LMP 1210H: Basic Principles of Machine Learning in Biomedical Research

Bo Wang
AI Lead Scientist, PMCC, UHN
CIFAR AI Chair, Vector Institute
Assistant Professor, University of Toronto

Administrative Stuff

1. Homework 2 is out! Due Feb 20!
   Reminder : lots of work! Start early!

2. Project Handout is out!

   - Proposal: Due Feb 21, 11:59pm
   - Presentation: Due March 28 or April 4, in-class
   - Final Report: Due April 10, 11:59pm

More on the Final Project

1. Group size: 2 vs 3

Trade-off between bonus and workload

2. Project proposal

Proposal: The project proposal is limited to two pages. It should roughly have the following sections:

- 1/4 page introduction
- 1/2 page related works
- 1/2 page method / algorithm
- 1/4 page abstract and reference

The point of the proposal is mainly for us to give you feedback and formulate a plan for the final report. The proposal will not be graded. We will set up project consultation appointments after we have collected all the project proposals. You will submit your proposal report through Quercus. **Note: Groups without proposal submissions cannot proceed with the final presentations and reports!**

More on the Final Project

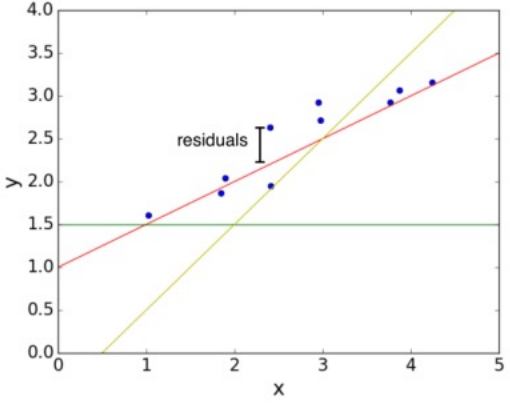
3. Project presentation

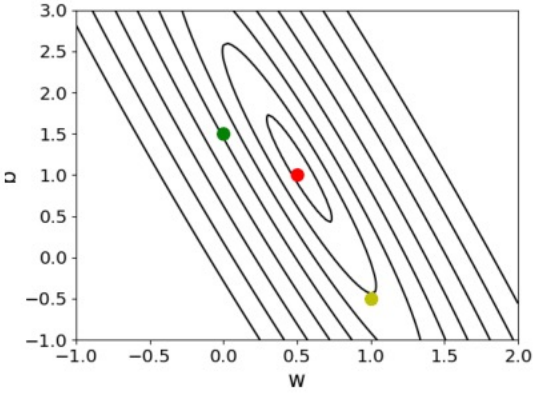
The slide presentation should be short, maximum 10 min. Presentations that are longer will be penalized, with a hard cutoff after 11 min. A good rule-of-thumb is one slide per minute (excluding title and reference slides). Effective presentations make use of large figures and minimal text.

4. Peer Review

- **Participation [10%] (We are counting on you!)** We will adopt a peer-review system through Quercus in which students will participate in reviewing the other classmates' reports. Reports will be made anonymously. We expect each student to review at least 2 reports. The participation score will be given based on the quality of the reviews by each student. You may find this guide helpful regarding how to write a good review. For participating in the review process, you will be awarded 10%.

Recap: Linear Classification and Gradient Descent





- Advantages: Easy to understand and implement; Widely-adopted;

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:

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

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

Logistic Regression

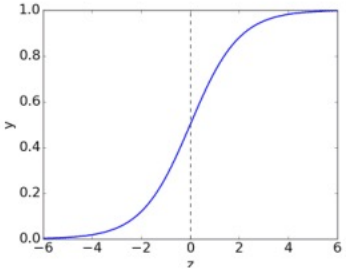
- We can't optimize classification accuracy directly with gradient descent because it's discontinuous.
- Instead, we typically define a continuous surrogate loss function which is easier to optimize. Logistic regression is a canonical example of this, in the context of classification.
- The model outputs 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:

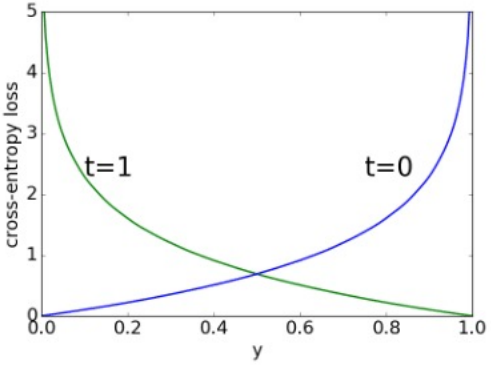
$$z = \mathbf{w}^\top \mathbf{x} + b$$
$$y = \sigma(z)$$

- 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:

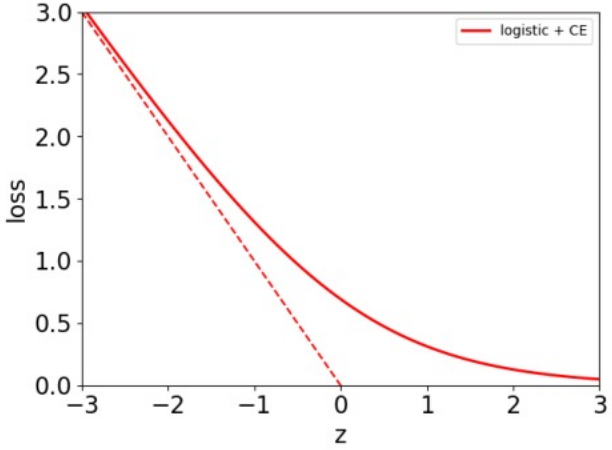
$$\begin{aligned}\mathcal{L}_{\text{CE}}(y, t) &= \begin{cases} -\log y & \text{if } t = 1 \\ -\log(1 - y) & \text{if } t = 0 \end{cases} \\ &= -t \log y - (1 - t) \log(1 - y)\end{aligned}$$



Logistic Regression

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

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



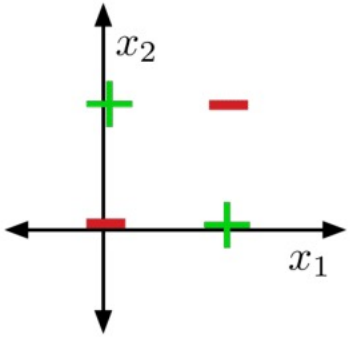
- Interestingly, the loss asymptotes to a linear function of the logit z.
- Full derivation in the readings.

Linear Regression v.s. Logistic Regression

Linear Regression	Logistic Regression
mostly used for continuous regression.	mostly used for binary classification.
Loss function: square error	Loss function: cross entropy
Optimization: gradient descent or closed form	Optimization: gradient descent
Output is linear in inputs	Output is *not* linear in inputs

Limits of Linear Classification

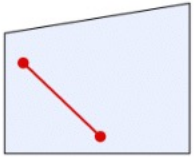
- Single neurons (linear classifiers) are very limited in expressive power.
- **XOR** is a classic example of a function that's not linearly separable.



- There's an elegant proof using convexity.

Limits of Linear Classification

Convex Sets



- A set $\mathcal{S}$ is convex if any line segment connecting points in $\mathcal{S}$ lies entirely within $\mathcal{S}$. Mathematically,

$$\mathbf{x}_1, \mathbf{x}_2 \in \mathcal{S} \quad \Longrightarrow \quad \lambda \mathbf{x}_1 + (1-\lambda)\mathbf{x}_2 \in \mathcal{S} \quad \text{for } 0 \leq \lambda \leq 1.$$

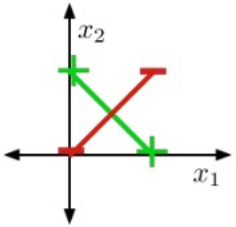
- A simple inductive argument shows that for $\mathbf{x}_1, \ldots, \mathbf{x}_N \in \mathcal{S}$, weighted averages, or convex combinations, lie within the set:

$$\lambda_1 \mathbf{x}_1 + \cdots + \lambda_N \mathbf{x}_N \in \mathcal{S} \quad \text{for } \lambda_i > 0,\ \lambda_1 + \cdots \lambda_N = 1.$$

Limits of Linear Classification

Showing that XOR is not linearly separable

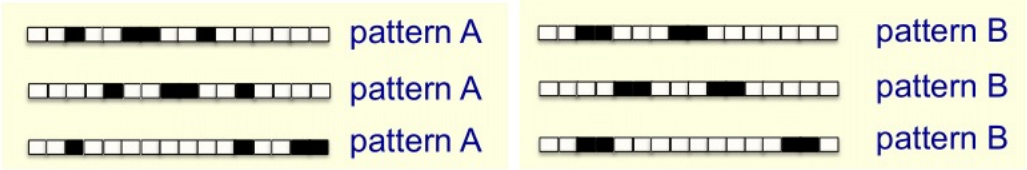
- Half-spaces are obviously convex.
- Suppose there were some feasible hypothesis. If the positive examples are in the positive half-space, then the green line segment must be as well.
- Similarly, the red line segment must line within the negative half-space.



- But the intersection can't lie in both half-spaces. Contradiction!

Limits of Linear Classification

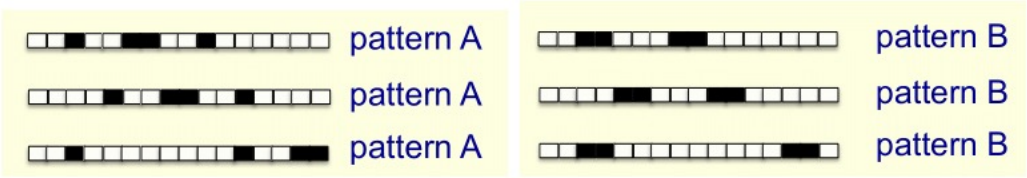
A more troubling example



- These images represent 16-dimensional vectors. White = 0, black = 1.
- Want to distinguish patterns A and B in all possible translations (with wrap-around)
- Translation invariance is commonly desired in vision!

Limits of Linear Classification

A more troubling example



- These images represent 16-dimensional vectors. White = 0, black = 1.
- Want to distinguish patterns A and B in all possible translations (with wrap-around)
- Translation invariance is commonly desired in vision!
- Suppose there's a feasible solution. The average of all translations of A is the vector $(0.25, 0.25, \ldots, 0.25)$. Therefore, this point must be classified as A.
- Similarly, the average of all translations of B is also $(0.25, 0.25, \ldots, 0.25)$. Therefore, it must be classified as B. Contradiction!

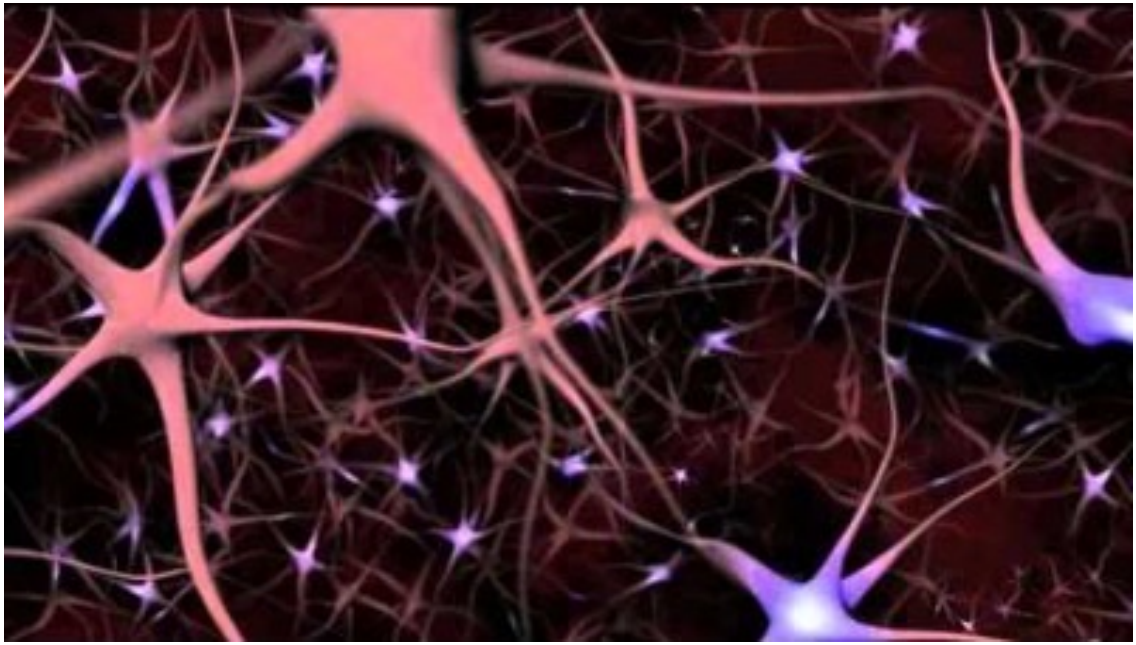
Limits of Linear Classification

- Sometimes we can overcome this limitation using feature maps, just like for linear regression. E.g., for **XOR**:

$$\boldsymbol{\psi}(\mathbf{x}) = \begin{pmatrix} x_1 \\ x_2 \\ x_1 x_2 \end{pmatrix}$$

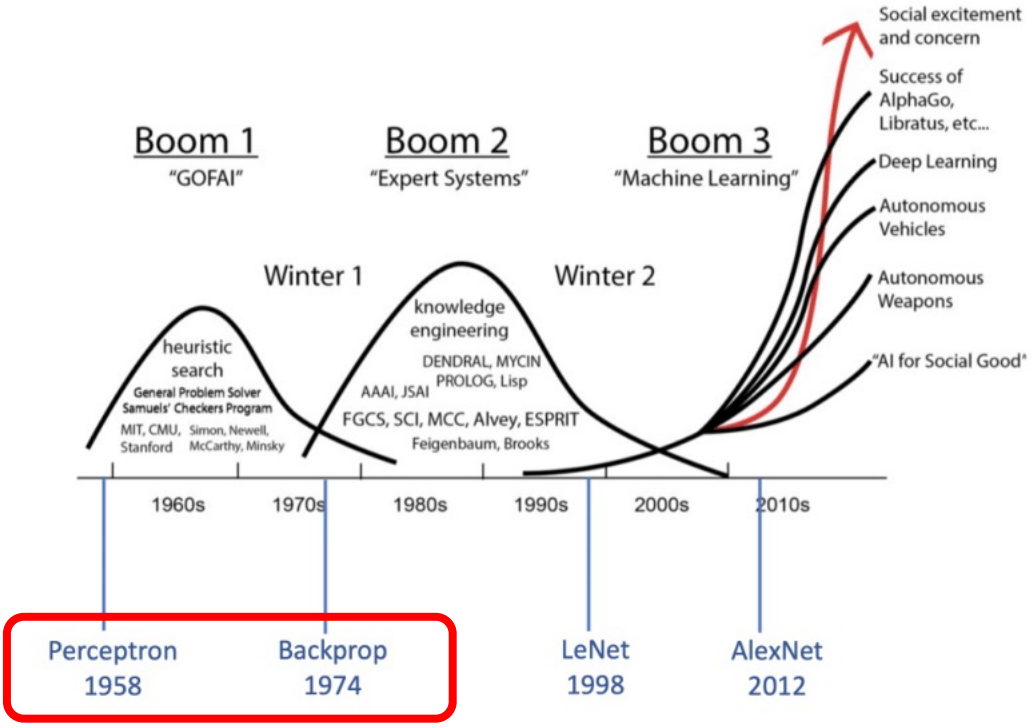
x_1	x_2	$\phi_1(\mathbf{x})$	$\phi_2(\mathbf{x})$	$\phi_3(\mathbf{x})$	t
0	0	0	0	0	0
0	1	0	1	0	1
1	0	1	0	0	1
1	1	1	1	1	0

- This is linearly separable. (Try it!)
- Not a general solution: it can be hard to pick good basis functions. Instead, we'll use neural nets to learn nonlinear hypotheses directly.



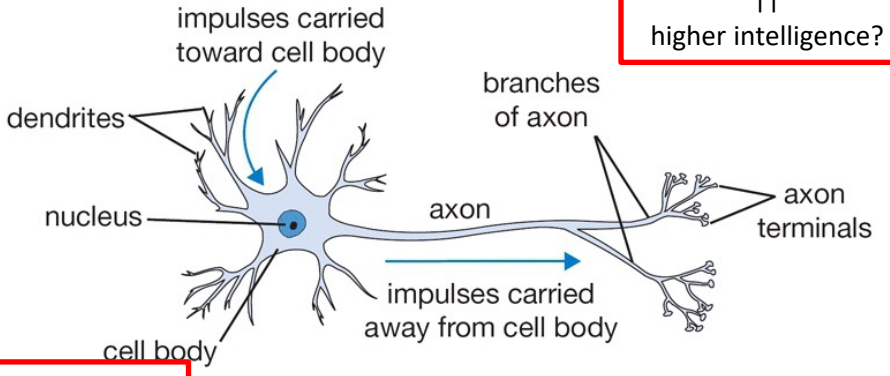
Source: https://www.youtube.com/watch?v=vyNkAuX29OU

A brief history



The Biological Motivation

more neurons
||
higher intelligence?



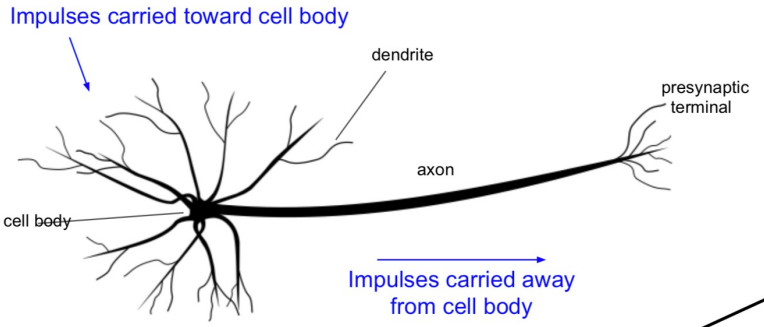
Some fun facts :

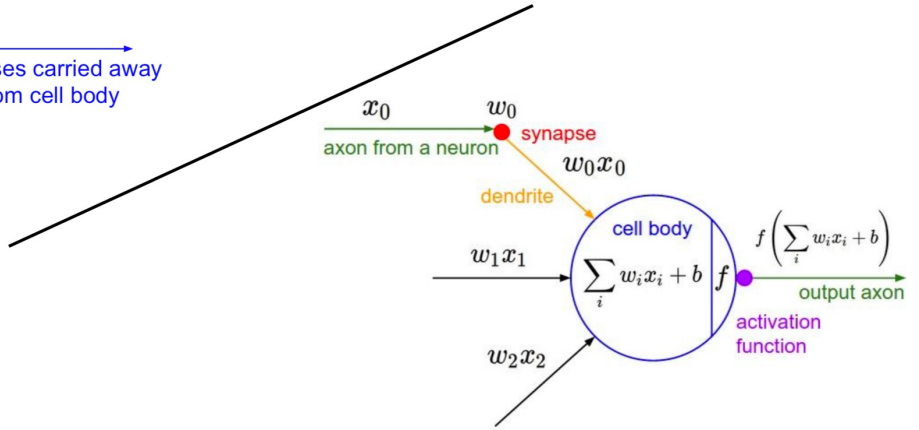
1 million x

100 billion neurons

100,000 neurons

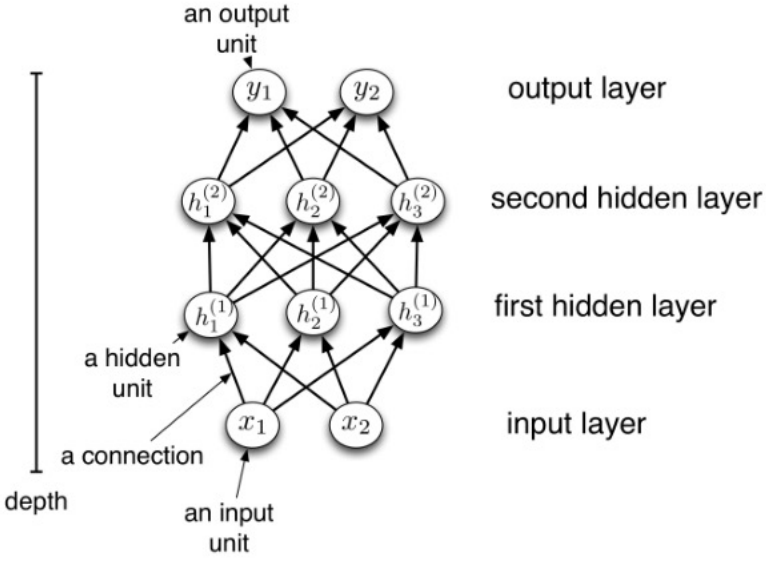
The Biological Motivation





Multilayer Perceptrons

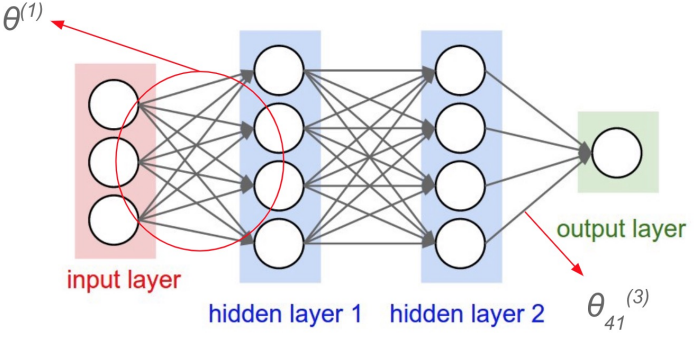
- We can connect lots of units together into a directed acyclic graph.
- This gives a feed-forward neural network. That's in contrast to recurrent neural networks, which can have cycles. (We'll talk about those later.)
- Typically, units are grouped together into layers.



Basic Components of MLP

What are we trying to learn? **Weights.**

- The value of each synaptic connection between neurons - **weights**.
- Let us denote the weight matrix of layer ***k*** as $\theta^{(k)}$ and the weight connecting the ***i-th*** neuron in layer ***k*** to the **j**-th neuron is layer ***k+1*** as $\theta_{ij}^{(k)}$



Basic Components of MLP

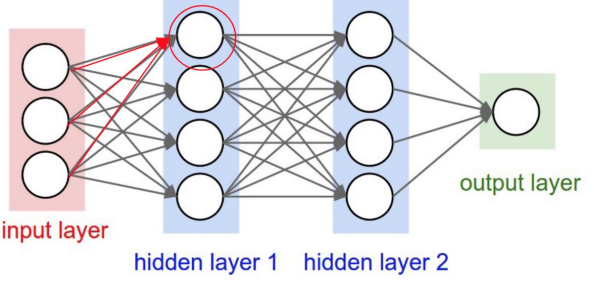
Forward Propagation

- **Goal:** Given input vector $\boldsymbol{x}$, we want to yield a mapping $\boldsymbol{f_\theta(x)}$.
- Propagate $\boldsymbol{x}$ through first set of weights $\theta^{(1)}$.
- Denote pre-activation values of layer $\boldsymbol{i}$ as $\boldsymbol{h^{(i)}}$.

Ex. Calculate the pre-activation value of $h_1^{(2)}$.

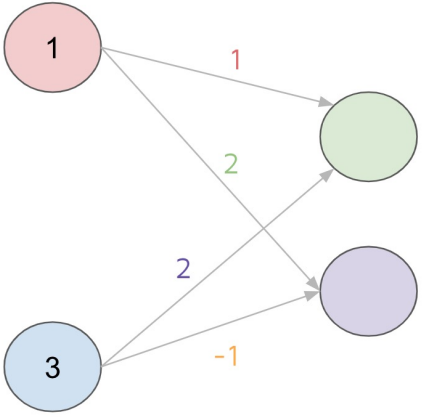
$$\boldsymbol{h_j^{(m)} = \Sigma_i \theta_{ij}^{(m-1)} x_i}$$

$$h_1^{(2)} = \theta_{11}^{(1)} x_1 + \theta_{21}^{(1)} x_2 + \theta_{31}^{(1)} x_3$$



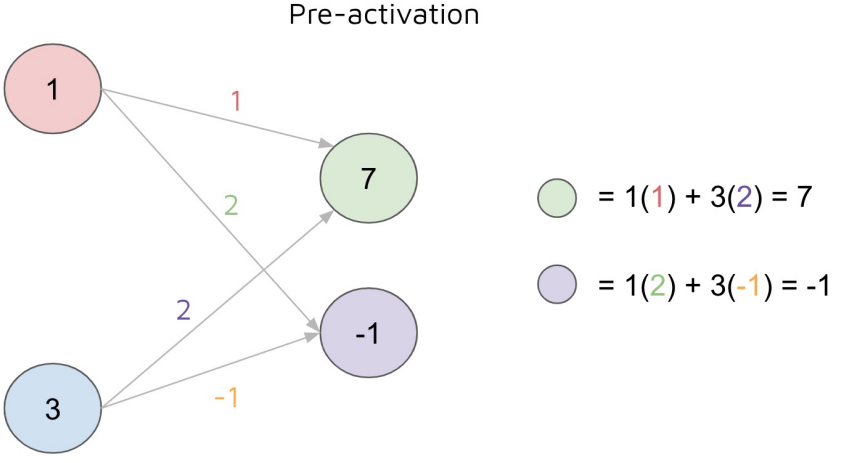
Basic Components of MLP

Example :What is the value of the green and purple neuron for this set of inputs and weights?



Basic Components of MLP

Example :What is the value of the green and purple neuron for this set of inputs and weights?



Basic Components of MLP

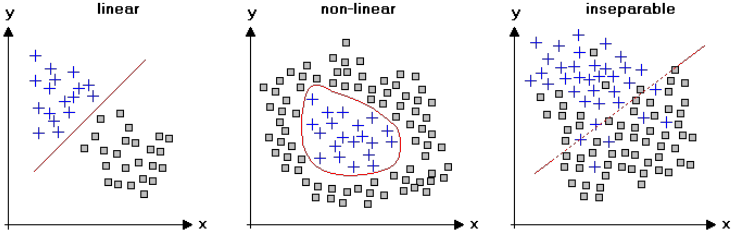
Activation Functions

- Successive weight matrices multiplied by the input **x** would just be a **linear transformation**.

- Non-linear activation functions allow us to learn **non-linearly separable** mappings.
  - Most data is **non-linearly separable**.

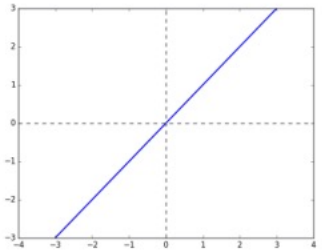
- Activation functions also allow us to calculate **gradients** used for optimizing the weight values.

- Let us denote the activation function as $\boldsymbol{a(x)} : \mathbb{R} \rightarrow \mathbb{R}$.



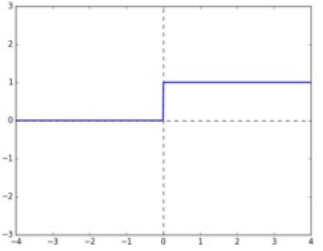
Basic Components of MLP

Some activation functions:



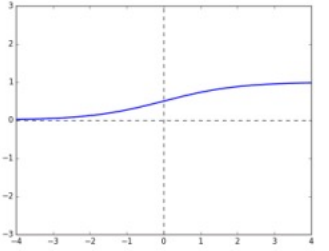
Linear

$$y = z$$



Hard Threshold

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

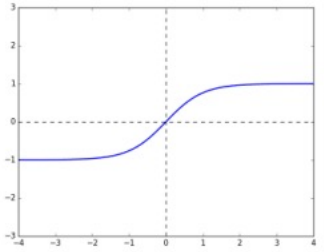


Logistic

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

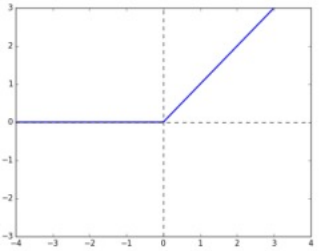
Basic Components of MLP

Some activation functions:



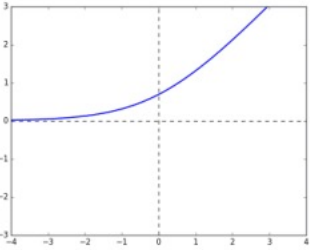
Hyperbolic Tangent (tanh)

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



Rectified Linear Unit (ReLU)

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



Soft ReLU

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

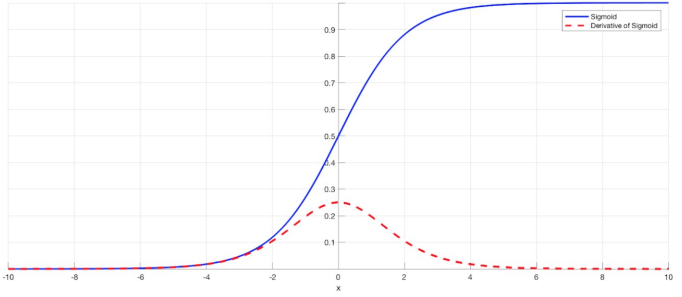
Basic Components of MLP

Activation Functions Sigmoid

$$a(x) = \text{Sigmoid}(x) = 1/(1+e^{-x})$$

$$a'(x) = e^{-x}/(1+e^{-x})^2$$

$$a'(x) = a(x)(1-a(x))$$



Activation Functions Sigmoid

Problems:

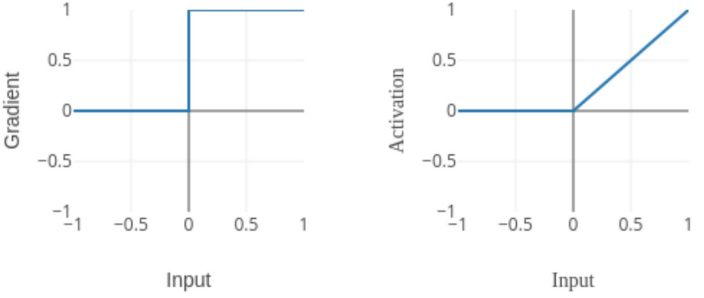
- **Vanishing Gradients:** The **gradients** approach zero resulting in almost no update to the **weights**.
  - Sigmoid squishes values to (0, 1).
- Gradients of larger and smaller x values approach zero.
  - No updates to those weights.
- Not used as the “go-to” activation function anymore.
  - Only used for cases where values should be scaled to (0, 1).

Basic Components of MLP

Activation Functions Rectified Linear Units (ReLU)

$$a(x) = \text{ReLU}(x) = \max\{0, x\}$$

$$a'(x) = \mathbb{1}_{x>0}$$



Basic Components of MLP

Activation Functions Rectified Linear Units (ReLU)

Pros

- Works well empirically.
  - More discriminability power as values no longer compressed in (0, 1).
  - Go-to activation function for almost all tasks.
- Fixes vanishing gradient issue sigmoid had.

Problems

- **Dying ReLU problem:** only output 0 for all input.
  - No activation = no classification.
  - The gradient is zero too so mathematically cannot update weights.

Forward Propagation Activating Features

- Apply an activation function *a(x)* on each $h_j^{(m)}$.
  - Analogous to a neuron firing.
  - Denote the **activation value** of the *j*-th neuron in layer *m* as $a_j^{(m)}$
- **Intuition:** Whether a neuron fires or not and the magnitude of its activation value is very useful in piecing together useful features for accomplishing the task.
  - **Extension:** unuseful features should be *zeroed out* by the activation function.

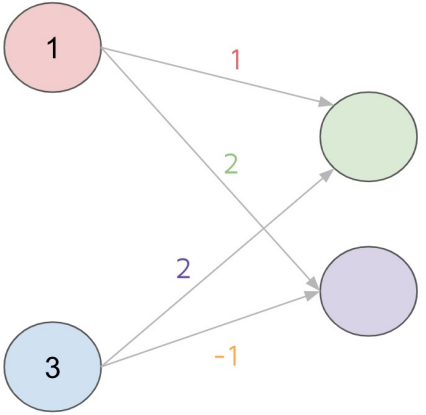
$$h_j^{(m)} = \Sigma_i \theta_{ij}^{(m-1)} x_i$$

$$a(h_j^{(m)}) = a(\Sigma_i \theta_{ij}^{(m-1)} x_i)$$

$$a_j^{(m)} = a(\Sigma_i \theta_{ij}^{(m-1)} x_i)$$

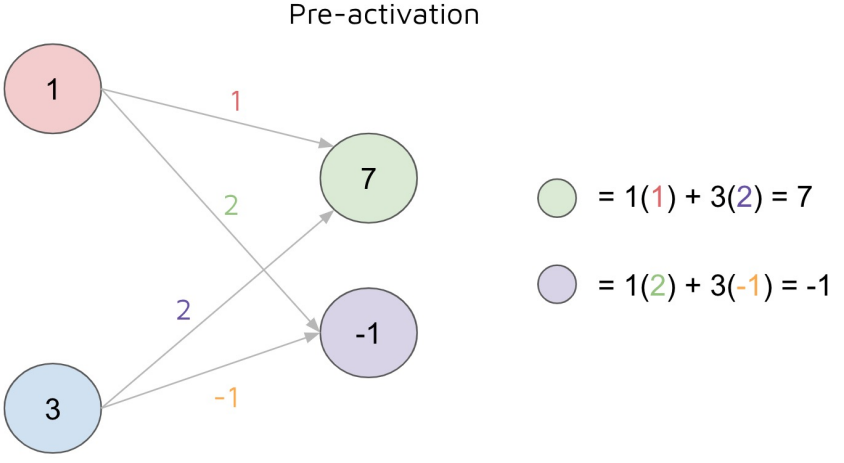
Basic Components of MLP

Example: What is the value of the green and purple neuron for this set of inputs and weights pre-activation? Post-activation?



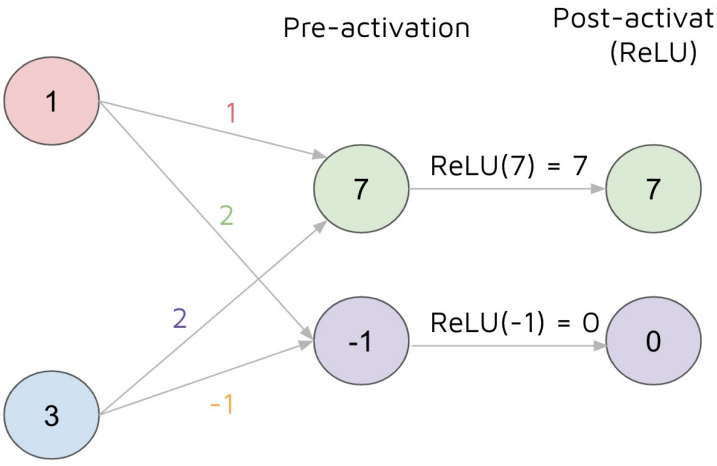
Basic Components of MLP

Example: What is the value of the green and purple neuron for this set of inputs and weights pre-activation? Post-activation?



Basic Components of MLP

Example: What is the value of the green and purple neuron for this set of inputs and weights pre-activation? Post-activation?



Recall: **ReLU(x) = max{0, x}**

Basic Components of MLP

Loss Functions Measuring Error

- Metric of how wrong the model performed.
- Ground truth label is given in supervised learning problems.
- Compares model output with the ground truth label so that *learning* can occur.
- **Goal of machine learning:** minimize the loss function.

Basic Components of MLP

Loss Functions Mean Squared Error

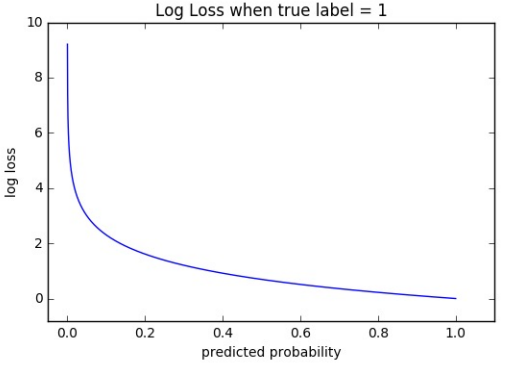
- Measure the variance of model output against target.
- t_i is ground truth label and y_i is predicted label.

$$J(\theta) = \frac{1}{m}\sum_{i}^{m}(y_i - t_i)^2$$

Loss Functions Cross Entropy Loss

- Measures the error of a model given the output is between [0, 1].
- Stronger gradients as loss diverges as the predicted probability diverges from the actual label.
- t_i is ground truth label {0, 1}, and y_i is predicted probability [0,1].

$$J(\theta) = -\sum_i t_i log(y_i) + (1 - t_i) log(1 - y_i)$$



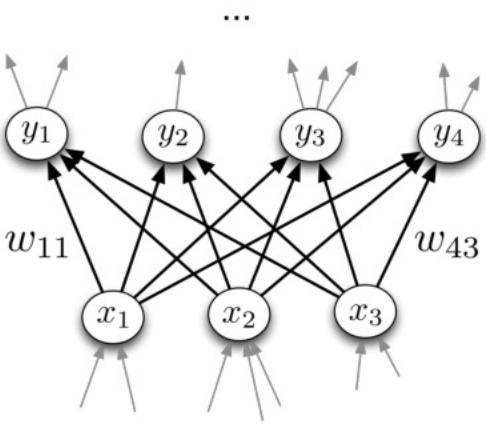
Multilayer Perceptrons

- Each layer connects N input units to M output units.
- In the simplest case, all input units are connected to all output units. We call this a fully connected layer. We'll consider other layer types later.
- Note: the inputs and outputs for a layer are distinct from the inputs and outputs to the network.

- Recall from softmax regression: this means we need an $M \times N$ weight matrix.
- The output units are a function of the input units:

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

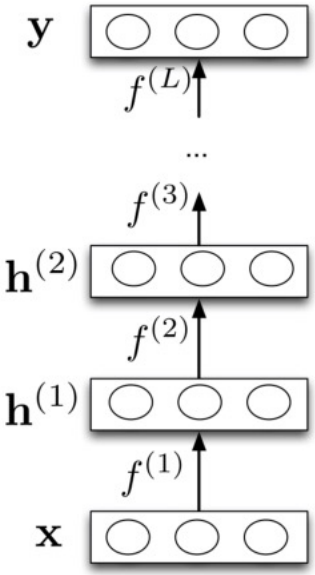
- A multilayer network consisting of fully connected layers is called a multilayer perceptron. Despite the name, it has nothing to do with perceptrons!



Multilayer Perceptrons

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

$$\mathbf{h}^{(1)} = f^{(1)}(\mathbf{x})$$
$$\mathbf{h}^{(2)} = f^{(2)}(\mathbf{h}^{(1)})$$
$$\vdots$$
$$\mathbf{y} = f^{(L)}(\mathbf{h}^{(L-1)})$$



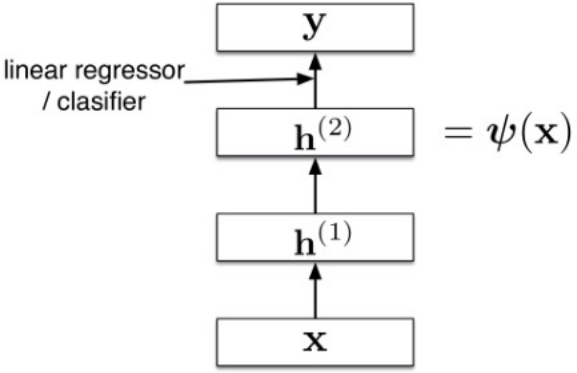
- Or more simply:

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

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

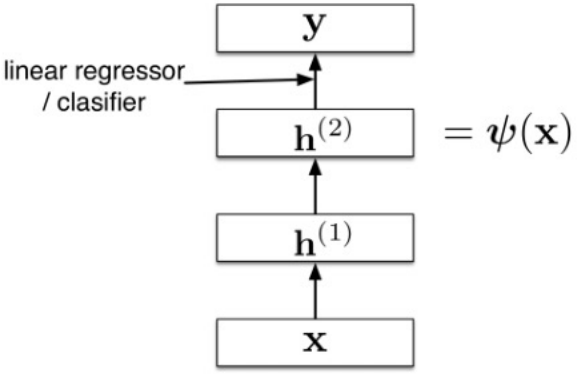
Feature Learning

- Neural nets can be viewed as a way of learning features:

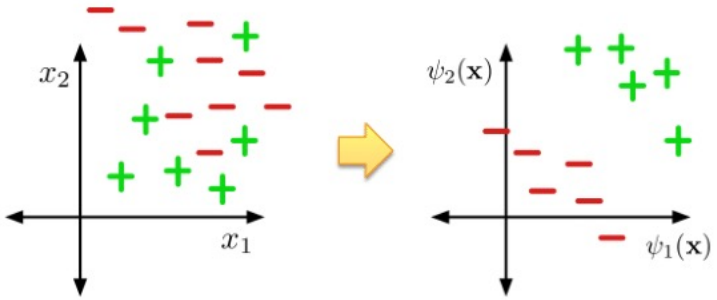


Feature Learning

- Neural nets can be viewed as a way of learning features:



- The goal:



Expressive Power

- We've seen that there are some functions that linear classifiers can't represent. Are deep networks any better?
- 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 regression!
  - Linear layers do have their uses — stay tuned!

https://arxiv.org/pdf/1610.00291.pdf

Expressive Power

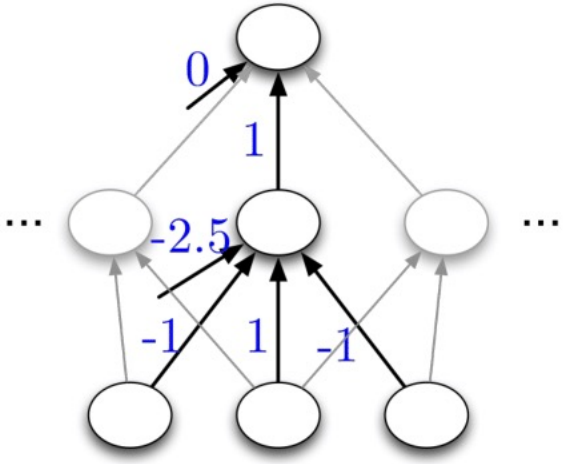
- Multilayer feed-forward neural nets with *nonlinear* activation functions are universal approximators: they can approximate any function arbitrarily well.
- This has been shown for various activation functions (thresholds, logistic, ReLU, etc.)
  - Even though ReLU is "almost" linear, it's nonlinear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

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

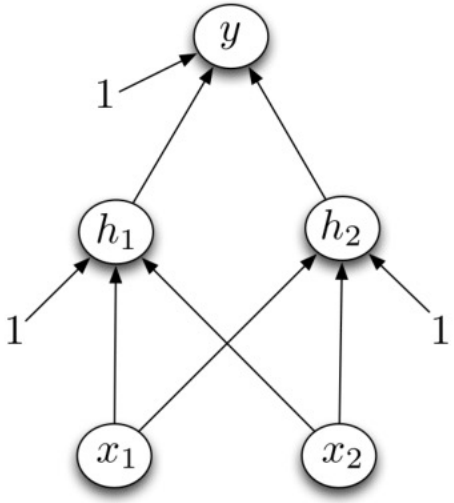
x_1	x_2	x_3	t
	$\vdots$		$\vdots$
-1	-1	1	-1
-1	1	-1	1
-1	1	1	1
	$\vdots$		$\vdots$



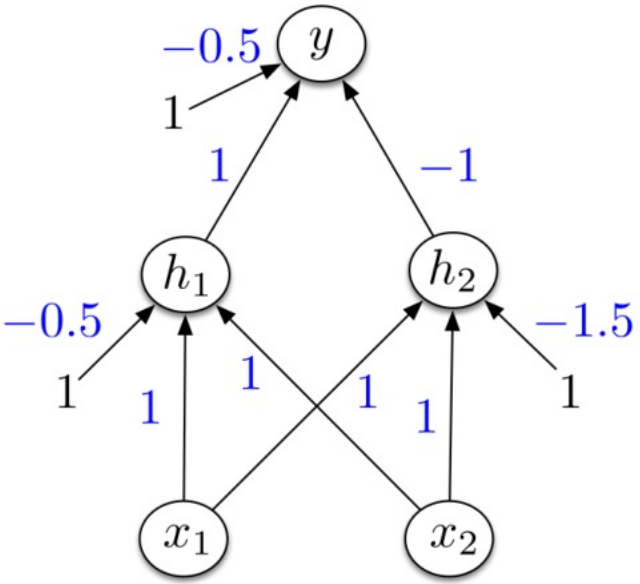
- Only requires one hidden layer, though it needs to be extremely wide!

Designing a network to compute XOR:

Assume hard threshold activation function



Multilayer Perceptrons



Exercise: Could you come up with another set of weights to compute XOR?

Expressive Power

- Limits of universality
  - You may need to represent an exponentially large network.
  - If you can learn any function, you'll just overfit.
  - Really, we desire a *compact* representation!

Expressive Power

- Limits of universality
  - You may need to represent an exponentially large network.
  - If you can learn any function, you'll just overfit.
  - Really, we desire a *compact* representation!
- We've derived units which compute the functions AND, OR, and NOT. Therefore, any Boolean circuit can be translated into a feed-forward neural net.
  - This suggests you might be able to learn *compact* representations of some complicated functions

After the break

After the break: **Backpropagation**

After the break Back-Propagation

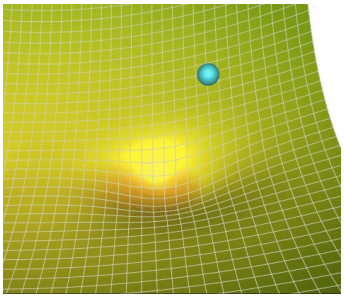
Source: https://www.youtube.com/watch?v=Suevq-kZdIw

Overview

- We've seen that multilayer neural networks are powerful. But how can we actually learn them?
- Backpropagation is the central algorithm in this course.
  - It's is an algorithm for computing gradients.
  - Really it's an instance of **reverse mode automatic differentiation**, which is much more broadly applicable than just neural nets.
    - This is "just" a clever and efficient use of the Chain Rule for derivatives.
    - We'll see how to implement an automatic differentiation system next week.

Recap: Gradient Descent

- **Recall:** gradient descent moves opposite the gradient (the direction of steepest descent)



- 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've seen so far — just higher dimensional and harder to visualize!
- We want to compute the cost gradient $\mathrm{d}\mathcal{J}/\mathrm{d}\mathbf{w}$, 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 $\mathrm{d}\mathcal{L}/\mathrm{d}\mathbf{w}$.

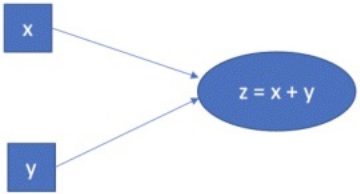
Recap : Univariate Chain Rule

- We've already been using the univariate Chain Rule.
- Recall: if $f(x)$ and $x(t)$ are univariate functions, then

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

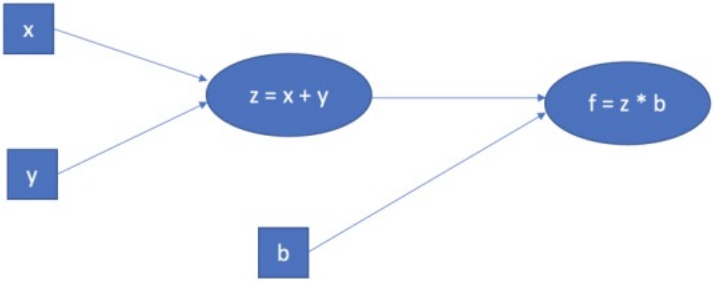
Recap: Computation Graph

- A computational graph is a directed graph where the **nodes** correspond to **operations** or **variables**.
- Variables can feed their value into operations, and operations can feed their output into other operations. This way, every node in the graph defines a function of the variables.
- For example : we want to plot the operation $z = x + y$, then



Recap: Computation Graph

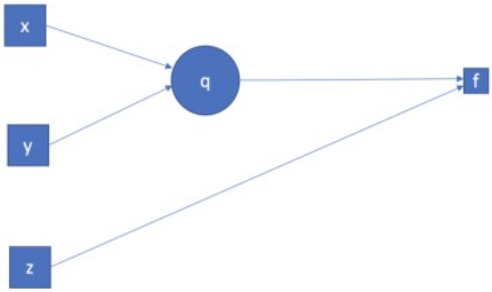
- A computational graph is a directed graph where the **nodes** correspond to **operations** or **variables**.
- Variables can feed their value into operations, and operations can feed their output into other operations. This way, every node in the graph defines a function of the variables.
- Another example : we want to plot the operation $f = (x + y) * b$, then



A simple example

$$f(x, y, z) = (x + y) * z$$

$$q = x + y; f = q * z$$



A simple example : Forward Pass

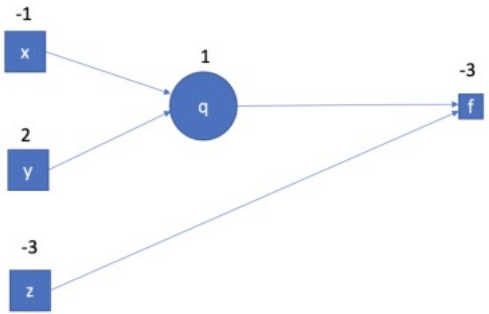
$$f(x, y, z) = (x + y) * z$$

$$q = x + y; f = q * z$$

$$e.g., x = -1, y = 2, z = 3$$

$$then, q = 1, f = -3$$

$$Want, \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$$



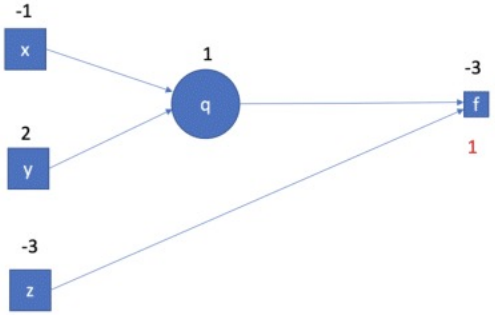
A simple example : Backward Pass

$$f(x, y, z) = (x + y) * z$$

$$q = x + y; f = q * z$$

$$e.g., x = -1, y = 2, z = 3$$

$$\textit{baseline} : \frac{\partial f}{\partial f} = 1$$



A simple example : Backward Pass

$$f(x, y, z) = (x + y) * z$$

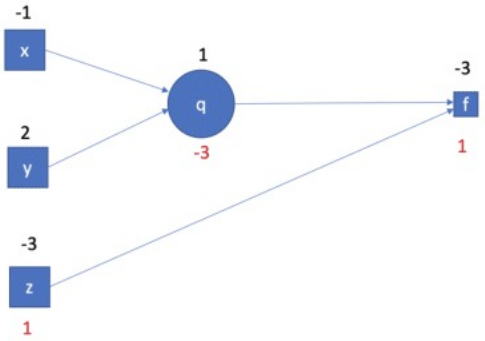
$$q = x + y; f = q * z$$

$$e.g., x = -1, y = 2, z = 3$$

$$\textit{baseline} : \frac{\partial f}{\partial f} = 1$$

$$\frac{\partial f}{\partial z} = \frac{\partial f}{\partial f}\frac{\partial f}{\partial z} = q = 1$$

$$\frac{\partial f}{\partial q} = \frac{\partial f}{\partial f}\frac{\partial f}{\partial q} = z = -3$$



A simple example : Backward Pass

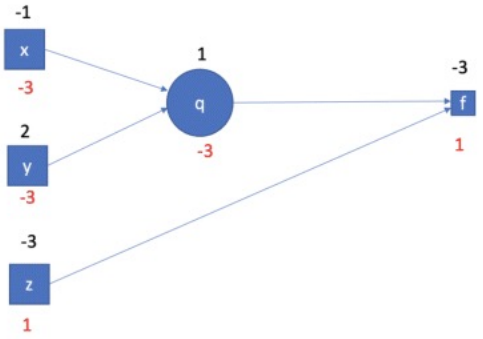
$$f(x, y, z) = (x + y) * z$$

$$q = x + y; f = q * z$$

$$e.g., x = -1, y = 2, z = 3$$

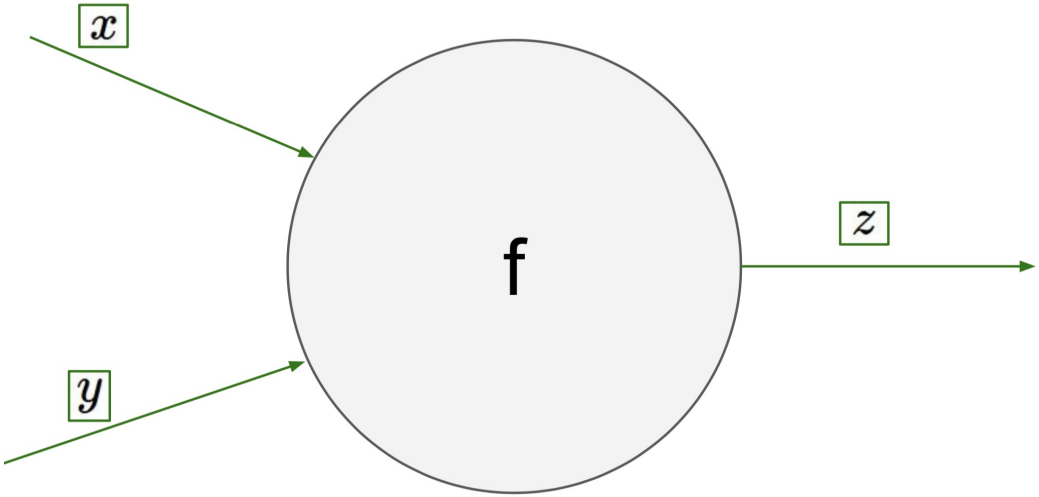
$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x} = (-3) * (1) = -3$$

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial y} = (-3) * (1) = -3$$



A simple example : Backward Pass

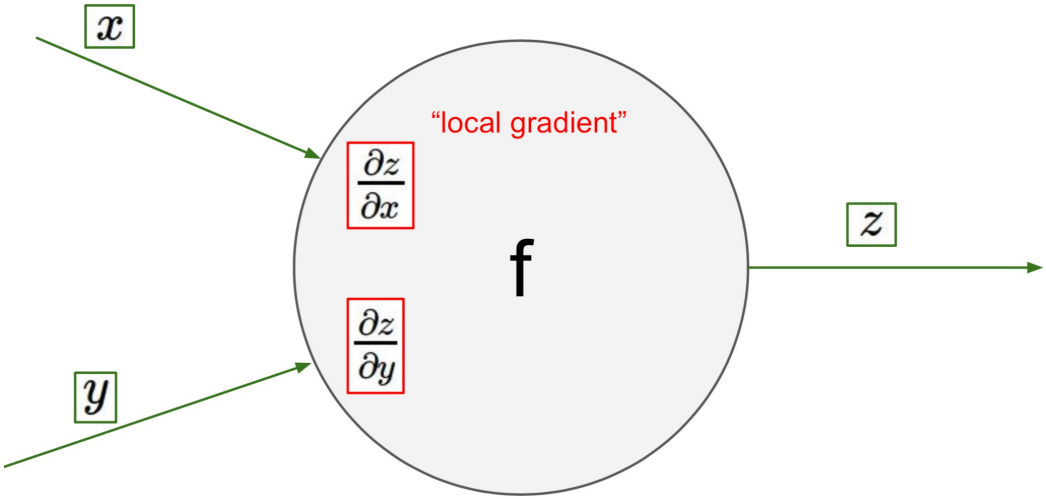
A quick summary:



Source: Fei-Fei Li & Justin Johnson & Serena Yeung, csc231N, Stanford University

A simple example : Backward Pass

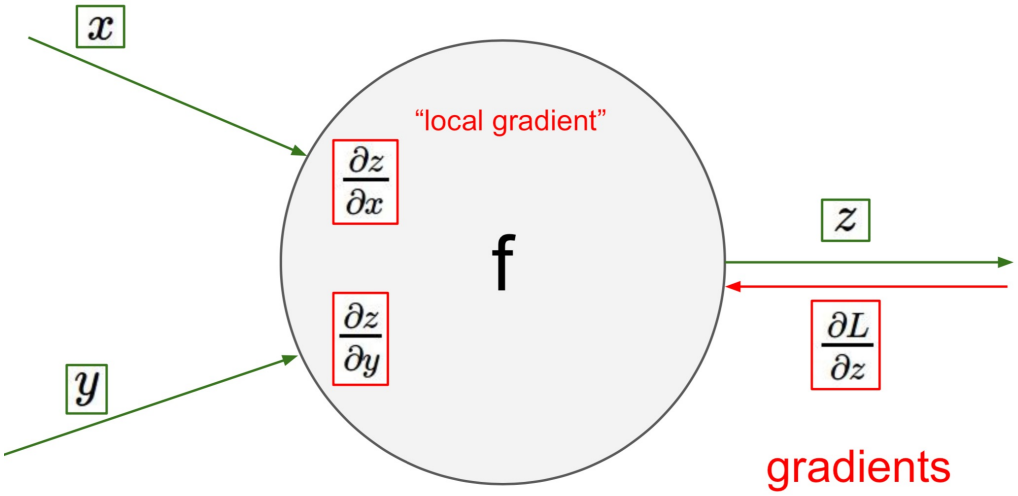
A quick summary:



Source: Fei-Fei Li & Justin Johnson & Serena Yeung, csc231N, Stanford University

A simple example : Backward Pass

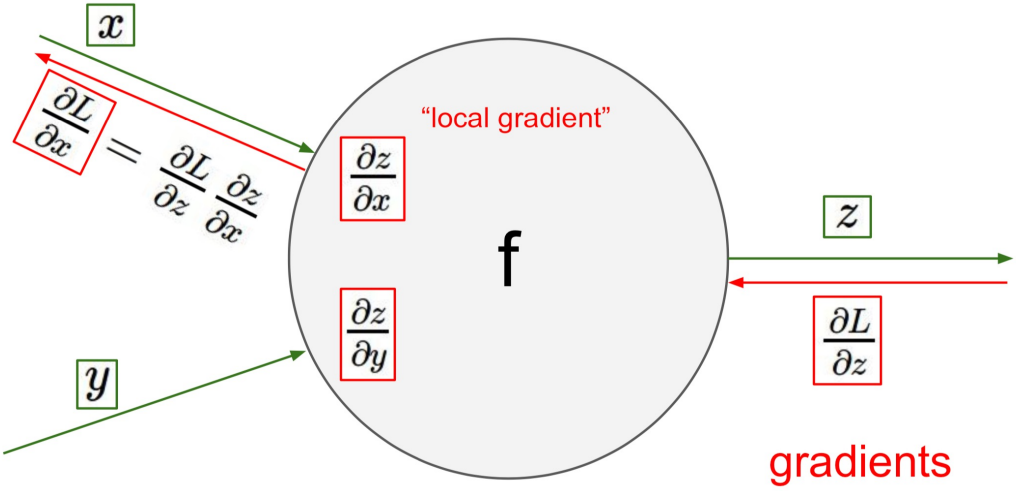
A quick summary:



Source: Fei-Fei Li & Justin Johnson & Serena Yeung, csc231N, Stanford University

A simple example : Backward Pass

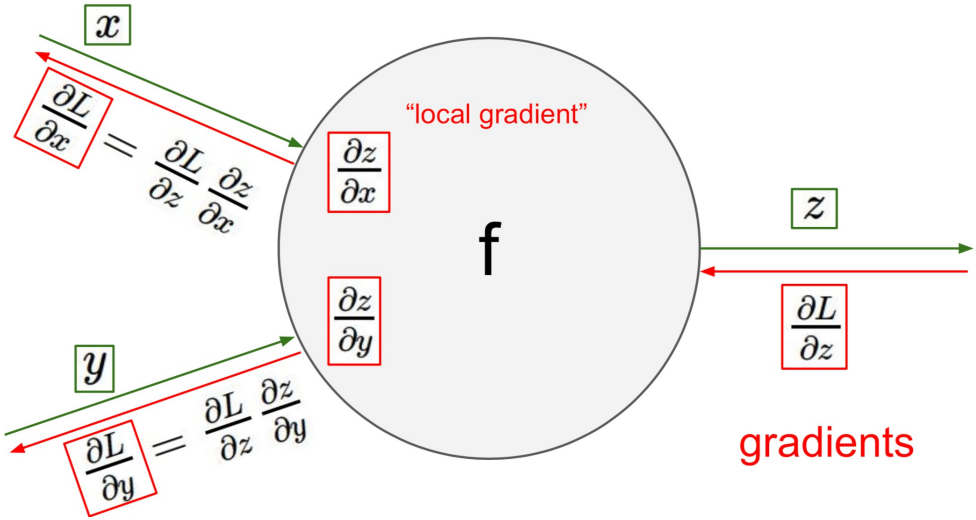
A quick summary:



Source: Fei-Fei Li & Justin Johnson & Serena Yeung, csc231N, Stanford University

A simple example : Backward Pass

A quick summary:



Source: Fei-Fei Li & Justin Johnson & Serena Yeung, csc231N, Stanford University

A more complex example: logistic least squares model

Recall: Univariate logistic least squares model

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

Let's compute the loss derivatives.

Univariate Chain Rule

How you would have done it in calculus class

$$\mathcal{L} = \frac{1}{2}(\sigma(wx+b)-t)^2$$

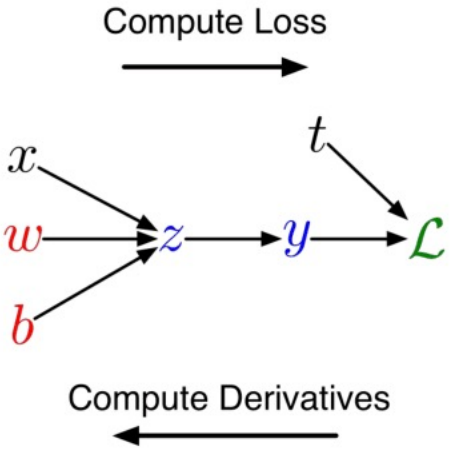
$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial w} &= \frac{\partial}{\partial w}\left[\frac{1}{2}(\sigma(wx+b)-t)^2\right] \\
&= \frac{1}{2}\frac{\partial}{\partial w}(\sigma(wx+b)-t)^2 \\
&= (\sigma(wx+b)-t)\frac{\partial}{\partial w}(\sigma(wx+b)-t) \\
&= (\sigma(wx+b)-t)\sigma'(wx+b)\frac{\partial}{\partial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

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

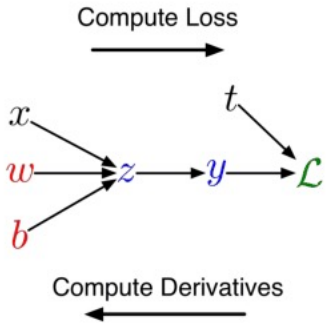
What are the disadvantages of this approach?

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.



A more structured way to do it



Computing the loss:

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

Computing the derivatives:

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

Univariate Chain Rule

A slightly more convenient notation:

- Use $\overline{y}$ to denote the derivative $\mathrm{d}\mathcal{L}/\mathrm{d}y$, sometimes called the error signal.
- This emphasizes that the error signals are just values our program is computing (rather than a mathematical operation).
- This is not a standard notation, but I couldn't find another one that I liked.

Computing the loss:

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

Computing the derivatives:

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

Backpropagation

Full backpropagation algorithm:

Let $v_1, \ldots, v_N$ 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 $i = 1, \ldots, N$

Compute v_i as a function of $\mathrm{Pa}(v_i)$

backward pass

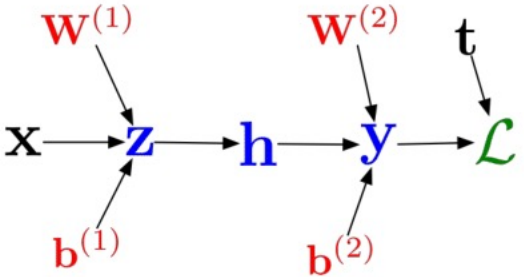
$\overline{v_N} = 1$

For $i = N-1, \ldots, 1$

$\overline{v_i} = \sum_{j \in \mathrm{Ch}(v_i)} \overline{v_j} \frac{\partial v_j}{\partial v_i}$

Vector Form

MLP example in vectorized form:



Forward pass:

$$\mathbf{z} = \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}$$

$$\mathbf{h} = \sigma(\mathbf{z})$$

$$\mathbf{y} = \mathbf{W}^{(2)}\mathbf{h} + \mathbf{b}^{(2)}$$

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

Backward pass:

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

$$\overline{\mathbf{y}} = \overline{\mathcal{L}}(\mathbf{y} - \mathbf{t})$$

$$\overline{\mathbf{W}^{(2)}} = \overline{\mathbf{y}}\mathbf{h}^\top$$

$$\overline{\mathbf{b}^{(2)}} = \overline{\mathbf{y}}$$

$$\overline{\mathbf{h}} = \mathbf{W}^{(2)\top}\overline{\mathbf{y}}$$

$$\overline{\mathbf{z}} = \overline{\mathbf{h}} \circ \sigma'(\mathbf{z})$$

$$\overline{\mathbf{W}^{(1)}} = \overline{\mathbf{z}}\mathbf{x}^\top$$

$$\overline{\mathbf{b}^{(1)}} = \overline{\mathbf{z}}$$

Closing Thoughts

- 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.
  - No evidence for biological signals analogous to error derivatives.
  - All the biologically plausible alternatives we know about learn much more slowly (on computers).
  - So how on earth does the brain learn?

Closing Thoughts

The psychological profiling [of a programmer] is mostly the ability to shift levels of abstraction, from low level to high level. To see something in the small and to see something in the large.

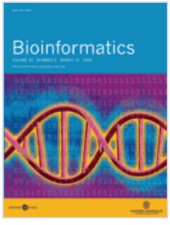
– Don Knuth

- By now, we've seen three different ways of looking at gradients:
  - **Geometric:** visualization of gradient in weight space
  - **Algebraic:** mechanics of computing the derivatives
  - **Implementational:** efficient implementation on the computer
- When thinking about neural nets, it's important to be able to shift between these different perspectives!

MLP for Molecular Data Analysis

Bioinformatics

Issues Advance articles Submit ▾ Purchase Alerts About ▾ All Bioinformatics



Volume 22, Issue 6
15 March 2006

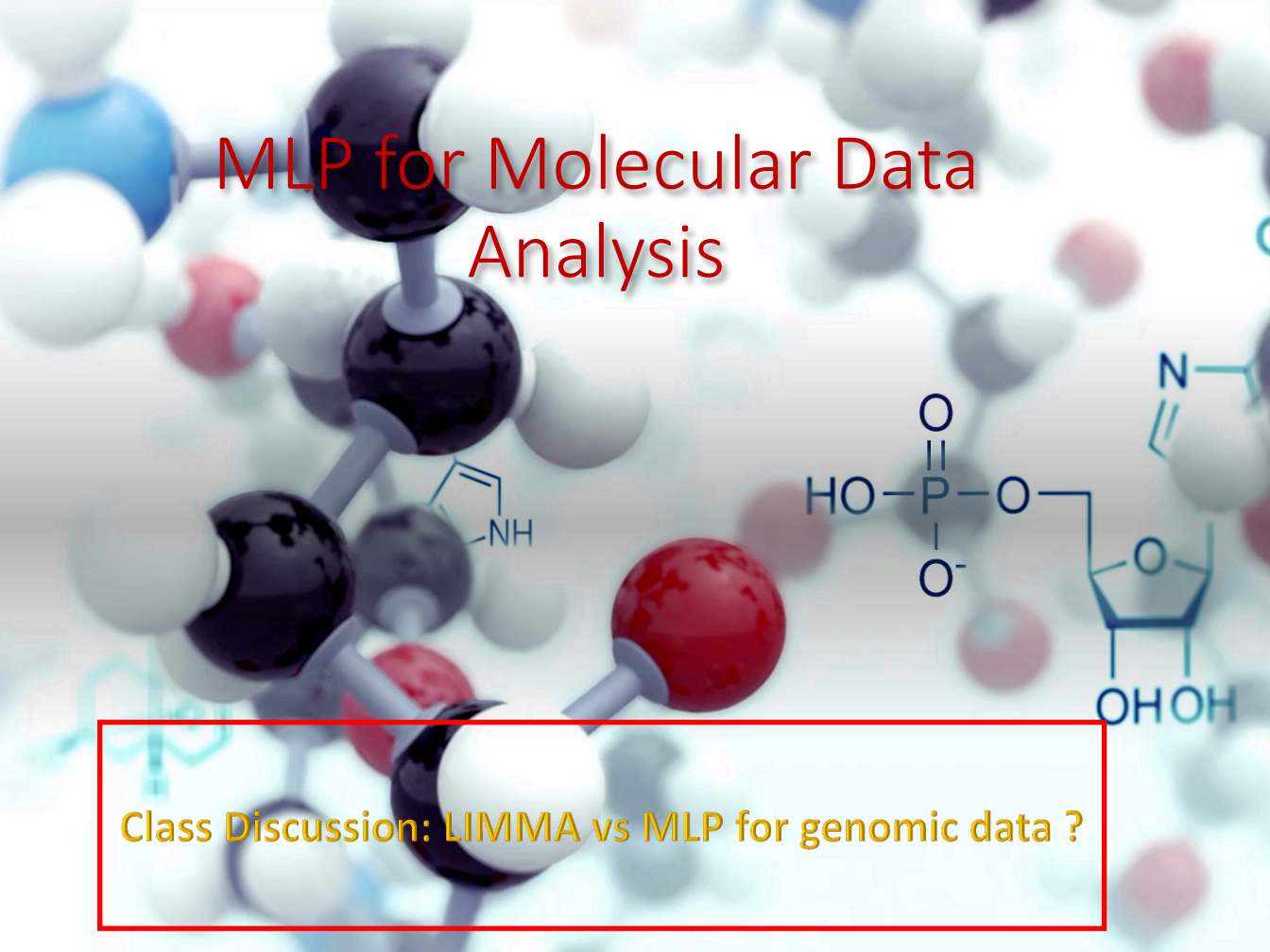
Article Contents

Optimized multilayer perceptrons for molecular classification and diagnosis using genomic data FREE

Zuyi Wang, Yue Wang ✉, Jianhua Xuan, Yibin Dong, Marina Bakay, Yuanjian Feng, Robert Clarke, Eric P. Hoffman Author Notes

Bioinformatics, Volume 22, Issue 6, 15 March 2006, Pages 755–761,
https://doi.org/10.1093/bioinformatics/btk036
Published: 10 January 2006 **Article history** ▾

MLP for Molecular Data Analysis



Class Discussion: LIMMA vs MLP for genomic data ?

What are in the next lecture?

1. Ensemble methods for classification

2. Tutorials on implementing supervised learning algorithms using ***sklearn***!