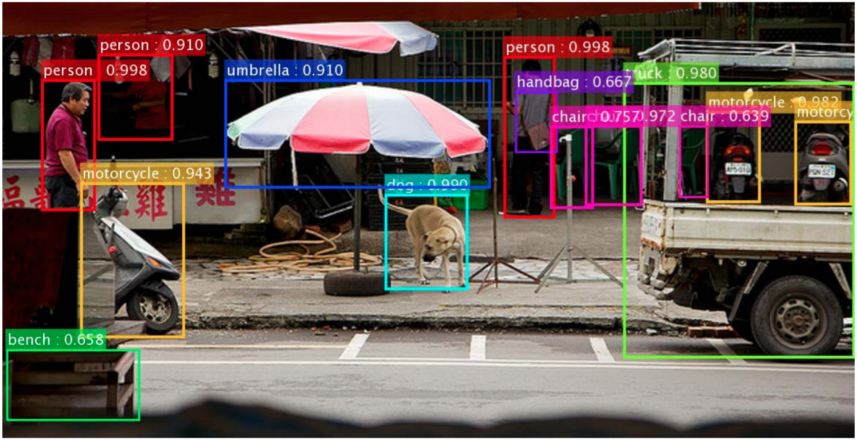
Slide: R. Liao, Paper: [He, K., Zhang, X., Ren, S. and Sun, J., 2015. Deep Residual Learning for Image Recognition. arXiv:1512.03385, 2016]

Results: Object Detection



Slide: R. Liao, Paper: [He, K., Zhang, X., Ren, S. and Sun, J., 2015. Deep Residual Learning for Image

Results: Object Detection

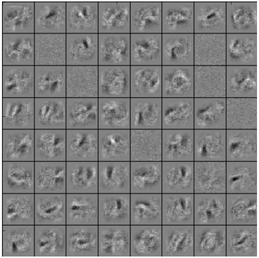


Slide: R. Liao, Paper: [He, K., Zhang, X., Ren, S. and Sun, J., 2015. Deep Residual Learning for Image Recognition. arXiv:1512.03385, 2016]

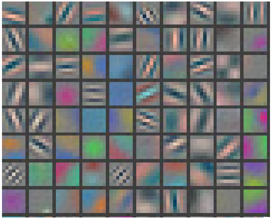
What Do Networks Learn?

- Recall: we can understand what first-layer features are doing by visualizing the weight matrices.

Fully connected (MNIST)



Convolutional (ImageNet)



- Higher-level weight matrices are hard to interpret.
- The better the input matches these weights, the more the feature activates.
  - Obvious generalization: visualize higher-level features by seeing what inputs activate them.