COMS 4995 NNDL Lecture 1: Introduction

Richard Zemel

Course information

- Second course in machine learning, with a focus on neural networks
    - This is an advanced machine learning course following Intro to ML with an in-depth focus on cutting-edge topics
    - Assumes knowledge of basic ML algorithms: linear regression, logistic regression, maximum likelihood, PCA, EM, etc.
    - First 2/3: supervised learning
    - Last 1/3: unsupervised learning and reinforcement learning

Course information

- Prerequisites:
  - **Machine Learning:** COMS 4771, or equivalent
  - **Multivariable Calculus**
  - **Linear Algebra**
  - **Probability & Statistics**
- It is your responsibility to ensure that you have these prerequisites. If you don't you should take this course next year after fulfilling them.

What should I know?

- Probability
  - Starting from the defition of independence, show that the independence of X and Y implies that their covariance is 0.
  - Write the transformation that takes $x \sim \mathcal{N}(0., 1.)$ to $z \sim \mathcal{N}(\mu, \sigma^2)$.
  - Write a code implementation to produce n independent samples from $\mathcal{N}(\mu, \sigma^2)$ by transforming n samples from $\mathcal{N}(0., 1.)$.
- Calculus
  - Let $x, y \in \Re^m, A \in \Re^{m \times n}$, and square matrix $B \in \Re^{m \times m}$. And where x' is the transpose of x. Answer the following questions in vector notation.
    - What is the gradient of $x'y$ with respect to x?
    - What is the gradient of $x'x$ with respect to x?
    - What is the Jacobian of A with respect to x?

What else should I know?

- Machine Learning
  - What is a validation set? Describe the trade-offs involved in assigning examples to the validation set versus the training set.
  - Suppose that you are training a decision tree but you would like to try an ensemble method. By random resampling, you create 100 copies of your data and train a separate decision tree based on each one of them, and predict outputs based on the majority vote of the trees. What is the effect of this procedure? How would your test error compare to a single decision tree predictor?
  - What are the advantages and disadvantages of *k*-nearest neighbors versus logistic regression? How do their decision boundaries compare?

Course information

- Expectations and marking
  - Written homeworks (36% of total mark)
    - Assignments will be a mix of written and programming problems
    - You will have 10 days to 2 weeks to do each assignment
    - The written part will consist of 2-3 short conceptual questions
    - They may also involve some mathematical derivations
    - The programming questions must be done in Python, PyTorch
    - They will involve 10-15 lines of code, and give you a chance to experiment with the algorithms
  - Exams
    - Midterm (16%)
    - Final (28%)
  - Project: 20%

How to get free GPUs

- **Colab (Mandatory)** Programming assignments are to be completed in Google Colab, which is a web-based iPython Notebook service that has access to a free Nvidia K80 GPU per Google account.
- **GCE (Recommended for course projects)** Google Compute Engine delivers virtual machines running in Google's data center. You get $300 free credit when you sign up.

Course information

Course web page:
`http://www.cs.columbia.edu/~zemel/Class/Nndl/index.html`

What is machine learning?

- For many problems, it's difficult to program the correct behavior by hand
    - recognizing people and objects
    - understanding human speech from audio files

What is machine learning?

- For many problems, it's difficult to program the correct behavior by hand
  - recognizing people and objects
  - understanding human speech from audio files
- Machine learning approach: program an algorithm to automatically learn from data, or from experience

What is machine learning?

- For many problems, it's difficult to program the correct behavior by hand
  - recognizing people and objects
  - understanding human speech from audio files
- Machine learning approach: program an algorithm to automatically learn from data, or from experience
- Some reasons you might want to use a learning algorithm:
  - hard to code up a solution by hand (e.g. vision, natural language processing)
  - system needs to adapt to a changing environment (e.g. spam detection)
  - want the system to perform *better* than the human programmers
  - privacy/fairness (e.g. ranking search results)

Relations to AI

- Nowadays, “machine learning” is often brought up with “artificial intelligence” (AI)

Relations to AI

- Nowadays, “machine learning” is often brought up with “artificial intelligence” (AI)
- AI often does not imply a learning based system
    - Symbolic reasoning
    - Rule based system
    - Tree search
    - etc.

Relations to AI

- Nowadays, “machine learning” is often brought up with “artificial intelligence” (AI)
- AI often does not imply a learning based system
    - Symbolic reasoning
    - Rule based system
    - Tree search
    - etc.
- Learning based system → learned based on the data → more flexibility, good at solving pattern recognition problems.

Relations to human learning

- It is tempting to imagine machine learning as a component in AI just like human learning in ourselves.

Relations to human learning

- It is tempting to imagine machine learning as a component in AI just like human learning in ourselves.
- Human learning is:
  - Very data efficient
  - An entire multitasking system (vision, language, motor control, etc.)
  - Takes at least a few years :)
- For serving specific purposes, machine learning doesn't have to look like human learning in the end.

Relations to human learning

- It is tempting to imagine machine learning as a component in AI just like human learning in ourselves.
- Human learning is:
  - Very data efficient
  - An entire multitasking system (vision, language, motor control, etc.)
  - Takes at least a few years :)
- For serving specific purposes, machine learning doesn't have to look like human learning in the end.
- It may borrow ideas from biological systems (e.g. neural networks).

Relations to human learning

- It is tempting to imagine machine learning as a component in AI just like human learning in ourselves.
- Human learning is:
  - Very data efficient
  - An entire multitasking system (vision, language, motor control, etc.)
  - Takes at least a few years :)
- For serving specific purposes, machine learning doesn't have to look like human learning in the end.
- It may borrow ideas from biological systems (e.g. neural networks).
- There may also be biological constraints.

History of machine learning

- 1957 — Perceptron algorithm (implemented as a circuit!)
- 1959 — Arthur Samuel wrote a learning-based checkers program that could defeat him
- 1969 — Minsky and Papert's book *Perceptrons* (limitations of linear models)

History of machine learning

- 1957 — Perceptron algorithm (implemented as a circuit!)
- 1959 — Arthur Samuel wrote a learning-based checkers program that could defeat him
- 1969 — Minsky and Papert's book *Perceptrons* (limitations of linear models)
- 1980s — Some foundational ideas
    - Connectionist psychologists explored neural models of cognition
    - 1984 — Leslie Valiant formalized the problem of learning as PAC learning
    - 1988 — Backpropagation (re-)discovered by Geoffrey Hinton and colleagues
    - 1988 — Judea Pearl's book *Probabilistic Reasoning in Intelligent Systems* introduced Bayesian networks

History of machine learning

- 1990s — the “AI Winter”, a time of pessimism and low funding

History of machine learning

- 1990s — the “AI Winter”, a time of pessimism and low funding
- But looking back, the ’90s were also sort of a golden age for ML research
    - Markov chain Monte Carlo
    - Variational inference
    - Kernels and support vector machines
    - Boosting
    - Convolutional networks

History of machine learning

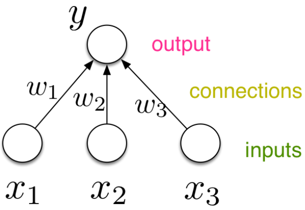
- 1990s — the “AI Winter”, a time of pessimism and low funding
- But looking back, the ’90s were also sort of a golden age for ML research
    - Markov chain Monte Carlo
    - Variational inference
    - Kernels and support vector machines
    - Boosting
    - Convolutional networks
- 2000s — applied AI fields (vision, NLP, etc.) adopted ML

History of machine learning

- 1990s — the “AI Winter”, a time of pessimism and low funding
- But looking back, the ’90s were also sort of a golden age for ML research
    - Markov chain Monte Carlo
    - Variational inference
    - Kernels and support vector machines
    - Boosting
    - Convolutional networks
- 2000s — applied AI fields (vision, NLP, etc.) adopted ML
- 2010s — deep learning
    - 2010–2012 — neural nets smashed previous records in speech-to-text and object recognition
    - increasing adoption by the tech industry
    - 2016 — AlphaGo defeated the human Go champion

What are neural networks?

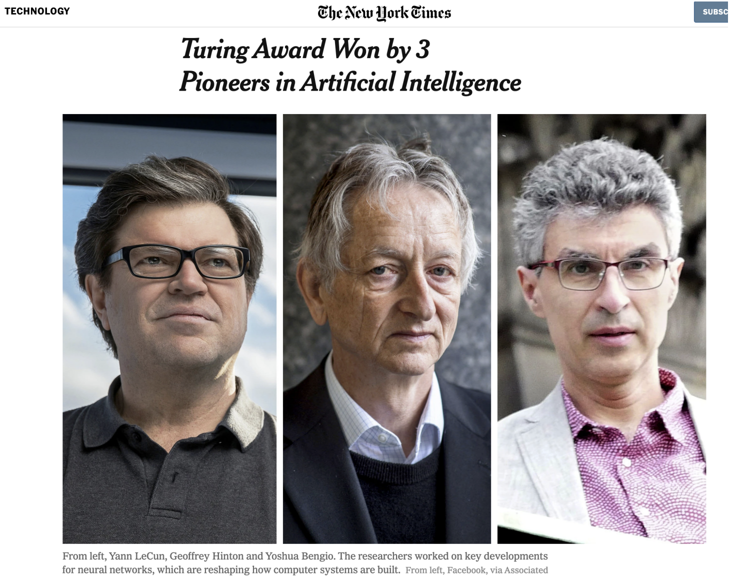
- Most of the biological details aren't essential, so we use vastly simplified models of neurons.
- While neural nets originally drew inspiration from the brain, nowadays we mostly think about math, statistics, etc.



$$\overset{\text{output}}{y} = \underset{\text{activation function}}{\phi}\left(\overset{\text{weights}}{\mathbf{w}}^{\top} \underset{\text{inputs}}{\mathbf{x}} + \overset{\text{bias}}{b}\right)$$

- Neural networks are collections of thousands (or millions) of these simple processing units that together perform useful computations.

What are neural networks?



TECHNOLOGY

The New York Times

SUBSC

Turing Award Won by 3 Pioneers in Artificial Intelligence

From left, Yann LeCun, Geoffrey Hinton and Yoshua Bengio. The researchers worked on key developments for neural networks, which are reshaping how computer systems are built. From left, Facebook, via Associated

What are neural networks?

Why neural nets?

- inspiration from the brain
  - proof of concept that a neural architecture can see and hear!
- very effective across a range of applications (vision, text, speech, medicine, robotics, etc.)
- widely used in both academia and the tech industry
- powerful software frameworks (PyTorch, TensorFlow, etc.) let us quickly implement sophisticated algorithms

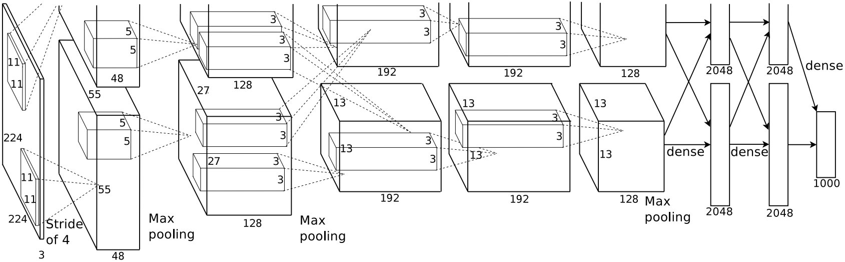
What are neural networks?

- Some near-synonyms for neural networks
  - “Deep learning”
    - Emphasizes that the algorithms often involve hierarchies with many stages of processing

“Deep learning”

Deep learning: many layers (stages) of processing

E.g. this network which recognizes objects in images:

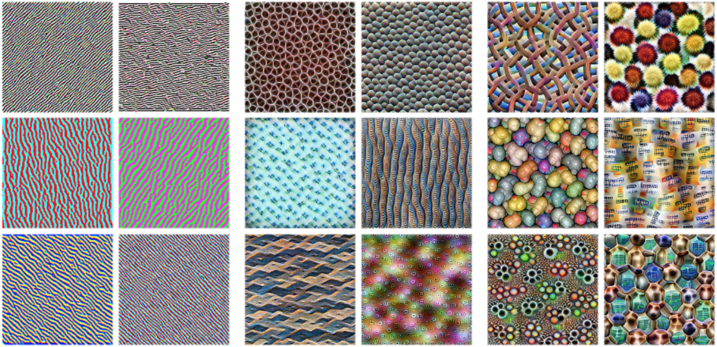


(Krizhevsky et al., 2012)

Each of the boxes consists of many neuron-like units similar to the one on the previous slide!

“Deep learning”

- You can visualize what a learned feature is responding to by finding an image that excites it. (We’ll see how to do this.)
- Higher layers in the network often learn higher-level, more interpretable representations

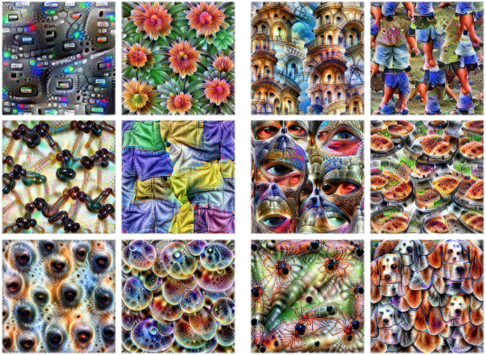


Edges (layer conv2d0)   **Textures** (layer mixed3a)   **Patterns** (layer mixed4a)

`https://distill.pub/2017/feature-visualization/`

“Deep learning”

- You can visualize what a learned feature is responding to by finding an image that excites it.
- Higher layers in the network often learn higher-level, more interpretable representations



Parts (layers mixed4b & mixed4c) **Objects** (layers mixed4d & mixed4e)

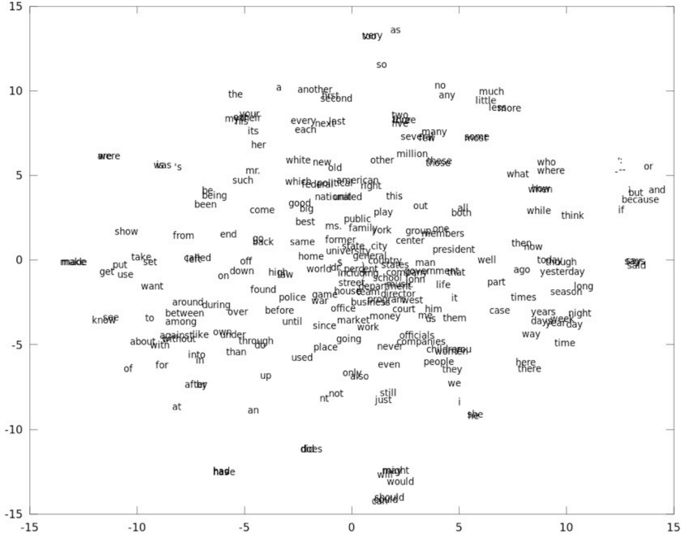
https://distill.pub/2017/feature-visualization/

What is a representation?

- How you represent your data determines what questions are easy to answer.
    - E.g. a dict of word counts is good for questions like “What is the most common word in *Hamlet*?”
    - It’s not so good for semantic questions like “if Alice liked *Harry Potter*, will she like *The Hunger Games*?”

What is a representation?

Idea: represent words as vectors



What is a representation?

- Mathematical relationships between vectors encode semantic relationships between words
  - Measure semantic similarity using the dot product (or dissimilarity using Euclidean distance)
  - Represent a web page with the average of its word vectors
  - Complete analogies by doing arithmetic on word vectors
    - e.g. “Paris is to France as London is to _____”
    - France – Paris + London = _____

What is a representation?

- Mathematical relationships between vectors encode semantic relationships between words
  - Measure semantic similarity using the dot product (or dissimilarity using Euclidean distance)
  - Represent a web page with the average of its word vectors
  - Complete analogies by doing arithmetic on word vectors
    - e.g. “Paris is to France as London is to ____”
    - France – Paris + London = ____
- It’s very hard to construct representations like these by hand, so we need to learn them from data
  - This is a big part of what neural nets do, whatever type of learning they are doing!

Types of machine learning

- **Supervised learning:** have labeled examples of the correct behavior, i.e. ground truth input/output response
- **Reinforcement learning:** learning system receives a reward signal, tries to learn to maximize the reward signal
- **Unsupervised learning:** no labeled examples – instead, looking for interesting patterns in the data

Supervised learning examples

Supervised learning: have labeled examples of the correct behavior

e.g. Handwritten digit classification with the MNIST dataset

- **Task:** given an image of a handwritten digit, predict the digit class
  - **Input:** the image
  - **Target:** the digit class

Supervised learning examples

Supervised learning: have labeled examples of the correct behavior

e.g. Handwritten digit classification with the MNIST dataset

- **Task:** given an image of a handwritten digit, predict the digit class
  - **Input:** the image
  - **Target:** the digit class
- **Data:** 70,000 images of handwritten digits labeled by humans
  - **Training set:** first 60,000 images, used to train the network
  - **Test set:** last 10,000 images, not available during training, used to evaluate performance

Supervised learning examples

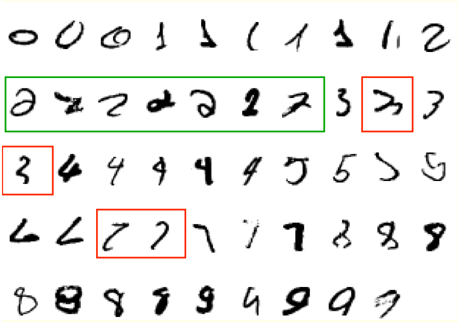
Supervised learning: have labeled examples of the correct behavior

e.g. Handwritten digit classification with the MNIST dataset

- **Task:** given an image of a handwritten digit, predict the digit class
    - **Input:** the image
    - **Target:** the digit class
- **Data:** 70,000 images of handwritten digits labeled by humans
    - **Training set:** first 60,000 images, used to train the network
    - **Test set:** last 10,000 images, not available during training, used to evaluate performance
- This dataset is the "fruit fly" of neural net research
- Neural nets already achieved $> 99\%$ accuracy in the 1990s, but we still continue to learn a lot from it

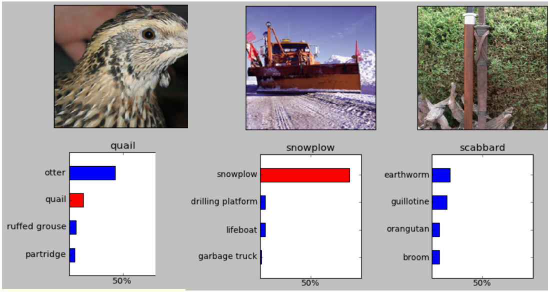
Supervised learning examples

What makes a “2”?



Supervised learning examples

Object recognition



(Krizhevsky and Hinton, 2012)

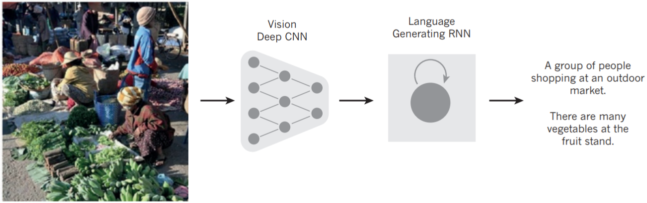
ImageNet dataset: one thousand categories, millions of labeled images

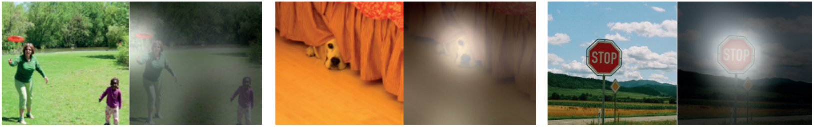
Lots of variability in viewpoint, lighting, etc.

Error rate dropped from 26% to under 4% over the course of a few years!

Supervised learning examples

Caption generation


Vision
Deep CNN
Language
Generating RNN
A group of people shopping at an outdoor market.
There are many vegetables at the fruit stand.



A woman is throwing a **frisbee** in a park.

A **dog** is standing on a hardwood floor.

A **stop** sign is on a road with a mountain in the background

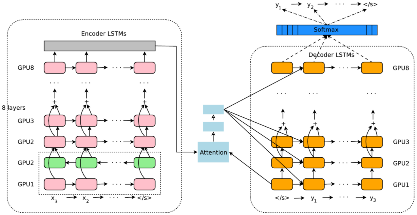
(Xu et al., 2015)

Given: dataset of Flickr images with captions

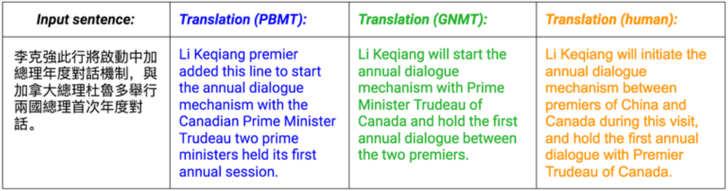
More examples at `http://deeplearning.cs.toronto.edu/i2t`

Supervised learning examples

Neural Machine Translation



(Wu et al., 2016)

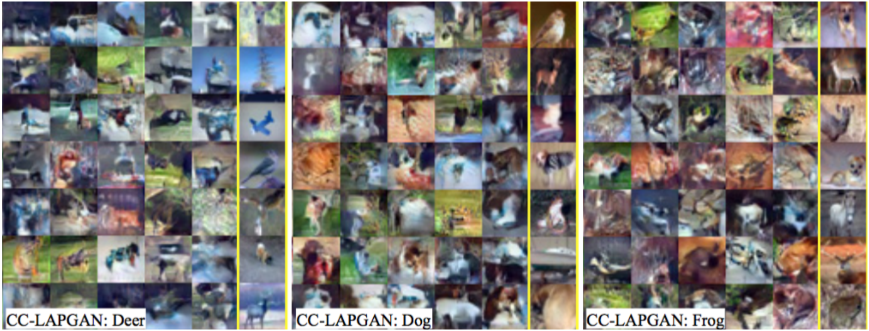


Input sentence:	***Translation (PBMT):***	***Translation (GNMT):***	***Translation (human):***
李克強此行將啟動中加總理年度對話機制，與加拿大總理杜魯多舉行兩國總理首次年度對話。	Li Keqiang premier added this line to start the annual dialogue mechanism with the Canadian Prime Minister Trudeau two prime ministers held its first annual session.	Li Keqiang will start the annual dialogue mechanism with Prime Minister Trudeau of Canada and hold the first annual dialogue between the two premiers.	Li Keqiang will initiate the annual dialogue mechanism between premiers of China and Canada during this visit, and hold the first annual dialogue with Premier Trudeau of Canada.

Now the production model on Google Translate

Unsupervised learning examples

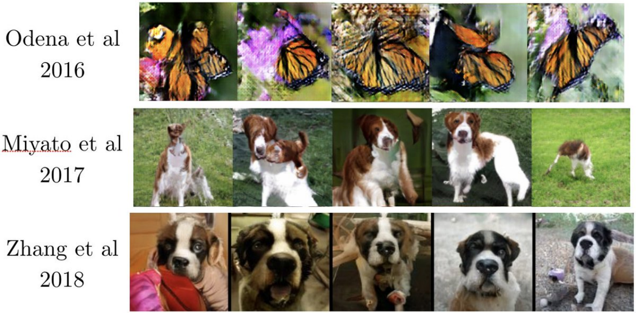
- In **generative modeling**, we want to learn a distribution over some dataset, such as natural images.
- We can evaluate a generative model by sampling from the model and seeing if it looks like the data.
- These results were considered impressive in 2014:



Denton et al., 2014, Deep generative image models using a Laplacian pyramid of adversarial networks

Unsupervised learning examples

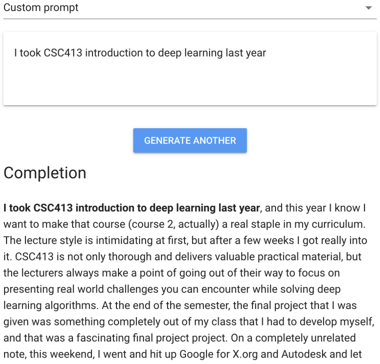
- The progress of generative models:



- Big GAN, Brock et al, 2019:

Unsupervised learning examples

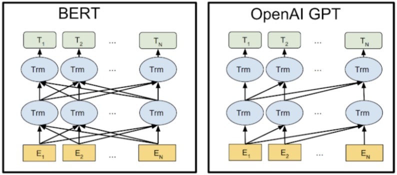
- Generative models of text. The models like BERT and GPT-2 perform unsupervised learning by reconstructing the next words in a sentence. The GPT-2 models learns from 40GB of Internet text.



I took CSC413 introduction to deep learning last year

Completion

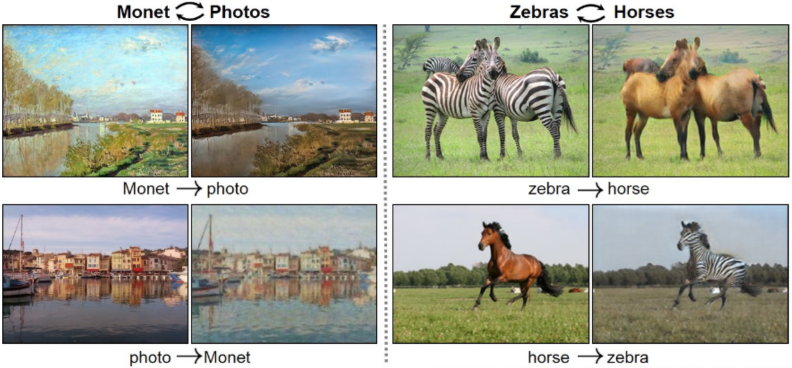
I took CSC413 introduction to deep learning last year, and this year I know I want to make that course (course 2, actually) a real staple in my curriculum. The lecture style is intimidating at first, but after a few weeks I got really into it. CSC413 is not only thorough and delivers valuable practical material, but the lecturers always make a point of going out of their way to focus on presenting real world challenges you can encounter while solving deep learning algorithms. At the end of the semester, the final project that I was given was something completely out of my class that I had to develop myself, and that was a fascinating final project project. On a completely unrelated note, this weekend, I went and hit up Google for X.org and Autodesk and let



https://talktotransformer.com/

Unsupervised learning examples

- Recent exciting result: a model called the CycleGAN takes lots of images of one category (e.g. horses) and lots of images of another category (e.g. zebras) and learns to translate between them.



https://github.com/junyanz/CycleGAN

Reinforcement learning

- An **agent** interacts with an **environment** (e.g. game of Breakout)
- In each time step,
  - the agent receives **observations** (e.g. pixels) which give it information about the **state** (e.g. positions of the ball and paddle)
  - the agent picks an **action** (e.g. keystrokes) which affects the state
- The agent periodically receives a **reward** (e.g. points)
- The agent wants to learn a **policy**, or mapping from observations to actions, which maximizes its average reward over time

Reinforcement learning

DeepMind trained neural networks to play many different Atari games

- given the raw screen as input, plus the score as a reward
- single network architecture shared between all the games
- in many cases, the networks learned to play better than humans (in terms of points in the first minute)

`https://www.youtube.com/watch?v=V1eYniJ0Rnk`

Reinforcement learning for control

Learning locomotion control from scratch

- The reward is to run as far as possible over all the obstacles
- single control policy that learns to adapt to different terrains

https://www.youtube.com/watch?v=hx_bgoTF7bs

Software frameworks

- Scientific computing (NumPy)
    - **vectorize** computations (express them in terms of matrix/vector operations) to exploit hardware efficiency
- Neural net frameworks: PyTorch, TensorFlow, etc.
    - automatic differentiation
    - compiling computation graphs
    - libraries of algorithms and network primitives
    - support for graphics processing units (GPUs)
- For this course:
    - Python, NumPy
    - **PyTorch**, a widely used neural net framework with a built-in automatic differentiation feature

Software frameworks

Why take this class, if PyTorch does so much for you?

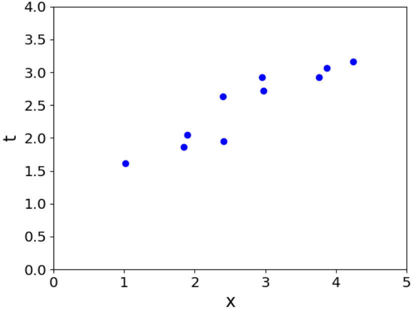
So you know what do to if something goes wrong!

- Debugging learning algorithms requires sophisticated detective work, which requires understanding what goes on beneath the hood.
- That's why we derive things by hand in this class!

Linear Models: Overview

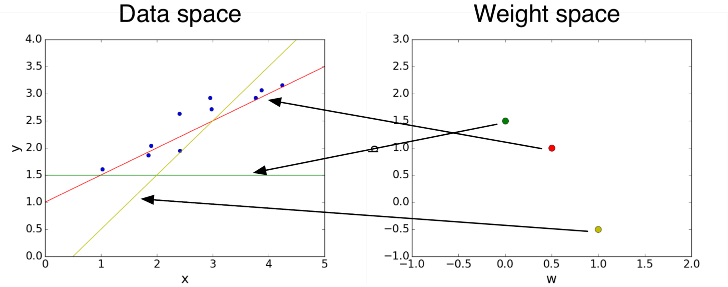
- One of the fundamental building blocks in deep learning are the linear models, where you decide based on a linear function of the input vector.
- Here, we will review linear models, some other fundamental concepts (e.g. gradient descent, generalization), and some of the common supervised learning problems:
    - Regression: predict a scalar-valued target (e.g. stock price)
    - Binary classification: predict a binary label (e.g. spam vs. non-spam email)
    - Multiway classification: predict a discrete label (e.g. object category, from a list)

Problem Setup



- Want to predict a scalar t as a function of a vector $\mathbf{x}$
- Given a dataset of pairs $\{(\mathbf{x}^{(i)}, t^{(i)})\}_{i=1}^{N}$
- The $\mathbf{x}^{(i)}$ are called input vectors, and the $t^{(i)}$ are called targets.

Problem Setup



- **Model:** y is a linear function of x:

$$y = \mathbf{w}^\top \mathbf{x} + b$$

- y is the prediction
- $\mathbf{w}$ is the weight vector
- b is the bias
- $\mathbf{w}$ and b together are the parameters
- Settings of the parameters are called hypotheses

Problem Setup

- **Loss function:** squared error

$$\mathcal{L}(y, t) = \frac{1}{2}(y - t)^2$$

- $y - t$ is the residual, and we want to make this small in magnitude
- The $\frac{1}{2}$ factor is just to make the calculations convenient.

Problem Setup

- **Loss function:** squared error

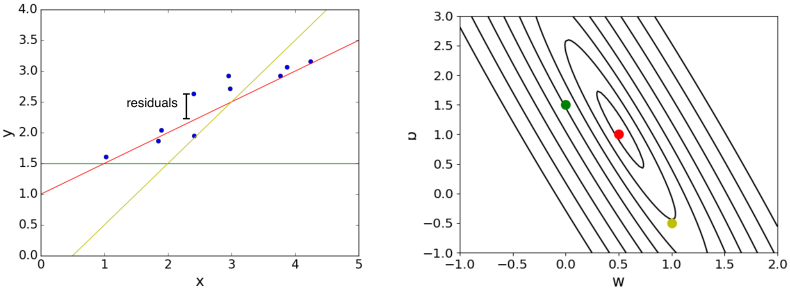
$$\mathcal{L}(y, t) = \frac{1}{2}(y - t)^2$$

- $y - t$ is the residual, and we want to make this small in magnitude
- The $\frac{1}{2}$ factor is just to make the calculations convenient.
- **Cost function:** loss function averaged over all training examples

$$\begin{aligned} \mathcal{J}(w, b) &= \frac{1}{2N} \sum_{i=1}^{N} \left(y^{(i)} - t^{(i)} \right)^2 \\ &= \frac{1}{2N} \sum_{i=1}^{N} \left(\mathbf{w}^\top \mathbf{x}^{(i)} + b - t^{(i)} \right)^2 \end{aligned}$$

Problem Setup

Visualizing the contours of the cost function:



Vectorization

- We can organize all the training examples into a matrix **X** with one row per training example, and all the targets into a vector **t**.

one feature across all training examples

$$\mathbf{X} = \begin{pmatrix} \mathbf{x}^{(1)\top} \\ \mathbf{x}^{(2)\top} \\ \mathbf{x}^{(3)\top} \end{pmatrix} = \begin{pmatrix} 8 & 0 & 3 & 0 \\ 6 & -1 & 5 & 3 \\ 2 & 5 & -2 & 8 \end{pmatrix}$$

one training example (vector)

- Computing the predictions for the whole dataset:

$$\mathbf{X}\mathbf{w} + b\mathbf{1} = \begin{pmatrix} \mathbf{w}^\top \mathbf{x}^{(1)} + b \\ \vdots \\ \mathbf{w}^\top \mathbf{x}^{(N)} + b \end{pmatrix} = \begin{pmatrix} y^{(1)} \\ \vdots \\ y^{(N)} \end{pmatrix} = \mathbf{y}$$

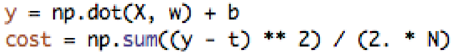
Vectorization

- Computing the squared error cost across the whole dataset:

$$\mathbf{y} = \mathbf{X}\mathbf{w} + b\mathbf{1}$$

$$\mathcal{J} = \frac{1}{2N}\|\mathbf{y} - \mathbf{t}\|^2$$

- In Python:



```
y = np.dot(X, w) + b
cost = np.sum((y - t) ** 2) / (2. * N)
```

Solving the optimization problem

- We defined a cost function. This is what we'd like to minimize.
- Recall from calculus class: the minimum of a smooth function (if it exists) occurs at a critical point, i.e. point where the partial derivatives are all 0.
- Two strategies for optimization:
  - Direct solution: derive a formula that sets the partial derivatives to 0. This works only in a handful of cases (e.g. linear regression).
  - Iterative methods (e.g. gradient descent): repeatedly apply an update rule which slightly improves the current solution. This is what we'll do throughout the course.

Direct solution

- Partial derivatives: derivatives of a multivariate function with respect to one of its arguments.

$$\frac{\partial}{\partial x_1} f(x_1, x_2) = \lim_{h \to 0} \frac{f(x_1 + h, x_2) - f(x_1, x_2)}{h}$$

- To compute, take the single variable derivatives, pretending the other arguments are constant.
- Example: partial derivatives of the prediction y

$$\begin{aligned} \frac{\partial y}{\partial w_j} &= \frac{\partial}{\partial w_j} \left[\sum_{j'} w_{j'} x_{j'} + b \right] \\ &= x_j \\ \frac{\partial y}{\partial b} &= \frac{\partial}{\partial b} \left[\sum_{j'} w_{j'} x_{j'} + b \right] \\ &= 1 \end{aligned}$$

Direct solution

- Chain rule for derivatives:

$$
\begin{aligned}
\frac{\partial \mathcal{L}}{\partial w_j} &= \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}y}\frac{\partial y}{\partial w_j} \\
&= \frac{\mathrm{d}}{\mathrm{d}y}\left[\frac{1}{2}(y-t)^2\right]\cdot x_j \\
&= (y-t)x_j \\
\frac{\partial \mathcal{L}}{\partial b} &= y-t
\end{aligned}
$$

- We will give a more precise statement of the Chain Rule next week. It's actually pretty complicated.
- Cost derivatives (average over data points):

$$
\begin{aligned}
\frac{\partial \mathcal{J}}{\partial w_j} &= \frac{1}{N}\sum_{i=1}^{N}(y^{(i)}-t^{(i)})\,x_j^{(i)} \\
\frac{\partial \mathcal{J}}{\partial b} &= \frac{1}{N}\sum_{i=1}^{N}y^{(i)}-t^{(i)}
\end{aligned}
$$

Gradient descent

- Gradient descent is an iterative algorithm, which means we apply an update repeatedly until some criterion is met.
- We initialize the weights to something reasonable (e.g. all zeros) and repeatedly adjust them in the direction of steepest descent.
- The gradient descent update decreases the cost function for small enough α:

$$\begin{aligned} w_j &\leftarrow w_j - \alpha \frac{\partial \mathcal{J}}{\partial w_j} \\ &= w_j - \frac{\alpha}{N} \sum_{i=1}^{N} (y^{(i)} - t^{(i)})\, x_j^{(i)} \end{aligned}$$

- α is a learning rate. The larger it is, the faster **w** changes.
  - We'll see later how to tune the learning rate, but values are typically small, e.g. 0.01 or 0.0001

Gradient descent

- This gets its name from the gradient:

$$\nabla \mathcal{J}(\mathbf{w}) = \frac{\partial \mathcal{J}}{\partial \mathbf{w}} = \begin{pmatrix} \frac{\partial \mathcal{J}}{\partial w_1} \\ \vdots \\ \frac{\partial \mathcal{J}}{\partial w_D} \end{pmatrix}$$

  - This is the direction of fastest increase in $\mathcal{J}$.

Gradient descent

- This gets its name from the gradient:

$$\nabla \mathcal{J}(\mathbf{w}) = \frac{\partial \mathcal{J}}{\partial \mathbf{w}} = \begin{pmatrix} \frac{\partial \mathcal{J}}{\partial w_1} \\ \vdots \\ \frac{\partial \mathcal{J}}{\partial w_D} \end{pmatrix}$$

  - This is the direction of fastest increase in $\mathcal{J}$.

- Update rule in vector form:

$$\begin{aligned} \mathbf{w} &\leftarrow \mathbf{w} - \alpha \nabla \mathcal{J}(\mathbf{w}) \\ &= \mathbf{w} - \frac{\alpha}{N} \sum_{i=1}^{N} (y^{(i)} - t^{(i)})\, \mathbf{x}^{(i)} \end{aligned}$$

- Hence, gradient descent updates the weights in the direction of fastest *decrease*.

Gradient descent

Visualization:
http://www.cs.toronto.edu/~guerzhoy/321/lec/W01/linear_regression.pdf#page=21

Gradient descent

- Why gradient descent, if we can find the optimum directly?
  - GD can be applied to a much broader set of models
  - GD can be easier to implement than direct solutions, especially with automatic differentiation software
  - For regression in high-dimensional spaces, GD is more efficient than direct solution (matrix inversion is an $\mathcal{O}(D^3)$ algorithm).

Feature maps

- We can convert linear models into nonlinear models using feature maps.

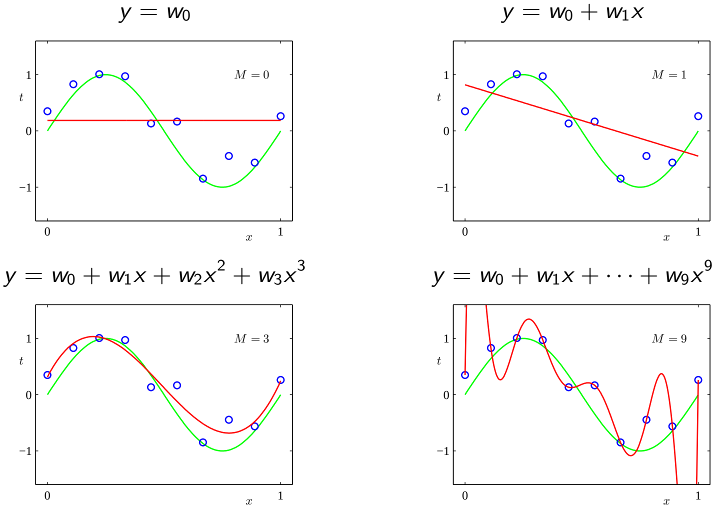
$$y = \mathbf{w}^\top \phi(\mathbf{x})$$

- E.g., if $\psi(x) = (1, \ x, \ \cdots, x^D)^\top$, then y is a polynomial in x. This model is known as polynomial regression:

$$y = w_0 + w_1 x + \cdots + w_D x^D$$

- This doesn't require changing the algorithm — just pretend $\psi(x)$ is the input vector.
- We don't need an expicit bias term, since it can be absorbed into ψ.
- Feature maps let us fit nonlinear models, but it can be hard to choose good features.
  - Before deep learning, most of the effort in building a practical machine learning system was feature engineering.

Feature maps



$y = w_0$

$y = w_0 + w_1 x$

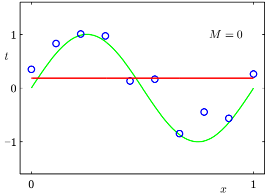
$y = w_0 + w_1 x + w_2 x^2 + w_3 x^3$

$y = w_0 + w_1 x + \cdots + w_9 x^9$

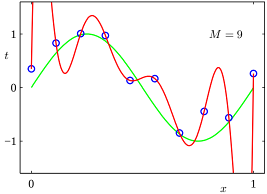
-Pattern Recognition and Machine Learning, Christopher Bishop.

Generalization

Underfitting : The model is too simple - does not fit the data.

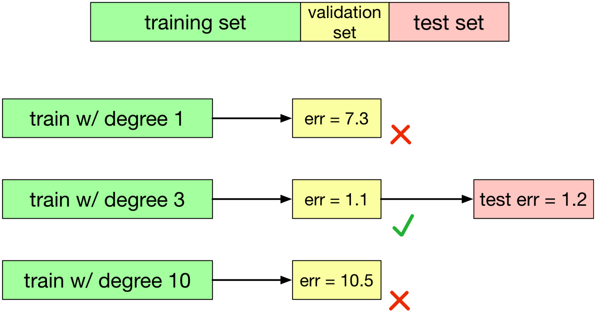


Overfitting : The model is too complex - fits perfectly, does not generalize.



Generalization

- We would like our models to generalize to data they haven't seen before
- The degree of the polynomial is an example of a hyperparameter, something we can't include in the training procedure itself
- We can tune hyperparameters using a validation set:



Classification

Binary linear classification

- **classification:** predict a discrete-valued target
- **binary:** predict a binary target $t \in \{0, 1\}$
  - Training examples with $t = 1$ are called positive examples, and training examples with $t = 0$ are called negative examples. Sorry.
- **linear:** model is a linear function of $\mathbf{x}$, thresholded at zero:

$$z = \mathbf{w}^T\mathbf{x} + b$$

$$\text{output} = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$

Logistic Regression

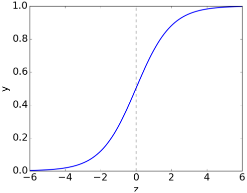
- We can't optimize classification accuracy directly with gradient descent because it's discontinuous.
- Instead, we typically define a continuous surrogate loss function which is easier to optimize. Logistic regression is a canonical example of this, in the context of classification.
- The model outputs a continuous value $y \in [0, 1]$, which you can think of as the probability of the example being positive.

Logistic Regression

- There's obviously no reason to predict values outside [0, 1]. Let's squash y into this interval.

- The logistic function is a kind of sigmoidal, or S-shaped, function:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$



- A linear model with a logistic nonlinearity is known as log-linear:

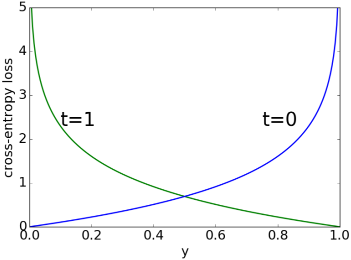
$$z = \mathbf{w}^\top \mathbf{x} + b$$
$$y = \sigma(z)$$

- Used in this way, σ is called an activation function, and z is called the logit.

Logistic Regression

- Because $y \in [0, 1]$, we can interpret it as the estimated probability that $t = 1$.
- Being 99% confident of the wrong answer is much worse than being 90% confident of the wrong answer. Cross-entropy loss captures this intuition:

$$\mathcal{L}_{\mathrm{CE}}(y, t) = \begin{cases} -\log y & \text{if } t = 1 \\ -\log(1 - y) & \text{if } t = 0 \end{cases}$$
$$= -t \log y - (1 - t) \log(1 - y)$$

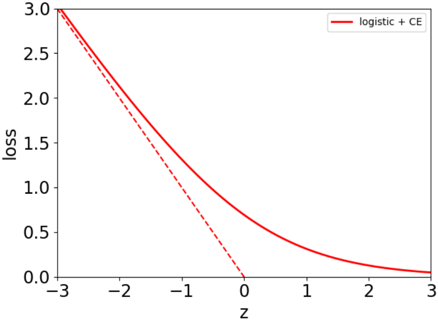


- Aside: why does it make sense to think of y as a probability? Because cross-entropy loss is a proper scoring rule, which means the optimal y is the true probability.

Logistic Regression

- Logistic regression combines the logistic activation function with cross-entropy loss.

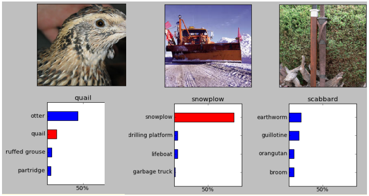
$$
\begin{aligned}
z &= \mathbf{w}^\top \mathbf{x} + b \\
y &= \sigma(z) \\
&= \frac{1}{1 + e^{-z}} \\
\mathcal{L}_{\mathrm{CE}} &= -t \log y - (1 - t) \log(1 - y)
\end{aligned}
$$



- Interestingly, the loss asymptotes to a linear function of the logit z.
- Full derivation in the readings.

Multiclass Classification

- What about classification tasks with more than two categories?



Multiclass Classification

- Targets form a discrete set $\{1, \ldots, K\}$.
- It's often more convenient to represent them as one-hot vectors, or a one-of-K encoding:

$$\mathbf{t} = \underbrace{(0, \ldots, 0, 1, 0, \ldots, 0)}_{\text{entry } k \text{ is } 1}$$

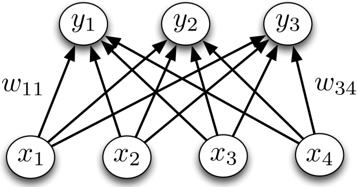
Multiclass Classification

- Now there are D input dimensions and K output dimensions, so we need $K \times D$ weights, which we arrange as a weight matrix $\mathbf{W}$.
- Also, we have a K-dimensional vector $\mathbf{b}$ of biases.
- Linear predictions:

$$z_k = \sum_j w_{kj} x_j + b_k$$

- Vectorized:

$$\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$$



Multiclass Classification

- A natural activation function to use is the softmax function, a multivariable generalization of the logistic function:

$$y_k = \mathrm{softmax}(z_1, \ldots, z_K)_k = \frac{e^{z_k}}{\sum_{k'} e^{z_{k'}}}$$

- The inputs z_k are called the logits.
- Properties:
  - Outputs are positive and sum to 1 (so they can be interpreted as probabilities)
  - If one of the z_k's is much larger than the others, $\mathrm{softmax}(\mathbf{z})$ is approximately the argmax. (So really it's more like "soft-argmax".)
  - **Exercise:** how does the case of $K = 2$ relate to the logistic function?
- Note: sometimes $\sigma(\mathbf{z})$ is used to denote the softmax function; in this class, it will denote the logistic function applied elementwise.

Multiclass Classification

- If a model outputs a vector of class probabilities, we can use cross-entropy as the loss function:

$$\begin{aligned}\mathcal{L}_{\mathrm{CE}}(\mathbf{y}, \mathbf{t}) &= -\sum_{k=1}^{K} t_k \log y_k \\ &= -\mathbf{t}^{\top}(\log \mathbf{y}),\end{aligned}$$

  where the log is applied elementwise.
- Just like with logistic regression, we typically combine the softmax and cross-entropy into a softmax-cross-entropy function.

Multiclass Classification

- Softmax regression, also called multiclass logistic regression:

$$\begin{aligned} \mathbf{z} &= \mathbf{W}\mathbf{x} + \mathbf{b} \\ \mathbf{y} &= \mathrm{softmax}(\mathbf{z}) \\ \mathcal{L}_{\mathrm{CE}} &= -\mathbf{t}^{\top}(\log \mathbf{y}) \end{aligned}$$

- It's possible to show the gradient descent updates have a convenient form:

$$\frac{\partial \mathcal{L}_{\mathrm{CE}}}{\partial \mathbf{z}} = \mathbf{y} - \mathbf{t}$$