lecture 5: learning deep learning for vision

Yannis Avrithis

Inria Rennes-Bretagne Atlantique

Rennes, Nov. 2017 - Jan. 2018



◆□▶ ◆□▶ ◆ □▶ ◆ □▶ ○ □ ○ ○ ○ ○

outline

machine learning binary classification binary classification, again multi-class classification regression multiple layers



supervised learning

• learn to map an input to a target output, which can be discrete (classification) or continuous (regression)

unsupervised learning

- learn a compact representation of the data that can be useful for other tasks, *e.g.* density estimation, clustering, sampling, dimension reduction, manifold learning
- but: in many cases, labels can be obtained automatically, transforming an unsupervised task to supervised

• also: semi-supervised, weakly supervised, ambiguous/noisy labels, self-supervised *etc*.

- learn to select actions, supervised by occasional rewards
- not studied here

supervised learning

• learn to map an input to a target output, which can be discrete (classification) or continuous (regression)

unsupervised learning

- learn a compact representation of the data that can be useful for other tasks, *e.g.* density estimation, clustering, sampling, dimension reduction, manifold learning
- but: in many cases, labels can be obtained automatically, transforming an unsupervised task to supervised

• also: semi-supervised, weakly supervised, ambiguous/noisy labels, self-supervised *etc*.

- learn to select actions, supervised by occasional rewards
- not studied here

supervised learning

• learn to map an input to a target output, which can be discrete (classification) or continuous (regression)

unsupervised learning

- learn a compact representation of the data that can be useful for other tasks, *e.g.* density estimation, clustering, sampling, dimension reduction, manifold learning
- but: in many cases, labels can be obtained automatically, transforming an unsupervised task to supervised

• also: semi-supervised, weakly supervised, ambiguous/noisy labels, self-supervised *etc*.

- learn to select actions, supervised by occasional rewards
- not studied here

supervised learning

• learn to map an input to a target output, which can be discrete (classification) or continuous (regression)

unsupervised learning

- learn a compact representation of the data that can be useful for other tasks, *e.g.* density estimation, clustering, sampling, dimension reduction, manifold learning
- but: in many cases, labels can be obtained automatically, transforming an unsupervised task to supervised

• also: semi-supervised, weakly supervised, ambiguous/noisy labels, self-supervised *etc*.

- learn to select actions, supervised by occasional rewards
- not studied here

- through a learning task/objective that may be unimportant, we are primarily interested in learning good representations for computer vision tasks
- we are interested in parametric models where we learn a fixed set of parameters, rather than non-parametric, where training data are memorized
- we are interested in learning explicit mappings from raw input to representation, rather than constructing a representation of an entire dataset that is hard to extend to new samples

 we may occasionally use "hand-crafted" features or matching methods, but with the objective of learning better ones

- through a learning task/objective that may be unimportant, we are primarily interested in learning good representations for computer vision tasks
- we are interested in parametric models where we learn a fixed set of parameters, rather than non-parametric, where training data are memorized
- we are interested in learning explicit mappings from raw input to representation, rather than constructing a representation of an entire dataset that is hard to extend to new samples

• we may occasionally use "hand-crafted" features or matching methods, but with the objective of learning better ones

- through a learning task/objective that may be unimportant, we are primarily interested in learning good representations for computer vision tasks
- we are interested in parametric models where we learn a fixed set of parameters, rather than non-parametric, where training data are memorized
- we are interested in learning explicit mappings from raw input to representation, rather than constructing a representation of an entire dataset that is hard to extend to new samples

 we may occasionally use "hand-crafted" features or matching methods, but with the objective of learning better ones

- through a learning task/objective that may be unimportant, we are primarily interested in learning good representations for computer vision tasks
- we are interested in parametric models where we learn a fixed set of parameters, rather than non-parametric, where training data are memorized
- we are interested in learning explicit mappings from raw input to representation, rather than constructing a representation of an entire dataset that is hard to extend to new samples

• we may occasionally use "hand-crafted" features or matching methods, but with the objective of learning better ones

learning and optimization

• in a supervised setting, given a distribution p of input data \mathbf{x} and target outputs t we want to learn the parameters $\boldsymbol{\theta}$ of a prediction model $f(\mathbf{x}, \boldsymbol{\theta})$ by minimizing the risk (objective, cost, or error) function

$$E^*(\boldsymbol{\theta}) := \mathbb{E}_{(\mathbf{x},t) \sim p} L(f(\mathbf{x}; \boldsymbol{\theta}), t)$$

where L is a per-sample loss function that compares predictions $f(\mathbf{x}; \pmb{\theta})$ to targets t

since the true distribution p is unknown, we use the empirical distribution p̂ of a training set x₁,..., x_m with associated target outputs t₁,..., t_n and minimize instead the empirical risk

$$E(\boldsymbol{\theta}) := \mathbb{E}_{(\mathbf{x},t)\sim\hat{p}} L(f(\mathbf{x};\boldsymbol{\theta}),t) = \frac{1}{n} \sum_{i=1}^{n} L(f(\mathbf{x}_i;\boldsymbol{\theta}),t_i),$$

converting the learning problem to optimization

learning and optimization

• in a supervised setting, given a distribution p of input data \mathbf{x} and target outputs t we want to learn the parameters $\boldsymbol{\theta}$ of a prediction model $f(\mathbf{x}, \boldsymbol{\theta})$ by minimizing the risk (objective, cost, or error) function

$$E^*(\boldsymbol{\theta}) := \mathbb{E}_{(\mathbf{x},t) \sim p} L(f(\mathbf{x}; \boldsymbol{\theta}), t)$$

where L is a per-sample loss function that compares predictions $f(\mathbf{x}; \pmb{\theta})$ to targets t

• since the true distribution p is unknown, we use the empirical distribution \hat{p} of a training set $\mathbf{x}_1, \ldots, \mathbf{x}_m$ with associated target outputs t_1, \ldots, t_n and minimize instead the empirical risk

$$E(\boldsymbol{\theta}) := \mathbb{E}_{(\mathbf{x},t)\sim\hat{p}} L(f(\mathbf{x};\boldsymbol{\theta}),t) = \frac{1}{n} \sum_{i=1}^{n} L(f(\mathbf{x}_{i};\boldsymbol{\theta}),t_{i}),$$

converting the learning problem to optimization

• the empirical risk is prone to overfitting the training set (even memorizing it), if non-parametric

- we need to balance our model's capacity with the amount of training data, find ways to regularize the objective function and use a validation set to select hyperparameters so that our model generalizes on new samples
- the ideal loss function may be hard to optimize, so we have to use a surrogate loss function that may as well improve generalization
- still, all functions encountered are non-convex so we can only hope for local minima

- the empirical risk is prone to overfitting the training set (even memorizing it), if non-parametric
- we need to balance our model's capacity with the amount of training data, find ways to regularize the objective function and use a validation set to select hyperparameters so that our model generalizes on new samples
- the ideal loss function may be hard to optimize, so we have to use a surrogate loss function that may as well improve generalization
- still, all functions encountered are non-convex so we can only hope for local minima

- the empirical risk is prone to overfitting the training set (even memorizing it), if non-parametric
- we need to balance our model's capacity with the amount of training data, find ways to regularize the objective function and use a validation set to select hyperparameters so that our model generalizes on new samples
- the ideal loss function may be hard to optimize, so we have to use a surrogate loss function that may as well improve generalization
- still, all functions encountered are non-convex so we can only hope for local minima

- the empirical risk is prone to overfitting the training set (even memorizing it), if non-parametric
- we need to balance our model's capacity with the amount of training data, find ways to regularize the objective function and use a validation set to select hyperparameters so that our model generalizes on new samples
- the ideal loss function may be hard to optimize, so we have to use a surrogate loss function that may as well improve generalization
- still, all functions encountered are non-convex so we can only hope for local minima

binary classification

k-nearest neighbor classifier



- an input sample is classified by majority voting (ties broken at random) over the class labels of its k-nearest neighbors in the training set
- no training needed, but prediction can be slow
- we are not interested in such an approach (for now) because it gives us no opportunity to learn a representation

Image credit: Bishop 2006. Pattern Recognition and Machine Learning.

k-nearest neighbor classifier



- an input sample is classified by majority voting (ties broken at random) over the class labels of its k-nearest neighbors in the training set
- no training needed, but prediction can be slow
- we are not interested in such an approach (for now) because it gives us no opportunity to learn a representation

Image credit: Bishop 2006. Pattern Recognition and Machine Learning.

perceptron [Rosenblatt 1962]



- perceptron, as introduced by Rosenblatt, refers to a wide range of network architectures, learning algorithms and hardware implementations
- due to Minsky and Papert, perceptron now refers to a binary linear classifier and an algorithm
- let's have a closer look at that

perceptron model

- given input $\mathbf{x} \in \mathbb{R}^d$, the perceptron is a generalized linear model

$$y = f(\mathbf{x}; \mathbf{w}) := \operatorname{sgn}(\mathbf{w}^{\top} \mathbf{x})$$

where $\mathbf{w} \in \mathbb{R}^d$ is a weight (parameter) vector to be learned, and

$$sgn(x) := \begin{cases} +1, & x \ge 0\\ -1, & x < 0 \end{cases}$$



▲ロ → ▲周 → ▲目 → ▲目 → ● ● ● ● ●

Rosenblatt 1962. Principles of Neurodynamics

- an input x with output $y = f(\mathbf{x}; \mathbf{w})$ is classified to class C_1 if y = 1 and to C_2 if y = -1
- given a training sample $\mathbf{x} \in \mathbb{R}^d$ and a target variable $s \in \{-1, 1\}$, \mathbf{x} is correctly classified iff output $y = f(\mathbf{x}; \mathbf{w})$ equals s, *i.e.* sy > 0
- we are given training samples $x_1, \ldots, x_n \in \mathbb{R}^d$ and target variables $s_1, \ldots, s_n \in \{-1, 1\}$
- starting from an initial parameter vector $\mathbf{w}^{(0)}$, the algorithm learns by iteratively choosing a random sample \mathbf{x}_i that is misclassified and updating

$$\mathbf{w}^{(\tau+1)} \leftarrow \mathbf{w}^{(\tau)} + \epsilon s_i \mathbf{x}_i$$

(日) (同) (三) (三) (三) (○) (○)

- an input x with output y = f(x; w) is classified to class C_1 if y = 1and to C_2 if y = -1
- given a training sample $\mathbf{x} \in \mathbb{R}^d$ and a target variable $s \in \{-1, 1\}$, \mathbf{x} is correctly classified iff output $y = f(\mathbf{x}; \mathbf{w})$ equals s, *i.e.* sy > 0
- we are given training samples $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$ and target variables $s_1, \ldots, s_n \in \{-1, 1\}$
- starting from an initial parameter vector $\mathbf{w}^{(0)}$, the algorithm learns by iteratively choosing a random sample \mathbf{x}_i that is misclassified and updating

$$\mathbf{w}^{(\tau+1)} \leftarrow \mathbf{w}^{(\tau)} + \epsilon s_i \mathbf{x}_i$$



• initial parameter vector \mathbf{w}_0 , normal to the decision boundary and pointing to the region to be classified as blue (+)



- pick a random point \mathbf{x}_0 that is misclassified: blue (+) in red (-) region



• because \mathbf{x}_0 is blue and \mathbf{w} is pointing at blue, we add $\epsilon \mathbf{x}_0$ to \mathbf{w}_0

▲□▶ ▲□▶ ▲目▶ ▲目▶ 二目 - わへぐ



• with the new parameter vector \mathbf{w}_1 , the decision boundary is updated

▲□▶ ▲□▶ ▲目▶ ▲目▶ 二目 - わへぐ



- pick a new random point \mathbf{x}_1 that is misclassified: red in blue region

▲□▶ ▲□▶ ▲目▶ ▲目▶ 二目 - わへぐ



• because x_1 is red and w is pointing at blue, we subtract ϵx_1 from w_1

◆□▶ ◆□▶ ◆臣▶ ◆臣▶ □臣 = のへぐ



• with the new \mathbf{w}_2 , the decision boundary is updated again



- again, random point \mathbf{x}_2 , blue misclassified in red region



• and we add $\epsilon \mathbf{x}_2$ to \mathbf{w}_2



• now at \mathbf{w}_3



• one last random point x_3 , red in blue region



• and we subtract
perceptron algorithm



イロト 不得 トイヨト イヨト ヨー うへつ

• finally at \mathbf{w}_4 , all points are classified correctly

perceptron algorithm



イロト 不得 トイヨト イヨト ヨー うへつ

• finally at \mathbf{w}_4 , all points are classified correctly

perceptron algorithm



イロト 不得 トイヨト イヨト ヨー うへつ

• finally at \mathbf{w}_4 , all points are classified correctly

- we do not say anything about convergence now; we will discuss later
- there is one more parameter to be learned: a more general linear model would be

$$y = f(\mathbf{x}; \mathbf{w}, b) := \operatorname{sgn}(\mathbf{w}^{\top}\mathbf{x} + b)$$

where $\mathbf{w} \in \mathbb{R}^d$ is a weight vector, and b is a bias

- this is often omitted because we can just add an extra dimension d+1 to x and w and always set $x_{d+1} = 1$; then w_{d+1} plays the role of bias
- but in many cases weights and bias need separate treatment
- it is common to use a (fixed) set of basis functions on the raw input and write $\phi({\bf x})$ instead of ${\bf x}$
- the linear model itself is not affected by this choice, but the classifier is; again, we discuss this later

- we do not say anything about convergence now; we will discuss later
- there is one more parameter to be learned: a more general linear model would be

$$y = f(\mathbf{x}; \mathbf{w}, b) := \operatorname{sgn}(\mathbf{w}^{\top}\mathbf{x} + b)$$

where $\mathbf{w} \in \mathbb{R}^d$ is a weight vector, and b is a bias

- this is often omitted because we can just add an extra dimension d+1 to x and w and always set x_{d+1} = 1; then w_{d+1} plays the role of bias
- but in many cases weights and bias need separate treatment
- it is common to use a (fixed) set of basis functions on the raw input and write $\phi({\bf x})$ instead of ${\bf x}$
- the linear model itself is not affected by this choice, but the classifier is; again, we discuss this later

ション 人口 マイビン トレート シックション

Rosenblatt 1962. Principles of Neurodynamics

- we do not say anything about convergence now; we will discuss later
- there is one more parameter to be learned: a more general linear model would be

$$y = f(\mathbf{x}; \mathbf{w}, b) := \operatorname{sgn}(\mathbf{w}^{\top} \mathbf{x} + b)$$

where $\mathbf{w} \in \mathbb{R}^d$ is a weight vector, and b is a bias

- this is often omitted because we can just add an extra dimension d+1 to x and w and always set $x_{d+1} = 1$; then w_{d+1} plays the role of bias
- but in many cases weights and bias need separate treatment
- it is common to use a (fixed) set of basis functions on the raw input and write $\phi(\mathbf{x})$ instead of \mathbf{x}
- the linear model itself is not affected by this choice, but the classifier is; again, we discuss this later

support vector machine (SVM) [Boser et al. 1992]

• given a decision boundary that classifies all points correctly, define the margin as its distance to the nearest point



• this was not optimal in the case of perceptron



• there is another decision boundary for which the margin is maximum; the vectors at this distance are the support vectors

SVM model

• there is now an explicit bias parameter *b*, but otherwise the SVM model is the same: activation

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

and output

$$y = f(\mathbf{x}; \mathbf{w}, b) := \operatorname{sgn}(\mathbf{w}^{\top}\mathbf{x} + b) = \operatorname{sgn}(a)$$

- again, an input \mathbf{x} with $a = \mathbf{w}^{\top}\mathbf{x} + b$ and output $y = \operatorname{sgn}(a)$ is classified to class C_1 if y = 1 ($a \ge 0$) and to C_2 if y = -1 (a < 0)
- again, given a training sample x and a target variable s, x is correctly classified iff sy > 0, i.e. sa = s(w^Tx + b) ≥ 0
- we are given training samples $x_1, \ldots, x_n \in \mathbb{R}^d$ and target variables $s_1, \ldots, s_n \in \{-1, 1\}$

SVM model

• there is now an explicit bias parameter b, but otherwise the SVM model is the same: activation

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

and output

$$y = f(\mathbf{x}; \mathbf{w}, b) := \operatorname{sgn}(\mathbf{w}^{\top}\mathbf{x} + b) = \operatorname{sgn}(a)$$

- again, an input x with $a = \mathbf{w}^{\top}\mathbf{x} + b$ and output $y = \operatorname{sgn}(a)$ is classified to class C_1 if y = 1 ($a \ge 0$) and to C_2 if y = -1 (a < 0)
- again, given a training sample x and a target variable s, x is correctly classified iff sy > 0, i.e. sa = s(w^Tx + b) ≥ 0
- we are given training samples $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$ and target variables $s_1, \ldots, s_n \in \{-1, 1\}$



- the distance of x to the boundary is $|\mathbf{w}^{\top}\mathbf{x} + b| / \|\mathbf{w}\|$
- this is $s(\mathbf{w}^{\top}\mathbf{x} + b) / \|\mathbf{w}\|$ if it is classified correctly
- if all points are classified correctly, then the margin is

$$\frac{1}{\|\mathbf{w}\|}\min_{i}(s_i(\mathbf{w}^{\top}\mathbf{x}_i+b))$$



- the distance of x to the boundary is $|\mathbf{w}^{\top}\mathbf{x} + b| / \|\mathbf{w}\|$
- this is $s(\mathbf{w}^{\top}\mathbf{x} + b)/\|\mathbf{w}\|$ if it is classified correctly

if all points are classified correctly, then the margin is

$$\frac{1}{\|\mathbf{w}\|}\min_{i}(s_i(\mathbf{w}^{\top}\mathbf{x}_i+b))$$



- the distance of x to the boundary is $|\mathbf{w}^{\top}\mathbf{x} + b| / \|\mathbf{w}\|$
- this is $s(\mathbf{w}^{\top}\mathbf{x} + b)/\|\mathbf{w}\|$ if it is classified correctly
- if all points are classified correctly, then the margin is

$$\frac{1}{\|\mathbf{w}\|}\min_{i}(s_i(\mathbf{w}^{\top}\mathbf{x}_i+b))$$

maximum margin

• the margin is invariant to scaling of \mathbf{w} and b, so we choose $s_i a_i = s_i (\mathbf{w}^\top \mathbf{x}_i + b) = 1$ for the point that is nearest to the boundary

then, the margin is maximized by

$$\arg\min_{\mathbf{w},b}\frac{1}{2}\|\mathbf{w}\|^2$$

subject to

 $s_i a_i \ge 1$

for all training samples *i*, where $a_i := \mathbf{w}^\top \mathbf{x}_i + b$

this is a quadratic programming problem

maximum margin

- the margin is invariant to scaling of \mathbf{w} and b, so we choose $s_i a_i = s_i (\mathbf{w}^\top \mathbf{x}_i + b) = 1$ for the point that is nearest to the boundary
- then, the margin is maximized by

$$\arg\min_{\mathbf{w},b}\frac{1}{2}\|\mathbf{w}\|^2$$

subject to

$$s_i a_i \ge 1$$

for all training samples *i*, where $a_i := \mathbf{w}^\top \mathbf{x}_i + b$

this is a quadratic programming problem

overlapping class distributions

[Cortes and Vapnik 1995]



 assuming that all training samples can be correctly classified is unrealistic



introduce slack variables ξ_i ≥ 0 that should be minimized; ξ_i ≤ 1 for correctly classified samples, ξ_i = 0 beyond the margin

overlapping class distributions

• the constraints $s_i a_i \ge 1$ are now replaced by

$$s_i a_i \ge 1 - \xi_i$$

$$\xi_i \ge 0$$

where $a_i := \mathbf{w}^\top \mathbf{x}_i + b$

• and the objective $rgmin_{\mathbf{w},b}rac{1}{2}\|\mathbf{w}\|^2$ is replaced by

$$\arg\min_{\mathbf{w},b} \frac{C}{n} \sum_{i=1}^{n} \xi_i + \frac{1}{2} \|\mathbf{w}\|^2$$

where hyperparameter ${\cal C}$ controls the trade-off between slack variables and margin

overlapping class distributions

• the constraints $s_i a_i \ge 1$ are now replaced by

$$s_i a_i \ge 1 - \xi_i$$

$$\xi_i \ge 0$$

where $a_i := \mathbf{w}^\top \mathbf{x}_i + b$

- and the objective $\arg\min_{\mathbf{w},b} \frac{1}{2} \|\mathbf{w}\|^2$ is replaced by

$$\arg\min_{\mathbf{w},b} \frac{C}{n} \sum_{i=1}^{n} \xi_i + \frac{1}{2} \|\mathbf{w}\|^2$$

where hyperparameter ${\cal C}$ controls the trade-off between slack variables and margin

- we do not say anything about how to solve this problem yet
- the standard treatment of SVM introduces Lagrange multipliers for the constraints and results in the dual formulation where coordinates only appear in dot products
- at this point, writing $\phi(\mathbf{x})$ instead of \mathbf{x} , gives rise to

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

- this kernel trick can make the classifier nonlinear assuming an appropriate positive-definite kernel function κ for the problem at hand
- we are not interested in this approach here because
 - we want to learn a parametric model and discard the training data after learning
 - we do not want to design a matching function κ any more than designing the representation ϕ ; we want to learn from raw data

イロト 不得 トイヨト イヨト ヨー ろくで

- we do not say anything about how to solve this problem yet
- the standard treatment of SVM introduces Lagrange multipliers for the constraints and results in the dual formulation where coordinates only appear in dot products
- at this point, writing $\phi(\mathbf{x})$ instead of $\mathbf{x},$ gives rise to

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

- this kernel trick can make the classifier nonlinear assuming an appropriate positive-definite kernel function κ for the problem at hand
- we are not interested in this approach here because
 - we want to learn a parametric model and discard the training data after learning
 - we do not want to design a matching function κ any more than designing the representation ϕ ; we want to learn from raw data

ション 人口 マイビン トレート シックション

- we do not say anything about how to solve this problem yet
- the standard treatment of SVM introduces Lagrange multipliers for the constraints and results in the dual formulation where coordinates only appear in dot products
- at this point, writing $\phi(\mathbf{x})$ instead of \mathbf{x} gives rise to

• this kernel trick can make the classifier nonlinear assuming an appropriate positive-definite kernel function κ for the problem at hand

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

- we are not interested in this approach here because
 - we want to learn a parametric model and discard the training data after learning
 - we do not want to design a matching function κ any more than designing the representation ϕ ; we want to learn from raw data

(binary) logistic regression

[Cox 1958]

• again, activation (but here we omit the bias)

$$a = \mathbf{w}^\top \mathbf{x}$$

and output

$$y = f(\mathbf{x}; \mathbf{w}) := \sigma(\mathbf{w}^{\top} \mathbf{x}) = \sigma(a)$$

• but now we have a different nonlinearity: σ is the sigmoid function

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



(binary) logistic regression

[Cox 1958]

• again, activation (but here we omit the bias)

$$a = \mathbf{w}^\top \mathbf{x}$$

and output

$$y = f(\mathbf{x}; \mathbf{w}) := \sigma(\mathbf{w}^{\top} \mathbf{x}) = \sigma(a)$$

• but now we have a different nonlinearity: σ is the sigmoid function

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



▲ロ → ▲ 冊 → ▲ 目 → ▲ 目 → の Q ()



probabilistic interpretation

• the output y represents the posterior probability of class C₁ given input x, which by Bayes rule is

$$y = p(C_1 | \mathbf{x}) = \frac{p(\mathbf{x} | C_1) p(C_1)}{p(\mathbf{x} | C_1) p(C_1) + p(\mathbf{x} | C_2) p(C_2)}$$
$$= \frac{1}{1 + e^{-a}} = \sigma(a)$$

• here the activation a is defined to represent the log-odds

$$a = \ln \frac{p(C_1|\mathbf{x})}{p(C_2|\mathbf{x})} = \ln \frac{p(\mathbf{x}|C_1)p(C_1)}{p(\mathbf{x}|C_2)p(C_2)}$$

ション 人口 マイビン トレート シックション

maximum likelihood

- we are given training samples $X = (\mathbf{x}_1, \dots, \mathbf{x}_n)$ with $\mathbf{x}_i \in \mathbb{R}^d$ and target variables $T = (t_1, \dots, t_n)$ with $t_i \in \{0, 1\}$
- watch out: target variables are in $\{0,1\}$ here, not $\{-1,1\}$
- the probabilistic interpretation allows us to define the learning objective: maximize the likelihood function

$$p(T|X, \mathbf{w}) = \prod_{i=1}^{n} y_i^{t_i} (1 - y_i)^{1 - t_i}$$

or, minimize the (average) cross-entropy error function

$$E(\mathbf{w}) := -\frac{1}{n} \sum_{i=1}^{n} (t_i \ln y_i + (1 - t_i) \ln(1 - y_i))$$

where
$$y_i = \sigma(a_i) = \sigma(\mathbf{w}^\top \mathbf{x}_i)$$

maximum likelihood

- we are given training samples $X = (\mathbf{x}_1, \dots, \mathbf{x}_n)$ with $\mathbf{x}_i \in \mathbb{R}^d$ and target variables $T = (t_1, \dots, t_n)$ with $t_i \in \{0, 1\}$
- watch out: target variables are in $\{0,1\}$ here, not $\{-1,1\}$
- the probabilistic interpretation allows us to define the learning objective: maximize the likelihood function

$$p(T|X, \mathbf{w}) = \prod_{i=1}^{n} y_i^{t_i} (1 - y_i)^{1 - t_i}$$

• or, minimize the (average) cross-entropy error function

$$E(\mathbf{w}) := -\frac{1}{n} \sum_{i=1}^{n} (t_i \ln y_i + (1 - t_i) \ln(1 - y_i))$$

where
$$y_i = \sigma(a_i) = \sigma(\mathbf{w}^\top \mathbf{x}_i)$$

maximum likelihood

- we are given training samples $X = (\mathbf{x}_1, \dots, \mathbf{x}_n)$ with $\mathbf{x}_i \in \mathbb{R}^d$ and target variables $T = (t_1, \dots, t_n)$ with $t_i \in \{0, 1\}$
- watch out: target variables are in $\{0,1\}$ here, not $\{-1,1\}$
- the probabilistic interpretation allows us to define the learning objective: maximize the likelihood function

$$p(T|X, \mathbf{w}) = \prod_{i=1}^{n} y_i^{t_i} (1 - y_i)^{1 - t_i}$$

• or, minimize the (average) cross-entropy error function

$$E(\mathbf{w}) := -\frac{1}{n} \sum_{i=1}^{n} (t_i \ln y_i + (1 - t_i) \ln(1 - y_i))$$

where
$$y_i = \sigma(a_i) = \sigma(\mathbf{w}^\top \mathbf{x}_i)$$



raw data

・ロン ・聞と ・注と ・注と

Э



perceptron

イロト イ団ト イヨト イヨト 三臣



SVM

メロト メポト メヨト メヨト 二百



logistic regression

イロト イポト イモト イモト

э

binary classification, again

three solutions so far

	perceptron	SVM	logistic
objective		yes	yes
constraints	—	yes	—
regularizer	—	yes	—
algorithm	yes	—	—
probabilistic			yes

perceptron, again

"choose a random sample i that is misclassified and update"

$$\mathbf{w}^{(\tau+1)} \leftarrow \mathbf{w}^{(\tau)} + \epsilon s_i \mathbf{x}_i$$

• given sample \mathbf{x}_i , if $s_i y_i > 0$ (*i.e.* $s_i a_i \ge 0$) the sample is correctly classified and there is no action; otherwise, we attempt to minimize $-s_i a_i = -s_i \mathbf{w}^\top \mathbf{x}_i$: the error function is

$$E(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^{n} E_i(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^{n} [-s_i a_i]_+ = \frac{1}{n} \sum_{i=1}^{n} [-s_i \mathbf{w}^\top \mathbf{x}_i]_+$$

• indeed, given any random sample \mathbf{x}_i (correctly classified or not), the update is

$$\mathbf{w}^{(\tau+1)} \leftarrow \mathbf{w}^{(\tau)} - \epsilon \nabla_{\mathbf{w}} E_i(\mathbf{w}^{(\tau)})$$

◆□▶ ◆□▶ ◆目▶ ◆目▶ ◆□▶
perceptron, again

"choose a random sample i that is misclassified and update"

$$\mathbf{w}^{(\tau+1)} \leftarrow \mathbf{w}^{(\tau)} + \epsilon s_i \mathbf{x}_i$$

• given sample \mathbf{x}_i , if $s_i y_i > 0$ (*i.e.* $s_i a_i \ge 0$) the sample is correctly classified and there is no action; otherwise, we attempt to minimize $-s_i a_i = -s_i \mathbf{w}^\top \mathbf{x}_i$: the error function is

$$E(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^{n} E_i(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^{n} [-s_i a_i]_+ = \frac{1}{n} \sum_{i=1}^{n} [-s_i \mathbf{w}^\top \mathbf{x}_i]_+$$

• indeed, given any random sample \mathbf{x}_i (correctly classified or not), the update is

$$\mathbf{w}^{(\tau+1)} \leftarrow \mathbf{w}^{(\tau)} - \epsilon \nabla_{\mathbf{w}} E_i(\mathbf{w}^{(\tau)})$$

perceptron, again

"choose a random sample i that is misclassified and update"

$$\mathbf{w}^{(\tau+1)} \leftarrow \mathbf{w}^{(\tau)} + \epsilon s_i \mathbf{x}_i$$

• given sample \mathbf{x}_i , if $s_i y_i > 0$ (*i.e.* $s_i a_i \ge 0$) the sample is correctly classified and there is no action; otherwise, we attempt to minimize $-s_i a_i = -s_i \mathbf{w}^\top \mathbf{x}_i$: the error function is

$$E(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^{n} E_i(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^{n} [-s_i a_i]_+ = \frac{1}{n} \sum_{i=1}^{n} [-s_i \mathbf{w}^\top \mathbf{x}_i]_+$$

- indeed, given any random sample \mathbf{x}_i (correctly classified or not), the update is

$$\mathbf{w}^{(\tau+1)} \leftarrow \mathbf{w}^{(\tau)} - \epsilon \nabla_{\mathbf{w}} E_i(\mathbf{w}^{(\tau)})$$

positive part

• quantity $[x]_+$ is the positive part of x; this function also known as rectified linear unit (ReLU):

$$relu(x) := [x]_+ := max(0, x)$$



• in general, given an error function in parameters heta of the additive form

$$E(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^{n} E_i(\boldsymbol{\theta}),$$

• online (or stochastic) gradient descent updates the parameters after seeing one random sample *i*, according to

$$\boldsymbol{\theta}^{(\tau+1)} \leftarrow \boldsymbol{\theta}^{(\tau)} - \epsilon \nabla_{\boldsymbol{\theta}} E_i(\boldsymbol{\theta}^{(\tau)})$$

• batch gradient descent updates the parameters once after seeing the entire dataset, according to

$$\boldsymbol{\theta}^{(\tau+1)} \leftarrow \boldsymbol{\theta}^{(\tau)} - \epsilon \nabla_{\boldsymbol{\theta}} E(\boldsymbol{\theta}^{(\tau)})$$

• in general, given an error function in parameters heta of the additive form

$$E(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^{n} E_i(\boldsymbol{\theta}),$$

• online (or stochastic) gradient descent updates the parameters after seeing one random sample *i*, according to

$$\boldsymbol{\theta}^{(\tau+1)} \leftarrow \boldsymbol{\theta}^{(\tau)} - \epsilon \nabla_{\boldsymbol{\theta}} E_i(\boldsymbol{\theta}^{(\tau)})$$

• batch gradient descent updates the parameters once after seeing the entire dataset, according to

$$\boldsymbol{\theta}^{(\tau+1)} \leftarrow \boldsymbol{\theta}^{(\tau)} - \epsilon \nabla_{\boldsymbol{\theta}} E(\boldsymbol{\theta}^{(\tau)})$$

• in general, given an error function in parameters heta of the additive form

$$E(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^{n} E_i(\boldsymbol{\theta}),$$

• online (or stochastic) gradient descent updates the parameters after seeing one random sample *i*, according to

$$\boldsymbol{\theta}^{(\tau+1)} \leftarrow \boldsymbol{\theta}^{(\tau)} - \epsilon \nabla_{\boldsymbol{\theta}} E_i(\boldsymbol{\theta}^{(\tau)})$$

• batch gradient descent updates the parameters once after seeing the entire dataset, according to

$$\boldsymbol{\theta}^{(\tau+1)} \leftarrow \boldsymbol{\theta}^{(\tau)} - \epsilon \nabla_{\boldsymbol{\theta}} E(\boldsymbol{\theta}^{(\tau)})$$

 mini-batch (or stochastic) gradient descent (SGD) is the most common option and updates the parameters after seeing a random subset I ⊂ {1,...,n} of samples of fixed size m = |I| according to

$$\boldsymbol{\theta}^{(\tau+1)} \leftarrow \boldsymbol{\theta}^{(\tau)} - \epsilon \frac{1}{m} \sum_{i \in I} \nabla_{\boldsymbol{\theta}} E_i(\boldsymbol{\theta}^{(\tau)})$$

- ϵ is the learning rate and is a hyperparameter; we will discuss later the convergence to a local minimum of E and conditions on ϵ
- whatever the choice, an iteration over the entire dataset is called an epoch

- stochastic versions make more sense when dataset is redundant
- it is important to take random samples

 mini-batch (or stochastic) gradient descent (SGD) is the most common option and updates the parameters after seeing a random subset I ⊂ {1,...,n} of samples of fixed size m = |I| according to

$$\boldsymbol{\theta}^{(\tau+1)} \leftarrow \boldsymbol{\theta}^{(\tau)} - \epsilon \frac{1}{m} \sum_{i \in I} \nabla_{\boldsymbol{\theta}} E_i(\boldsymbol{\theta}^{(\tau)})$$

- ϵ is the learning rate and is a hyperparameter; we will discuss later the convergence to a local minimum of E and conditions on ϵ
- whatever the choice, an iteration over the entire dataset is called an epoch

- stochastic versions make more sense when dataset is redundant
- it is important to take random samples

 mini-batch (or stochastic) gradient descent (SGD) is the most common option and updates the parameters after seeing a random subset I ⊂ {1,...,n} of samples of fixed size m = |I| according to

$$\boldsymbol{\theta}^{(\tau+1)} \leftarrow \boldsymbol{\theta}^{(\tau)} - \epsilon \frac{1}{m} \sum_{i \in I} \nabla_{\boldsymbol{\theta}} E_i(\boldsymbol{\theta}^{(\tau)})$$

- ϵ is the learning rate and is a hyperparameter; we will discuss later the convergence to a local minimum of E and conditions on ϵ
- whatever the choice, an iteration over the entire dataset is called an epoch
- stochastic versions make more sense when dataset is redundant
- it is important to take random samples



• either $s_i a_i \ge 1$ and $\xi_i = 0$ (correct side of margin) or $\xi_i = 1 - s_i a_i$

the constraints

$$s_i a_i \ge 1 - \xi_i$$

$$\xi_i \ge 0$$

do not tell the whole truth

- either $s_i a_i \ge 1$ and $\xi_i = 0$ (correct side of margin) or $\xi_i = 1 s_i a_i$, that is, $\xi_i = [1 s_i a_i]_+$
- the error function becomes

$$E(\mathbf{w}, b) = \frac{1}{n} \sum_{i=1}^{n} [1 - s_i a_i]_+ + \frac{\lambda}{2} \|\mathbf{w}\|^2$$

without ξ_i and without constraints, where $\lambda = 1/C$

Cortes and Vapnik. ML 1995. Support-Vector Networks.

the constraints

$$s_i a_i \ge 1 - \xi_i$$
$$\xi_i \ge 0$$

do not tell the whole truth

• either $s_i a_i \ge 1$ and $\xi_i = 0$ (correct side of margin) or $\xi_i = 1 - s_i a_i$, that is, $\xi_i = [1 - s_i a_i]_+$

the error function becomes

$$E(\mathbf{w}, b) = \frac{1}{n} \sum_{i=1}^{n} [1 - s_i a_i]_+ + \frac{\lambda}{2} \|\mathbf{w}\|^2$$

without ξ_i and without constraints, where $\lambda = 1/C$

Cortes and Vapnik. ML 1995. Support-Vector Networks.

the constraints

$$s_i a_i \ge 1 - \xi_i$$

$$\xi_i \ge 0$$

do not tell the whole truth

- either $s_i a_i \ge 1$ and $\xi_i = 0$ (correct side of margin) or $\xi_i = 1 s_i a_i$, that is, $\xi_i = [1 s_i a_i]_+$
- the error function becomes

$$E(\mathbf{w}, b) = \frac{1}{n} \sum_{i=1}^{n} [1 - s_i a_i]_+ + \frac{\lambda}{2} \|\mathbf{w}\|^2$$

without ξ_i and without constraints, where $\lambda = 1/C$

Cortes and Vapnik. ML 1995. Support-Vector Networks.

- recall that the margin in SVM is invariant to scaling ${\bf w}$ and b
- same for perceptron error function
- in logistic regression, the sigmoid tends to a non-smooth step function as $\|\mathbf{w}\|$ becomes larger
- as $\|\mathbf{w}\|$ increases, the classifier function becomes more sensitive to perturbations in the input and is harder to generalize to new data
- the term

$$\frac{\lambda}{2} \|\mathbf{w}\|^2$$

helps to keep $\|\mathbf{w}\|$ low because its gradient is $-\lambda \mathbf{w}$; it is a standard regularization method and we can add it to any method including perceptron and logistic regression

- λ is another hyperparameter
- weight decay is only applied to weights, not to bias

- recall that the margin in SVM is invariant to scaling ${\bf w}$ and b
- same for perceptron error function
- in logistic regression, the sigmoid tends to a non-smooth step function as $\|\mathbf{w}\|$ becomes larger
- as $\|\mathbf{w}\|$ increases, the classifier function becomes more sensitive to perturbations in the input and is harder to generalize to new data
- the term



helps to keep $\|\mathbf{w}\|$ low because its gradient is $-\lambda \mathbf{w}$; it is a standard regularization method and we can add it to any method including perceptron and logistic regression

- λ is another hyperparameter
- weight decay is only applied to weights, not to bias

- recall that the margin in SVM is invariant to scaling ${\bf w}$ and b
- same for perceptron error function
- in logistic regression, the sigmoid tends to a non-smooth step function as $\|\mathbf{w}\|$ becomes larger
- as ||w|| increases, the classifier function becomes more sensitive to perturbations in the input and is harder to generalize to new data
- the term

$$\frac{\lambda}{2} \|\mathbf{w}\|^2$$

helps to keep $\|\mathbf{w}\|$ low because its gradient is $-\lambda \mathbf{w}$; it is a standard regularization method and we can add it to any method including perceptron and logistic regression

- λ is another hyperparameter
- weight decay is only applied to weights, not to bias

- recall that the margin in SVM is invariant to scaling ${\bf w}$ and b
- same for perceptron error function
- in logistic regression, the sigmoid tends to a non-smooth step function as $\|\mathbf{w}\|$ becomes larger
- as ||w|| increases, the classifier function becomes more sensitive to perturbations in the input and is harder to generalize to new data
- the term

$$\frac{\lambda}{2} \|\mathbf{w}\|^2$$

helps to keep $\|\mathbf{w}\|$ low because its gradient is $-\lambda \mathbf{w}$; it is a standard regularization method and we can add it to any method including perceptron and logistic regression

- λ is another hyperparameter
- weight decay is only applied to weights, not to bias

logistic regression, again

recall that

$$E(\mathbf{w}) := -\frac{1}{n} \sum_{i=1}^{n} (t_i \ln y_i + (1 - t_i) \ln(1 - y_i))$$

where
$$y_i = \sigma(a_i) = \sigma(\mathbf{w}^\top \mathbf{x}_i)$$

• using variables $s_i = 2t_i - 1$ in $\{-1, 1\}$, each term is

 $\begin{array}{ll} \text{if } t_i = 1 \ (s_i = 1) & \ln \sigma(a_i) \\ \text{if } t_i = 0 \ (s_i = -1) & \ln(1 - \sigma(a_i)) = \ln \sigma(-a_i) \\ \text{in either case} & \ln \sigma(s_i a_i) \end{array}$

the error function becomes

$$E(\mathbf{w}) = -\frac{1}{n} \sum_{i=1}^{n} \ln \sigma(s_i a_i) = \frac{1}{n} \sum_{i=1}^{n} \ln(1 + e^{-s_i a_i})$$

◆□ ▶ ◆□ ▶ ◆ □ ▶ ◆ □ ▶ ● □ ● ● ● ●

logistic regression, again

recall that

$$E(\mathbf{w}) := -\frac{1}{n} \sum_{i=1}^{n} (t_i \ln y_i + (1 - t_i) \ln(1 - y_i))$$

where
$$y_i = \sigma(a_i) = \sigma(\mathbf{w}^\top \mathbf{x}_i)$$

• using variables $s_i = 2t_i - 1$ in $\{-1, 1\}$, each term is

$$\begin{array}{ll} \text{if } t_i = 1 \ (s_i = 1) & \ln \sigma(a_i) \\ \text{if } t_i = 0 \ (s_i = -1) & \ln(1 - \sigma(a_i)) = \ln \sigma(-a_i) \\ \text{in either case} & \ln \sigma(s_i a_i) \end{array}$$

the error function becomes

$$E(\mathbf{w}) = -\frac{1}{n} \sum_{i=1}^{n} \ln \sigma(s_i a_i) = \frac{1}{n} \sum_{i=1}^{n} \ln(1 + e^{-s_i a_i})$$

▲□▶ ▲圖▶ ▲臣▶ ▲臣▶ 三臣 - わへぐ

logistic regression, again

recall that

$$E(\mathbf{w}) := -\frac{1}{n} \sum_{i=1}^{n} (t_i \ln y_i + (1 - t_i) \ln(1 - y_i))$$

where
$$y_i = \sigma(a_i) = \sigma(\mathbf{w}^\top \mathbf{x}_i)$$

• using variables $s_i = 2t_i - 1$ in $\{-1, 1\}$, each term is

$$\begin{array}{ll} \text{if } t_i = 1 \ (s_i = 1) & \ln \sigma(a_i) \\ \text{if } t_i = 0 \ (s_i = -1) & \ln(1 - \sigma(a_i)) = \ln \sigma(-a_i) \\ \text{in either case} & \ln \sigma(s_i a_i) \end{array}$$

• the error function becomes

$$E(\mathbf{w}) = -\frac{1}{n} \sum_{i=1}^{n} \ln \sigma(s_i a_i) = \frac{1}{n} \sum_{i=1}^{n} \ln(1 + e^{-s_i a_i})$$

◆□ > ◆□ > ◆豆 > ◆豆 > ̄豆 = のへぐ

maximum posterior

• weight decay also appears in probabilistic formulations by considering the weight vector a random variable and incorporating a Gaussian prior for ${\bf w}$

$$p(\mathbf{w}|\lambda) = \exp(-\frac{\lambda}{2} \|\mathbf{w}\|^2)$$

• the posterior distribution given the dataset X, T is

$$p(\mathbf{w}|X,T) \propto p(T|X,\mathbf{w})p(\mathbf{w}|\lambda)$$

taking negative logarithm, the error function to minimize is

$$E(\mathbf{w}) = -\ln p(T|X, \mathbf{w}) + \frac{\lambda}{2} \|\mathbf{w}\|^2$$

maximum posterior

• weight decay also appears in probabilistic formulations by considering the weight vector a random variable and incorporating a Gaussian prior for ${\bf w}$

$$p(\mathbf{w}|\lambda) = \exp(-\frac{\lambda}{2} \|\mathbf{w}\|^2)$$

• the posterior distribution given the dataset X, T is

$$p(\mathbf{w}|X,T) \propto p(T|X,\mathbf{w})p(\mathbf{w}|\lambda)$$

taking negative logarithm, the error function to minimize is

$$E(\mathbf{w}) = -\ln p(T|X, \mathbf{w}) + \frac{\lambda}{2} \|\mathbf{w}\|^2$$

maximum posterior

• weight decay also appears in probabilistic formulations by considering the weight vector a random variable and incorporating a Gaussian prior for ${\bf w}$

$$p(\mathbf{w}|\lambda) = \exp(-\frac{\lambda}{2} \|\mathbf{w}\|^2)$$

• the posterior distribution given the dataset X, T is

$$p(\mathbf{w}|X,T) \propto p(T|X,\mathbf{w})p(\mathbf{w}|\lambda)$$

taking negative logarithm, the error function to minimize is

$$E(\mathbf{w}) = -\ln p(T|X, \mathbf{w}) + \frac{\lambda}{2} \|\mathbf{w}\|^2$$

• in all three cases, we can define the error function (or cost function)

$$E(\boldsymbol{\theta}) := \frac{1}{n} \sum_{i=1}^{n} L(\hat{f}(\mathbf{x}_i; \boldsymbol{\theta}), s_i) + \frac{\lambda}{2} \|\mathbf{w}\|^2$$

• there are no constraints: in all three cases, we can use (stochastic) gradient descent to minimize the error function with respect to parameters θ

• in all three cases, we can define the error function (or cost function)

$$E(\boldsymbol{\theta}) := \frac{1}{n} \sum_{i=1}^{n} L(\hat{f}(\mathbf{x}_i; \boldsymbol{\theta}), s_i) + \frac{\lambda}{2} \|\mathbf{w}\|^2$$
data term

▲ロ → ▲周 → ▲目 → ▲目 → □ → の Q (~

• there are no constraints: in all three cases, we can use (stochastic) gradient descent to minimize the error function with respect to parameters θ

• in all three cases, we can define the error function (or cost function)

$$E(\boldsymbol{\theta}) := \boxed{\frac{1}{n} \sum_{i=1}^{n} L(\hat{f}(\mathbf{x}_i; \boldsymbol{\theta}), s_i)} + \frac{\lambda}{2} \|\mathbf{w}\|^2}$$
data term regularization term

▲ロ → ▲周 → ▲目 → ▲目 → □ → の Q (~

• there are no constraints: in all three cases, we can use (stochastic) gradient descent to minimize the error function with respect to parameters θ

• in all three cases, we can define the error function (or cost function)

$$E(\boldsymbol{\theta}) := \frac{1}{n} \sum_{i=1}^{n} L(\hat{f}(\mathbf{x}_i; \boldsymbol{\theta}), s_i) + \frac{\lambda}{2} \|\mathbf{w}\|^2$$
data term regularization term

ション 人口 マイビン トレート シックション

• there are no constraints: in all three cases, we can use (stochastic) gradient descent to minimize the error function with respect to parameters θ

prediction function

• in all three cases, we can use parameters $oldsymbol{ heta}=(\mathbf{w},b)$ and

$$\hat{f}(\mathbf{x}; \mathbf{w}, b) = \mathbf{w}^\top \mathbf{x} + b$$

to make predictions during learning (training); this is the activation, without the nonlinearity

• in all three cases, when the optimal parameters $\theta^* = \arg \min_{\theta} E(\theta)$ are found, use the prediction function

$$f(\mathbf{x}; \mathbf{w}^*, b^*) = \operatorname{sgn}(\mathbf{w}^{*\top} \mathbf{x} + b^*) = \begin{cases} +1, & \mathbf{w}^{*\top} \mathbf{x} + b^* \ge 0\\ -1, & \mathbf{w}^{*\top} \mathbf{x} + b^* < 0 \end{cases}$$

ション 人口 マイビン トレート シックション

to classify new samples during inference (testing)

prediction function

• in all three cases, we can use parameters $oldsymbol{ heta}=(\mathbf{w},b)$ and

$$\hat{f}(\mathbf{x}; \mathbf{w}, b) = \mathbf{w}^\top \mathbf{x} + b$$

to make predictions during learning (training); this is the activation, without the nonlinearity

• in all three cases, when the optimal parameters $\theta^* = \arg \min_{\theta} E(\theta)$ are found, use the prediction function

$$f(\mathbf{x}; \mathbf{w}^*, b^*) = \operatorname{sgn}(\mathbf{w}^{*\top}\mathbf{x} + b^*) = \begin{cases} +1, & \mathbf{w}^{*\top}\mathbf{x} + b^* \ge 0\\ -1, & \mathbf{w}^{*\top}\mathbf{x} + b^* < 0 \end{cases}$$

to classify new samples during inference (testing)

• in all cases, we can use loss function

$$L(a,s) = \ell(sa)$$

where a is the activation and s the target variable in $\{-1,1\}$ ("sign") the only difference is



• in all cases, we can use loss function

$$L(a,s) = \ell(sa)$$

where a is the activation and s the target variable in $\{-1,1\}$ ("sign")

• the only difference is

	$\ell(x)$
perceptron	$[-x]_{+}$
SVM (hinge)	$[1-x]_+$
logistic	$\ln(1+e^{-x})$

▲□▶ ▲圖▶ ▲臣▶ ▲臣▶ 三臣 - わへぐ

- perceptron and logistic are asymptotically equivalent
- both SVM and logistic penalize small positive inputs



- perceptron and logistic are asymptotically equivalent
- both SVM and logistic penalize small positive inputs



x

▲ロ → ▲周 → ▲目 → ▲目 → ● ● ● ● ●

derivatives

• the actual value of the loss is never used; all that matters is its derivative



x

derivatives

• the actual value of the loss is never used; all that matters is its derivative



x

◆□▶ ◆□▶ ◆臣▶ ◆臣▶ 三臣 - のへぐ

derivatives

- in all cases, a sample that is correctly classified with an activation well above some margin does not contribute at all to the error function: the loss derivative is zero
- in all cases, a sample that is correctly classified with an activation well below some margin has a fixed negative contribution: the loss derivative is -1
- the same holds for logistic regression, which is unexpected if one looks at the saturating form of the sigmoid $\left(\frac{d\sigma}{dx}(x)\right)$ tends to zero for $|x| \to \infty$)
- this is because the log of cross-entropy cancels the effect of the exp of the sigmoid and is a good reason the treat these two as one function operating directly on the activation
derivatives

- in all cases, a sample that is correctly classified with an activation well above some margin does not contribute at all to the error function: the loss derivative is zero
- in all cases, a sample that is correctly classified with an activation well below some margin has a fixed negative contribution: the loss derivative is -1
- the same holds for logistic regression, which is unexpected if one looks at the saturating form of the sigmoid $\left(\frac{d\sigma}{dx}(x)\right)$ tends to zero for $|x| \to \infty$)
- this is because the log of cross-entropy cancels the effect of the exp of the sigmoid and is a good reason the treat these two as one function operating directly on the activation



• perceptron and hinge loss differ only by a shift; once the bias is learned, aren't they equivalent?



•
$$k = 2, n = 200, m = 10, \epsilon = 10^{-3}, \lambda = 10^{-3}$$



•
$$k = 2, n = 200, m = 10, \epsilon = 10^{-3}, \lambda = 10^{-3}$$



•
$$k = 2, n = 200, m = 10, \epsilon = 10^{-3}, \lambda = 10^{-3}$$



(□) (@) (E) (E) [E]

•
$$k = 2, n = 200, m = 10, \epsilon = 10^{-3}, \lambda = 10^{-3}$$



•
$$k = 2, n = 200, m = 10, \epsilon = 10^{-3}, \lambda = 10^{-3}$$



•
$$k = 2, n = 200, m = 10, \epsilon = 10^{-3}, \lambda = 10^{-3}$$



(□) (@) (E) (E) [E]

•
$$k = 2, n = 200, m = 10, \epsilon = 10^{-3}, \lambda = 10^{-3}$$



(□) (@) (E) (E) [E]

•
$$k = 2, n = 200, m = 10, \epsilon = 10^{-3}, \lambda = 10^{-3}$$



•
$$k = 2, n = 200, m = 10, \epsilon = 10^{-3}, \lambda = 10^{-3}$$

▲□▶ ▲圖▶ ▲ 臣▶ ▲ 臣▶ ▲ 臣 → のQC



•
$$k = 2, n = 200, m = 10, \epsilon = 10^{-3}, \lambda = 10^{-3}$$

▲□▶ ▲圖▶ ▲目▶ ▲目▶ 目 のQC

multi-class classification

multi-class logistic regression

• there are now k classes C_1, \ldots, C_k and, given input $\mathbf{x} \in \mathbb{R}^d$, one activation per class for $j = 1, \ldots, k$

$$a_j = \mathbf{w}_j^\top \mathbf{x} + b_j$$

or, in matrix form

$$\mathbf{a} = (a_1, \dots, a_k) = W^\top \mathbf{x} + \mathbf{b}$$

where $W = (\mathbf{w}_1, \dots, \mathbf{w}_k)$ is a $d \times k$ weight matrix and $\mathbf{b} = (b_1, \dots, b_k)$ a bias vector

• and one output $y_j \in [0,1]$ per class for $j=1,\ldots,k$

$$y_j = f_j(\mathbf{x}; W, \mathbf{b}) := \boldsymbol{\sigma}_j(W^\top \mathbf{x} + \mathbf{b}) = \boldsymbol{\sigma}_j(\mathbf{a})$$

or output vector $\mathbf{y} \in [0,1]^k$

$$\mathbf{y} = (y_1, \dots, y_k) = f(\mathbf{x}; W, \mathbf{b}) := \boldsymbol{\sigma}(W^\top \mathbf{x} + \mathbf{b})$$

multi-class logistic regression

• there are now k classes C_1, \ldots, C_k and, given input $\mathbf{x} \in \mathbb{R}^d$, one activation per class for $j = 1, \ldots, k$

$$a_j = \mathbf{w}_j^\top \mathbf{x} + b_j$$

or, in matrix form

$$\mathbf{a} = (a_1, \dots, a_k) = W^\top \mathbf{x} + \mathbf{b}$$

where $W = (\mathbf{w}_1, \dots, \mathbf{w}_k)$ is a $d \times k$ weight matrix and $\mathbf{b} = (b_1, \dots, b_k)$ a bias vector

- and one output $y_j \in [0,1]$ per class for $j=1,\ldots,k$

$$y_j = f_j(\mathbf{x}; W, \mathbf{b}) := \boldsymbol{\sigma}_j(W^\top \mathbf{x} + \mathbf{b}) = \boldsymbol{\sigma}_j(\mathbf{a})$$

or output vector $\mathbf{y} \in [0, 1]^k$

$$\mathbf{y} = (y_1, \dots, y_k) = f(\mathbf{x}; W, \mathbf{b}) := \boldsymbol{\sigma}(W^\top \mathbf{x} + \mathbf{b})$$

• the softmax function generalizes the sigmoid function and yields a vector of k values in [0, 1] by exponentiating and then normalizing

$$\boldsymbol{\sigma}(\mathbf{a}) := ext{softmax}(\mathbf{a}) := rac{1}{\sum_{j} e^{a_j}} (e^{a_1}, \dots, e^{a_k})$$

as activation values increase, softmax tends to focus on the maximum



• the softmax function generalizes the sigmoid function and yields a vector of k values in [0, 1] by exponentiating and then normalizing

$$\boldsymbol{\sigma}(\mathbf{a}) := ext{softmax}(\mathbf{a}) := rac{1}{\sum_{j} e^{a_j}} (e^{a_1}, \dots, e^{a_k})$$

as activation values increase, softmax tends to focus on the maximum



• the softmax function generalizes the sigmoid function and yields a vector of k values in [0, 1] by exponentiating and then normalizing

$$\boldsymbol{\sigma}(\mathbf{a}) := ext{softmax}(\mathbf{a}) := rac{1}{\sum_{j} e^{a_j}} (e^{a_1}, \dots, e^{a_k})$$

as activation values increase, softmax tends to focus on the maximum



イロト 不得 トイヨト イヨト ヨー ろくで

• the softmax function generalizes the sigmoid function and yields a vector of k values in [0, 1] by exponentiating and then normalizing

$$\boldsymbol{\sigma}(\mathbf{a}) := ext{softmax}(\mathbf{a}) := rac{1}{\sum_{j} e^{a_j}} (e^{a_1}, \dots, e^{a_k})$$

as activation values increase, softmax tends to focus on the maximum



イロト 不得 トイヨト イヨト ヨー ろくで

• the softmax function generalizes the sigmoid function and yields a vector of k values in [0, 1] by exponentiating and then normalizing

$$\boldsymbol{\sigma}(\mathbf{a}) := ext{softmax}(\mathbf{a}) := rac{1}{\sum_{j} e^{a_j}} (e^{a_1}, \dots, e^{a_k})$$

as activation values increase, softmax tends to focus on the maximum



イロト 不得 トイヨト イヨト ヨー ろくで

• the softmax function generalizes the sigmoid function and yields a vector of k values in [0, 1] by exponentiating and then normalizing

$$\boldsymbol{\sigma}(\mathbf{a}) := ext{softmax}(\mathbf{a}) := rac{1}{\sum_{j} e^{a_j}} (e^{a_1}, \dots, e^{a_k})$$

as activation values increase, softmax tends to focus on the maximum



• the softmax function generalizes the sigmoid function and yields a vector of k values in [0, 1] by exponentiating and then normalizing

$$\boldsymbol{\sigma}(\mathbf{a}) := ext{softmax}(\mathbf{a}) := rac{1}{\sum_{j} e^{a_j}} (e^{a_1}, \dots, e^{a_k})$$

as activation values increase, softmax tends to focus on the maximum



• the softmax function generalizes the sigmoid function and yields a vector of k values in [0, 1] by exponentiating and then normalizing

$$\boldsymbol{\sigma}(\mathbf{a}) := ext{softmax}(\mathbf{a}) := rac{1}{\sum_{j} e^{a_j}} (e^{a_1}, \dots, e^{a_k})$$

as activation values increase, softmax tends to focus on the maximum



cross-entropy error

- we are given training samples $X = (\mathbf{x}_1, \dots, \mathbf{x}_n) \in \mathbb{R}^{d \times n}$ and target variables $T = (\mathbf{t}_1, \dots, \mathbf{t}_n) \in \{0, 1\}^{k \times n}$
- this is an 1-of-k or one-hot encoding scheme: $t_{ji} = \mathbb{1}[\mathbf{x}_i \in C_j]$
- there is a similar probabilistic interpretation: output y_{ji} represents the posterior class probability p(C_j|x_i)
- again, maximizing the likelihood function yields the average cross-entropy error function

$$E(W, \mathbf{b}) = \frac{1}{n} \sum_{i=1}^{n} L(\mathbf{a}_i, \mathbf{t}_i) = -\frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{k} t_{ji} \ln y_{ji}$$

where $Y = (\mathbf{y}_1, \dots, \mathbf{y}_n)$ and $\mathbf{y}_i = \boldsymbol{\sigma}(\mathbf{a}_i) = \boldsymbol{\sigma}(W^\top \mathbf{x}_i + \mathbf{b})$

cross-entropy error

- we are given training samples $X = (\mathbf{x}_1, \dots, \mathbf{x}_n) \in \mathbb{R}^{d \times n}$ and target variables $T = (\mathbf{t}_1, \dots, \mathbf{t}_n) \in \{0, 1\}^{k \times n}$
- this is an 1-of-k or one-hot encoding scheme: $t_{ji} = \mathbb{1}[\mathbf{x}_i \in C_j]$
- there is a similar probabilistic interpretation: output y_{ji} represents the posterior class probability $p(C_j|\mathbf{x}_i)$
- again, maximizing the likelihood function yields the average cross-entropy error function

$$E(W, \mathbf{b}) = \frac{1}{n} \sum_{i=1}^{n} L(\mathbf{a}_i, \mathbf{t}_i) = -\frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{k} t_{ji} \ln y_{ji}$$

where $Y = (\mathbf{y}_1, \dots, \mathbf{y}_n)$ and $\mathbf{y}_i = \boldsymbol{\sigma}(\mathbf{a}_i) = \boldsymbol{\sigma}(W^\top \mathbf{x}_i + \mathbf{b})$

・ロト・西ト・ヨト・ヨー シック

cross-entropy loss

• given a single sample x and target variable t, and corresponding producing activation $\mathbf{a} = W^{\top}\mathbf{x} + \mathbf{b}$, the loss function is

$$egin{aligned} L(\mathbf{a},\mathbf{t}) &= -\mathbf{t}^{ op} \ln oldsymbol{\sigma}(\mathbf{a}) \ &= -\mathbf{t}^{ op} \left(\mathbf{a} - \ln \left(\sum_{j=1}^k e^{a_j}
ight)
ight) \end{aligned}$$

- suppose the correct label (nonzero element of ${f t})$ is $l,~i.e.~~{f t}={f e}_l$
- also this term can be approximated by the maximum element of a:

$$L(\mathbf{a}, \mathbf{t}) \approx \max \mathbf{a} - a_l = \max_j a_j - a_l$$

ション 人口 マイビン トレート シックション

so there is loss if the activation of the correct class is not maximum

cross-entropy loss

• given a single sample x and target variable t, and corresponding producing activation $\mathbf{a} = W^{\top}\mathbf{x} + \mathbf{b}$, the loss function is

$$L(\mathbf{a}, \mathbf{t}) = -\mathbf{t}^{\top} \ln \boldsymbol{\sigma}(\mathbf{a})$$
$$= -\mathbf{t}^{\top} \left(\mathbf{a} - \ln \left(\sum_{j=1}^{k} e^{a_j} \right) \right)$$

suppose the correct label (nonzero element of t) is l, i.e. t = e_l
also this term can be approximated by the maximum element of a:

$$L(\mathbf{a}, \mathbf{t}) \approx \max \mathbf{a} - a_l = \max_j a_j - a_l$$

so there is loss if the activation of the correct class is not maximum

cross-entropy loss derivative

- remember, it's only derivatives that matter
- the derivative of the cross-entropy loss with respect to the activation is particularly simple, no approximation needed:

$$\nabla_{\mathbf{a}} L(\mathbf{a}, \mathbf{t}) = \boldsymbol{\sigma}(\mathbf{a}) - \mathbf{t} = \mathbf{y} - \mathbf{t}$$

- again, exp and log cancel, and that's a reason to keep softmax followed by cross-entropy as one function
- example (correct label l = 3):

t	0	0	0	0	0
a	0.3	0.1	0.4	0.0	0.2
$\frac{dL}{d\mathbf{a}}$	0.3	0.1	0.4	0.0	0.2

• by increasing a class activation, the loss decreases if the class is correct, and increases otherwise

cross-entropy loss derivative

- remember, it's only derivatives that matter
- the derivative of the cross-entropy loss with respect to the activation is particularly simple, no approximation needed:

$$\nabla_{\mathbf{a}} L(\mathbf{a}, \mathbf{t}) = \boldsymbol{\sigma}(\mathbf{a}) - \mathbf{t} = \mathbf{y} - \mathbf{t}$$

- again, exp and log cancel, and that's a reason to keep softmax followed by cross-entropy as one function
- example (correct label l = 3):

\mathbf{t}	0	0	1	0	0	0
а	0.3	0.1	0.8	0.4	0.0	0.2
$\frac{dL}{d\mathbf{a}}$	0.3	0.1	-0.2	0.4	0.0	0.2

• by increasing a class activation, the loss decreases if the class is correct, and increases otherwise

cross-entropy loss derivative

- remember, it's only derivatives that matter
- the derivative of the cross-entropy loss with respect to the activation is particularly simple, no approximation needed:

$$\nabla_{\mathbf{a}} L(\mathbf{a}, \mathbf{t}) = \boldsymbol{\sigma}(\mathbf{a}) - \mathbf{t} = \mathbf{y} - \mathbf{t}$$

- again, exp and log cancel, and that's a reason to keep softmax followed by cross-entropy as one function
- example (correct label l = 3):

\mathbf{t}	0	0	1	0	0	0
а	0.3	0.1	0.8	0.4	0.0	0.2
$\frac{dL}{d\mathbf{a}}$	0.3	0.1	-0.2	0.4	0.0	0.2

• by increasing a class activation, the loss decreases if the class is correct, and increases otherwise

multiclass SVM

- following the representation of correct label $l \in \{1, \ldots, k\}$
- several extensions, e.g. Weston and Watkins

$$L(\mathbf{a}, l) := \left[1 + \max_{j \neq l} a_j - a_l\right]_+ = \max_{j \neq l} \left[1 + a_j - a_l\right]_+$$

similar to the previous approximation of cross-entropy, plus marginCrammer and Singer

$$L(\mathbf{a},l) := \sum_{j \neq l} \left[1 + a_j - a_l\right]_+$$

penalizes all labels that have better activation than the correct oneboth interpretable with simple derivatives

Weston and Watkins. ESANN 1999. Support Vector Machines for Multi-Class Pattern Recognition. Crammer and Singer. JMLR 2001. On the Algorithmic Implementation of Multiclass Kernel-Based Vector Machines.

multiclass SVM

- we now apply logistic regression and SVM (W&W) to classify three classes in 2d
- soft assignment: to visualize the class confidences, we apply softmax to activations in each case, even if SVM is not probabilistic
- hard assignment: now we threshold activations with sgn instead, as we do in testing

• we repeat at different epochs during training

epoch 00



epoch 05



epoch 10



epoch 15


epoch 20



epoch 25



epoch 30



epoch 35



epoch 00



hinge (W & W)

logistic

epoch 04



hinge (W & W)

logistic

epoch 08



epoch 12



hinge (W & W)

logistic

epoch 16



epoch 20



epoch 24



logistic

epoch 28



logistic

MNIST digits dataset

4 5 6 5

• 10 classes, 60k training images, 10k test images, 28 imes 28 images

LeCun, Bottou, Bengio and Haffner IEEE Proc. 1998. Gradient-Based Learning Applied to Document Recognition.

from images to vectors

- all classifiers considered so far work with vectors
- we have seen how to extract a descriptor—a vector—from an image
- however, the point now is how to learn to extract a descriptor
- so we start from raw pixels: a gray-scale input image is just a 28×28 matrix, and we vectorize it into 784×1



◆□▶ ◆□▶ ◆臣▶ ◆臣▶ 三臣 - 釣��





input - weights - bias - softmax - parameters to be learned

◆□ > ◆□ > ◆豆 > ◆豆 > ̄豆 = 釣�@



• input - weights - bias - softmax - parameters to be learned

◆□ > ◆□ > ◆豆 > ◆豆 > ̄豆 = 釣�@



• input - weights - bias - softmax - parameters to be learned

◆□ > ◆□ > ◆豆 > ◆豆 > ̄豆 = 釣�@



• input - weights - bias - softmax - parameters to be learned

◆□▶ ◆□▶ ◆臣▶ ◆臣▶ □臣 = のへぐ

what is being learned?

- the columns of W are multiplied with \mathbf{x} ; they live in the same space
- we can reshape each one back from 784×1 to $28\times 28:$ it should look like a digit

◆□▶ ◆□▶ ◆臣▶ ◆臣▶ □臣 = のへぐ



- $k = 3, n = 60000, m = 6000, \epsilon = 10^{-1}, \lambda = 10^{-4}$
- test error 7.67%



- $k = 3, n = 60000, m = 6000, \epsilon = 10^{-1}, \lambda = 10^{-4}$
- test error 7.67%



- $k = 3, n = 60000, m = 6000, \epsilon = 10^{-1}, \lambda = 10^{-4}$
- test error 7.67%



- $k = 3, n = 60000, m = 6000, \epsilon = 10^{-1}, \lambda = 10^{-4}$
- test error 7.67%



- $k = 3, n = 60000, m = 6000, \epsilon = 10^{-1}, \lambda = 10^{-4}$
- test error 7.67%



▲□▶ ▲圖▶ ▲圖▶ ▲圖▶ / 圖 / のへで



• linear model with parameters $\mathbf{w} = (a, b)$

$$y = ax + b = (a, b)^{\top}(x, 1) = \mathbf{w}^{\top}\phi(x)$$

• least squares error given samples (x_1, \ldots, x_n) , targets $\mathbf{t} = (t_1, \ldots, t_n)$

$$E(\mathbf{w}) = \sum_{i=1}^{n} (\mathbf{w}^{\top} \phi(x_i) - t_i)^2$$



• linear model with parameters $\mathbf{w} = (a, b)$

$$y = ax + b = (a, b)^{\top}(x, 1) = \mathbf{w}^{\top}\phi(x)$$

• least squares solution, where $\Phi = (\phi(x_1); \ldots; \phi(x_n)) \in \mathbb{R}^{n \times 2}$

$$\mathbf{w}^* = (\Phi^\top \Phi)^{-1} \Phi^\top \mathbf{t}$$

・ロト ・ 同ト ・ ヨト ・ ヨト … ヨ



• linear model with parameters $\mathbf{w} \in \mathbb{R}^4$

$$y = \mathbf{w}^{\top} \phi(x) = \mathbf{w}^{\top}(1, x, x^2, x^3)$$

• least squares solution, where $\Phi = (\phi(x_1); \ldots; \phi(x_n)) \in \mathbb{R}^{n \times 4}$

$$\mathbf{w}^* = (\Phi^{\top} \Phi)^{-1} \Phi^{\top} \mathbf{t}$$

ション 人口 マイビン トレート シックション



$$y = \mathbf{w}^{\top} \phi(x) = \mathbf{w}^{\top}(1, x, x^2, \dots, x^{10})$$

• least squares solution, where $\Phi = (\phi(x_1); \ldots; \phi(x_n)) \in \mathbb{R}^{n \times 11}$

$$\mathbf{w}^* = (\Phi^{\top} \Phi)^{-1} \Phi^{\top} \mathbf{t}$$

イロト 不得 トイヨト イヨト ヨー ろくで



$$y = \mathbf{w}^{\top} \phi(x) = \mathbf{w}^{\top}(1, x, x^2, \dots, x^{10})$$

• least squares solution, where $\Phi = (\phi(x_1); \ldots; \phi(x_n)) \in \mathbb{R}^{n \times 11}$

$$\mathbf{w}^* = (\Phi^{\top} \Phi)^{-1} \Phi^{\top} \mathbf{t}$$

イロト 不得 トイヨト イヨト ヨー ろくで



$$y = \mathbf{w}^{\top} \phi(x) = \mathbf{w}^{\top}(1, x, x^2, \dots, x^{10})$$

• regularized least squares error with parameter λ

$$E(\mathbf{w}) = \sum_{i=1}^{n} (\mathbf{w}^{\top} \phi(x_i) - t_i)^2 + \frac{\lambda}{2} \|\mathbf{w}\|^2$$



$$y = \mathbf{w}^{\top} \phi(x) = \mathbf{w}^{\top}(1, x, x^2, \dots, x^{10})$$

• regularized least squares solution with parameter $\lambda = 10^{-3}$

$$\mathbf{w}^* = (\lambda I + \Phi^\top \Phi)^{-1} \Phi^\top \mathbf{t}$$

・ロト ・ 同ト ・ ヨト ・ ヨト … ヨ



$$y = \mathbf{w}^{\top} \phi(x) = \mathbf{w}^{\top}(1, x, x^2, \dots, x^{10})$$

• regularized least squares solution with parameter $\lambda = 1$

$$\mathbf{w}^* = (\lambda I + \Phi^\top \Phi)^{-1} \Phi^\top \mathbf{t}$$

generalization error



• linear model with parameters $\mathbf{w} \in \mathbb{R}^{11}$

$$y = \mathbf{w}^{\top} \phi(x) = \mathbf{w}^{\top}(1, x, x^2, \dots, x^{10})$$

• regularized least squares solution with parameter $\lambda \in [10^{-8}, 10^0]$

$$\mathbf{w}^* = (\lambda I + \Phi^\top \Phi)^{-1} \Phi^\top \mathbf{t}$$

イロト 不得 トイヨト イヨト ヨー ろくで
- optimize both parameters and hyperparameters on the training set: could work perfectly on training set, no idea how it works on test set train
- train parameters on training set, hyperparameters on test set: no idea how it works no new data; the test set represents new data and should never be touched but for evaluation at the very end

trair

test

 train parameters on training set, hyperparameters on validation set: great, validation data are new so we test our model's generalization; test data are also new and are only used for evaluation

train

イロト (理) (ヨ) (ヨ) (ヨ) (の)

- optimize both parameters and hyperparameters on the training set: could work perfectly on training set, no idea how it works on test set train
- train parameters on training set, hyperparameters on test set: no idea how it works no new data; the test set represents new data and should never be touched but for evaluation at the very end

trair

test

 train parameters on training set, hyperparameters on validation set: great, validation data are new so we test our model's generalization; test data are also new and are only used for evaluation

train

イロト (理) (ヨ) (ヨ) (ヨ) (の)

- optimize both parameters and hyperparameters on the training set: could work perfectly on training set, no idea how it works on test set train
- train parameters on training set, hyperparameters on test set: no idea how it works no new data; the test set represents new data and should never be touched but for evaluation at the very end

train

test

 train parameters on training set, hyperparameters on validation set: great, validation data are new so we test our model's generalization; test data are also new and are only used for evaluation

train

test

イロト 不得 トイヨト イヨト ヨー ろくで

- optimize both parameters and hyperparameters on the training set: could work perfectly on training set, no idea how it works on test set train
- train parameters on training set, hyperparameters on test set: no idea how it works no new data; the test set represents new data and should never be touched but for evaluation at the very end

train

test

 train parameters on training set, hyperparameters on validation set: great, validation data are new so we test our model's generalization; test data are also new and are only used for evaluation

train

1

test

イロト 不得 トイヨト イヨト ヨー ろくで

- optimize both parameters and hyperparameters on the training set: could work perfectly on training set, no idea how it works on test set train
- train parameters on training set, hyperparameters on test set: no idea how it works no new data; the test set represents new data and should never be touched but for evaluation at the very end

train

test

 train parameters on training set, hyperparameters on validation set: great, validation data are new so we test our model's generalization; test data are also new and are only used for evaluation

train

val

test

- optimize both parameters and hyperparameters on the training set: could work perfectly on training set, no idea how it works on test set train
- train parameters on training set, hyperparameters on test set: no idea how it works no new data; the test set represents new data and should never be touched but for evaluation at the very end

train

test

 train parameters on training set, hyperparameters on validation set: great, validation data are new so we test our model's generalization; test data are also new and are only used for evaluation

train

val

test

• split data into k groups; treat k-1 as training and 1 as validation, measure on test set; repeat over all splits and average the results

val				run 1	test
				1	
	val			run 2	test
		val			test
			val		test

• too expensive for large datasets: better use only one split; even better, each dataset has an official validation set so results are comparable

• split data into k groups; treat k - 1 as training and 1 as validation, measure on test set; repeat over all splits and average the results

val				run 1	test
	val			run 2	test
		val		run 3	test
			val	run 4	test

• too expensive for large datasets: better use only one split; even better, each dataset has an official validation set so results are comparable

• split data into k groups; treat k - 1 as training and 1 as validation, measure on test set; repeat over all splits and average the results



• too expensive for large datasets: better use only one split; even better, each dataset has an official validation set so results are comparable

• split data into k groups; treat k - 1 as training and 1 as validation, measure on test set; repeat over all splits and average the results



• too expensive for large datasets: better use only one split; even better, each dataset has an official validation set so results are comparable

• split data into k groups; treat k - 1 as training and 1 as validation, measure on test set; repeat over all splits and average the results



• too expensive for large datasets: better use only one split; even better, each dataset has an official validation set so results are comparable

• split data into k groups; treat k - 1 as training and 1 as validation, measure on test set; repeat over all splits and average the results



• too expensive for large datasets: better use only one split; even better, each dataset has an official validation set so results are comparable

"basis" functions

- the most interesting idea discussed here is that the model becomes nonlinear in the raw input by expressing the unknown function as a linear combination (with unknown weights) of a number of fixed nonlinear "basis" functions
- we can re-use this idea in classification because classification is really regression followed by thresholding (or comparison)

"basis" functions

- the most interesting idea discussed here is that the model becomes nonlinear in the raw input by expressing the unknown function as a linear combination (with unknown weights) of a number of fixed nonlinear "basis" functions
- we can re-use this idea in classification because classification is really regression followed by thresholding (or comparison)

basis functions



ma C

basis function derivatives



basis functions

- we want basis functions to cover the entire space so that any arbitrary input can be expressed as a linear of combination of such functions
- the Gaussian is localized, the others have larger support
- polynomials and their derivatives can get extremely large; the range of all the others can be easily controlled

• the derivatives of the Gaussian and sigmoid are localized; the derivative of softplus is nonzero over half of the space

multiple layers

・ロト ・ 理 ト ・ ヨ ト ・ ヨ ・ の へ ()・



- two point sets $X_1, X_2 \subset \mathbb{R}^d$ are linearly separable iff there is \mathbf{w}, b such that $\mathbf{w}^\top x_1 < b < \mathbf{w}^\top x_2$ for $\mathbf{x}_1 \in X_1, \mathbf{x}_2 \in X_2$
- or, they can be separated by a perceptron

non-linearly separable classes



()

credit: dataset adapted from Andrej Karpathy

epoch 00



◆□> <圖> < E> < E> E のQ@

•
$$k = 3, n = 300, m = 100, \epsilon = 10^{0}, \lambda = 10^{-3}$$

epoch 05



•
$$k = 3, n = 300, m = 100, \epsilon = 10^{0}, \lambda = 10^{-3}$$

epoch 10



• $k = 3, n = 300, m = 100, \epsilon = 10^0, \lambda = 10^{-3}$

epoch 15



•
$$k = 3, n = 300, m = 100, \epsilon = 10^0, \lambda = 10^{-3}$$

epoch 20



イロト イロト イヨト イヨト 三日

• $k = 3, n = 300, m = 100, \epsilon = 10^0, \lambda = 10^{-3}$

epoch 25



イロト イポト イヨト イヨト 二百

• $k = 3, n = 300, m = 100, \epsilon = 10^0, \lambda = 10^{-3}$

epoch 30



•
$$k = 3, n = 300, m = 100, \epsilon = 10^0, \lambda = 10^{-3}$$

epoch 35



•
$$k = 3, n = 300, m = 100, \epsilon = 10^{0}, \lambda = 10^{-3}$$

epoch 40



•
$$k = 3, n = 300, m = 100, \epsilon = 10^0, \lambda = 10^{-3}$$

epoch 45



•
$$k = 3, n = 300, m = 100, \epsilon = 10^0, \lambda = 10^{-3}$$



◆□▶ ◆□▶ ◆臣▶ ◆臣▶ ─臣 ─の�?

• so how do we make our classifier nonlinear?



▲□▶ ▲圖▶ ▲臣▶ ★臣▶ ―臣 …のへで

- define a 10×10 grid over the entire space



▲□▶ ▲圖▶ ▲臣▶ ★臣▶ ―臣 …のへで

• and a (Gaussian?) basis function centered at every cell



• then, a linear classifier can separate the 3 classes in 100 dimensions!

the curse of dimensionality



• but, in 3 dimensions we would need 1000 basis functions; and remember, a 320×200 image is a vector in $\mathbb{R}^{64,000}$

◆□ ▶ ◆□ ▶ ◆ □ ▶ ◆ □ ▶ ◆ □ ▶ ◆ □ ▶ ◆ □ ▶

basis functions

- we need a small set of basis functions to cover the entire space, or at least the regions where our data live
- we did use fixed basis functions before: the Gabor filters discretized the 2d space of scales and orientations in uniform bins and their responses were used as vectors
- but right in the next layer, the dimensions increase and we cannot afford to have fixed basis functions everywhere: we have to learn from the data, as we did with the codebooks
- codebooks were trained by clustering the features of the observed data, in an unsupervised fashion; but, now, we have the opportunity to learn them jointly with the classifier, in a supervised fashion
- so, each basis function will have itself some parameters to learn, but what form should the function have?
- why not just like a classifier?
basis functions

- we need a small set of basis functions to cover the entire space, or at least the regions where our data live
- we did use fixed basis functions before: the Gabor filters discretized the 2d space of scales and orientations in uniform bins and their responses were used as vectors
- but right in the next layer, the dimensions increase and we cannot afford to have fixed basis functions everywhere: we have to learn from the data, as we did with the codebooks
- codebooks were trained by clustering the features of the observed data, in an unsupervised fashion; but, now, we have the opportunity to learn them jointly with the classifier, in a supervised fashion
- so, each basis function will have itself some parameters to learn, but what form should the function have?
- why not just like a classifier?

basis functions

- we need a small set of basis functions to cover the entire space, or at least the regions where our data live
- we did use fixed basis functions before: the Gabor filters discretized the 2d space of scales and orientations in uniform bins and their responses were used as vectors
- but right in the next layer, the dimensions increase and we cannot afford to have fixed basis functions everywhere: we have to learn from the data, as we did with the codebooks
- codebooks were trained by clustering the features of the observed data, in an unsupervised fashion; but, now, we have the opportunity to learn them jointly with the classifier, in a supervised fashion
- so, each basis function will have itself some parameters to learn, but what form should the function have?
- why not just like a classifier?

- we describe each sample with a feature vector obtained by a nonlinear function
- we model this function after a (binary) logistic regression unit: much like this unit can act as a classifier, it might also "detect" features that can be useful in the final classification

• layer 1
ightarrow "features"

$$\mathbf{a}_1 = W_1^\top \mathbf{x} + \mathbf{b}_1, \quad \mathbf{z} = h(\mathbf{a}_1) = h(W_1^\top \mathbf{x} + \mathbf{b}_1)$$

where h is a nonlinear activation function

• layer 2 \rightarrow class probabilities

 $\mathbf{a}_2 = W_2^\top \mathbf{z} + \mathbf{b}_2, \quad \mathbf{y} = \boldsymbol{\sigma}(\mathbf{a}_2) = \boldsymbol{\sigma}(W_2^\top \mathbf{z} + \mathbf{b}_2)$

• $oldsymbol{ heta} := (W_1, \mathbf{b}_1, W_2, \mathbf{b}_2)$ is the set of parameters to learn

- we describe each sample with a feature vector obtained by a nonlinear function
- we model this function after a (binary) logistic regression unit: much like this unit can act as a classifier, it might also "detect" features that can be useful in the final classification
- layer $1 \rightarrow$ "features"

$$\mathbf{a}_1 = W_1^\top \mathbf{x} + \mathbf{b}_1, \quad \mathbf{z} = h(\mathbf{a}_1) = h(W_1^\top \mathbf{x} + \mathbf{b}_1)$$

where h is a nonlinear activation function

• layer 2 \rightarrow class probabilities

$$\mathbf{a}_2 = W_2^\top \mathbf{z} + \mathbf{b}_2, \quad \mathbf{y} = \boldsymbol{\sigma}(\mathbf{a}_2) = \boldsymbol{\sigma}(W_2^\top \mathbf{z} + \mathbf{b}_2)$$

• $oldsymbol{ heta} := (W_1, \mathbf{b}_1, W_2, \mathbf{b}_2)$ is the set of parameters to learn

- we describe each sample with a feature vector obtained by a nonlinear function
- we model this function after a (binary) logistic regression unit: much like this unit can act as a classifier, it might also "detect" features that can be useful in the final classification
- layer $1 \rightarrow$ "features"

$$\mathbf{a}_1 = W_1^\top \mathbf{x} + \mathbf{b}_1, \quad \mathbf{z} = h(\mathbf{a}_1) = h(W_1^\top \mathbf{x} + \mathbf{b}_1)$$

where h is a nonlinear activation function

• layer 2 \rightarrow class probabilities

$$\mathbf{a}_2 = W_2^\top \mathbf{z} + \mathbf{b}_2, \quad \mathbf{y} = \boldsymbol{\sigma}(\mathbf{a}_2) = \boldsymbol{\sigma}(W_2^\top \mathbf{z} + \mathbf{b}_2)$$

• $oldsymbol{ heta} := (W_1, \mathbf{b}_1, W_2, \mathbf{b}_2)$ is the set of parameters to learn

- we describe each sample with a feature vector obtained by a nonlinear function
- we model this function after a (binary) logistic regression unit: much like this unit can act as a classifier, it might also "detect" features that can be useful in the final classification
- layer $1 \rightarrow$ "features"

$$\mathbf{a}_1 = W_1^\top \mathbf{x} + \mathbf{b}_1, \quad \mathbf{z} = h(\mathbf{a}_1) = h(W_1^\top \mathbf{x} + \mathbf{b}_1)$$

where h is a nonlinear activation function

• layer 2 \rightarrow class probabilities

$$\mathbf{a}_2 = W_2^\top \mathbf{z} + \mathbf{b}_2, \quad \mathbf{y} = \boldsymbol{\sigma}(\mathbf{a}_2) = \boldsymbol{\sigma}(W_2^\top \mathbf{z} + \mathbf{b}_2)$$

• $oldsymbol{ heta}:=(W_1,\mathbf{b}_1,W_2,\mathbf{b}_2)$ is the set of parameters to learn

activation function \boldsymbol{h}

- this should be nonlinear, otherwise the whole network will be linear and we don't gain much by the hierarchy (but: linear layers can be useful sometimes)
- it shouldn't have any more parameters, at least for now: all the parameters in a layer are $W, {\bf b}$
- it is a vector-to-vector function and there are still endless choices of nonlinear functions
- so we make the simplest choice for now: an element-wise function

• from the functions we saw previously, we leave polynomials and Gaussians out, and bring a couple more

activation function \boldsymbol{h}

- this should be nonlinear, otherwise the whole network will be linear and we don't gain much by the hierarchy (but: linear layers can be useful sometimes)
- it shouldn't have any more parameters, at least for now: all the parameters in a layer are W, \mathbf{b}
- it is a vector-to-vector function and there are still endless choices of nonlinear functions
- so we make the simplest choice for now: an element-wise function

• from the functions we saw previously, we leave polynomials and Gaussians out, and bring a couple more

activation function \boldsymbol{h}

- this should be nonlinear, otherwise the whole network will be linear and we don't gain much by the hierarchy (but: linear layers can be useful sometimes)
- it shouldn't have any more parameters, at least for now: all the parameters in a layer are W, \mathbf{b}
- it is a vector-to-vector function and there are still endless choices of nonlinear functions
- so we make the simplest choice for now: an element-wise function

• from the functions we saw previously, we leave polynomials and Gaussians out, and bring a couple more

▲□▶ ▲圖▶ ▲圖▶ ▲圖▶ → 圖 - 釣��







- tanh and sigmoid model exactly what a classifier makes (a decision), but they are smooth unlike sgn whose derivative is zero everywhere: indeed, they have been standard choices for decades.
- relu and its "soft" version softplus are like which functions we have seen?

・ロト・日本・モート モー うへぐ

- tanh and sigmoid model exactly what a classifier makes (a decision), but they are smooth unlike sgn whose derivative is zero everywhere: indeed, they have been standard choices for decades.
- relu and its "soft" version softplus are like which functions we have seen?

・ロト・日本・モート モー うへぐ



- $\operatorname{relu}(x) = [x]_+$ and $\zeta(x) = \log(1 + e^x)$ are the flipped versions of the perceptron and logistic loss functions, respectively
- also shown is the 0-1 misclassification loss, which is what we actually evaluate during testing: and why didn't we optimize that instead?

• because it's difficult: its derivative is zero everywhere



- $\operatorname{relu}(x) = [x]_+$ and $\zeta(x) = \log(1 + e^x)$ are the flipped versions of the perceptron and logistic loss functions, respectively
- also shown is the 0-1 misclassification loss, which is what we actually evaluate during testing: and why didn't we optimize that instead?

• because it's difficult: its derivative is zero everywhere



- $\operatorname{relu}(x) = [x]_+$ and $\zeta(x) = \log(1 + e^x)$ are the flipped versions of the perceptron and logistic loss functions, respectively
- also shown is the 0-1 misclassification loss, which is what we actually evaluate during testing: and why didn't we optimize that instead?

• because it's difficult: its derivative is zero everywhere



- $\operatorname{relu}(x) = [x]_+$ and $\zeta(x) = \log(1 + e^x)$ are the flipped versions of the perceptron and logistic loss functions, respectively
- also shown is the 0-1 misclassification loss, which is what we actually evaluate during testing: and why didn't we optimize that instead?
- because it's difficult: its derivative is zero everywhere

surrogate loss functions

- all three loss functions we have seen are surrogate (proxy) for the 0-1 loss: their derivative is constant for $x \to -\infty$
- they also often work better because even if the 0-1 loss is low (or zero) on the training set, they improve on the test set
- even so, we could have used a sigmoid instead, which is the smooth version of the 0-1 loss, but we didn't: its derivative tends to zero for $x\to -\infty$
- so if just one sigmoid is harder than relu, softplus *etc.* in a linear classifier, why use 100 of those in the hidden units of a two-layer network?

ション 人口 マイビン トレート シックション

surrogate loss functions

- all three loss functions we have seen are surrogate (proxy) for the 0-1 loss: their derivative is constant for $x \to -\infty$
- they also often work better because even if the 0-1 loss is low (or zero) on the training set, they improve on the test set
- even so, we could have used a sigmoid instead, which is the smooth version of the 0-1 loss, but we didn't: its derivative tends to zero for $x\to -\infty$
- so if just one sigmoid is harder than relu, softplus *etc.* in a linear classifier, why use 100 of those in the hidden units of a two-layer network?

surrogate loss functions

- all three loss functions we have seen are surrogate (proxy) for the 0-1 loss: their derivative is constant for $x \to -\infty$
- they also often work better because even if the 0-1 loss is low (or zero) on the training set, they improve on the test set
- even so, we could have used a sigmoid instead, which is the smooth version of the 0-1 loss, but we didn't: its derivative tends to zero for $x\to -\infty$
- so if just one sigmoid is harder than relu, softplus *etc.* in a linear classifier, why use 100 of those in the hidden units of a two-layer network?



- in LSH, we used a number of random projections followed by sgn to produce a binary code as a description of x
- here, we use again a number of (initially) random projections followed by relu instead, acting like a switch: half space is zeroed out, the other half passes through
- in the nonzero part, gradients are also nonzero and we can use them to adapt the projections themselves

Charikar. STOC 2002. Similarity Estimation Techniques From Rounding Algorithms.



linear

two-layer

▲□▶ ▲圖▶ ▲臣▶ ▲臣▶ 二臣 - わへぐ



linear

two-layer

▲□▶ ▲圖▶ ▲臣▶ ▲臣▶ 二臣 - わへぐ



linear

two-layer

▲□▶ ▲圖▶ ▲臣▶ ▲臣▶ 二臣 - わへぐ



linear

two-layer

▲□▶ ▲圖▶ ▲臣▶ ▲臣▶ 二臣 - わへぐ



linear

two-layer

▲□▶ ▲圖▶ ▲臣▶ ▲臣▶ 二臣 - わへぐ



linear

two-layer

▲□▶ ▲圖▶ ▲臣▶ ▲臣▶ 二臣 - わへぐ



linear

two-layer

◆□▶ ◆□▶ ◆臣▶ ◆臣▶ 三臣 - のへぐ

•
$$k = 3, n = 300, m = 100, \epsilon = 10^{0}, \lambda = 10^{-3}$$



linear

two-layer

◆□▶ ◆□▶ ◆臣▶ ◆臣▶ 三臣 - のへぐ

•
$$k = 3, n = 300, m = 100, \epsilon = 10^{0}, \lambda = 10^{-3}$$



linear

two-layer

◆□▶ ◆□▶ ◆臣▶ ◆臣▶ 三臣 - のへぐ

•
$$k = 3, n = 300, m = 100, \epsilon = 10^{0}, \lambda = 10^{-3}$$



linear

two-layer

▲□▶ ▲圖▶ ▲臣▶ ▲臣▶ 二臣 - わへぐ

MNIST digits dataset

4 5 6 5

• 10 classes, 60k training images, 10k test images, 28 imes 28 images

LeCun, Bottou, Bengio and Haffner IEEE Proc. 1998. Gradient-Based Learning Applied to Document Recognition.



- input layer 1 weights and bias relu activation function layer 2 weights and bias softmax
- parameter learning using cross-entropy on ${f y}$ (or rather, directly on ${f a}_2)$



- input layer 1 weights and bias relu activation function layer 2 weights and bias softmax
- ullet parameter learning using cross-entropy on ${f y}$ (or rather, directly on ${f a}_2)$



- input layer 1 weights and bias relu activation function layer 2 weights and bias softmax
- parameter learning using cross-entropy on ${f y}$ (or rather, directly on ${f a}_2)$



- input layer 1 weights and bias relu activation function layer 2 weights and bias softmax
- ullet parameter learning using cross-entropy on ${f y}$ (or rather, directly on ${f a}_2)$


- input layer 1 weights and bias relu activation function layer 2 weights and bias softmax
- ullet parameter learning using cross-entropy on ${f y}$ (or rather, directly on ${f a}_2)$

▲ロ → ▲周 → ▲目 → ▲目 → ● ● ● ● ●



- input layer 1 weights and bias relu activation function layer 2 weights and bias softmax
- ullet parameter learning using cross-entropy on ${f y}$ (or rather, directly on ${f a}_2)$



- input layer 1 weights and bias relu activation function layer 2 weights and bias softmax
- ullet parameter learning using cross-entropy on ${f y}$ (or rather, directly on ${f a}_2)$



- input layer 1 weights and bias relu activation function layer 2 weights and bias softmax
- ullet parameter learning using cross-entropy on ${f y}$ (or rather, directly on ${f a}_2)$



- input layer 1 weights and bias relu activation function layer 2 weights and bias - softmax
- parameter learning using cross-entropy on ${f y}$ (or rather, directly on ${f a}_2)$

what is being learned?

- the columns of W_1 are multiplied with ${\bf x};$ they live in the same space, as in the linear classifier
- we can reshape each one back from 784×1 to 28×28 : but now it shouldn't look like a digit; rather, like a pattern that might help in recognizing digits
- these patterns are shared: once the activations are computed, they can be used in the next layer to score any of the digits
- the columns of W_2 are in an 100-dimensional space that we can't make much sense of now; but we'll revisit this later

layer 1 weights 00-09



- $k = 10, n = 60000, m = 6000, \epsilon = 10^{-1}, \lambda = 10^{-4}$
- hidden layer width: 100; test error 2.54%

layer 1 weights 10-19



- $k = 10, n = 60000, m = 6000, \epsilon = 10^{-1}, \lambda = 10^{-4}$
- hidden layer width: 100; test error 2.54%

layer 1 weights 20-29



- $k = 10, n = 60000, m = 6000, \epsilon = 10^{-1}, \lambda = 10^{-4}$
- hidden layer width: 100; test error 2.54%

layer 1 weights 30-39



- $k = 10, n = 60000, m = 6000, \epsilon = 10^{-1}, \lambda = 10^{-4}$
- hidden layer width: 100; test error 2.54%

layer 1 weights 40-49



- $k = 10, n = 60000, m = 6000, \epsilon = 10^{-1}, \lambda = 10^{-4}$
- hidden layer width: 100; test error 2.54%

layer 1 weights 50-59



- $k = 10, n = 60000, m = 6000, \epsilon = 10^{-1}, \lambda = 10^{-4}$
- hidden layer width: 100; test error 2.54%

layer 1 weights 60-69



- $k = 10, n = 60000, m = 6000, \epsilon = 10^{-1}, \lambda = 10^{-4}$
- hidden layer width: 100; test error 2.54%

layer 1 weights 70-79



- $k = 10, n = 60000, m = 6000, \epsilon = 10^{-1}, \lambda = 10^{-4}$
- hidden layer width: 100; test error 2.54%

layer 1 weights 80-89



- $k = 10, n = 60000, m = 6000, \epsilon = 10^{-1}, \lambda = 10^{-4}$
- hidden layer width: 100; test error 2.54%

layer 1 weights 90-99



- $k = 10, n = 60000, m = 6000, \epsilon = 10^{-1}, \lambda = 10^{-4}$
- hidden layer width: 100; test error 2.54%

summary

- only care about learning features: so, not interested *e.g.* in nearest neighbor search or dual SVM formulation
- three different linear classifiers, perceptron, SVM and logistic regression, only differ slightly in their loss function, which is similar to relu in all cases
- stochastic gradient descent optimization
- multi-class classification, $\operatorname{softmax}$ and MNIST
- linear regression, basis functions, overfitting, validation, hyperparameter optimization
- learning basis functions, two-layer networks, activation functions, connection to classifier loss functions

• why relu makes sense