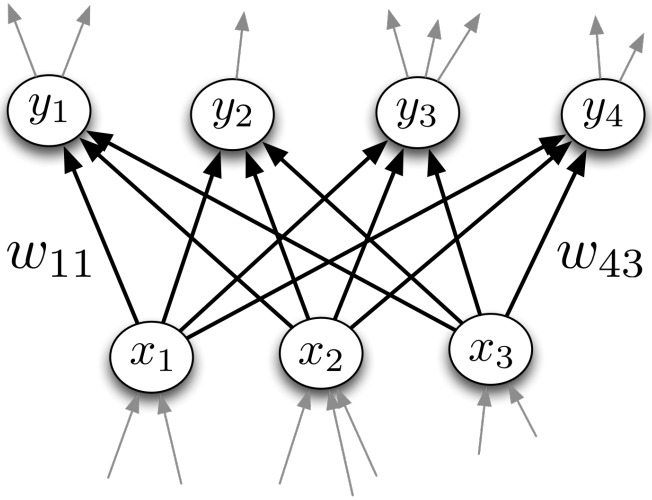
STA414/2104

Statistical Methods for Machine Learning II

Murat A. Erdogdu

Department of Computer Science
Department of Statistical Sciences

Lecture 6



UNIVERSITY OF TORONTO

Announcements

- Midterm exam is next week
  - If you are enrolled to Monday section, you will take the exam on March 1st, at 14-16.
  - If you are enrolled to Tuesday section, you will take the exam on March 2nd, at 19-21.
  - Exceptions will be made **only** if you currently live in a time zone that "strictly" conflicts with your section's exam time. You need to contact the instructor by Feb 24 and get approval. Requests made after this date will not be considered.
  - If neither of these times work for you, your final exam (scheduled by FAS) will be worth 50% of your course mark.
  - Further instructions will be posted on the course website.
  - Exam will be on mostly ML concepts and derivations, covered in lectures 1-6 (including this one).
  - Practice midterm will be posted on the course webpage. Solutions will be posted on Friday.

Last time

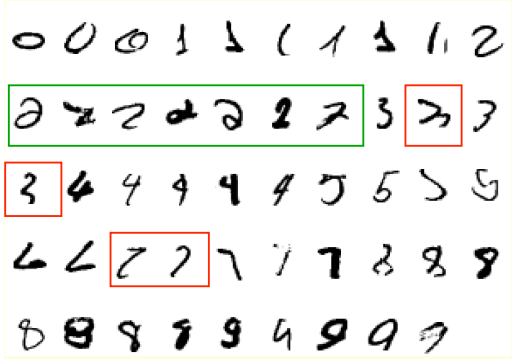
- Naïve Bayes classifier
- Statistical decision theory
- Bias-variance decomposition
- Optimization in ML

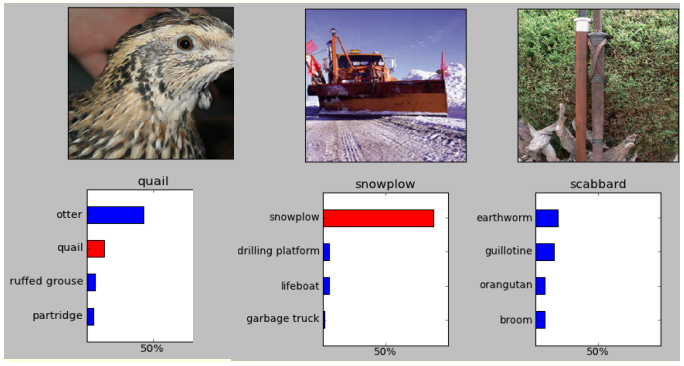
Today

- Multiclass logistic regression (recap)
- Neural networks
- Midterm review (if time permits)

Multiclass Classification

- Classification tasks with more than two categories:





- We use multiclass logistic regression to demonstrate computation graph.

Multiclass Classification

- Consider a single data point $(\mathbf{t}, \mathbf{x})$.
- 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:

$$\mathbf{t} = \underbrace{(0, \ldots, 0, 1, 0, \ldots, 0)}_{\text{entry } k \text{ is } 1} \in \mathbb{R}^K$$

- We saw the MLE interpretation. Now we minimize errors.

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:

$$z_k = \sum_{j=1}^{D} w_{kj} x_j + b_k \quad \text{for} \quad k = 1, 2, ..., K$$

- Vectorized as in assignment 2:

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

Multiclass Classification

- Predictions are like probabilities: want $1 \geq y_k \geq 0$ and $\sum_k y_k = 1$
- A natural activation function to use is the softmax function, a multivariable generalization of the sigmoid 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 is much larger than the others, $\mathrm{softmax}(\mathbf{z})_k \approx 1$ (behaves like argmax).

Multiclass Classification

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

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

  where the $\log$ is applied elementwise.

- Error E is the sum (or average) of loss functions across data points $(\mathbf{y}_i, \mathbf{t}_i)$ for $i = 1, .., N$.

$$E(\mathbf{w}) = \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}_{\mathrm{CE}}(\mathbf{y}_i, \mathbf{t}_i)$$

- This is exactly the negative of the log-likelihood (up to constants). Recall that $1/N$ doesn't change the problem.

Multiclass Classification

- All we need is the gradient of the loss for one data point $\frac{\partial}{\partial \mathbf{w}} \mathcal{L}_{\mathrm{CE}}(\mathbf{y}_i, \mathbf{t}_i)$ since:
- Error E is the sum (or average) of loss functions across data points.

$$\begin{aligned}\frac{\partial}{\partial \mathbf{w}} E(\mathbf{w}) &= \frac{\partial}{\partial \mathbf{w}} \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}_{\mathrm{CE}}(\mathbf{y}_i, \mathbf{t}_i) \\ &= \frac{1}{N} \sum_{i=1}^{N} \frac{\partial}{\partial \mathbf{w}} \mathcal{L}_{\mathrm{CE}}(\mathbf{y}_i, \mathbf{t}_i)\end{aligned}$$

- This is exactly the negative of the log-likelihood (up to constants). Recall that $1/N$ doesn't change the problem.

Multiclass Classification

- Multiclass logistic regression (for one sample):

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

- Gradient descent updates can be derived for each row of $\mathbf{W}$:

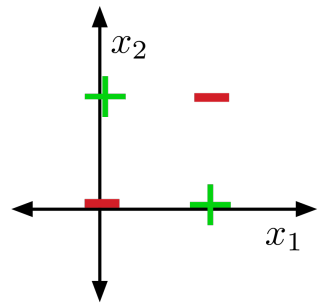
$$\frac{\partial \mathcal{L}_{\mathrm{CE}}}{\partial \mathbf{w}_k} = \frac{\partial \mathcal{L}_{\mathrm{CE}}}{\partial z_k} \cdot \frac{\partial z_k}{\partial \mathbf{w}_k} = (y_k - t_k)\mathbf{x} \qquad \text{(for 1 sample)}$$

$$\mathbf{w}_k \leftarrow \mathbf{w}_k - \eta \frac{1}{N} \sum_{i=1}^{N} (y_{ik} - t_{ik})\mathbf{x}_i \qquad \text{(for N samples)}$$

- Compare to the update given in lecture 4 and hw 2.

Limits of Linear Classification

- Visually, it's obvious that **XOR** is not linearly separable. But how to show this?



x_1	x_2	t
0	0	0
0	1	1
1	0	1
1	1	0

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	$\psi_1(\mathbf{x})$	$\psi_2(\mathbf{x})$	$\psi_3(\mathbf{x})$	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

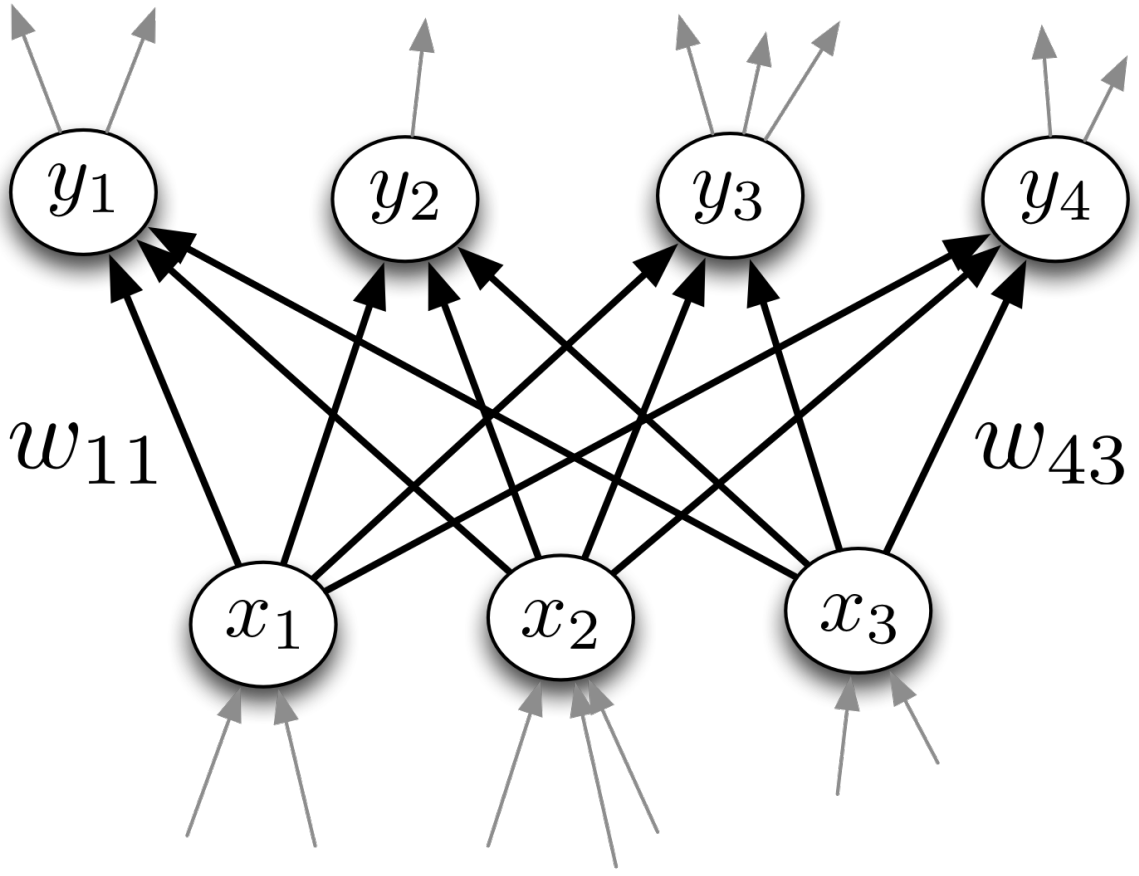
- This is linearly separable. (Try it!)
- Not a general solution: it can be hard to pick good basis functions. Instead, we'll use neural nets to learn nonlinear hypotheses directly.

Feature maps are hard

- Can we automate the feature extraction somehow?

Neural Networks

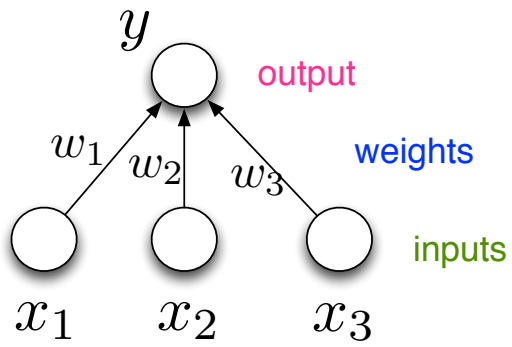
...

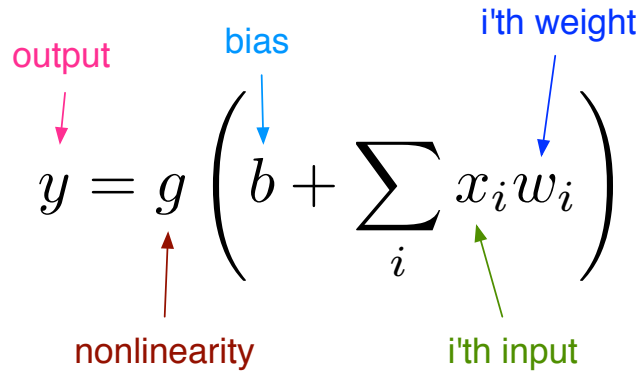


...

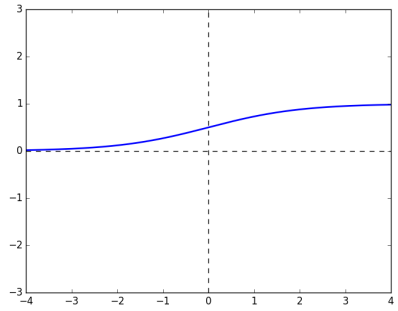
Single Neuron (Unit)

- For neural nets, we use a neuron, or **unit** to encode non-linearities:





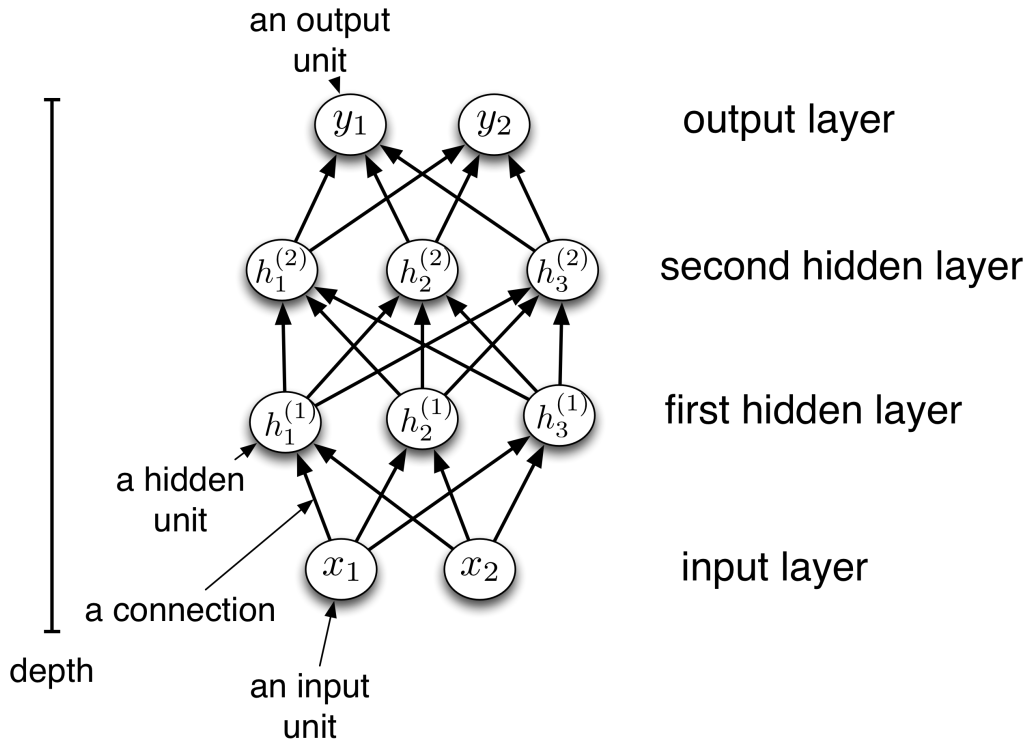
- Compare with logistic regression: $y = \sigma(\mathbf{w}^\top \mathbf{x} + b)$



- By throwing together lots of these incredibly simplistic neuron-like processing units, we can do some powerful computations!

Multilayer Perceptrons

- We can connect lots of units together into a **directed graph**.
- Typically, units are grouped together into **layers**.
- This gives a **feed-forward neural network**.

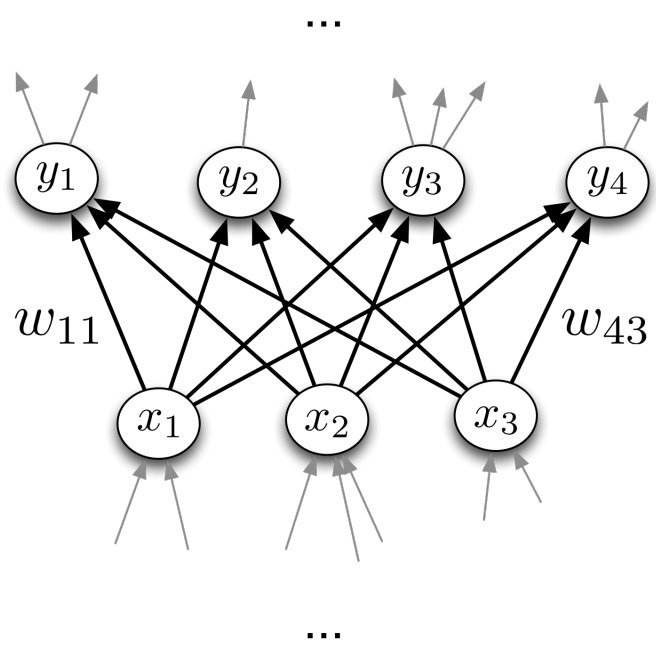


Multilayer Perceptrons

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

- If we need to compute M outputs from N inputs, we can do so in parallel using matrix multiplication. This means we'll be using a $M \times N$ matrix
- The output units are a function of the input units:

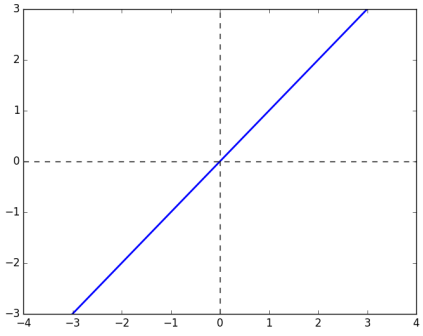
$$\mathbf{y} = f(\mathbf{x}) = \phi(\mathbf{W}\mathbf{x} + \mathbf{b})$$



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

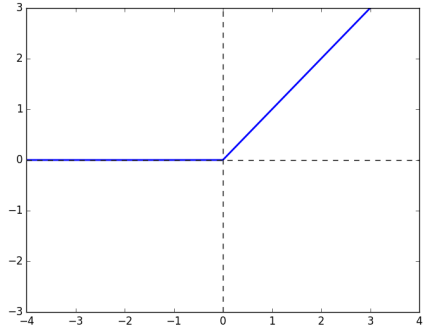
Multilayer Perceptrons

Some activation functions:



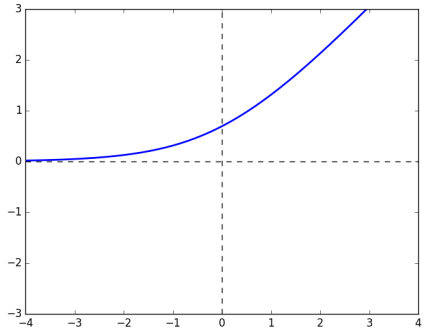
Identity

$$y = z$$



Rectified Linear Unit (ReLU)

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

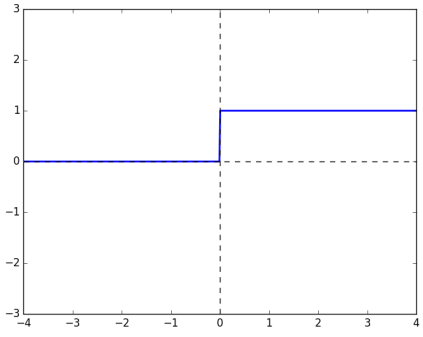


Soft ReLU

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

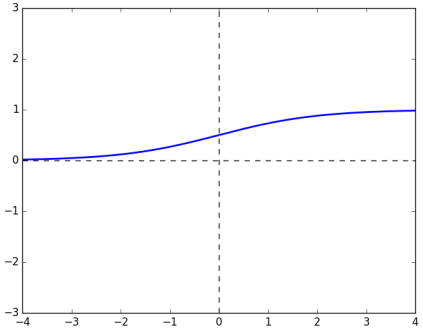
Multilayer Perceptrons

Some activation functions:



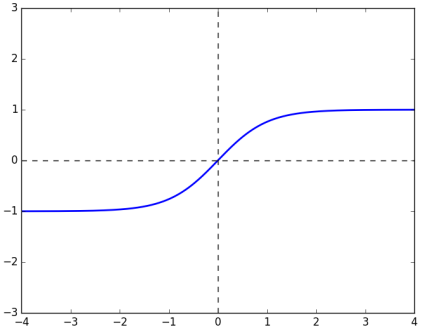
Hard Threshold

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



Logistic

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



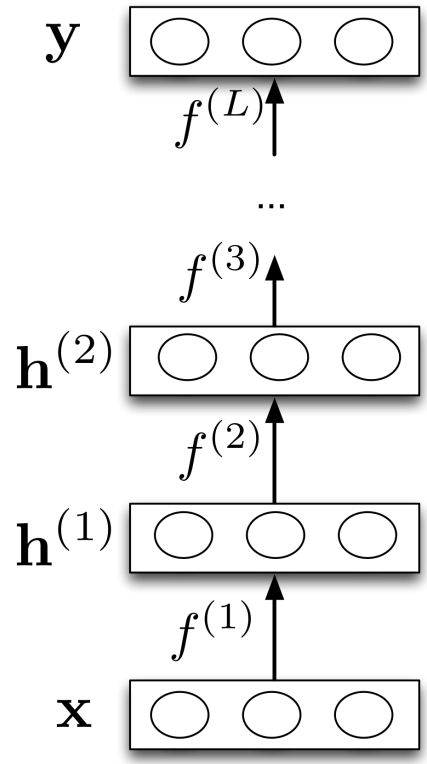
Hyperbolic Tangent (tanh)

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

Multilayer Perceptrons

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

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



- Or more simply:

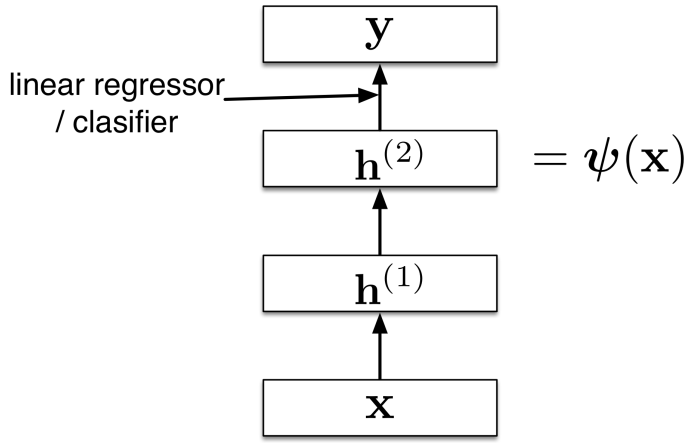
$$\mathsf{y} = f^{(L)} \circ \cdots \circ f^{(1)}(\mathsf{x}).$$

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

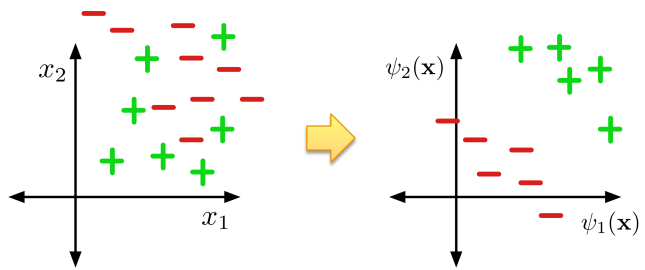
Feature Learning

Last layer:

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

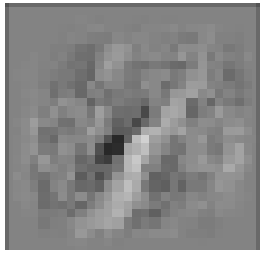


- The goal:



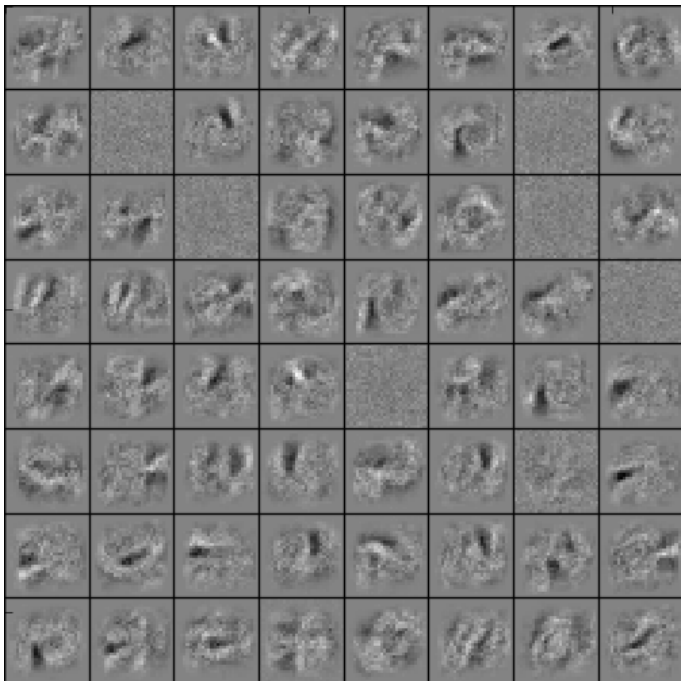
Feature Learning

- Suppose we're trying to classify images of handwritten digits. Each image is represented as a vector of $28 \times 28 = 784$ pixel values.
- Each first-layer hidden unit computes $\phi(\mathbf{w}_i^T \mathbf{x})$. It acts as a **feature detector**.
- We can visualize $\mathbf{w}$ by reshaping it into an image. Here's an example that responds to a diagonal stroke.



Feature Learning

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



Expressive Power

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

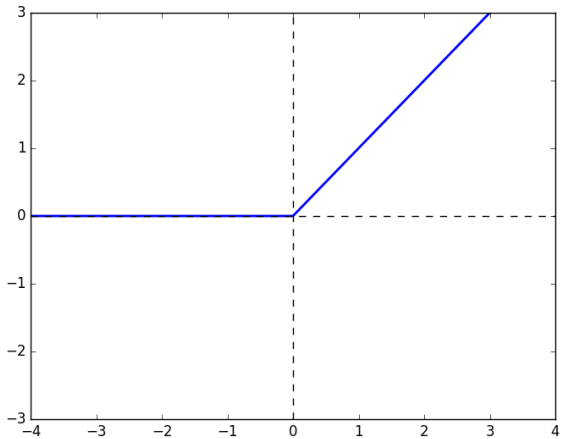
$$y = \underbrace{W^{(3)}W^{(2)}W^{(1)}}_{\triangleq W'}x$$

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

Expressive Power

- Multilayer feed-forward neural nets with *nonlinear* activation functions are **universal function approximators**: they can approximate any function arbitrarily well.

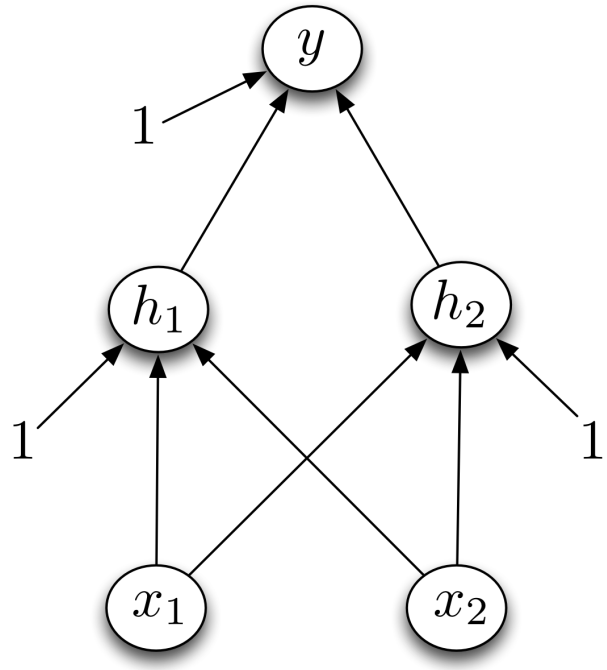
- This has been shown for various activation functions (thresholds, logistic, ReLU, etc.)
  - Even though ReLU is "almost" linear, it's nonlinear enough.



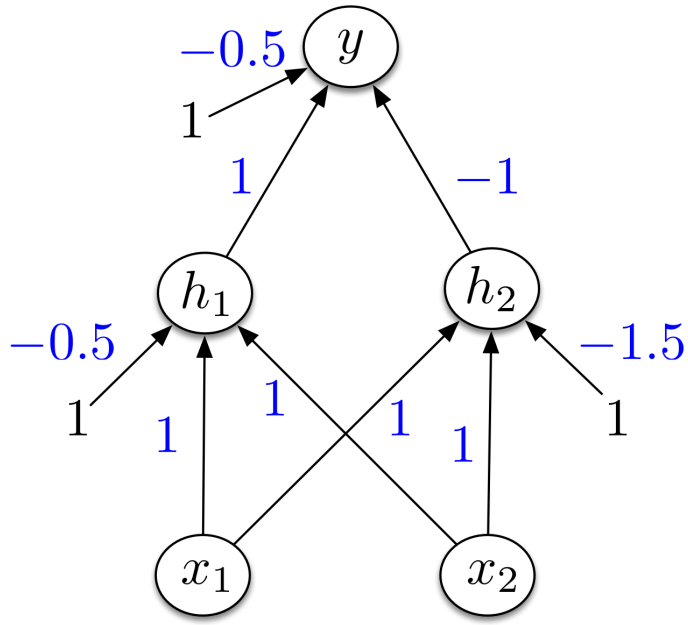
Multilayer Perceptrons

Designing a network to classify XOR:

Assume hard threshold activation function



Multilayer Perceptrons



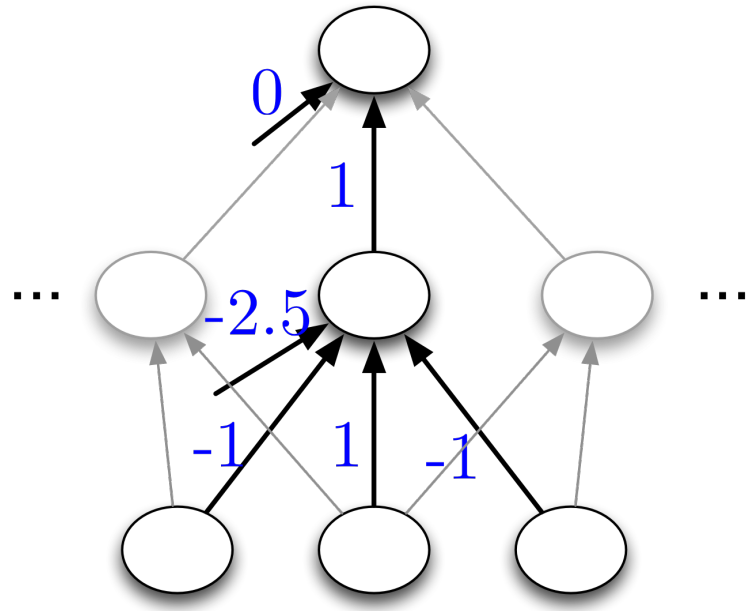
- h_1 computes $\mathbb{I}[x_1 + x_2 - 0.5 > 0]$
  - i.e. x_1 OR x_2
- h_2 computes $\mathbb{I}[x_1 + x_2 - 1.5 > 0]$
  - i.e. x_1 AND x_2
- y computes $\mathbb{I}[h_1 - h_2 - 0.5 > 0] \equiv \mathbb{I}[h_1 + (1 - h_2) - 1.5 > 0]$
  - i.e. h_1 AND (NOT h_2) $= x_1$ XOR x_2

Expressive Power

Universality for binary inputs and targets $x_i, t \in \{-1, +1\}$**:**

- 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
	$\vdots$		$\vdots$



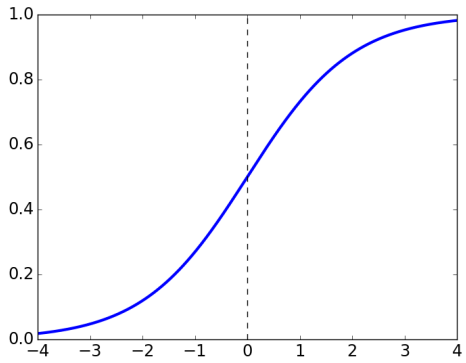
- Only requires one hidden layer, though it needs to be extremely wide.

Expressive Power

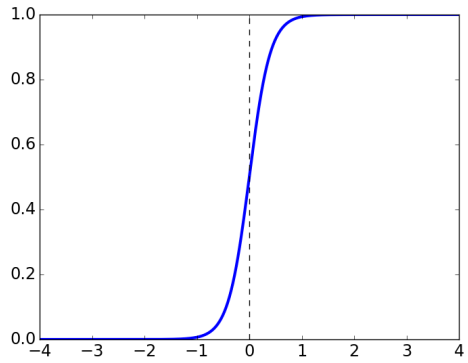
- Limits of universality
  - You may need to represent an exponentially large network.
  - How can you find the appropriate weights to represent a given function?
  - If you can learn any function, you'll just overfit.
  - Really, we desire a *compact* representation.

Expressive Power

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



$y = \sigma(x)$



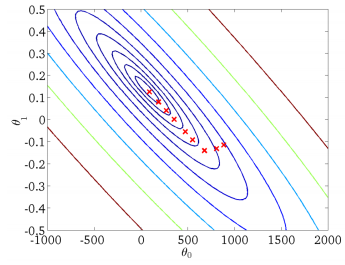
$y = \sigma(5x)$

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

Training neural networks with backpropagation

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 define a loss $\mathcal{L}$ and compute the gradient of the cost $\mathrm{d}E/\mathrm{d}\mathbf{w}$, which is the vector of partial derivatives.
  - This is the average of $\mathrm{d}\mathcal{L}/\mathrm{d}\mathbf{w}$ over all the training examples, so in this lecture we focus on computing $\mathrm{d}\mathcal{L}/\mathrm{d}\mathbf{w}$.

Univariate Chain Rule

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

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

Univariate Chain Rule

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 $\frac{\partial \mathcal{L}}{\partial w}, \frac{\partial \mathcal{L}}{\partial b}$

Univariate Chain Rule

How you would have done it in calculus class

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

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

What are the disadvantages of this approach?

Univariate Chain Rule

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:

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

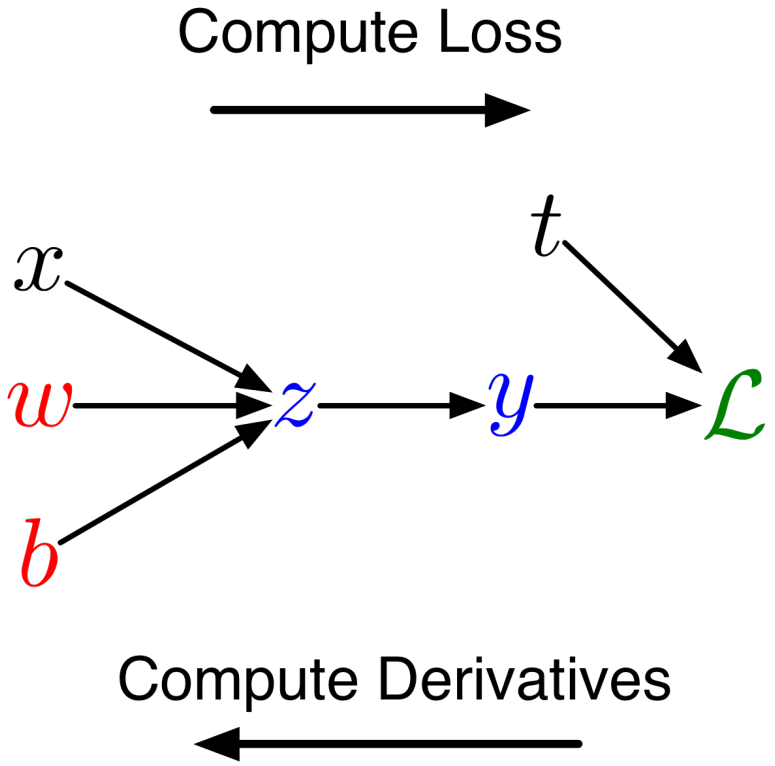
Remember, the goal isn't to obtain closed-form solutions, but to be able to write a program that efficiently computes the derivatives.

Univariate Chain Rule

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

Computing the loss:

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



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).

Computing the loss:

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

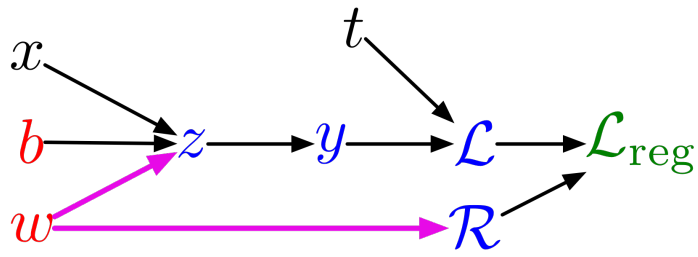
Computing the derivatives:

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

Multivariate Chain Rule

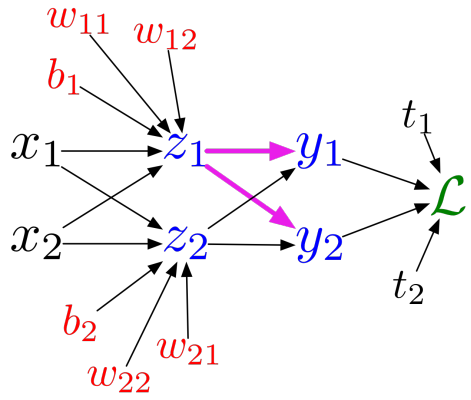
Problem: what if the computation graph has **fan-out** > 1?
This requires the **multivariate Chain Rule**!

L_2-Regularized regression



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

Logistic regression

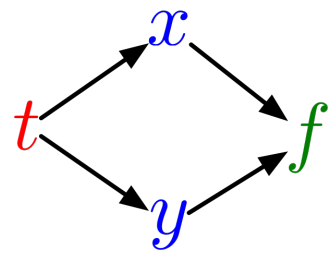


$$z_\ell = \sum_j w_{\ell j} x_j + b_\ell$$
$$y_k = \frac{e^{z_k}}{\sum_\ell e^{z_\ell}}$$
$$\mathcal{L} = -\sum_k t_k \log y_k$$

Multivariate Chain Rule

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

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



- Example:

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

- Plug in to Chain Rule:

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

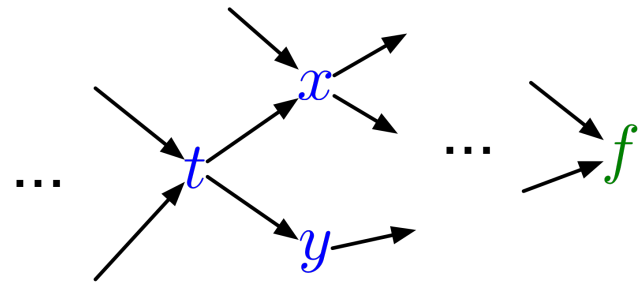
Multivariable Chain Rule

- In the context of backpropagation:

Mathematical expressions to be evaluated

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

Values already computed by our program

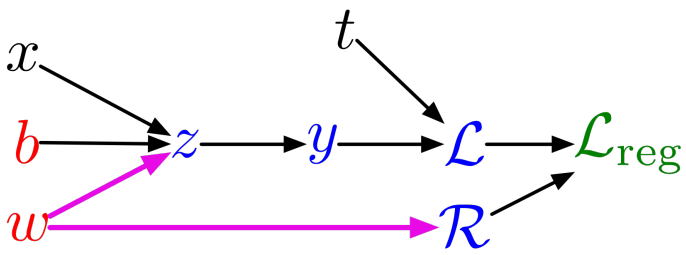


- In our notation:

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

Backpropagation

Example: univariate logistic least squares regression



Forward pass:

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

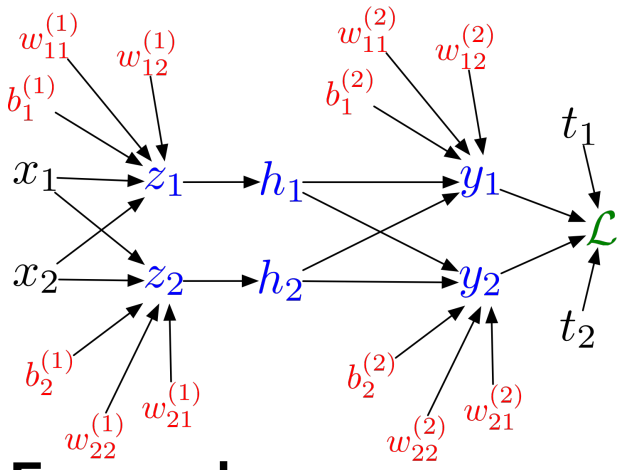
Backward pass:

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

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

Backpropagation

Multilayer Perceptron (multiple outputs):



Forward pass:

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

$$h_i = \sigma(z_i)$$

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

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

Backward pass:

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

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

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

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

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

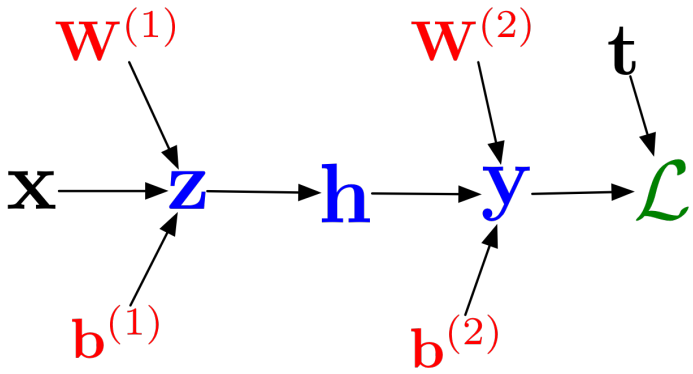
$$\overline{z_i} = \overline{h_i} \, \sigma'(z_i)$$

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

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

Backpropagation

In vectorized form:



Forward pass:

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

Backward pass:

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

Computational Cost

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

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

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

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

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

Backpropagation

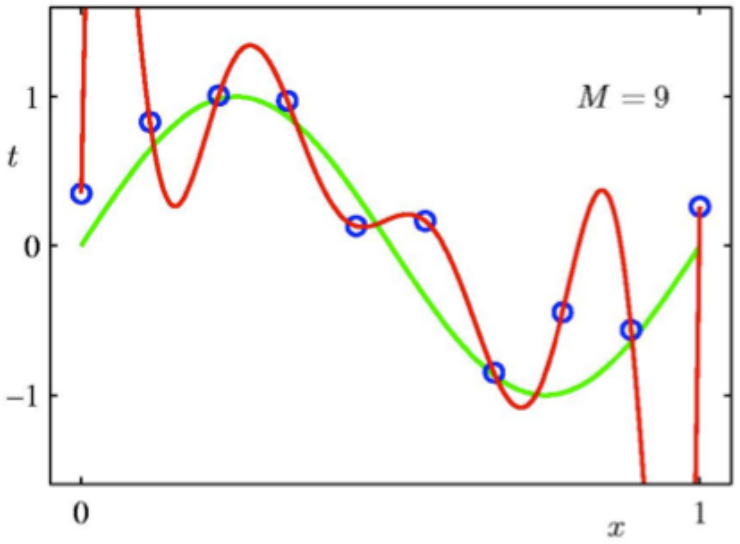
- Backprop is used to train the overwhelming majority of neural nets today.
  - Even optimization algorithms much fancier than gradient descent (e.g. second-order methods) use backprop to compute the gradients.

Midterm review

- Review of some important ML concepts.
- Practice midterm will be posted on the course webpage. Solutions will be posted on Friday.

Generalization

- The goal in ML is to achieve good generalization by making accurate predictions for **new test data** that is not known during learning.
- Choosing the values of parameters that minimize the loss function on the training data may not be the best option.
- We would like to model the true regularities in the data and ignore the noise in the data:
  - It is hard to know which regularities are real and which are accidental due to the particular training examples we happen to pick.



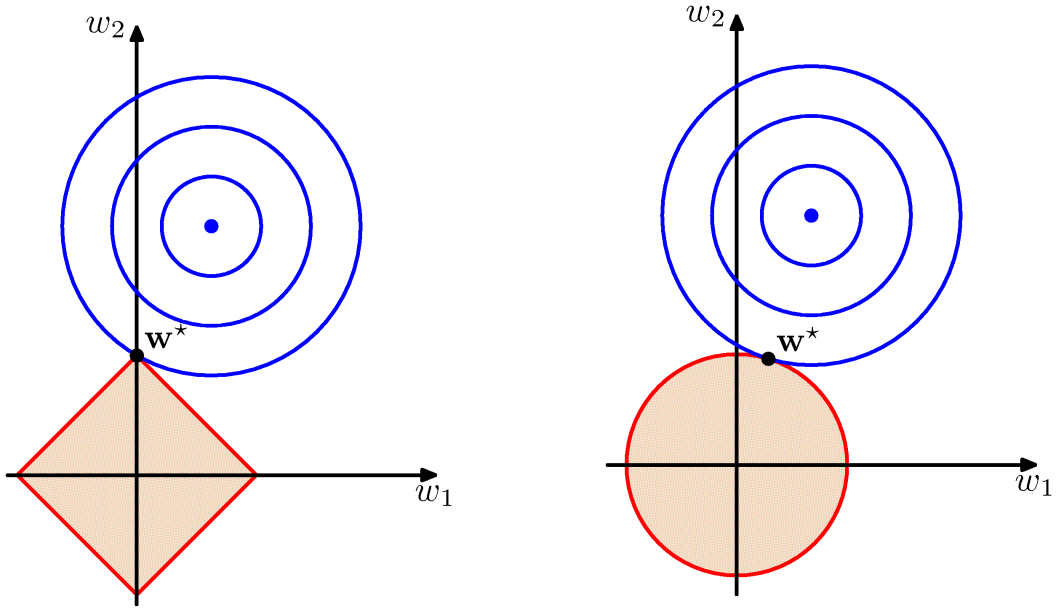
- Intuition: We expect the model to generalize if it explains the data well given the complexity of the model is low.
- A model can fit the data perfectly. But this is not very informative.

Regularization in ML

- We can write the problem $\min\ E_D(\mathbf{w}) + \lambda E_W(\mathbf{w})$

Training error + Regularization term

Lasso tends to generate sparser solutions compared to a quadratic regularizer (often referred to as L_1 and L_2 regularizers).



This constrains the complexity of the model and helps with generalization.

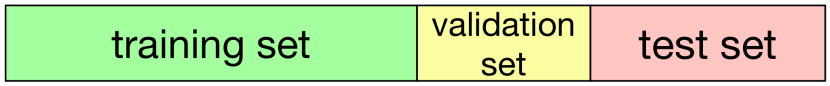
But how to test our model or choose the penalty level?

Validation

If the data is plentiful, we can divide the dataset into three subsets:

- Training Data: used to fitting/learning the parameters of the model.
- Validation Data: not used for learning but for selecting the model, choosing the amount of regularization that works best (i.e. M or any other hyperparameter tuning).
- Test Data: used to get performance of the final model.

Rule of thumb: split 50% training, 25% validation, 25% test



- For a range of λ (say $\lambda = \{0.1, 0.5, 1, 1.5, 2\}$)
  - For each λ, train model on training data and compute its error on validation data set
- Choose λ that has the smallest error.
- Test the final performance of your model on test data.

2.4. *Validation.* What is cross-validation? Discuss why we use this method and the main computational trade-offs involved.

Maximum Likelihood Estimation

- Recipe:
  - Observe data: $\mathcal{D} = \{x_1, x_2, ..., x_N\}$
  - Assume data is iid from a distribution $x_i \sim p(x|\theta)$
  - Write down the joint density

$$p(x_1, x_2, ..., x_N|\theta) = \prod_{i=1}^{N} p(x_i|\theta) = \mathcal{L}(\theta; x_1, x_2, ..., x_N)$$

  - Plug in the observed values (data) and see it as a function of the unknown parameters (at this stage, we call this function the likelihood)
  - Maximize the likelihood. $\hat{\theta}^{\mathrm{ML}} = \arg\max_{\theta} \mathcal{L}(\theta; x_1, x_2, ..., x_N)$

- We generally minimize negative log-likelihood since log is monotone strictly increasing function, and converts products to summations (which behave nicely taking derivatives). For example for linear regression,

$$\ln p(\mathbf{t}|\mathbf{x}, \mathbf{w}, \beta) = \underbrace{-\frac{\beta}{2}\sum_{n=1}^{N}(y(\mathbf{x}_n, \mathbf{w}) - t_n)^2}_{\beta E(\mathbf{w})} + \frac{N}{2}\ln\beta - \frac{N}{2}\ln(2\pi).$$

The Exponential Family

- The exponential family of distributions over x is defined to be a set of distributions of the form:

$$p(\mathbf{x}|\boldsymbol{\eta}) = h(\mathbf{x})g(\boldsymbol{\eta})\exp\left\{\boldsymbol{\eta}^{\mathrm{T}}\mathbf{u}(\mathbf{x})\right\}$$

where

- η is the vector of natural parameters
- u(x) is the vector of sufficient statistics

- The function g(η) can be interpreted as the coefficient that ensures that the distribution p(x|η) is normalized:

$$g(\boldsymbol{\eta})\int h(\mathbf{x})\exp\left\{\boldsymbol{\eta}^{\mathrm{T}}\mathbf{u}(\mathbf{x})\right\}\,\mathrm{d}\mathbf{x} = 1$$

1.1. *Geometric distribution.* Probability mass function of a random variable X distributed as geometric distribution with parameter γ is given as

$$\mathbb{P}(X = k) = \gamma(1-\gamma)^{k-1} \quad \text{for} \quad k = 1, 2, \ldots$$

(a) Show that this is a probability mass function.
(b) Write the above distribution as an exponential family, and identify its sufficient statistics, natural parameter, and normalizing function (or cumulant generating function or partition function).

We can easily compute the expectation and the variance of the sufficient statistics! How?

MAP: Maximum A posteriori Probability

- In Maximum A posteriori Probability (MAP) estimation, we maximize the posterior

$$p(\mathbf{w}|\mathcal{D}) = \frac{p(\mathcal{D}|\mathbf{w})P(\mathbf{w})}{P(\mathcal{D})},$$

$$\propto p(\mathcal{D}|\mathbf{w})P(\mathbf{w})$$

$$\mathbf{w}^{\mathrm{MAP}} = \arg\max_{\mathbf{w}} \frac{p(\mathcal{D}|\mathbf{w})P(\mathbf{w})}{P(\mathcal{D})},$$

$$= \arg\max_{\mathbf{w}} p(\mathcal{D}|\mathbf{w})P(\mathbf{w})$$

Bayesian Linear Regression

• Consider a zero mean isotropic Gaussian prior, which is governed by a single precision parameter:

$$p(\mathbf{w}) = \mathcal{N}(\mathbf{w}|\mathbf{0}, \alpha^{-1}\mathbf{I})$$

means diagonal covariance

for which the posterior is Gaussian with:

$$\begin{aligned} \mathbf{m}_N &= \beta \mathbf{S}_N \mathbf{\Phi}^{\mathrm{T}} \mathbf{t} \\ \mathbf{S}_N^{-1} &= \alpha \mathbf{I} + \beta \mathbf{\Phi}^{\mathrm{T}} \mathbf{\Phi}. \end{aligned}$$

$$\mathbf{w}_{ML} = \left(\mathbf{\Phi}^T \mathbf{\Phi}\right)^{-1} \mathbf{\Phi}^T \mathbf{t}.$$

• If we consider an infinitely broad prior, α = 0, the mean m_N of the posterior distribution reduces to maximum likelihood value w_{ML}.

• The log of the posterior distribution is given by the sum of the log-likelihood and the log of the prior:

$$\ln p(\mathbf{w}|\mathcal{D}) = -\frac{\beta}{2} \sum_{n=1}^{N} \left(t_n - \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}_n)\right)^2 - \frac{\alpha}{2} \mathbf{w}^T \mathbf{w} + \text{const.}$$

• Maximizing this posterior with respect to w is equivalent to minimizing the sum-of-squares error function with a quadratic regulation term (ridge regression).

Approaches to Classification

• First attempt: Construct a discriminant function that directly maps each input vector to a specific class.

• There are two alternative approaches:

- Discriminative Approach: Model $p(\mathcal{C}_k|\mathbf{x})$, directly, for example by representing them as parametric models, and optimize for parameters using the training set (e.g. logistic regression).
- Generative Approach: Model class conditional densities $p(\mathbf{x}|\mathcal{C}_k)$ together with the prior probabilities $p(\mathcal{C}_k)$ for the classes. Infer posterior probability using Bayes' rule:

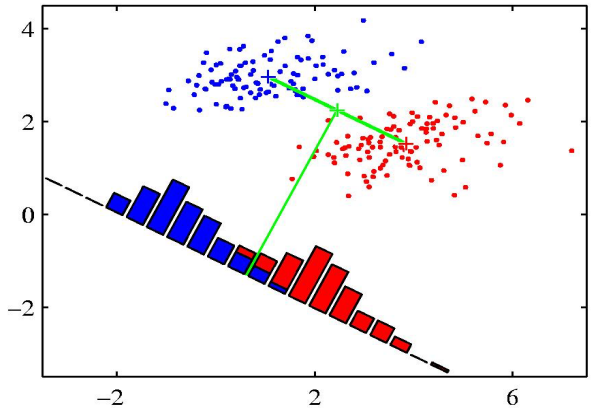
$$p(\mathcal{C}_k|\mathbf{x}) = \frac{p(\mathbf{x}|\mathcal{C}_k)p(\mathcal{C}_k)}{p(\mathbf{x})}.$$

  • For example, we could fit multivariate Gaussians to the input vectors of each class. Given a test vector, we see under which Gaussian the test vector is most probable.

Fisher's Linear Discriminant

- Let the mean of two classes be $\mathbf{m}_1 = \frac{1}{N_1} \sum_{n \in \mathcal{C}_1} \mathbf{x}_n, \quad \mathbf{m}_2 = \frac{1}{N_2} \sum_{n \in \mathcal{C}_2} \mathbf{x}_n,$

- Projecting onto the vector separating the two classes is reasonable (say it is a unit vector for now): $\mathbf{w} \propto \mathbf{m}_1 - \mathbf{m}_2.$



- But we also want to minimize the within-class variance: $s_1^2 = \sum_{n \in \mathcal{C}_1} (y_n - m_1)^2, \quad s_2^2 = \sum_{n \in \mathcal{C}_2} (y_n - m_2)^2,$

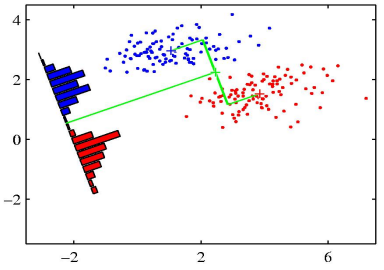
where $m_k = \mathbf{w}^T \mathbf{m}_k. \quad y_n = \mathbf{w}^T \mathbf{x}_n.$

- Fisher's criterion: maximize ratio of the between-class variance to within-class variance:

$$J(\mathbf{w}) = \frac{(m_2 - m_1)^2}{s_1^2 + s_2^2}.$$

$$\mathbf{w} \propto S_w^{-1} S_b \mathbf{w}$$

$$= S_w^{-1} (\mathbf{m}_2 - \mathbf{m}_1)$$



Probabilistic Generative Models

- Model class conditional densities $p(\mathbf{x}|\mathcal{C}_k)$ separately for each class, as well as the class priors $p(\mathcal{C}_k)$.
- Consider the case of two classes. The posterior probability of class C_1 is given by:

$$p(\mathcal{C}_1|\mathbf{x}) = \frac{p(\mathbf{x}|\mathcal{C}_1)p(\mathcal{C}_1)}{p(\mathbf{x}|\mathcal{C}_1)p(\mathcal{C}_1) + p(\mathbf{x}|\mathcal{C}_2)p(\mathcal{C}_2)}$$
$$= \frac{1}{1 + \exp(-a)} = \sigma(a),$$

Logistic sigmoid function

where we defined:

$$a = \ln \frac{p(\mathbf{x}|\mathcal{C}_1)p(\mathcal{C}_1)}{p(\mathbf{x}|\mathcal{C}_2)p(\mathcal{C}_2)} = \ln \frac{p(\mathcal{C}_1|\mathbf{x})}{1 - p(\mathcal{C}_1|\mathbf{x})},$$

which is known as the logit function. It represents the log of the ratio of probabilities of two classes, also known as the log-odds.

Gaussian class conditionals

$$p(\mathbf{t}, \mathbf{X}|\pi, \boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \boldsymbol{\Sigma}) = \prod_{n=1}^{N} \Big[\pi \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_1, \boldsymbol{\Sigma})\Big]^{t_n} \Big[(1-\pi)\mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_2, \boldsymbol{\Sigma})\Big]^{1-t_n}.$$

• Maximizing with respect to π, we look at the terms of the log-likelihood functions that depend on π:

$$\sum_n \big[t_n \ln \pi + (1-t_n)\ln(1-\pi)\big] + \text{const.}$$

Differentiating, we get:

$$\pi = \frac{1}{N}\sum_{n=1}^{N} t_n = \frac{N_1}{N_1 + N_2}.$$

• Maximizing with respect to μ_1, we look at the terms of the log-likelihood functions that depend on μ_1:

$$\sum_n t_n \ln \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_1, \boldsymbol{\Sigma}) = -\frac{1}{2}\sum_n t_n (\mathbf{x}_n - \boldsymbol{\mu}_1)^T \boldsymbol{\Sigma}^{-1}(\mathbf{x}_n - \boldsymbol{\mu}_1) + \text{const.}$$

Differentiating, we get:

$$\boldsymbol{\mu}_1 = \frac{1}{N_1}\sum_{n=1}^{N} t_n \mathbf{x}_n.$$

And similarly:

$$\boldsymbol{\mu}_2 = \frac{1}{N_2}\sum_{n=1}^{N} (1-t_n)\mathbf{x}_n.$$

Logistic Regression

- Consider the problem of two-class classification.
- We have seen that the posterior probability of class C_1 can be written as a logistic sigmoid function:

$$p(\mathcal{C}_1|\mathbf{x}) = \frac{1}{1+\exp(-\mathbf{w}^T\mathbf{x})} = \sigma(\mathbf{w}^T\mathbf{x}),$$

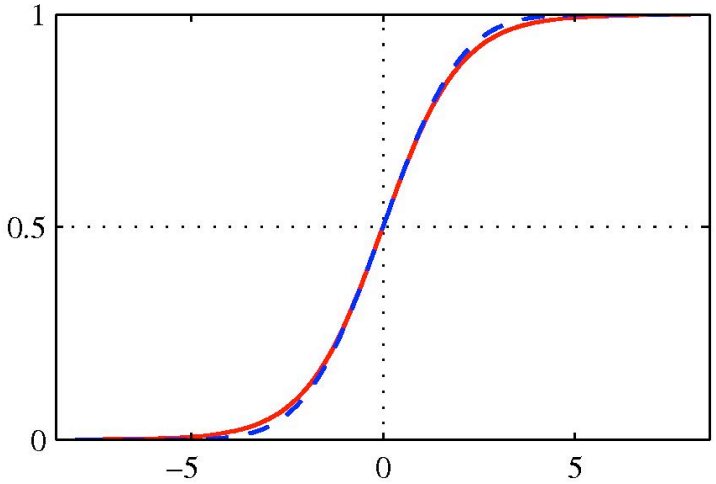
where $p(\mathcal{C}_2|\mathbf{x}) = 1 - p(\mathcal{C}_1|\mathbf{x})$, we omit the bias term for clarity.

- This model is known as logistic regression (although this is a model for classification rather than regression).

Note that for generative models, we would first determine the class conditional densities and class-specific priors, and then use Bayes' rule to obtain the posterior probabilities.

Here we model $p(\mathcal{C}_k|\mathbf{x})$ directly.

logistic sigmoid function



ML for Logistic Regression

- We observed a training dataset $\{\mathbf{x}_n, t_n\},\ n = 1, .., N;\ t_n \in \{0, 1\}.$
- Maximize the probability of getting the label right, so the likelihood function takes form:

$$p(\mathbf{t}|\mathbf{X}, \mathbf{w}) = \prod_{n=1}^{N} \left[y_n^{t_n} (1 - y_n)^{1-t_n} \right], \quad y_n = \sigma(\mathbf{w}^T \mathbf{x}_n).$$

- Taking the negative log of the likelihood, we can define cross-entropy error function (that we want to minimize):

$$E(\mathbf{w}) = -\ln p(\mathbf{t}|\mathbf{X}, \mathbf{w}) = -\sum_{n=1}^{N} \left[t_n \ln y_n + (1 - t_n) \ln(1 - y_n) \right] = \sum_{n=1}^{N} E_n.$$

- Differentiating and using the chain rule:

$$\frac{\mathrm{d}}{\mathrm{d}y_n} E_n = \frac{y_n - t_n}{y_n(1 - y_n)}, \quad \frac{\mathrm{d}}{\mathrm{d}\mathbf{w}} y_n = y_n(1 - y_n)\mathbf{x}_n, \quad \boxed{\frac{\mathrm{d}}{\mathrm{d}a}\sigma(a) = \sigma(a)(1 - \sigma(a)).}$$

$$\frac{\mathrm{d}}{\mathrm{d}\mathbf{w}} E_n = \frac{\mathrm{d}E_n}{\mathrm{d}y_n} \frac{\mathrm{d}y_n}{\mathrm{d}\mathbf{w}} = (y_n - t_n)\mathbf{x}_n.$$

verify

- Note that the factor involving the derivative of the logistic function cancelled.

Bias-Variance Trade-off

$$\text{expected loss} = (\text{bias})^2 + \text{variance} + \text{noise}$$

Average predictions over all datasets differ from the optimal regression function.

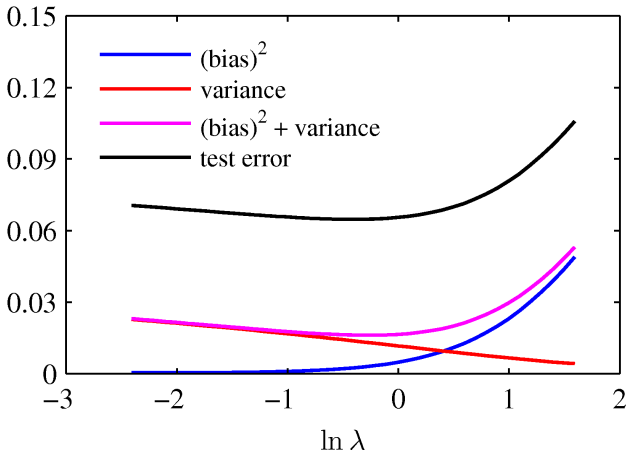
Solutions for individual datasets vary around their averages -- how sensitive is the function to the particular choice of the dataset.

Intrinsic variability of the target values.

$$(\text{bias})^2 = \int \{\mathbb{E}_{\mathcal{D}}[y(\mathbf{x};\mathcal{D})] - h(\mathbf{x})\}^2 p(\mathbf{x})\,\mathrm{d}\mathbf{x}$$

$$\text{variance} = \int \mathbb{E}_{\mathcal{D}}\left[\{y(\mathbf{x};\mathcal{D}) - \mathbb{E}_{\mathcal{D}}[y(\mathbf{x};\mathcal{D})]\}^2\right] p(\mathbf{x})\,\mathrm{d}\mathbf{x}$$

$$\text{noise} = \iint \{h(\mathbf{x}) - t\}^2 p(\mathbf{x},t)\,\mathrm{d}\mathbf{x}\,\mathrm{d}t$$



- Trade-off between bias and variance: With very flexible models (high complexity) we have low bias and high variance; With relatively rigid models (low complexity) we have high bias and low variance.
- The model with the optimal predictive capabilities has to balance between bias and variance.

Optimization: First and Second Order Methods

$$\beta^{t+1} = \text{argmin}_\beta q_t(\beta) = \beta^t - [\mathbf{H}^t]^{-1} \nabla E(\beta)$$

- $\mathbf{H}^t$ provides curvature information about the optimization landscape and determines the type of optimization method.
- $\mathbf{H}^t = \mathbf{I}$ reduces to gradient descent which is a first order method, i.e., only uses first order derivative information (ignoring the step size). Cheap per-iteration cost, but slow convergence rate.
- $\mathbf{H}^t = \nabla^2 E(\beta^t)$ reduces to Newton's method which is a second order method, i.e., uses second order derivative information, e.g.

$$\beta^{t+1} = \beta^t - \nabla^2 E(\beta^t)^{-1} \nabla E(\beta^t)$$

  - These methods get faster **convergence rate**, since they use curvature information.
- Computing Hessian is numerically expensive so Newton's method has high per-iteration cost.
- The performance of an algorithm is determined by both its convergence rate and per-iteration cost.