Online Learning with Support Vectors

Alexander Johannes Smola
Department of Engineering and RSISE
Australian National University
Canberra, 0200 ACT, Australia
Alex.Smola@anu.edu.au

Joint work with **Jyrki Kivinen** and **Bob Williamson**

THE AUSTRALIAN
NATIONAL UNIVERSITY

Outline

THE AUSTRALIAN NATIONAL UNIVERSITY

The Basic Ingredients of a Kernel Machine

Empirical Risk, Regularization, Feature Space, Quadratic Programming

Problems with Batch Learning

Scalability, Adaptation, Difficult to code

Stochastic Gradient Descent

Stochastic Approximation, Update Rules, Truncation

The Return of the Perceptron

Kernel Perceptron, + regularization, + margin

Examples

Ingredient 1: Empirical Risk

THE AUSTRALIAN NATIONAL UNIVERSITY

What we have

We have some observations $\mathbf{x}_1, \ldots, \mathbf{x}_m \subset \mathcal{X}$ with corresponding labels $y_1, \ldots, y_m$, drawn (independently identically distributed) from some $\Pr(\mathbf{x}, y)$.

Goal

Find some function $f : \mathcal{X} \to \mathcal{Y}$ which predicts y at some possibly new and unknown location $\mathbf{x}$ which minimizes the cost $c(\mathbf{x}, y, f(\mathbf{x}))$ for predicting $f(\mathbf{x})$ rather than y.

Standard Trick

We don't know $\Pr(\mathbf{x}, y)$, so we replace it by the data we have and we minimize the **training error**, also called the **empirical risk**

$$R_{\text{emp}}[f] := \frac{1}{m} \sum_{i=1}^{m} c(\mathbf{x}_i, y_i, f(\mathbf{x}_i))$$

Examples

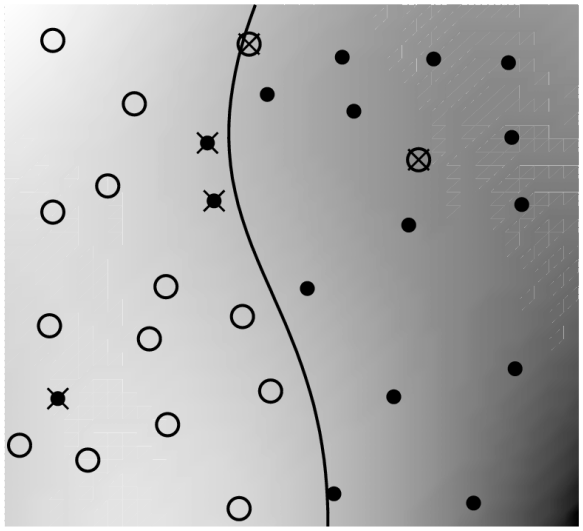
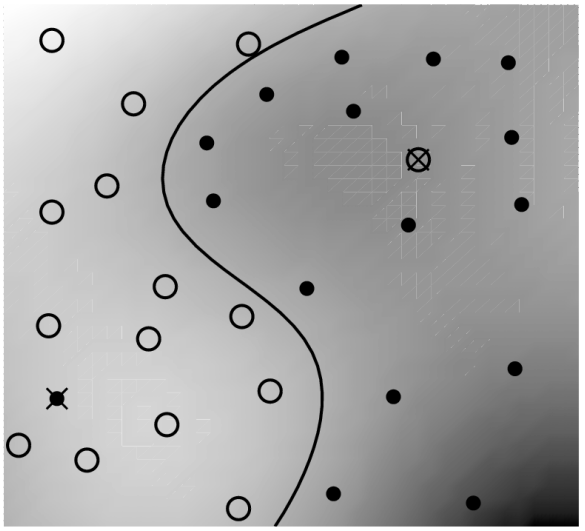
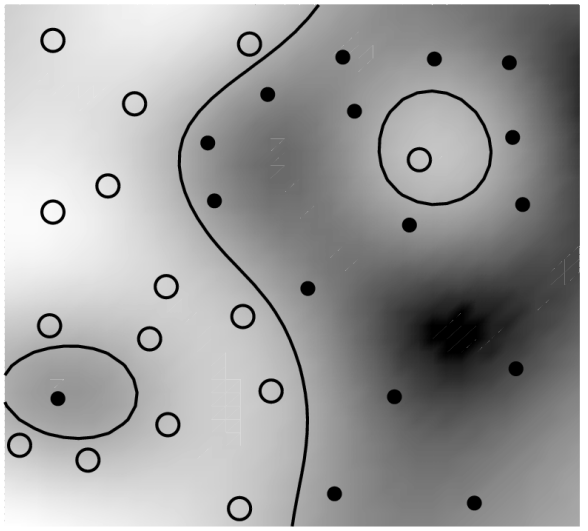
Misclassification (apples and oranges), confidence level (cancer diagnosis), prediction (value of a stock tomorrow), novelty detection (network intrusion, car alarm).

Ingredient 2: Regularization

THE AUSTRALIAN NATIONAL UNIVERSITY

Problem

If we allow just any function f as a predictor, we typically will get overfitting and numerical problems.

Solution

Restrict the class of possible functions, by introducing a trade-off term, to be added to $R_{\text{emp}}[f]$, also called regularization term $\Omega[f]$.

Ingredient 2: Regularization

Specific Solution for Kernel Methods

We choose linear function (we come to a fancier setting later) f and a quadratic regularization term $\Omega[f]$, given by

$$f(\mathbf{x}) := \langle \mathbf{w}, \mathbf{x} \rangle + b \text{ and } \Omega[f] := \frac{1}{2}\|\mathbf{w}\|^2$$

This means that we favour **flat** functions. Usually we do not regularize b, but this more a matter of taste than anything else.

Regularized Risk Functional

To specify how much more we prefer flat functions f than good minimizers of the empirical risk (training error) we use a regularization constant $\lambda > 0$. We obtain

$$R_{\text{reg}}[f] = R_{\text{emp}}[f] + \lambda\Omega[f] = \frac{1}{m}\sum_{i=1}^{m} c(\mathbf{x}_i, y_i, f(\mathbf{x}_i)) + \frac{\lambda}{2}\|\mathbf{w}\|^2.$$

This looks and feels like the **weight decay** term in Neural Networks ...

Ingredient 3: Feature Spaces

THE AUSTRALIAN NATIONAL UNIVERSITY

Problem

Linear functions are boring. We want something much fancier than that. And we want to use our prior knowledge of what f should be like.

Solution

Compute features $\Phi(\mathbf{x})$ which are more like the functions we want. This gives us nonlinearity by

$$f(\mathbf{x}) = \langle \mathbf{w}, \Phi(\mathbf{x}) \rangle + b$$

Problem

What, if we go and compute too many features, say all 5th order monomials of a 16×16 picture (that's almost 10^{10} features). Or, if we don't quite know which features to get?

Solution: Kernels (Aizerman et al, 1964, Boser et al, 1992)

If our algorithm only needs dot products $\langle \Phi(\mathbf{x}), \Phi(\mathbf{x}') \rangle$, we may compute the overlap between features implicitly by a kernel function $k(\mathbf{x}, \mathbf{x}') := \langle \Phi(\mathbf{x}), \Phi(\mathbf{x}') \rangle$

Ingredient 3: Feature Spaces — Examples

THE AUSTRALIAN NATIONAL UNIVERSITY

Dot Product Kernels (Schoenberg 1942, Poggio 1975, Smola et al, 2000)

For feature spaces with dth order kernels use

$$k(\mathbf{x}, \mathbf{x}') = (\langle \mathbf{x}, \mathbf{x}' \rangle + c)^d \text{ where } c \geq 0$$

This is quite similar to polynomial networks. We can show that any of the following kernel will do

$$k(\mathbf{x}, \mathbf{x}') = \kappa(\langle \mathbf{x}, \mathbf{x}' \rangle) = \sum_{l=1}^{\infty} a_l \langle \mathbf{x}, \mathbf{x}' \rangle^l \text{ where } a_l \geq 0$$

The 'tanh-kernel,' as used in sigmoid networks is not a proper kernel.

Radial Basis Function Kernels (Bochner 1932, Aizerman et al, 1964)

A useful kernel is the Gaussian RBF kernel

$$k(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\sigma^2}\right).$$

This is known from RBF networks. In general any of the following kernels will do

$$k(\mathbf{x}, \mathbf{x}') = \kappa(\|\mathbf{x} - \mathbf{x}'\|) \text{ where } \tilde{\kappa}(\omega) \geq 0 \text{ for all } \omega.$$

Problems with Batch 1: Scalability

THE AUSTRALIAN NATIONAL UNIVERSITY

Representer Theorem (Kimeldorf et al, 1971, Schölkopf et al, 2001)

When minimizing a regularized risk functional, the estimate $f(\mathbf{x})$ is given by

$$f(\mathbf{x}) = \sum_{i=1}^{m} \alpha_i k(\mathbf{x}_i, \mathbf{x}) + b$$

And a significant fraction of the α_i are nonzero (only for very clean data and special loss functions this can be different).

Problem (Joachims, 1999, Platt 1999, Kowalczyk 2000)

Training complexity increases with sample size and number of basis functions. Typically training time scales with $O(m^{2+\gamma})$ and prediction time with $O(m)$. $O(m^3)$ is an upper bound for the training time and $O(m^2)$ an upper bound for the storage requirements.

Problem

Coding an efficient algorithm is usually quite difficult ...

Problems with Batch 2: Drift and Offline

Problem

What happens if the distribution which generated $(\mathbf{x}, y)$ changes over time?

Example

Novelty detection with an aging device (parameter drift), predicting the stock market (adaptation to betting on variance, butterfly portfolio, ...), dynamical system.

Problem

What happens if we want to have a good predictor instantly.

Example

Real-time systems, security and surveillance applications ("someone hacked your computer three weeks ago", "this plane was going to crash one hour ago", ... is rather useless information).

Stochastic Gradient Descent 1: Risk

THE AUSTRALIAN NATIONAL UNIVERSITY

Idea

In batch learning we want to minimize the regularized risk functional

$$R_{\text{reg}}[f] = \frac{1}{m}\sum_{i=1}^{m} c(\mathbf{x}_i, y_i, f(\mathbf{x}_i)) + \frac{\lambda}{2}\|\mathbf{w}\|^2.$$

Approximate the empirical risk term by the loss at time t and perform gradient descent with respect to it. This yields

$$R_{\text{stoch}}[f, t] = c(\mathbf{x}_t, y_t, f(\mathbf{x}_t)) + \frac{\lambda}{2}\|\mathbf{w}\|^2.$$

We perform the updates $\mathbf{w} \leftarrow \mathbf{w} - \Lambda\partial_{\mathbf{w}} R_{\text{stoch}}[f, t]$.

Good News (Kivinen et al, 2001)

One can show that stochastic gradient descent with respect to $R_{\text{stoch}}[f, t]$ will converge to the minimum of $R_{\text{reg}}[f]$.

Stochastic Gradient Descent 2: Updates

THE AUSTRALIAN NATIONAL UNIVERSITY

Explicit Update Rule

For a given learning rate (we will come to that later) we have

$$\begin{aligned}
\mathbf{w} &\leftarrow \mathbf{w} - \Lambda \partial_{\mathbf{w}} R_{\text{stoch}}[f, t] \\
&= \mathbf{w} - \Lambda \partial_{\mathbf{w}} \left(c(\mathbf{x}_t, y_t, \langle \mathbf{w}, \Phi(\mathbf{x}_t) \rangle + b) + \frac{\lambda}{2} \|\mathbf{w}\|^2 \right) \\
&= (1 - \Lambda\lambda)\mathbf{w} - \Lambda c'(\mathbf{x}_t, y_t, f(\mathbf{x}_t))\Phi(\mathbf{x}_t) \\
b &\leftarrow b - \Lambda c'(\mathbf{x}_t, y_t, f(\mathbf{x}_t))
\end{aligned}$$

Explicit Kernel Notation

For a function $f(\mathbf{x})$ given by $f(\mathbf{x}) = \sum_l \alpha_l \langle \Phi(\mathbf{x}_l), \Phi(\mathbf{x}) + b = \sum_l \alpha_l k(\mathbf{x}_l, \mathbf{x}) + b$

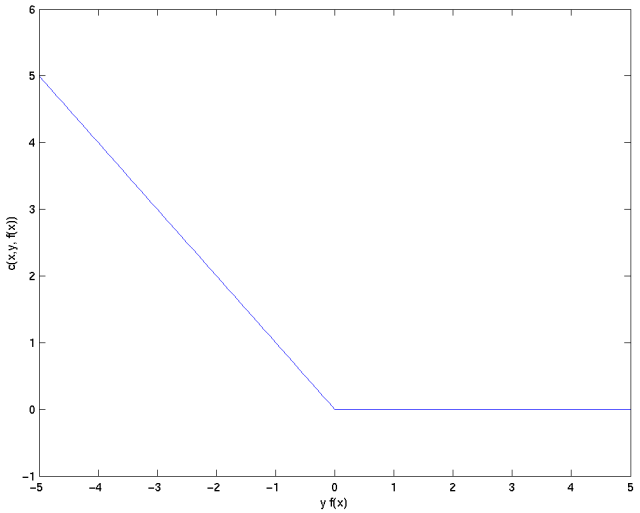
we get the update in the coefficients as

$$\alpha_i \leftarrow \begin{cases} (1 - \Lambda\lambda)\alpha_i & \text{for } i \neq t \\ -\Lambda c'(\mathbf{x}_t, y_t, f(\mathbf{x}_t)) & \text{for } i = t \end{cases} \quad \text{and} \quad b \leftarrow b - \Lambda c'(\mathbf{x}_t, y_t, f(\mathbf{x}_t))$$

Stochastic Gradient Descent 3: Examples

THE AUSTRALIAN NATIONAL UNIVERSITY

The Perceptron Learning Rule



We set the regularization term $\lambda = 0$ and use the hinge loss

$$c(\mathbf{x}, y, f(\mathbf{x})) = \max(0, -yf(\mathbf{x}))$$

Here the derivative of c is given by

$$c'(\mathbf{x}, y, f(\mathbf{x})) = \begin{cases} -y & \text{if } yf(\mathbf{x}) < 0 \\ 0 & \text{otherwise} \end{cases}$$

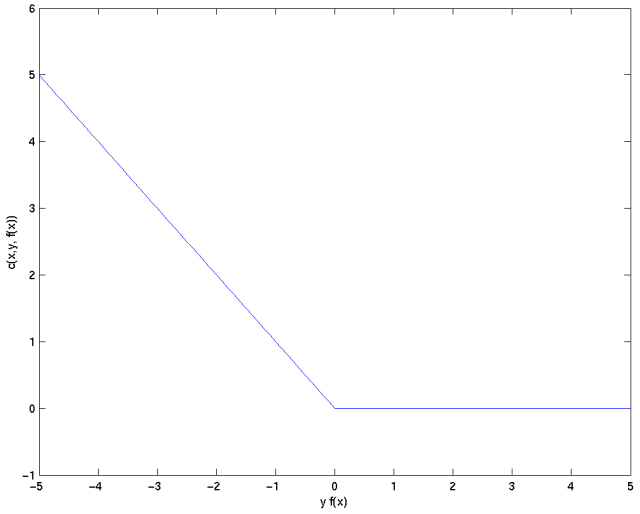
Kernel Perceptron

$$\alpha_t = \begin{cases} \Lambda y & \text{if } yf(\mathbf{x}) < 0 \\ 0 & \text{otherwise} \end{cases} \quad \text{and } b \leftarrow b + \begin{cases} \Lambda y & \text{if } yf(\mathbf{x}) < 0 \\ 0 & \text{otherwise} \end{cases}$$

Stochastic Gradient Descent 3: Examples

THE AUSTRALIAN NATIONAL UNIVERSITY

The Perceptron with Regularization (weight decay)



We use a nonzero regularization term λ and the hinge loss

$$c(\mathbf{x}, y, f(\mathbf{x})) = \max(0, -yf(\mathbf{x}))$$

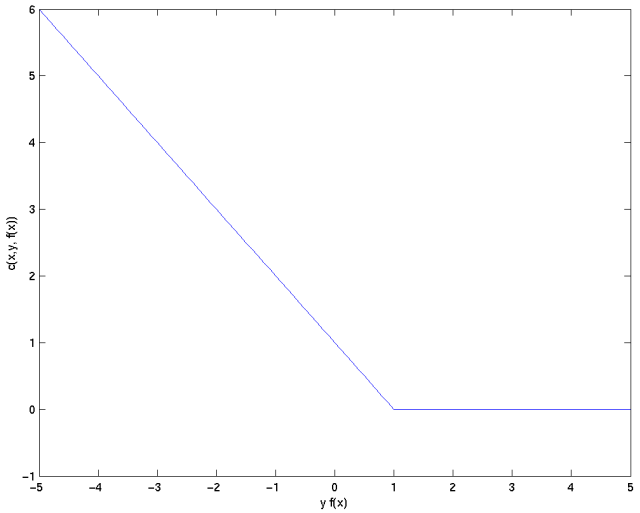
Here the derivative of c is given by

$$c'(\mathbf{x}, y, f(\mathbf{x})) = \begin{cases} -y & \text{if } yf(\mathbf{x}) < 0 \\ 0 & \text{otherwise} \end{cases}$$

Regularized Kernel Perceptron

$$\alpha_i = (1 - \lambda\Lambda)\alpha_i \text{ and } \alpha_t = \begin{cases} \Lambda y_t & \text{if } y_t f(\mathbf{x}_t) < 0 \\ 0 & \text{otherwise} \end{cases} \text{ and } b \leftarrow b + \begin{cases} \Lambda y_t & \text{if } y_t f(\mathbf{x}_t) < 0 \\ 0 & \text{otherwise} \end{cases}$$

Large Margin Perceptron with Regularization



We use a nonzero regularization term λ and the soft margin loss

$$c(\mathbf{x}, y, f(\mathbf{x})) = \max(0, 1 - yf(\mathbf{x}))$$

Here the derivative of c is given by

$$c'(\mathbf{x}, y, f(\mathbf{x})) = \begin{cases} -y & \text{if } yf(\mathbf{x}) < 1 \\ 0 & \text{otherwise} \end{cases}$$

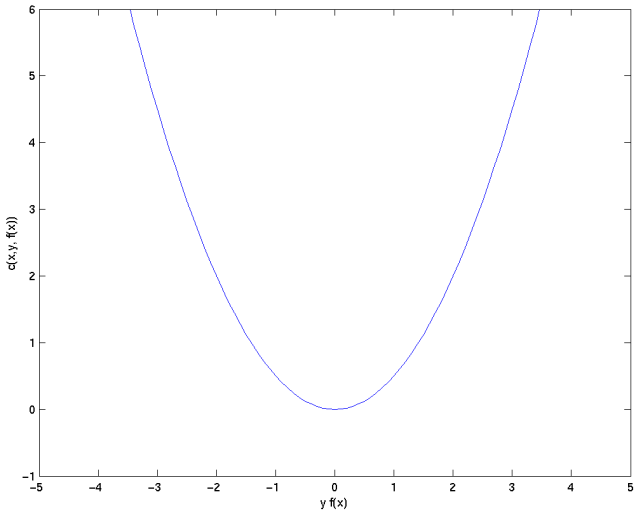
Regularized Kernel Perceptron

$$\alpha_i = (1 - \lambda\Lambda)\alpha_i \text{ and } \alpha_t = \begin{cases} \Lambda y_t & \text{if } y_t f(\mathbf{x}_t) < 1 \\ 0 & \text{otherwise} \end{cases} \text{ and } b \leftarrow b + \begin{cases} \Lambda y_t & \text{if } y_t f(\mathbf{x}_t) < 1 \\ 0 & \text{otherwise} \end{cases}$$

Stochastic Gradient Descent 3: Examples

THE AUSTRALIAN NATIONAL UNIVERSITY

Least Mean Square Regression with Regularization



We use a nonzero regularization term λ and squared loss

$$c(\mathbf{x}, y, f(\mathbf{x})) = (f(\mathbf{x}) - y)^2$$

Here the derivative of c is given by

$$c'(\mathbf{x}, y, f(\mathbf{x})) = f(\mathbf{x}) - y$$

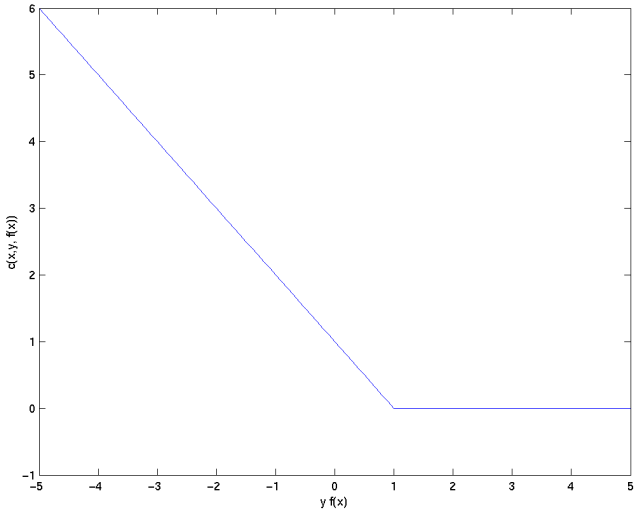
Explicit Update Equations

$$\alpha_i = (1 - \lambda\Lambda)\alpha_i \text{ and } \alpha_t = \Lambda(y - f(\mathbf{x}_t)) \text{ and } b \leftarrow b + \Lambda(y - f(\mathbf{x}_t))$$

Stochastic Gradient Descent 3: Examples

THE AUSTRALIAN NATIONAL UNIVERSITY

Novelty Detection



We use a nonzero regularization term λ and the novelty detection loss

$$c(\mathbf{x}, f(\mathbf{x})) = \max(0, 1 - f(\mathbf{x}))$$

Here the derivative of c is given by

$$c'(\mathbf{x}, y, f(\mathbf{x})) = \begin{cases} -1 & \text{if } f(\mathbf{x}) < 1 \\ 0 & \text{otherwise} \end{cases}$$

We classify an event as novel if $f(\mathbf{x}) < 1$.

Explicit Update Equations

$$\alpha_i = (1 - \lambda\Lambda)\alpha_i \text{ and } \alpha_t = \begin{cases} \Lambda & \text{if } f(\mathbf{x}_t) < 1 \\ 0 & \text{otherwise} \end{cases} \text{ and } b \leftarrow b + \begin{cases} \Lambda & \text{if } f(\mathbf{x}_t) < 1 \\ 0 & \text{otherwise} \end{cases}$$

Stochastic Gradient Descent 3: Truncation

Problem

We have to add one kernel function $k(\mathbf{x}_t, \mathbf{x})$ every time we make a mistake, commit a margin error, see a novel event, and also pretty much every time in regression.

This means that prediction time increases linearly with the number of data we see.

Good News

All coefficients α_i decay with $(1 - \lambda\Lambda)$ per iteration. So, all we need to do is wait until they are small enough. In particular, after τ steps, we have

$$\begin{aligned} \|\alpha_{t-\tau}\Phi(\mathbf{x}_{t-\tau})\| &\leq \Lambda(1-\lambda\Lambda)^\tau |c'(\mathbf{x}_{t-\tau}, y_{t-\tau}, f_{t-\tau}(\mathbf{x}_{t-\tau}))| \|\Phi(\mathbf{x}_{t-\tau}\| \\ &\leq \Lambda C\kappa(1-\lambda\Lambda)^\tau \end{aligned}$$

Here $|c'(\mathbf{x}, y, f(\mathbf{x}))| \leq C$ (for the hinge loss $C = 1$), and $k(\mathbf{x}, \mathbf{x}) \leq \kappa^2$ (for RBF kernels $\kappa = 1$).

So, by dumping $\alpha_{t-\tau}\Phi(\mathbf{x}_{t-\tau})$ we commit an error of no more than $\Lambda C\kappa(1-\lambda\Lambda)^\tau$.

Stochastic Gradient Descent 3: Truncation

Geometric Series Expansion

The error by discarding all terms older than τ is bounded by $\lambda^{-1}C\kappa(1-\lambda\Lambda)^{\tau}$.

Interpretation

The term $(1-\lambda\Lambda)$ determines how many past instances we keep. The larger $\lambda\Lambda$ becomes, the fewer instances we keep. This can be done for two reasons:

Small Memory Footprint: we do not have enough memory (or CPU power) to deal with many terms in the expansion

Rapidly Changing Distribution: if $\Pr(\mathbf{x}, y)$ changes too rapidly, it does not make sense to store too old data.

Effective Limit on Learning

The fact that we can only store a finite number of examples limits the capacity of the estimator. It depends on the regularization constant and the speed of updates.

Stochastic Gradient Descent 4: ν-trick

THE AUSTRALIAN NATIONAL UNIVERSITY

Goal

We want to limit the number of updates we make. And get rid of one parameter.

Solution

Make the threshold parameter (so far set to 1) adaptive (works for classification, regression, novelty detection) and decrease it if too many updates are made and vice versa.

Example: Novelty Detection

Standard Loss Function

$$c(\mathbf{x}, f(\mathbf{x})) = \max(0, 1 - f(\mathbf{x}))$$

We classify an event as novel if $f(\mathbf{x}) < 1$.

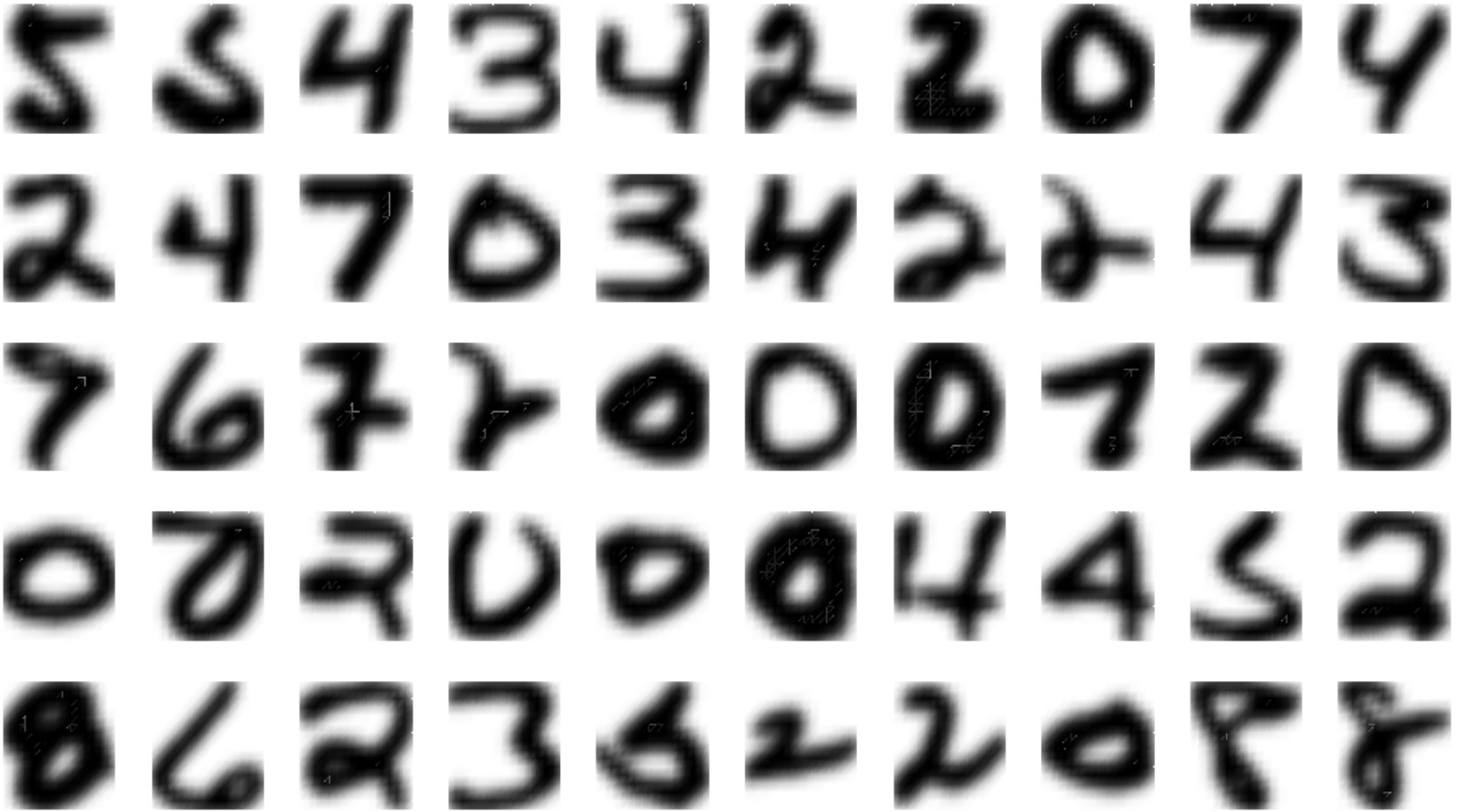
Loss Function with ν

$$c(\mathbf{x}, f(\mathbf{x})) = \max(0, \rho - f(\mathbf{x})) - \nu\rho$$

The updates on ρ are $\Lambda\nu$ if the pattern is **not novel**, and $\Lambda(\nu - 1)$ if the pattern **is novel**. On average, a fraction of ν points will be classified as novel.

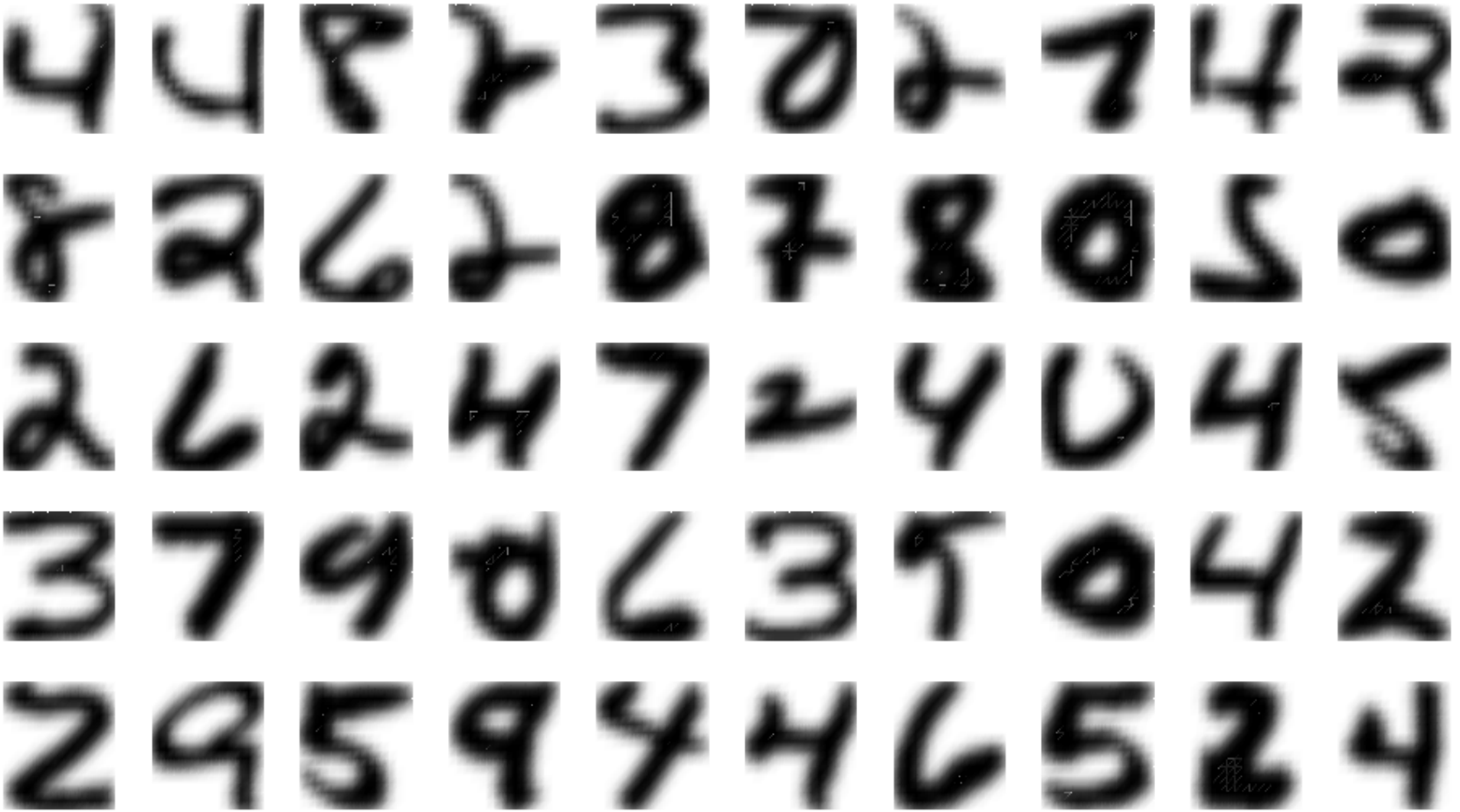
Online Training Run

THE AUSTRALIAN NATIONAL UNIVERSITY

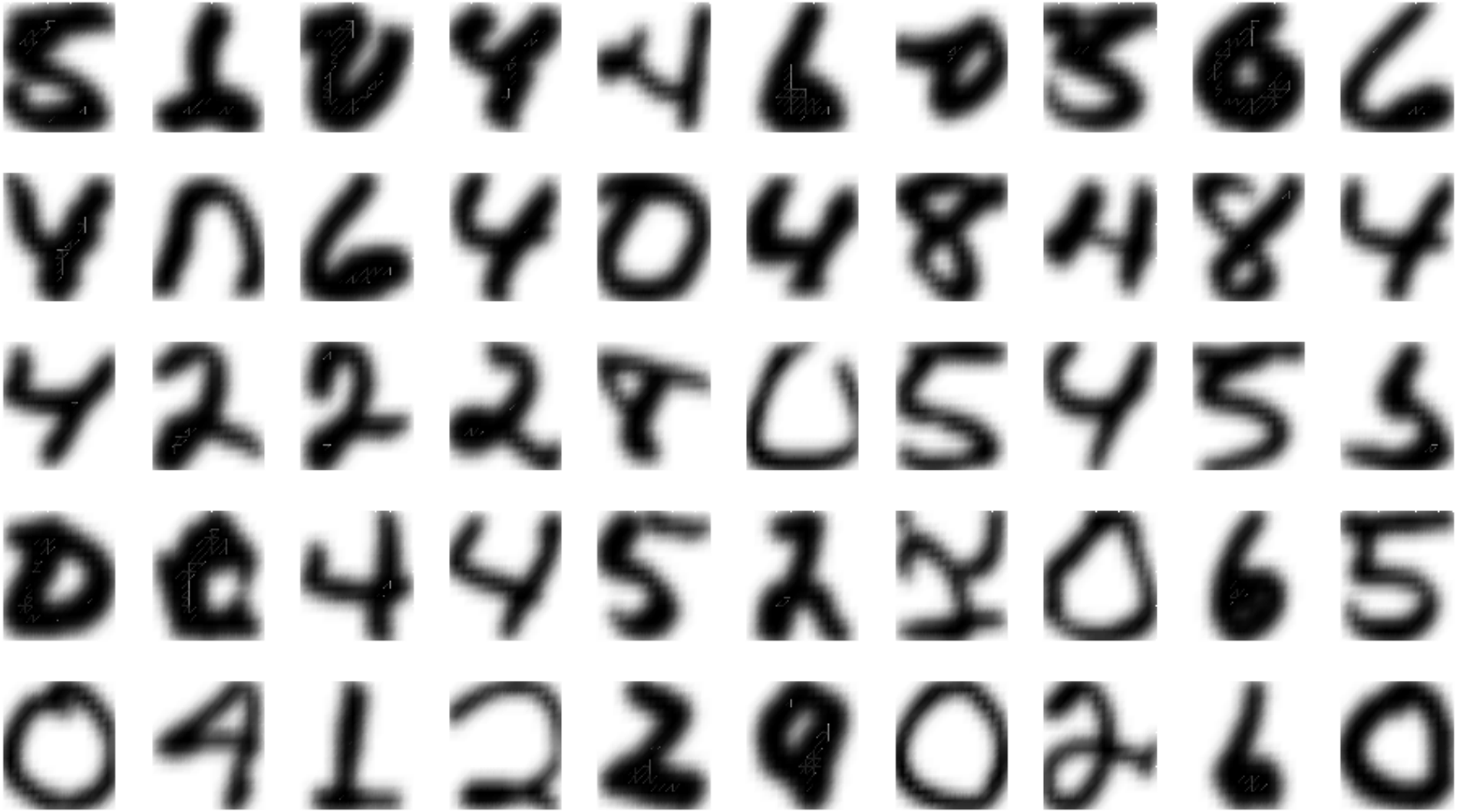


Worst Training Examples

THE AUSTRALIAN NATIONAL UNIVERSITY



Worst Test Examples



Summary and Outlook

Related Work

(Csato and Opper, 2001, Gentile 2001, Herbster 2001, Tresp 2000)

Extensions

We can use the ν trick also for Huber's robust loss function (not possible for SV batch learning). This gives us trimmed mean estimators.

Stochastic Gradient Descent in Feature Space

Can be used for many other applications (pretty much anything that works with kernels).

Weight Decay Term

Keeps expansion tractable.

For more information see

http://www.kernel-machines.org