CSCI567 Machine Learning (Spring 2018)

Michael Shindler

Lecture 8: February 5, 2018

Outline

Outline

Linear regression

Setup

- Input: $\boldsymbol{x} \in \mathbb{R}^{\mathsf{D}}$ (covariates, predictors, features, etc)
- Output: $y \in \mathbb{R}$ (responses, targets, outcomes, outputs, etc)
- Training data: $\mathcal{D} = \{(\boldsymbol{x}_n, y_n), n = 1, 2, \ldots, \mathsf{N}\}$
- Model: $f : \boldsymbol{x} \to y$, with $f(\boldsymbol{x}) = w_0 + \sum_d w_d x_d = w_0 + \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}$

Goal: Minimize prediction error as much as possible

$$RSS(\tilde{\boldsymbol{w}}) = \sum_n [y_n - f(\boldsymbol{x}_n)]^2 = \sum_n [y_n - (w_0 + \sum_d w_d x_{nd})]^2$$

Why minimizing RSS is a sensible thing?

Why minimizing RSS is a sensible thing?

Probabilistic interpretation

- Noisy observation model (for simplicity, we have assumed 1-dimensional data)

$$Y = w_0 + w_1 X + \eta$$

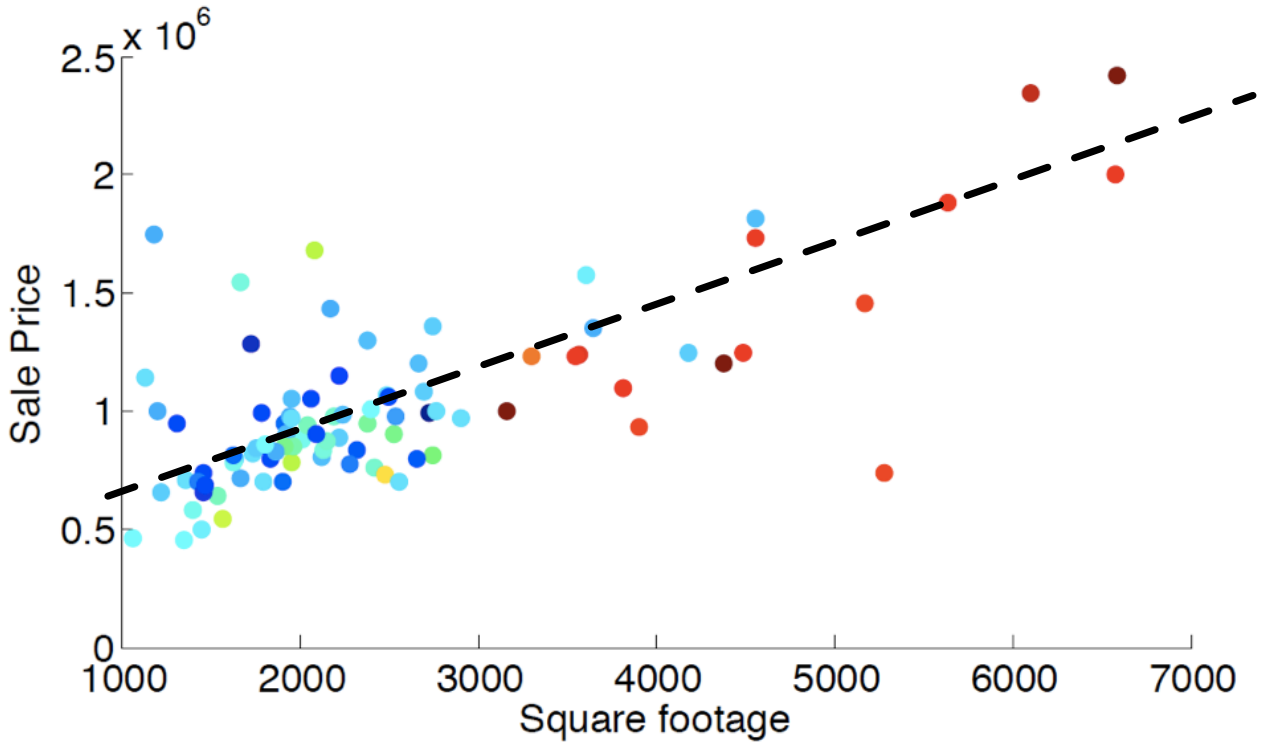
  where $\eta \sim N(0, \sigma^2)$ is a Gaussian random variable

- Likelihood of one training sample (x_n, y_n)

$$p(y_n|x_n) = N(w_0 + w_1 x_n, \sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{[y_n - (w_0 + w_1 x_n)]^2}{2\sigma^2}}$$

Possibly linear relationship

Sale price = price_per_sqft × square_footage + fixed_expense + unexplainable_stuff



Namely, we are saying the unexplainable_stuff is a Gaussian random variable

Probabilistic interpretation (cont'd)

Log-likelihood of the training data $\mathcal{D}$ (assuming i.i.d)

$$\log P(\mathcal{D}) = \log \prod_{n=1}^{\mathsf{N}} p(y_n|x_n) = \sum_n \log p(y_n|x_n)$$

Probabilistic interpretation (cont'd)

Log-likelihood of the training data $\mathcal{D}$ (assuming i.i.d)

$$\log P(\mathcal{D}) = \log \prod_{n=1}^{\mathsf{N}} p(y_n|x_n) = \sum_n \log p(y_n|x_n)$$

$$= \sum_n \left\{ -\frac{[y_n - (w_0 + w_1 x_n)]^2}{2\sigma^2} - \log \sqrt{2\pi}\sigma \right\}$$

Probabilistic interpretation (cont'd)

Log-likelihood of the training data $\mathcal{D}$ (assuming i.i.d)

$$\begin{aligned}
\log P(\mathcal{D}) &= \log \prod_{n=1}^{\mathsf{N}} p(y_n|x_n) = \sum_n \log p(y_n|x_n) \\
&= \sum_n \left\{ -\frac{[y_n - (w_0 + w_1 x_n)]^2}{2\sigma^2} - \log \sqrt{2\pi}\sigma \right\} \\
&= -\frac{1}{2\sigma^2} \sum_n [y_n - (w_0 + w_1 x_n)]^2 - \frac{\mathsf{N}}{2} \log \sigma^2 - \mathsf{N} \log \sqrt{2\pi}
\end{aligned}$$

Probabilistic interpretation (cont'd)

Log-likelihood of the training data $\mathcal{D}$ (assuming i.i.d)

$$\begin{aligned}
\log P(\mathcal{D}) &= \log \prod_{n=1}^{\mathsf{N}} p(y_n|x_n) = \sum_n \log p(y_n|x_n) \\
&= \sum_n \left\{ -\frac{[y_n - (w_0 + w_1 x_n)]^2}{2\sigma^2} - \log \sqrt{2\pi}\sigma \right\} \\
&= -\frac{1}{2\sigma^2} \sum_n [y_n - (w_0 + w_1 x_n)]^2 - \frac{\mathsf{N}}{2} \log \sigma^2 - \mathsf{N} \log \sqrt{2\pi} \\
&= -\frac{1}{2} \left\{ \frac{1}{\sigma^2} \sum_n [y_n - (w_0 + w_1 x_n)]^2 + \mathsf{N} \log \sigma^2 \right\} + \text{const}
\end{aligned}$$

i.i.d stands for independently and identically distributed.

Maximum likelihood estimation

Estimating σ, w_0 and w_1 can be done in two steps[1]

- Maximize over w_0 and w_1

$$\max \ \log P(\mathcal{D}) \Leftrightarrow \min \ \sum_n [y_n - (w_0 + w_1 x_n)]^2 \leftarrow \text{That is RSS}(\tilde{\boldsymbol{w}})!$$

This is not generally true but in this particular case, we can do so.

Maximum likelihood estimation

Estimating σ, w_0 and w_1 can be done in two steps[1]

- Maximize over w_0 and w_1

$$\max \ \log P(\mathcal{D}) \Leftrightarrow \min \ \sum_n [y_n - (w_0 + w_1 x_n)]^2 \leftarrow \text{That is RSS}(\tilde{\boldsymbol{w}})!$$

- Maximize over $s = \sigma^2$ (we could estimate σ directly)

$$\frac{\partial \log P(\mathcal{D})}{\partial s} = -\frac{1}{2}\left\{-\frac{1}{s^2}\sum_n [y_n - (w_0 + w_1 x_n)]^2 + \mathsf{N}\frac{1}{s}\right\} = 0$$

This is not generally true but in this particular case, we can do so.

Maximum likelihood estimation

Estimating σ, w_0 and w_1 can be done in two steps[1]

- Maximize over w_0 and w_1

$$\max \ \log P(\mathcal{D}) \Leftrightarrow \min \ \sum_n [y_n - (w_0 + w_1 x_n)]^2 \leftarrow \text{That is RSS}(\tilde{\boldsymbol{w}})!$$

- Maximize over $s = \sigma^2$ (we could estimate σ directly)

$$\frac{\partial \log P(\mathcal{D})}{\partial s} = -\frac{1}{2}\left\{-\frac{1}{s^2}\sum_n [y_n - (w_0 + w_1 x_n)]^2 + \mathsf{N}\frac{1}{s}\right\} = 0$$

$$\rightarrow \sigma^{*2} = s^* = \frac{1}{\mathsf{N}}\sum_n [y_n - (w_0 + w_1 x_n)]^2$$

This is not generally true but in this particular case, we can do so.

Why we want to have the probabilistic interpretation?

- It gives a solid footing to our intuition: minimizing $\text{RSS}(\tilde{\boldsymbol{w}})$ is a sensible thing to do as it grows naturally out of the probabilistic model.
- The ability of having estimated σ^* — how much noise could be present in our prediction – is valuable. For example, it allows us to make confidence intervals about our predictions.

Outline

Setup

Suppose we need to predict multiple classes/outcomes: $C_1, C_2, \ldots, C_K$

- Weather prediction: sunny, cloudy, raining, etc
- Optical character recognition: 10 digits + 26 characters (lower and upper cases) + special characters, etc

Studied methods

- One versus Rest
- One versus One

Definition of multinomial logistic regression

Model

For each class C_k, we have a parameter vector $\boldsymbol{w}_k$ and model the conditional probability as

$$p(y = k|\boldsymbol{x}) = \frac{e^{\boldsymbol{w}_k^{\mathrm{T}}\boldsymbol{x}}}{\sum_{k'} e^{\boldsymbol{w}_{k'}^{\mathrm{T}}\boldsymbol{x}}}$$

Decision boundary: assign $\boldsymbol{x}$ with the label that is the maximum of the conditional probabilities

$$\arg\max_k p(y = k|\boldsymbol{x}) = \arg\max_k \boldsymbol{w}_k^{\mathrm{T}}\boldsymbol{x}$$

Cross-entropy error function

Definition: negated likelihood

$$\begin{aligned}\mathcal{E}(\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_K) &= -\sum_n \sum_k y_{nk} \log p(y = |\boldsymbol{x}_n) \\ &= -\sum_n \sum_k y_{nk} \left\{ \boldsymbol{w}_k^{\mathrm{T}} \boldsymbol{x} - \log \sum_{k'} e^{\boldsymbol{w}_{k'}^{\mathrm{T}} \boldsymbol{x}} \right\} \quad (1)\end{aligned}$$

Cross-entropy error function

Definition: negated likelihood

$$\begin{aligned}\mathcal{E}(\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_K) &= -\sum_n \sum_k y_{nk} \log p(y = |\boldsymbol{x}_n) \\ &= -\sum_n \sum_k y_{nk} \left\{ \boldsymbol{w}_k^{\mathrm{T}} \boldsymbol{x} - \log \sum_{k'} e^{\boldsymbol{w}_{k'}^{\mathrm{T}} \boldsymbol{x}} \right\} \quad (1)\end{aligned}$$

Properties

- Optimization requires numerical procedures, analogous to those used for binary logistic regression
  Large-scale implementation, in both the number of classes and the training examples, is non-trivial.

Summary

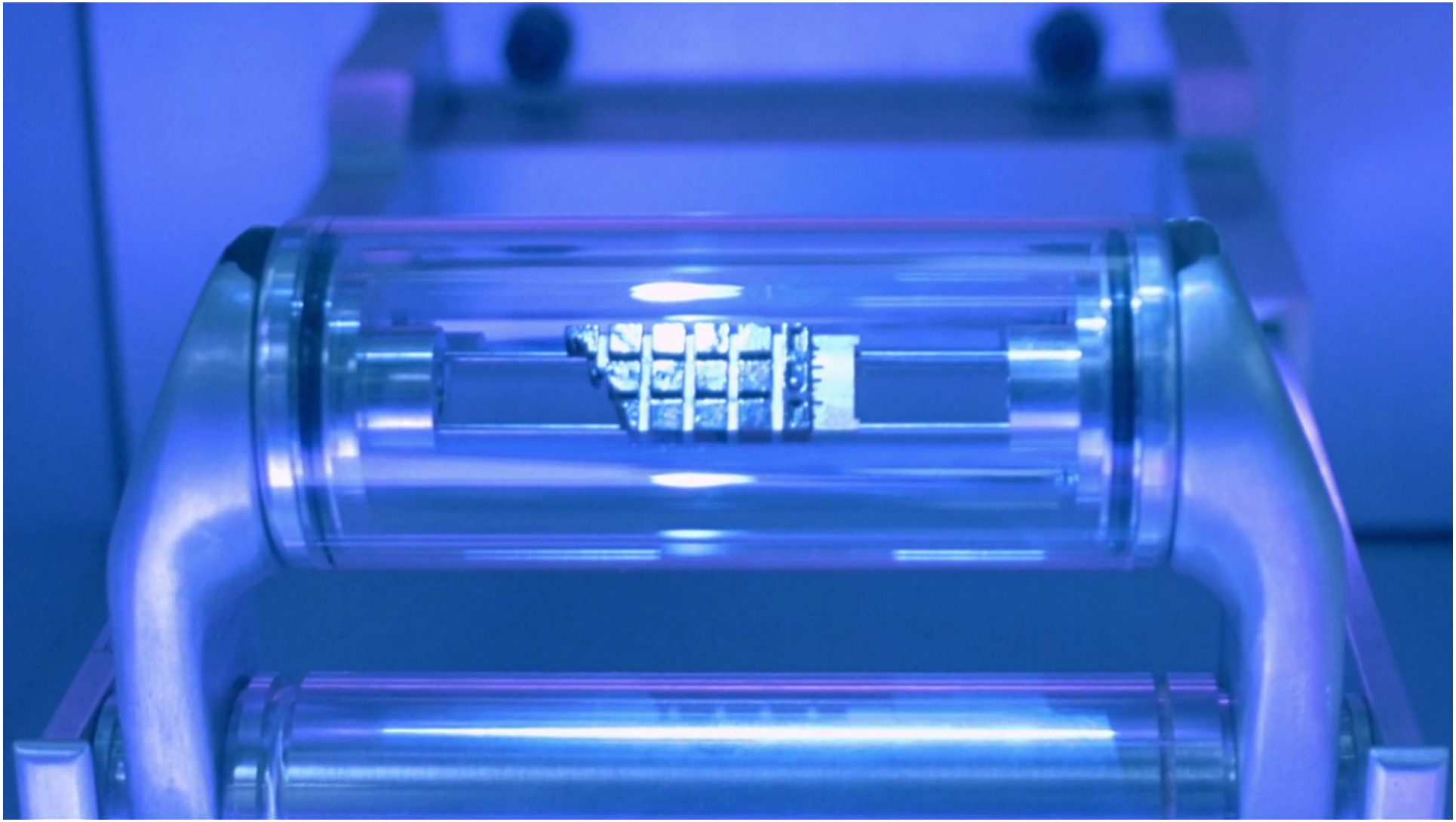
- Supervised learning
  regression and classification: continuous versus discrete outputs
- Methods
  parametric and nonparametric: linear classifier/regression versus nearest neighbor
  Linear and nonlinear: linear regression versus regression with nonlinear basis
- Learning objectives
  Probabilistic model: conditional probabilistic models for either regression or classification
  Non-probabilistic model: perceptron that minimizes a loss function

  These two objectives are called *discriminative* – we will see *generative* models for unsupervised learning later in the semester.

Outline

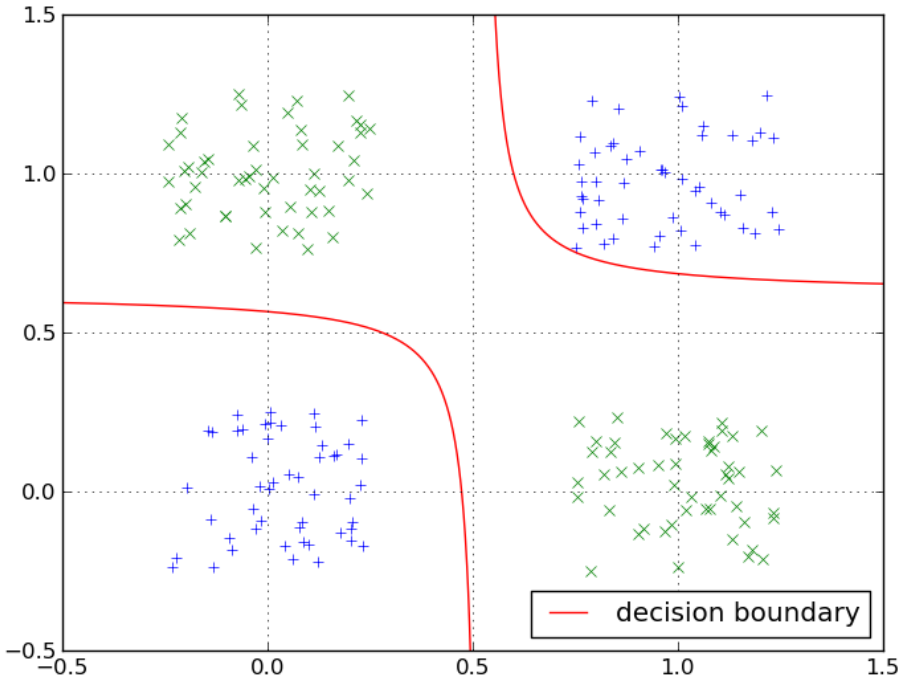
Acknowledgement: several figures from this lecture are borrowed from the lecture slides on the website `www.deeplearningbook.org`

Example neural network (this one is broken)



Outline

Linear functions are not always adequate



In particular, the following table displays the famous XOR problem

x_1	x_2	label
0	0	0
0	1	1
1	0	1
1	1	0

We already showed in previous lectures that there exists no linear classifier that can classify those 4 points correctly.

General nonlinear basis functions

We can use a nonlinear mapping

$$\boldsymbol{\phi}(\boldsymbol{x}) : \boldsymbol{x} \in \mathbb{R}^D \rightarrow \boldsymbol{z} \in \mathbb{R}^M$$

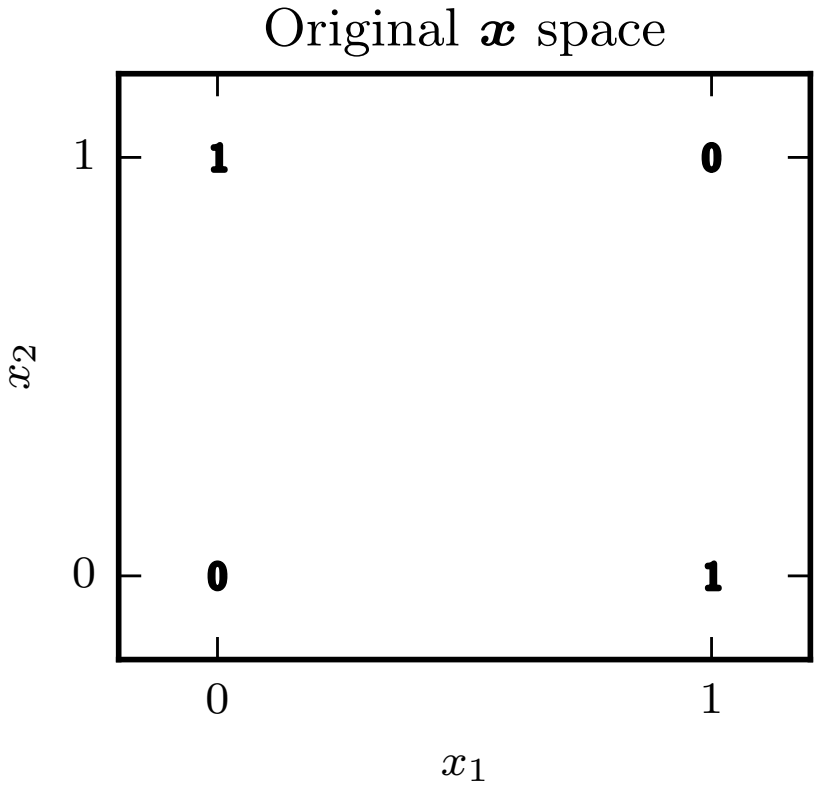
where M is the dimensionality of the new feature/input $\boldsymbol{z}$ (or $\boldsymbol{\phi}(\boldsymbol{x})$). Note that M could be either greater than D or less than or the same.

Then apply linear method, such that

$$y = \text{sign}[\boldsymbol{w}^{\mathrm{T}} \boldsymbol{\phi}(\boldsymbol{x})]$$

might work. *But what kind of nonlinear mapping ϕ we should use?*

Solving XOR



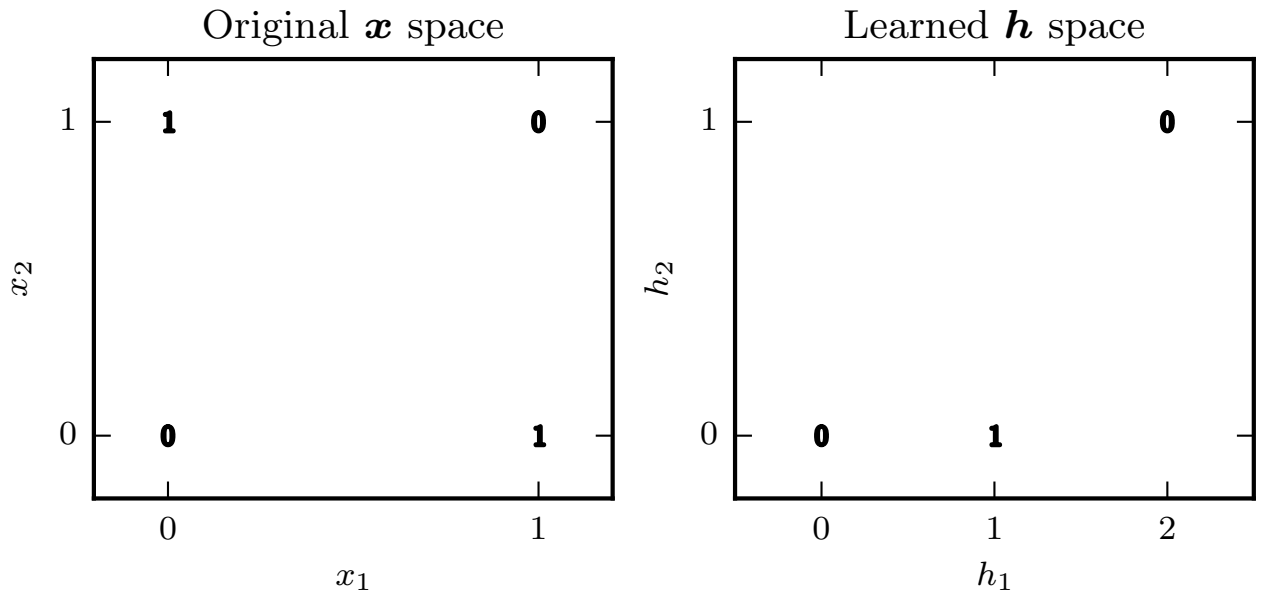
Define two mappings

$$h_1 = \max\{0, x_1 + x_2\} \quad (2)$$

$$h_2 = \max\{0, x_1 + x_2 - 1\} \quad (3)$$

x_1	x_2	h_1	h_2	label
0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	2	1	0

Solving XOR: linearly separable in the new feature space

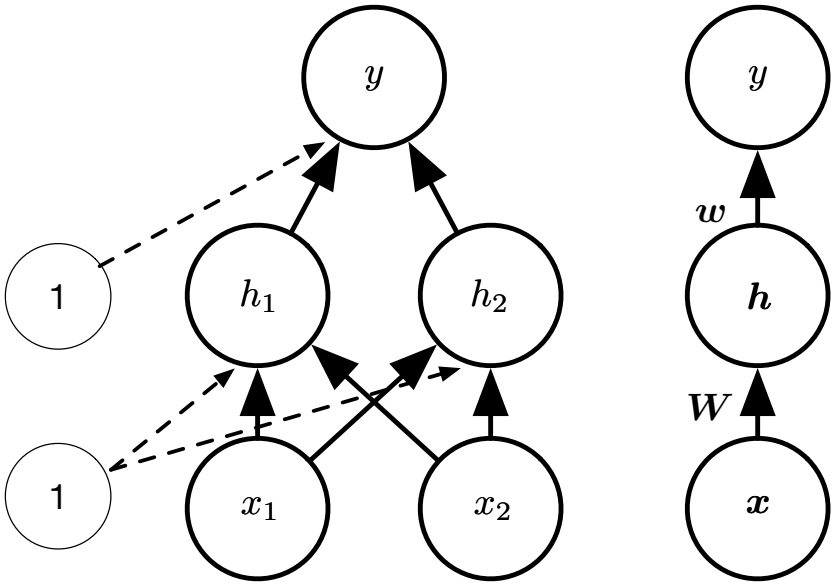


Define linear decision boundary

$$y = \text{sign}[h_1 - 2h_2]$$

(We assume $\text{sign}[0] = 0$)

The computation shown as a layered-network structure



We have introduced shorthands:

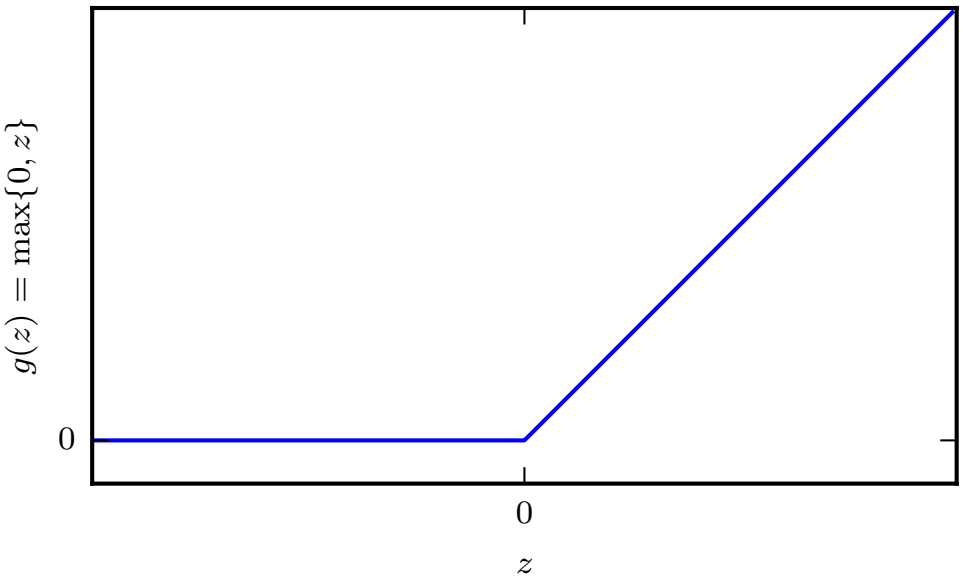
- We implicitly assume there is always a constant "1" (dashed lines) at every stage of computation, representing the "biases".
- Between layers, there are linear transformations as *inputs* to the nodes

$$\boldsymbol{W}^{\mathrm{T}}\boldsymbol{x},\ \boldsymbol{w}^{\mathrm{T}}\boldsymbol{h}$$

- The outputs (of nodes) could be nonlinear transformation of their inputs

$$\boldsymbol{h} = \max\{0, \boldsymbol{W}^{\mathrm{T}}\boldsymbol{x}\},\ y = \mathrm{sign}[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{h}]$$

The importance of nonlinearity



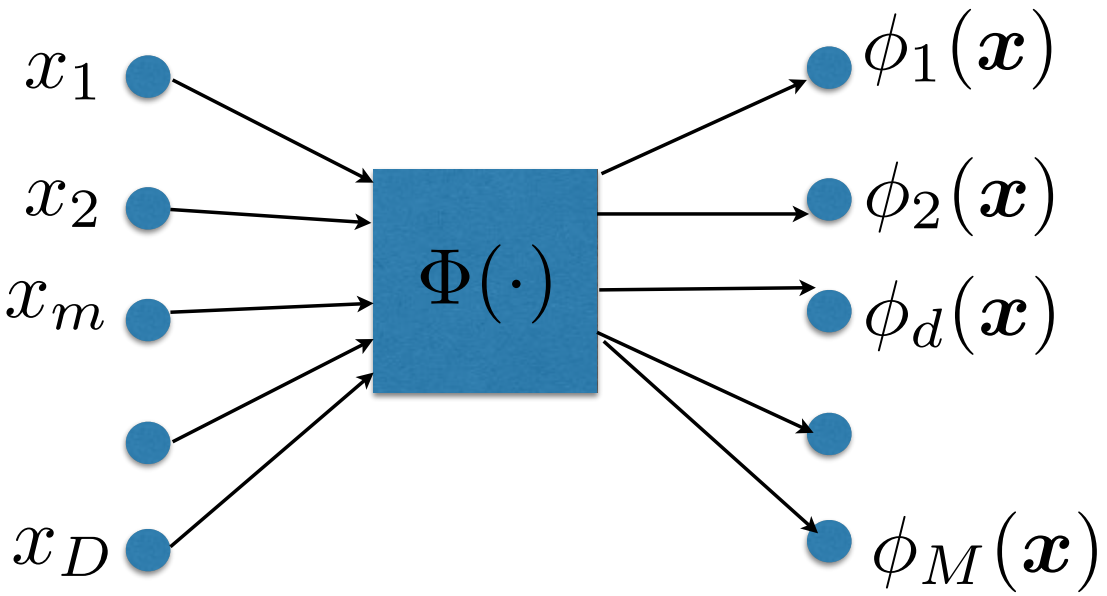
Many other nonlinearities also work too!

Any nonlinear basis functions seen as a neural nets

Transform the input feature with nonlinear function

Original input/feature $\boldsymbol{x} \in \mathbb{R}^D$

new features $\boldsymbol{\Phi}(\boldsymbol{x}) \in \mathbb{R}^M$



Linear regression

$$y = \boldsymbol{w}^{\mathrm{T}} \boldsymbol{\Phi}(\boldsymbol{x}) + w_0$$

$$y = \mathrm{sgn}(\boldsymbol{w}^{\mathrm{T}} \boldsymbol{\Phi}(\boldsymbol{x}) + w_0)$$

Linear classification

Terminology

Layered architecture of "neurons"

Input layer: features

hidden layer: nonlinear transformation (transfer function)

Output layer: targets

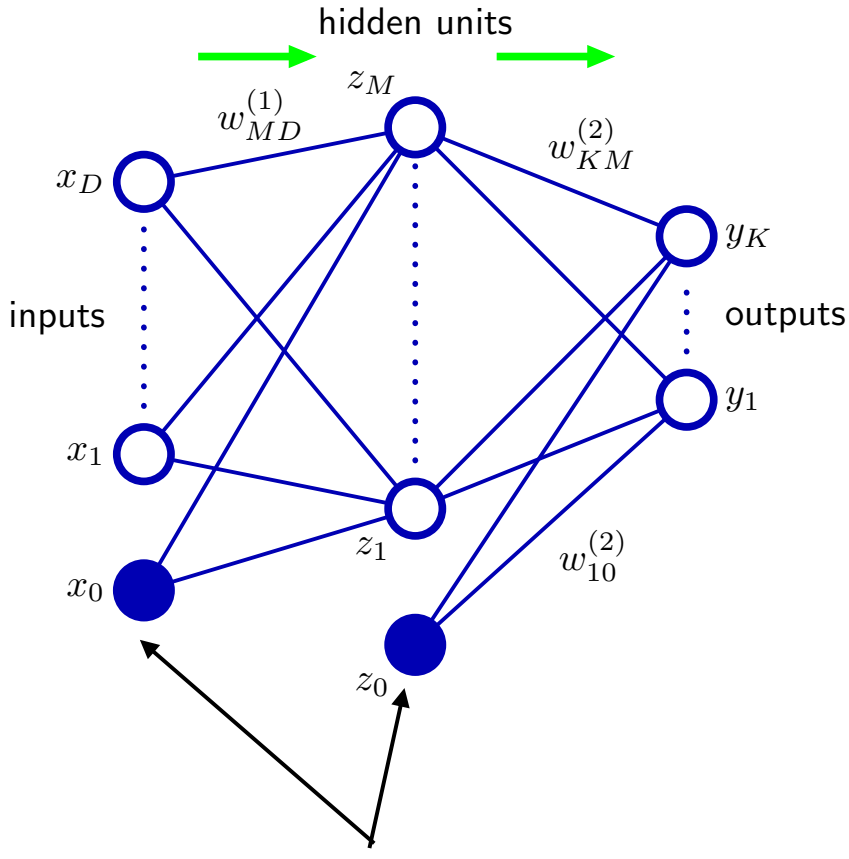
Feedforward computation

hidden layer output:

$$z_j = h(a_j) = h\left(\sum_{i=0}^{D} w_{ji}^{(1)} x_i\right)$$

Output layer output

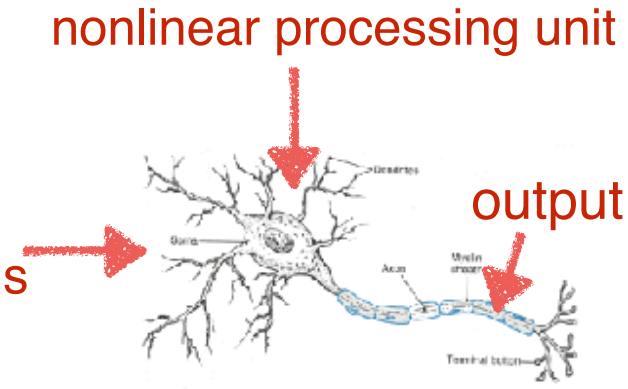
$$y_k = g\left(\sum_{j=0}^{M} w_{kj}^{(2)} z_j\right)$$



We set these two have a constant value of 1, thus "bias"

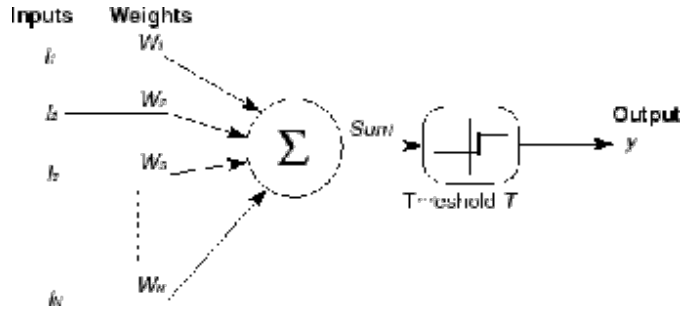
A very concise history

1943 McCulloch-Pitts model of single neurons



1960's Rosenblatt's perceptron learning

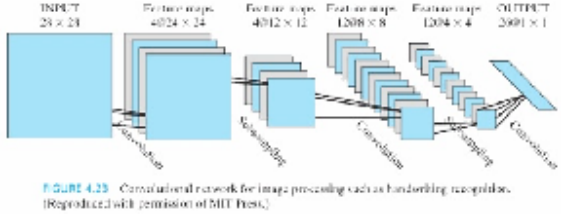
1969 Minsky and Papert's perceptron



1985 Hopfield neural nets

1986 Parallel and Distributed Processing (PDP book) and Connectionisms

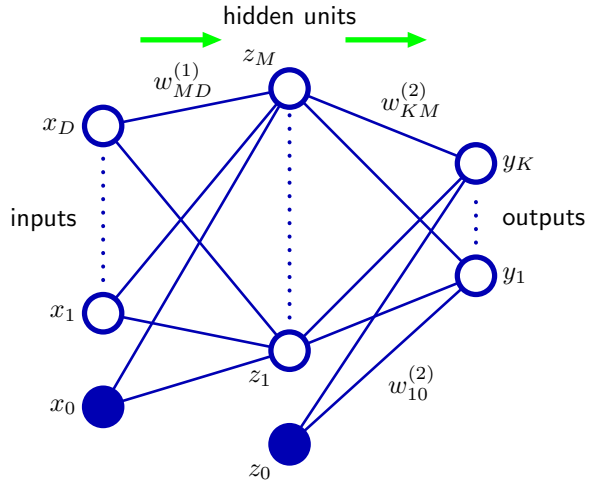
2006 Deep nets



Neural networks are very powerful

Sufficient

Universal approximator: with sufficient number of nonlinear hidden units, linear output unit can approximate any continuous functions



Transfer function for the neurons

sigmoid function

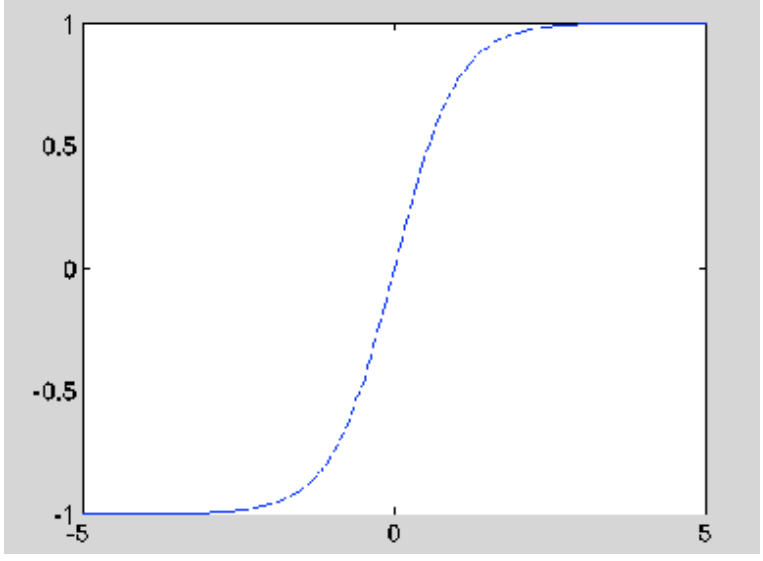
$$h(z) = \frac{1}{1 + e^{-z}}$$

tanh function:

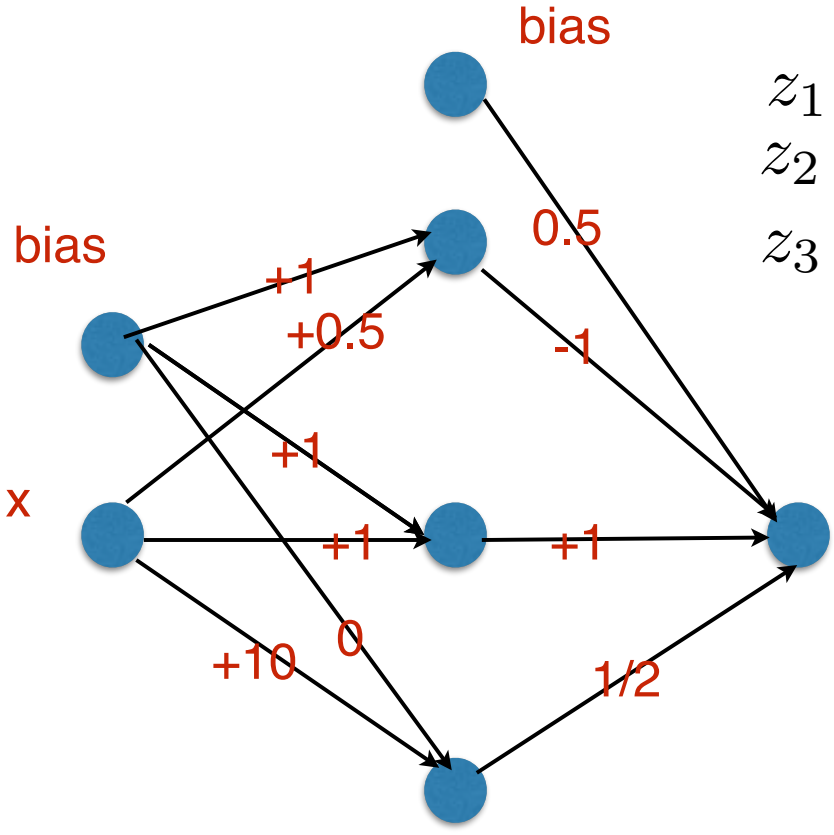
$$h(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

piecewise linear/rectified linear units

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



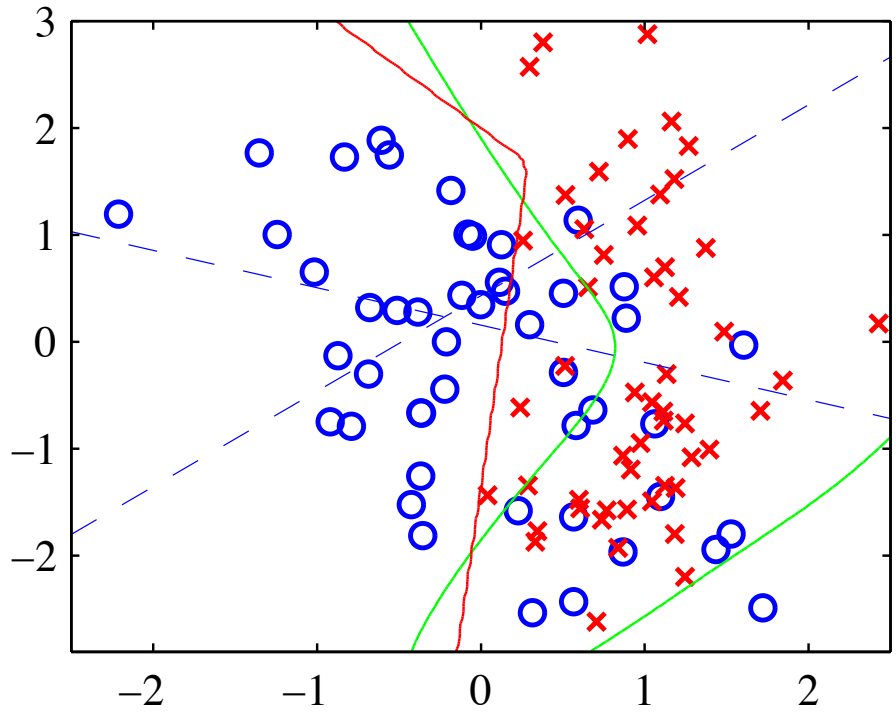
Ex: computing highly nonlinear function



$$z_1 = h(0.5x + 1)$$
$$z_2 = h(x + 1)$$
$$z_3 = h(10x)$$

$$y = -z_1 + z_2 + 0.5 * z3 + 0.5$$

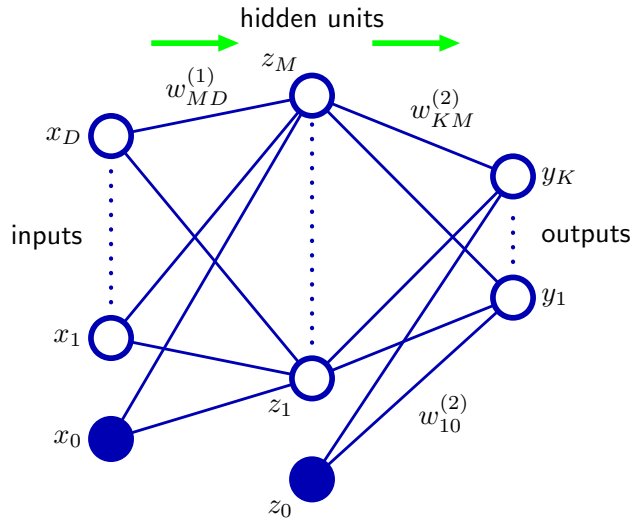
Complicated decision boundaries



Choice of output nodes

Regression

Linear output $$y_k = \sum_k w_{kj}^{(2)} h\left(\sum_i w_{ji}^{(1)} x_i\right)$$



Classification

sigmoid (for binary classification)

$$y = \sigma\left(\sum_k w_{kj}^{(2)} h\left(\sum_i w_{ji}^{(1)} x_i\right)\right)$$

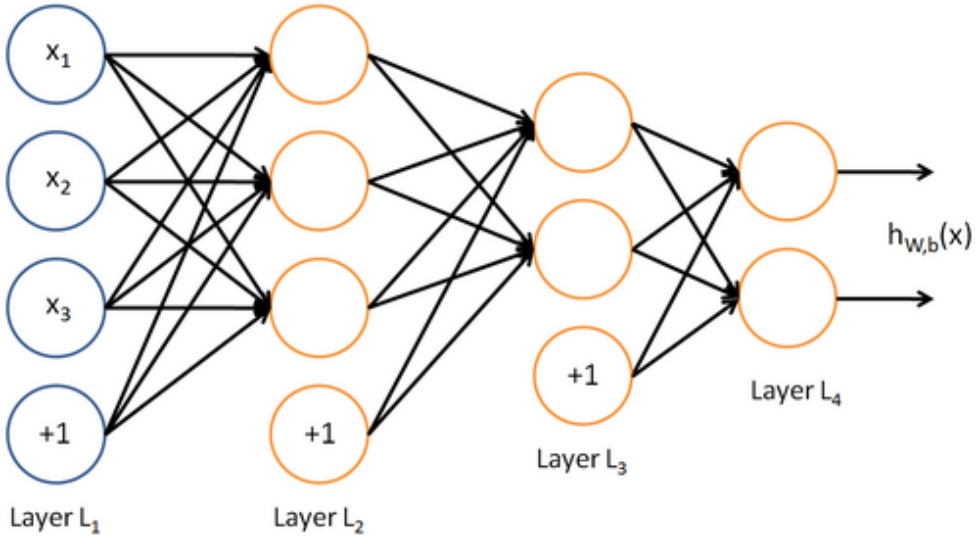
softmax (for multiclass classification)

$$z_k = \sum_k w_{kj}^{(2)} h\left(\sum_i w_{ji}^{(1)} x_i\right) \qquad y_k = \frac{e^{z_k}}{\sum_{k'} e^{z_{k'}}}$$

Can have multiple (ie, deep) layers

Implements highly complicated nonlinear mapping

$$\boldsymbol{y} = f(\boldsymbol{x})$$



We can this type architecture deep
neural networks, deep nets, or multi-layer perceptrons

How to learn the parameters?

Choose the right loss function

Regression: least-square loss

$$\min \sum_n (f(\boldsymbol{x}_n) - y_n)^2$$

Classification: cross-entropy loss

$$\min -\sum_n \sum_k y_{nk} \log f_k(\boldsymbol{x}_n)$$

Very hard optimization problem

Stochastic gradient descent is commonly used

Many optimization tricks are applied

Stochastic gradient descent

High-level idea

Randomly pick a data point (x_n, y_n)

Compute the gradient using only this data point, for example,

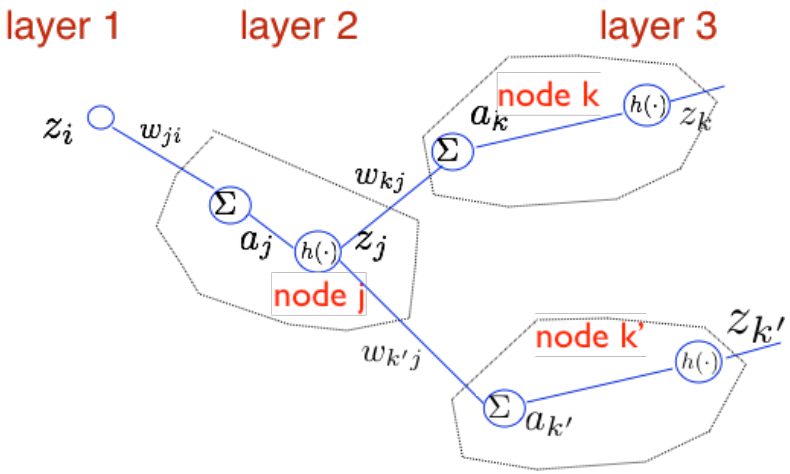
$$g = \frac{\partial [f(\boldsymbol{x}_n) - y_n)^2]}{\partial \boldsymbol{w}}$$

Update the parameter right away

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta g$$

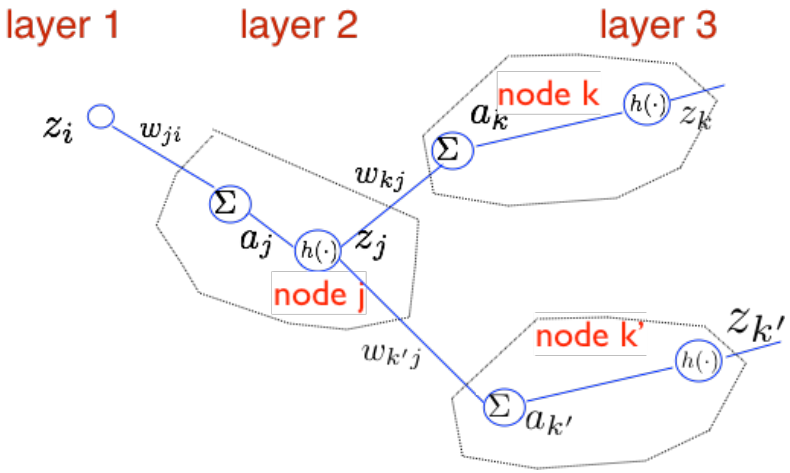
Iterate the process until some stop criteria

Derivation of the error-backpropagation



Key steps (essentially, chain rule in calculus)

To compute

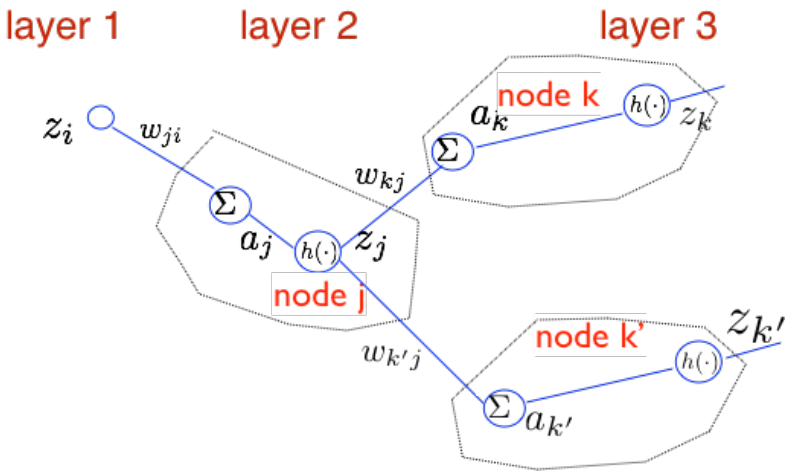


$$\frac{\partial \ell}{\partial w_{ji}} = \frac{\partial \ell}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}} = z_i \frac{\partial \ell}{\partial a_j}$$

as w_{ji} affects only a_j

Key steps (essentially, chain rule in calculus)

To compute



$$\frac{\partial \ell}{\partial w_{ji}} = \frac{\partial \ell}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}} = z_i \frac{\partial \ell}{\partial a_j}$$

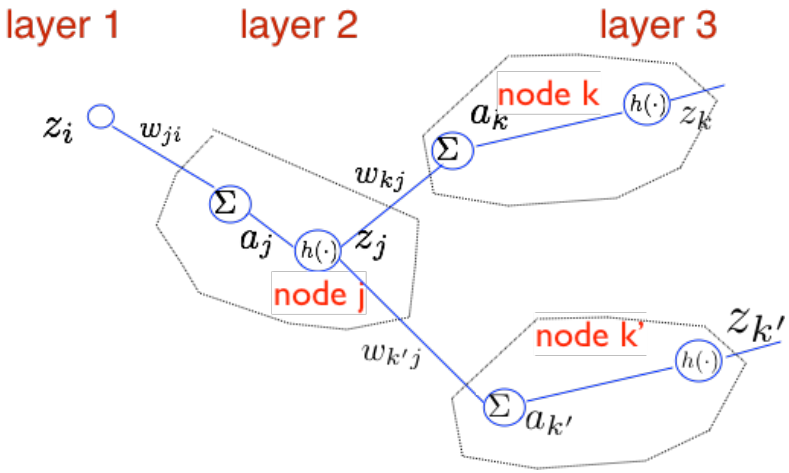
as w_{ji} affects only a_j

Passthru the nonlinear transfer function

$$\frac{\partial \ell}{\partial a_j} = \frac{\partial \ell}{\partial z_j}\frac{\partial z_j}{\partial a_j} = h'(a_j)\frac{\partial \ell}{\partial z_j}$$

Key steps (essentially, chain rule in calculus)

To compute



$$\frac{\partial \ell}{\partial w_{ji}} = \frac{\partial \ell}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}} = z_i \frac{\partial \ell}{\partial a_j}$$

as w_{ji} affects only a_j

Passthru the nonlinear transfer function

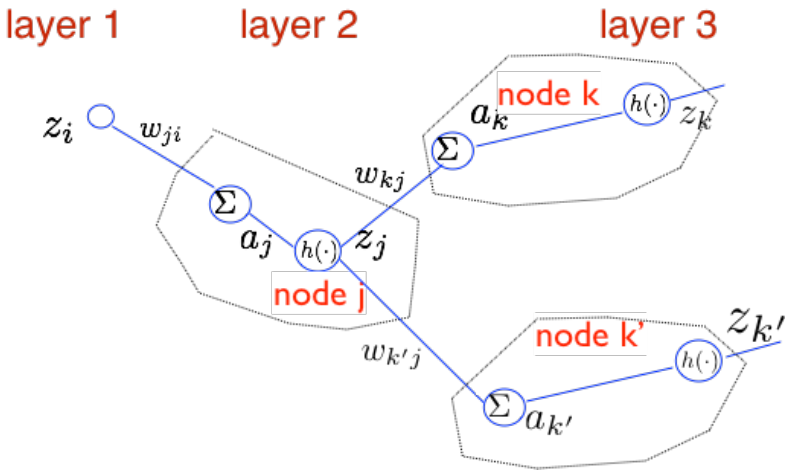
$$\frac{\partial \ell}{\partial a_j} = \frac{\partial \ell}{\partial z_j}\frac{\partial z_j}{\partial a_j} = h'(a_j)\frac{\partial \ell}{\partial z_j}$$

Collect errors from next layer

$$\frac{\partial \ell}{\partial z_j} = \sum_k \frac{\partial \ell}{\partial a_k}\frac{\partial a_k}{\partial z_j} = \sum_k \frac{\partial \ell}{\partial a_k} w_{kj}$$

Key steps (essentially, chain rule in calculus)

To compute



$$\frac{\partial \ell}{\partial w_{ji}} = \frac{\partial \ell}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}} = z_i \frac{\partial \ell}{\partial a_j}$$

as w_{ji} affects only a_j

Passthru the nonlinear transfer function

$$\frac{\partial \ell}{\partial a_j} = \frac{\partial \ell}{\partial z_j}\frac{\partial z_j}{\partial a_j} = h'(a_j)\frac{\partial \ell}{\partial z_j}$$

Collect errors from next layer

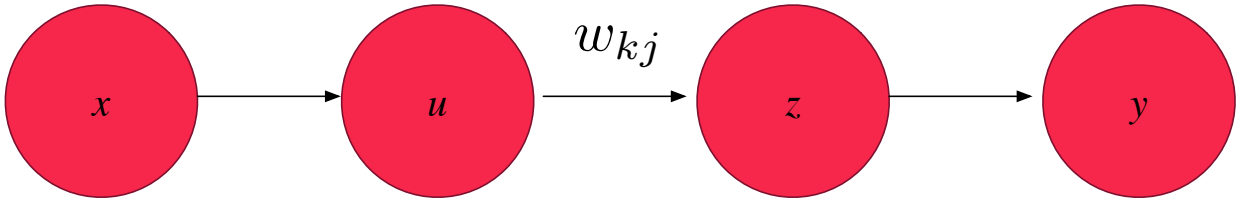
$$\frac{\partial \ell}{\partial z_j} = \sum_k \frac{\partial \ell}{\partial a_k}\frac{\partial a_k}{\partial z_j} = \sum_k \frac{\partial \ell}{\partial a_k} w_{kj}$$

Recursion

$$\frac{\partial \ell}{\partial a_j} = h'(a_j)\sum_k \frac{\partial \ell}{\partial a_k} w_{kj}$$

Challenge: How to compute the first error?

Example: a multi-layer neural net for multinomial classification



$$a_k = \sum_j w_{kj} u_j \tag{4}$$

$$z_k = a_k \tag{5}$$

$$y_k = \log p(y = k|x) = \frac{e^{z_k}}{\sum_{k'} e^{z_{k'}}} \tag{6}$$

So what is

$$\frac{\partial \ell}{\partial a_k}$$

with the cross-entropy loss

$$\ell = -\sum_k t_k \log y_k$$

Overfitting is very possible

Methods for overcoming overfitting

- Regularization
- Early stopping
- Data augmentation
- Inject noise

Regularization: weight decay in neural nets

Original loss function, eg:

$$\ell(\boldsymbol{w}) = -\sum_n \sum_k y_{nk} \log p(y = k|x_n)$$

Adding L_2 norm to regularize

$$\ell'(\boldsymbol{w}) = \ell(\boldsymbol{w}) + \frac{1}{2}\lambda \|\boldsymbol{w}\|_2^2$$

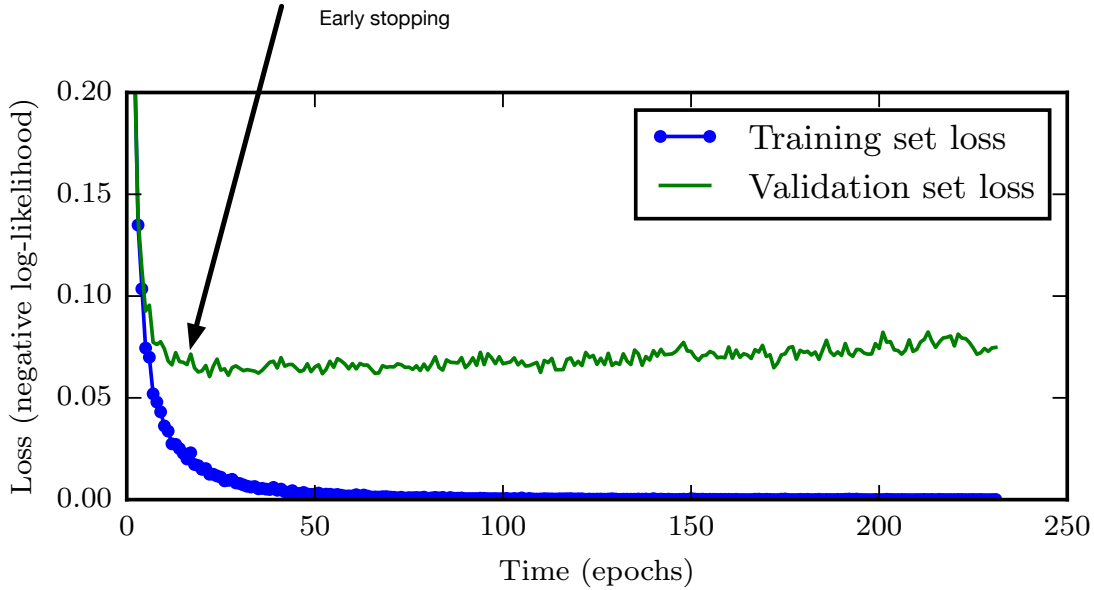
Gradient changed to

$$\frac{\partial \ell'(\boldsymbol{w})}{\partial \boldsymbol{w}} = \frac{\partial \ell(\boldsymbol{w})}{\partial \boldsymbol{w}} + \lambda \boldsymbol{w}$$

Effect: (slowly) decaying $\boldsymbol{w}$ to zero.

Early stopping

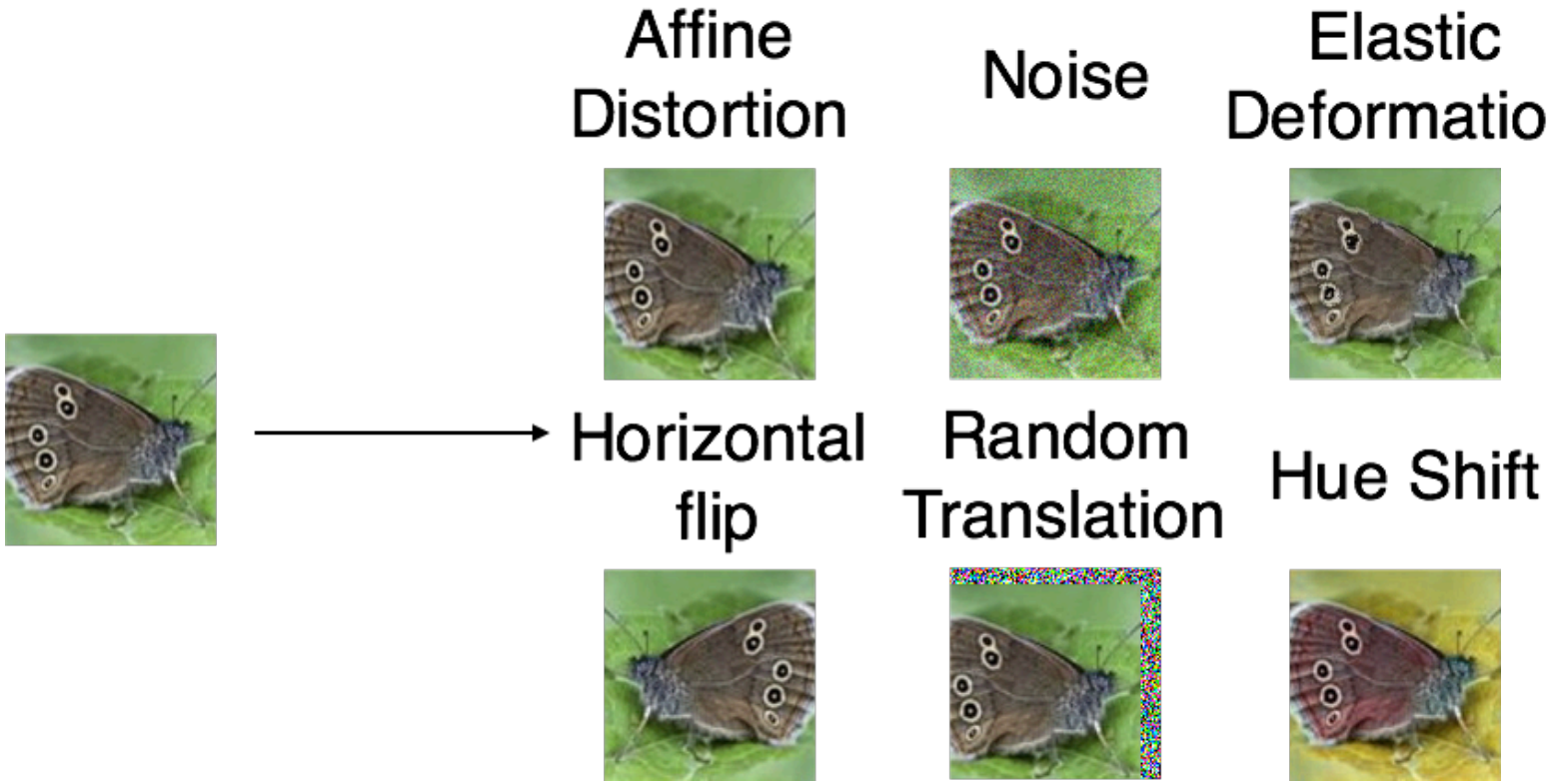
Stopping training before the training error is reduced too much



Thus, we should *monitor* the change of performance on the validation set every iteration.

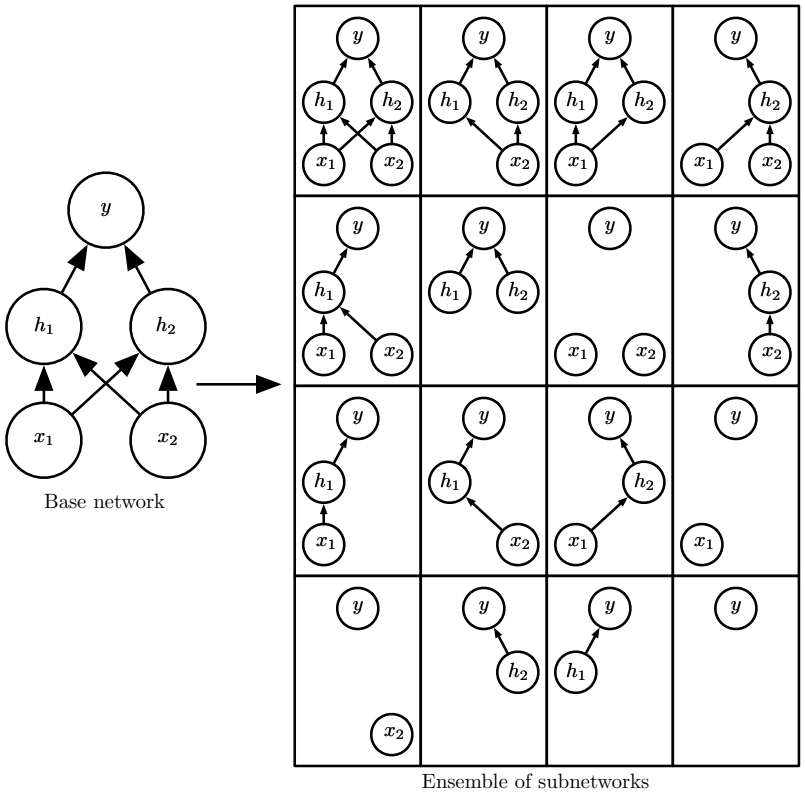
Data Augmentation

Exploiting prior knowledge to add more labeled data



Injecting noise

During training, randomly delete nodes (this is called *dropout*)



Optimization in neural networks is challenging

Common approaches

- Better initialization
- Stochastic gradient descent with minibatch
- Stochastic gradient descent with momentum
- Adaptive learning rate

Stochastic Gradient Descent with minibatch

Recipe

- Decide a minibatch size m
- Initialize $\boldsymbol{w}$ to $\boldsymbol{w}^{(0)}$; set $t = 0$; choose $\eta > 0$
- Loop *until convergence*
  1. random choose m training samples $\boldsymbol{x}_1$, $\boldsymbol{x}_2$, ..., $\boldsymbol{x}_m$
  2. Compute their contribution to the gradient

$$\boldsymbol{g} = \frac{1}{m} \sum_i \frac{\partial \ell(\boldsymbol{x}_m, y_m)}{\partial \boldsymbol{w}}$$

  3. Update the parameters $\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \eta \boldsymbol{g}$
  4. $t \leftarrow t + 1$

How to choose m?

Stochastic Gradient Descent with Momentum

Intuition

Stochastic gradient descent is "greedy" in using the current minibatch of samples.

Can we use the past gradient to help us? Namely how the parameters move in the past (the velocity) could be useful to us.

Recipe

- Decide a minibatch size m, a momentum strength of $\alpha \in (0, 1)$
- Initialize $\boldsymbol{w}$ to $\boldsymbol{w}^{(0)}$, initialize the velocity vector $\boldsymbol{v}$ (say, 0); set $t = 0$; choose $\eta > 0$
- Loop *until convergence*
  1. random choose m training samples $\boldsymbol{x}_1$, $\boldsymbol{x}_2$, ..., $\boldsymbol{x}_m$
  2. Compute their contribution to the gradient

$$\boldsymbol{g} = \frac{1}{m} \sum_i \frac{\partial \ell(\boldsymbol{x}_m, y_m)}{\partial \boldsymbol{w}}$$

  3. Update the velocity

$$\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \eta \boldsymbol{g}$$

  4. Update the parameters $\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} + \boldsymbol{v}$
  5. $t \leftarrow t + 1$

How history of update can affect present

Assume initial velocity is 0

time	velocity $\boldsymbol{v}$	how much parameters are to be updated
0	$\mathbf{0}$	$-\eta \boldsymbol{g}^1$
1	$-\eta \boldsymbol{g}^1$	$-\alpha\eta \boldsymbol{g}^1 - \eta \boldsymbol{g}^2$
2	$-\alpha\eta \boldsymbol{g}^1 - \eta \boldsymbol{g}^2$	$-\alpha^2\eta \boldsymbol{g}^1 - \alpha\eta \boldsymbol{g}^2 - \eta \boldsymbol{g}^3$

$\boldsymbol{g}^1$ stands for the gradient computed on the first minibatch and $\boldsymbol{g}^2$ stands for the gradient computed on the second batch, so on and so forth.

Challenge At time t, what is the parameter vector (if the initial parameter vector is $\boldsymbol{w}^0$) – expressed in all the $\boldsymbol{g}^t$s?

Outline

Summary

- Deep neural networks are hugely popular.
- They achieve stellar performance on many problems.
- They do need a lot of data points to work well.
- They do have optimization headaches to overcome: local optima, convergence, hyperparameters (minibatch size, step size, momentum, etc)
- They do tend to require heavy computation: use a GPU for massive parallel computing
- Selecting good models takes time: how many hidden layers, how many units/neurons each layer etc

Good reference: Goodfellow, Bengio and Courville's textbook *Deep Learning* `www.deeplearningbook.org`

Acknowledgement: several figures from this lecture are borrowed from the lecture slides on the website `www.deeplearningbook.org`