

# CSC413 Lecture 1: Introduction

Jimmy Ba and Bo Wang

# Course information

- Second course in machine learning, with a focus on neural networks
  - CSC413 is an advanced machine learning course following CSC411 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
- Four sections
  - Equivalent content, same assignments and midterms

# Course information

- Formal prerequisites or equivalent in the tri-campus system:
  - **Multivariable Calculus:** MAT235/MAT237/MAT257/equivalent
  - **Linear Algebra:** MAT221H1/MAT223H1/MAT240H1/equivalent
  - **Machine Learning:** CSC311/STA314/ECE421/ROB313/equivalent
- Prerequisites will be enforced, including for grad students. See details on the FAS calendar.

# Course information

- Expectations and marking (**undergrads**)
  - 4 Assignments (60% of total mark)
    - Due Friday nights at 11:59pm
    - First homework will be out 1/15, due 2/03
    - Written part: 2-3 conceptual questions
    - Programming part: 10-15 lines of python code using PyTorch
  - Exams
    - Midterm quiz (openbook) (10%)
    - Final project (30%)
- See Course Information handout for detailed policies
- **Important policy:** A minimum of 3 out of 4 assignments must be submitted on time (with grace days) to pass the course.
  - Every student has a total of 7 grace days to extend the coursework deadlines through the semester.

# Course information

- Expectations and marking (**grad students**)
  - Same as undergrads:
    - Assignments: 60%
    - **Final project: 40%**
- See Course Information handout for detailed policies
- Waitlists expire  $\approx$ 1 week after the course starts.
  - After that, students are responsible for trying to enroll in the course on ACORN in a first-come, first-serve fashion.

# The Good, the Bad and the Ugly

- This course will be “easy” but there are rules.
  - **Grade days:** every student has a total of 7 grace days to extend the coursework deadlines through the semester.
  - **Late penalty:** 10% per day up to 3 days. i.e. maximum of 100%, 90%, 80%, 70%, 0% within 72 hours after the deadline.
  - **Completion requirement:** a minimum of 3 out of 4 assignments must be submitted on time (with grace days) to pass the course.
  - **Collaboration:** individual work except for the final project.
- Our guarantees to you: this will be one of the most unique learning experiences at UofT. But, you will have to put in a lot of work.

# Resources and communication

- A lot of learning in this course happens outside of the classroom.
- We are fully committed to help you succeed:
  - **Piazza:** main platform for fast, asynchronous, collaborative learning.
  - **Support ticketing system:** default mailing list.
  - **Office hours:** in-person interaction.
  - **Video lectures:** catch up on the missing bits.

# Compute

- **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.
- **OpenAI API** Variety of language models, e.g., GPT-3, ChatGPT, you will encounter and try out in the course. Recommend register early and apply for Codex beta access.
- See Course Information handout for the details

# Course information

Course web page: <http://uoft-csc413.github.io/2023/>

Includes detailed course information handout:

<https://uoft-csc413.github.io/2023/assets/misc/syllabus.pdf>

# Hello world

- Final project: you must form a group of two or three.

# Hello world

- Final project: you must form a group of two or three.
- Now, everyone, please stand up.

# 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?

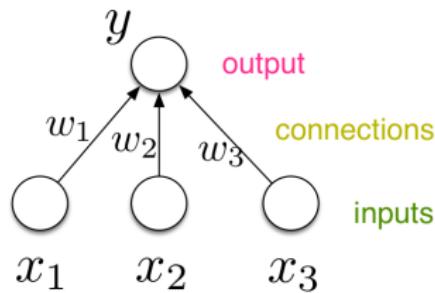
- For many problems, it's difficult to program the correct behavior by hand
  - recognizing people and objects
  - understanding human speech from audio files
- Machine learning approach: program an algorithm to automatically learn from data, or from experience
- Some reasons you might want to use a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

# What is machine learning?

- 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

# 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.



The equation for a single neuron's output is  $y = \phi(\mathbf{w}^\top \mathbf{x} + b)$ . Arrows point from the labels to the corresponding parts of the equation:

- "output" points to the variable  $y$ .
- "weights" points to the term  $\mathbf{w}^\top \mathbf{x}$ .
- "bias" points to the term  $b$ .
- "activation function" points to the symbol  $\phi$ .
- "inputs" points to the term  $\mathbf{x}$ .

- 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

# 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

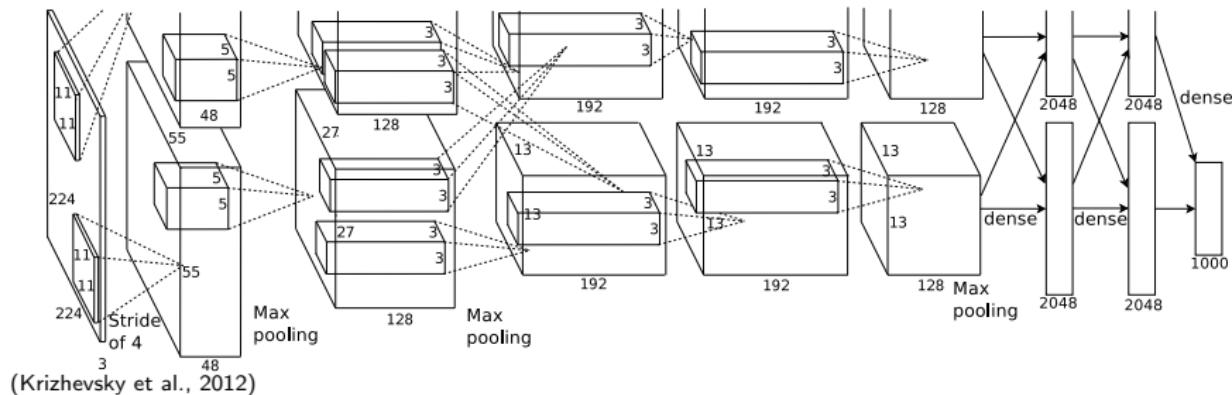
# What are neural networks?

- 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

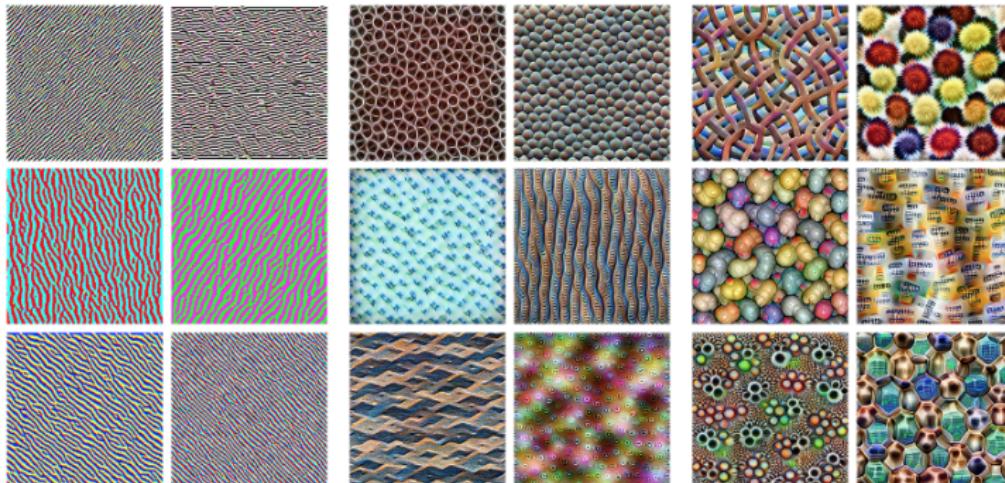
E.g. this network which recognizes objects in images:



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)

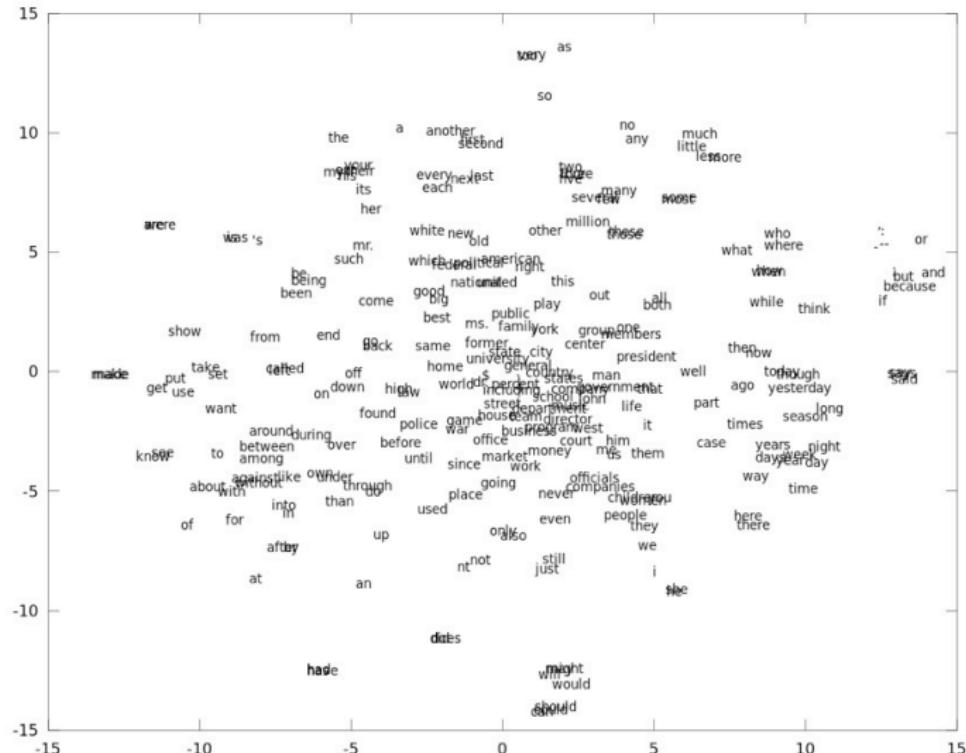
<https://distill.pub/2017/feature-visualization/>

# 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 \_\_\_\_\_”
    - $\text{France} - \text{Paris} + \text{London} = \underline{\hspace{2cm}}$

# 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, whether it's supervised, unsupervised, or reinforcement learning!

# 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

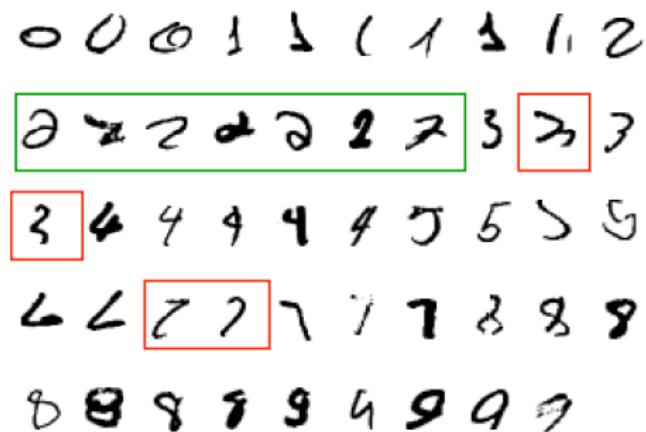
**Supervised learning:** have labeled examples of the correct behavior

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

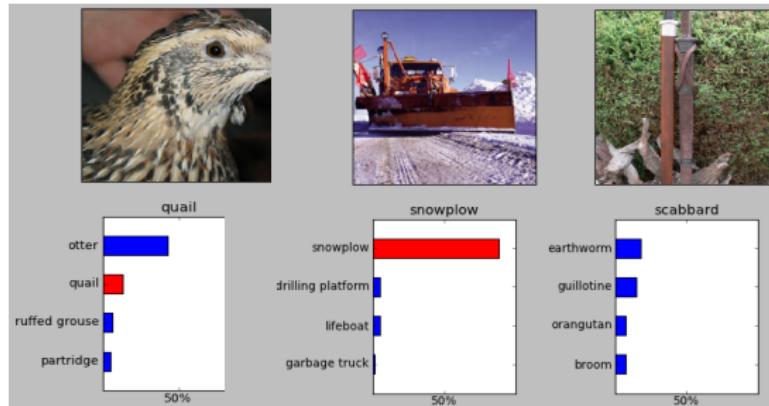
# Supervised learning examples

What makes a “2”?



# Supervised learning examples

## Object recognition

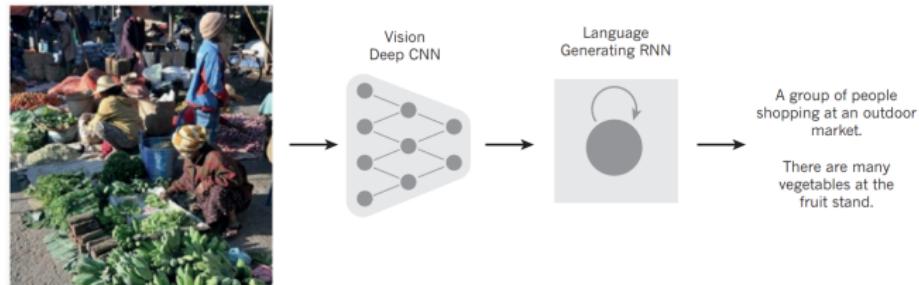


(Krizhevsky and Hinton, 2012)

ImageNet dataset: thousands of 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



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 mountain in the background

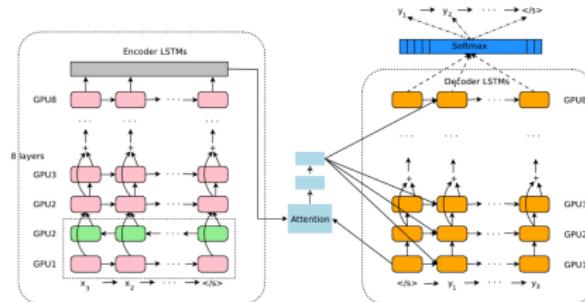
(Xu et al., 2015)

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)

<i>Input sentence:</i>	<i>Translation (PBMT):</i>	<i>Translation (GNMT):</i>	<i>Translation (human):</i>
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:

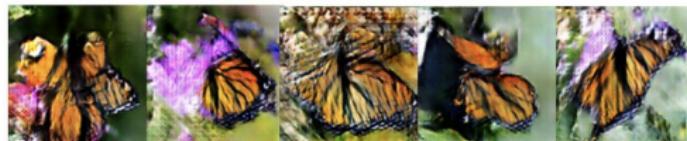


Denton et al., 2014, Deep generative image models using a Laplacian pyramid of adversarial networks

# Unsupervised learning examples

- The progress of generative models:

Odena et al  
2016



Miyato et al  
2017



Zhang et al  
2018



# Unsupervised learning examples

- The progress of generative models:
- Stable Diffusion, Robini et al, 2022:



# Unsupervised learning examples

- The progress of generative models:
- DreamStudio, Stability AI:



# 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 model learns from 40GB of Internet text.

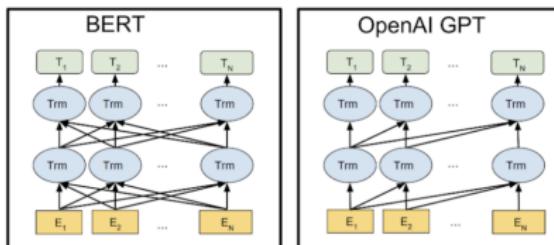
Custom prompt

I took CSC413 introduction to deep learning last year

**GENERATE ANOTHER**

## 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://takttotransformer.com/>

# Unsupervised learning examples

- The GPT-3 models now are learning from 2TB of Internet text.  
Similar models are helping with code generation.



## Playground

Load a preset...

Save

View code

Share

...

⚙

Write a short poem about introduction to deep learning and neural networks in lord's prayer style.

Our Father in deep learning,  
Hallowed be thy name.  
Thy algorithm come,  
Thy neural networks be done,  
On deep learning as it is in heaven.  
Give us this day our daily data,  
And forgive us our model errors,  
As we forgive those who train against us.  
And lead us not into overfitting,  
But deliver us from underfitting.  
For thine is the power,  
The accuracy, and the glory,  
Forever and ever. Amen.

write a matplotlib function to make a line plot on data=[32e3,254e5,255e7]

dates=["2018", "2020", "2022"] where the x axis label is replaced by strings in the date list. Make sure the y axis labels are displayed in millions.

Here's an example of a function that takes in a list of data points and a corresponding list of dates, and creates a line plot with the x-axis labels replaced by the date strings:

```
import matplotlib.pyplot as plt

def plot_line_data(data, dates):
    # Create a figure and axis
    fig, ax = plt.subplots()
    # Plot the data as a line
    ax.plot(data)
    # Set the x-axis labels to the date strings
    ax.set_xticks(range(len(dates)))
    ax.set_xticklabels(dates)
    # Use a ScalarFormatter to format the y-axis labels in millions
    ax.yaxis.set_major_formatter(plt.ScalarFormatter(useMathText=True))
    ax.yaxis.offsetText.set_fontsize(10)
    ax.yaxis.set_major_formatter(plt.FuncFormatter(lambda x, _:
        '{:.1e}'.format(x)))
    # Add axis labels
    ax.set_xlabel("Date")
    ax.set_ylabel("Data (millions)")

    # Show the plot
    plt.show()
```

# 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](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!

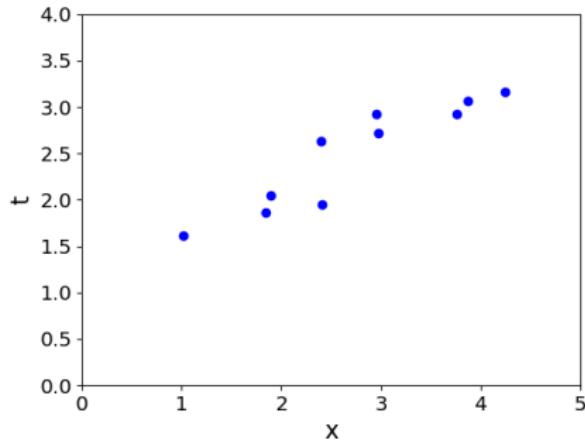
After break

## Linear models

# Overview

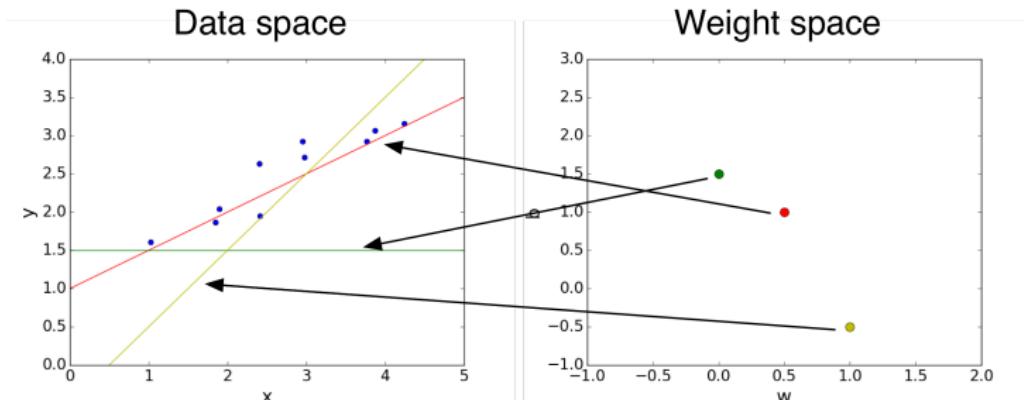
- One of the fundamental building blocks in deep learning are the **linear models**, where you decide based on a linear function of the input vector.
- Here, we will review linear models, some other fundamental concepts (e.g. gradient descent, generalization), and some of the common supervised learning 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  $x$
- Given a dataset of pairs  $\{(x^{(i)}, t^{(i)})\}_{i=1}^N$
- The  $x^{(i)}$  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

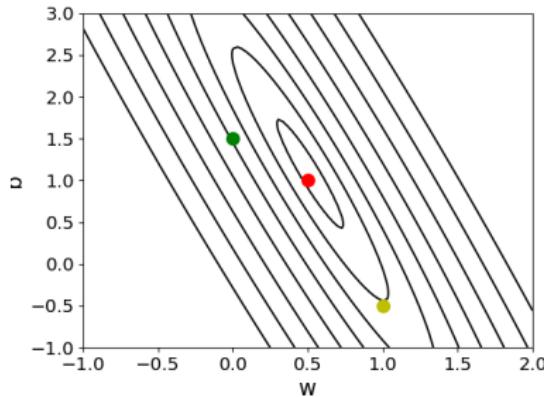
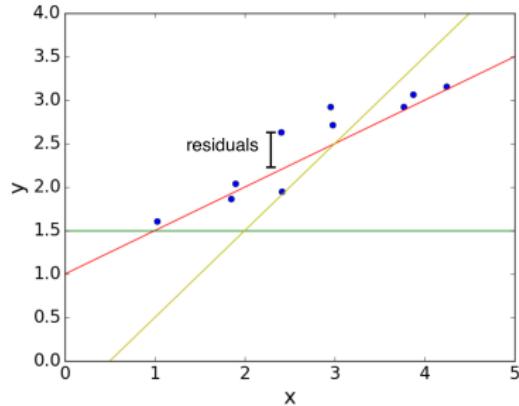
$$\mathcal{L}(y, t) = \frac{1}{2}(y - t)^2$$

- $y - t$  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

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

# 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

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

one training  
example (vector)

- Computing the predictions for the whole dataset:

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

# Vectorization

- Computing the squared error cost across the whole dataset:

$$y = Xw + b$$

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

- 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.

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

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

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

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

# Direct solution

- Chain rule for derivatives:

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

- 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  $\alpha$ :

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

- $\alpha$  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**:

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

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

# Gradient descent

- This gets its name from the **gradient**:

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

- This is the direction of fastest increase in  $\mathcal{J}$ .
- Update rule in vector form:

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

- 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](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.

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

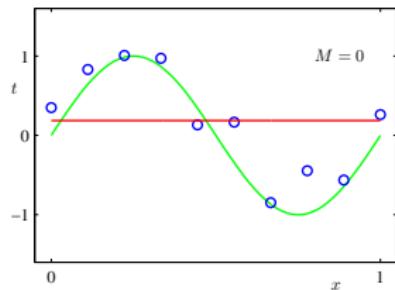
- E.g., if  $\psi(\mathbf{x}) = (1, \mathbf{x}, \dots, \mathbf{x}^D)^\top$ , then  $y$  is a polynomial in  $\mathbf{x}$ . This model is known as **polynomial regression**:

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

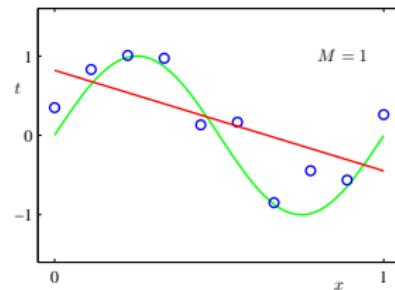
- This doesn't require changing the algorithm — just pretend  $\psi(\mathbf{x})$  is the input vector.
- We don't need an explicit bias term, since it can be absorbed into  $\psi$ .
- Feature maps let us fit nonlinear models, but it can be hard to choose good features.
  - Before deep learning, most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

# Feature maps

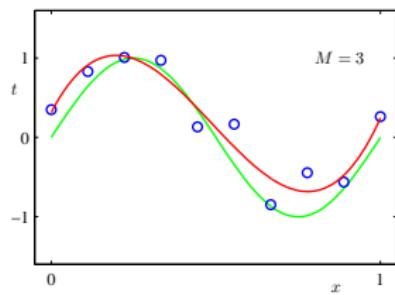
$$y = w_0$$



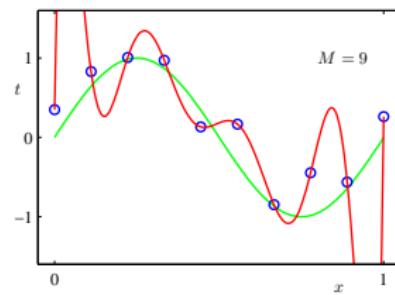
$$y = w_0 + w_1 x$$



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

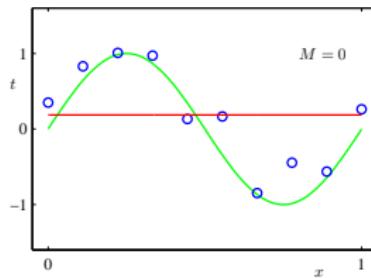


$$y = w_0 + w_1 x + \cdots + w_9 x^9$$

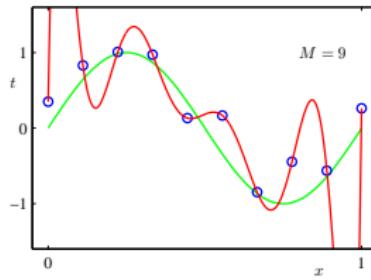


# Generalization

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

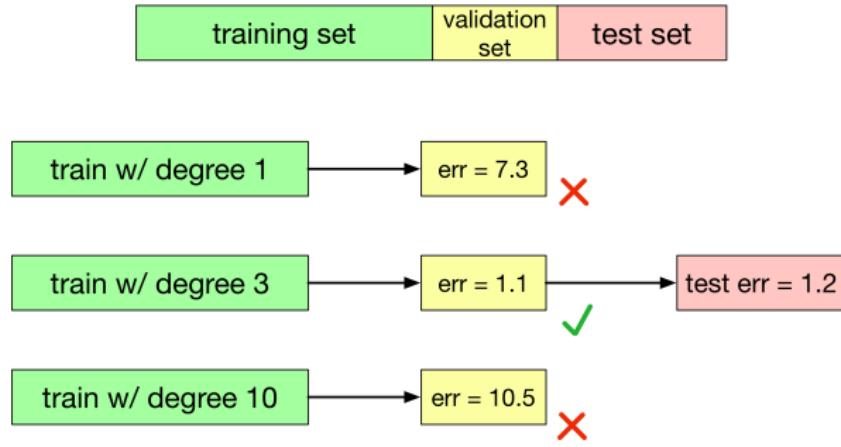


**Overfitting** : The model is too complex - fits perfectly, does not generalize.



# 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  $x$ , thresholded at zero:

$$z = w^T x + b$$

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

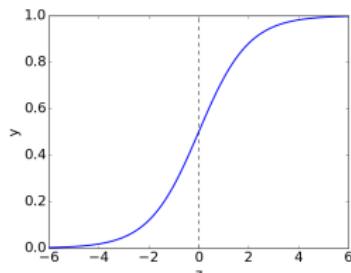
# Logistic Regression

- 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**:

$$z = w^\top x + b$$

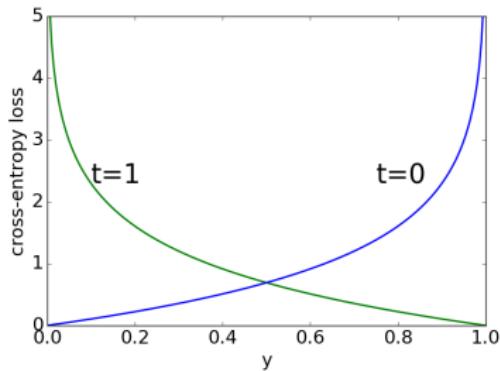
$$y = \sigma(z)$$

- Used in this way,  $\sigma$  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:

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



# Logistic Regression

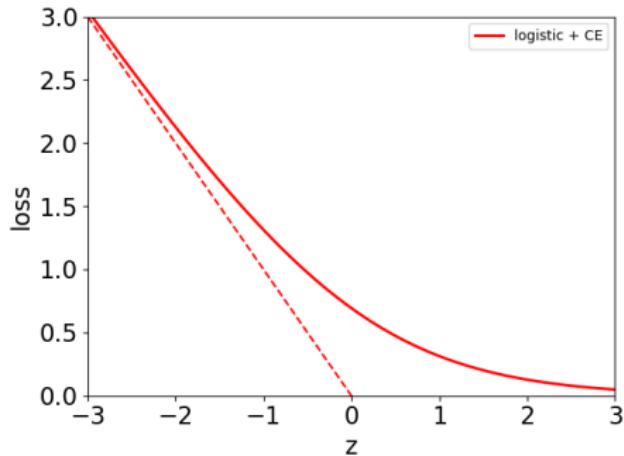
- Logistic regression combines the logistic activation function with cross-entropy loss.

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

$$y = \sigma(z)$$

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

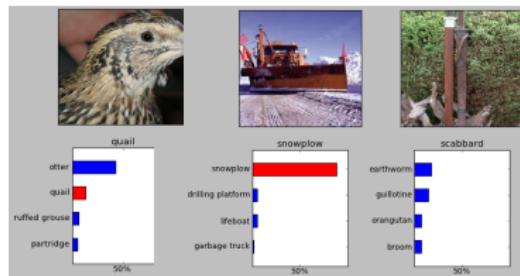
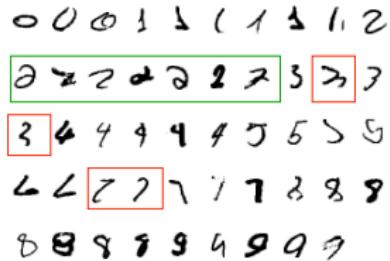
$$\mathcal{L}_{\text{CE}} = -t \log y - (1 - t) \log(1 - y)$$



- 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  $\{1, \dots, K\}$ .
- It's often more convenient to represent them as **one-hot vectors**, or a **one-of-K encoding**:

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

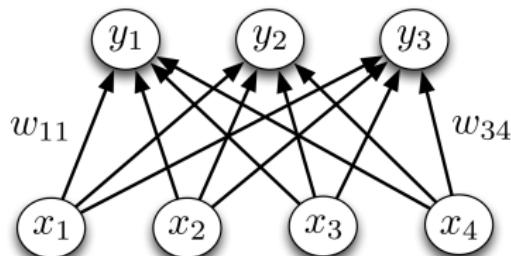
# Multiclass Classification

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

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

- 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,  $\text{softmax}(z)$  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  $\sigma(z)$  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:

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

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**:

$$z = Wx + b$$

$$y = \text{softmax}(z)$$

$$\mathcal{L}_{\text{CE}} = -t^\top (\log y)$$

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

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