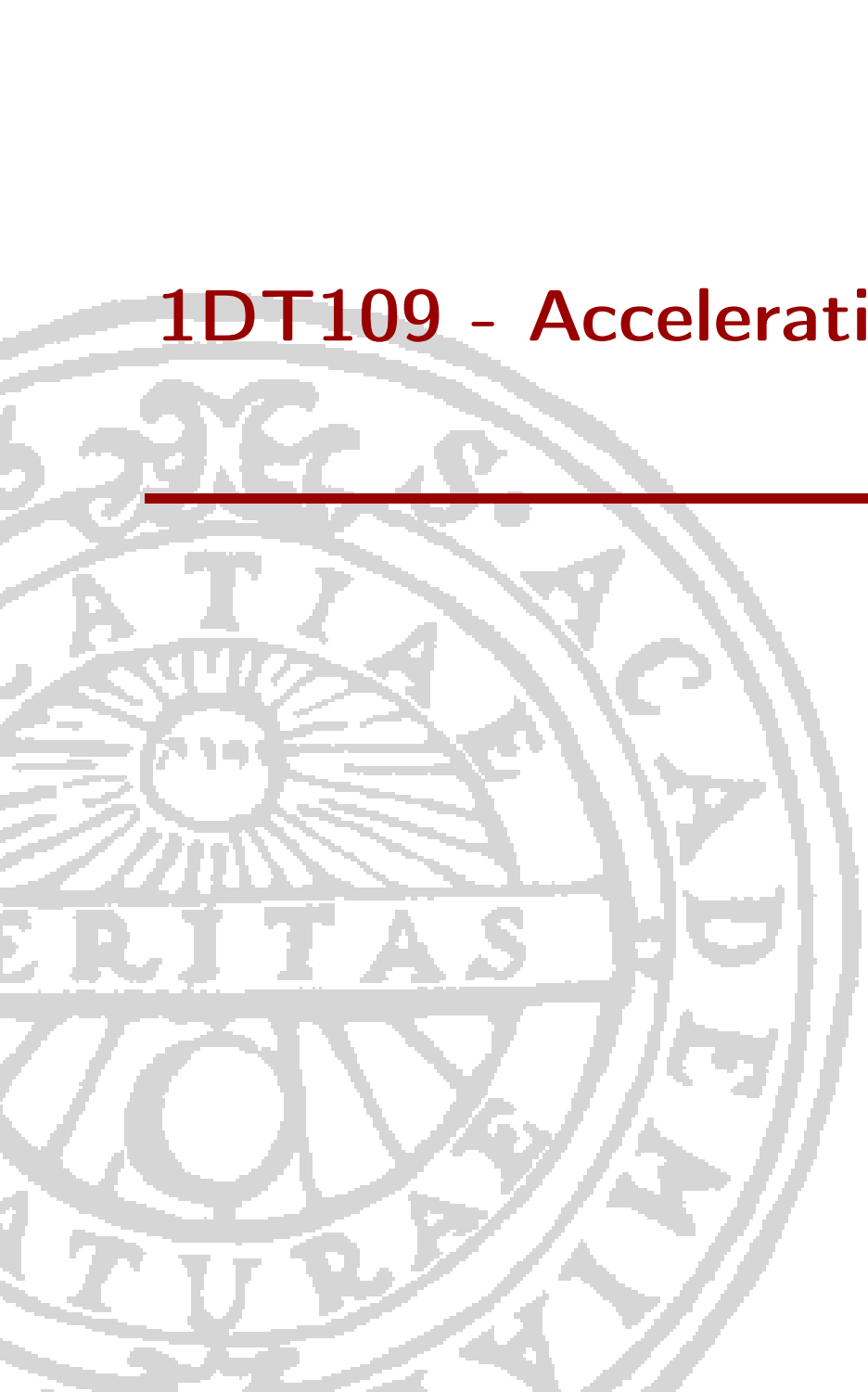


1DT109 - Accelerating systems with FPGAs

How to improve learning

Riccardo De Masellis

Uppsala University

December 6, 2021

The basis

Supervised learning:

- $\boldsymbol{x}$ input vector;
- $y(\boldsymbol{x})$ unknown function we want to approximate;
- $\sigma(\boldsymbol{x}, \boldsymbol{w})$ hypothesis function;
- loss and cost function.

What we have seen so far

We used squared error loss (SE) function:

$$L_{sel}(\boldsymbol{x}, y, \sigma) = (y(\boldsymbol{x}) - \sigma(\boldsymbol{x}, \boldsymbol{w}))^2 = (y(\boldsymbol{x}) - \frac{1}{1 + e^{-(\boldsymbol{w}^T \boldsymbol{x})}})^2$$

What we have seen so far

We used squared error loss (SE) function:

$$L_{sel}(\boldsymbol{x}, y, \sigma) = (y(\boldsymbol{x}) - \sigma(\boldsymbol{x}, \boldsymbol{w}))^2 = (y(\boldsymbol{x}) - \frac{1}{1 + e^{-(\boldsymbol{w}^T \boldsymbol{x})}})^2$$

The cost function is therefore:

$$C(\boldsymbol{w}) = \frac{1}{2n} \sum_{i=1}^{n} L_{sel}(\boldsymbol{x}, y, \sigma) = \frac{1}{2n} \sum_{i=1}^{n} (y(\boldsymbol{x}_i) - \frac{1}{1 + e^{-(\boldsymbol{w}^T \boldsymbol{x}_i)}})^2$$

where n is the number of examples in the training set.

Gradient descent

$\boldsymbol{w}^{new} := [-\varepsilon, +\varepsilon]$ (ε close to zero, e.g., 0.05) IMPORTANT!

do {

$\boldsymbol{w}^{old} := \boldsymbol{w}^{new}$

$w_0^{new} := w_0^{old} - \eta \frac{\partial C}{\partial w_0}(\boldsymbol{w}^{old})$

...

$w_m^{new} := w_m^{old} - \eta \frac{\partial C}{\partial w_m}(\boldsymbol{w}^{old})$

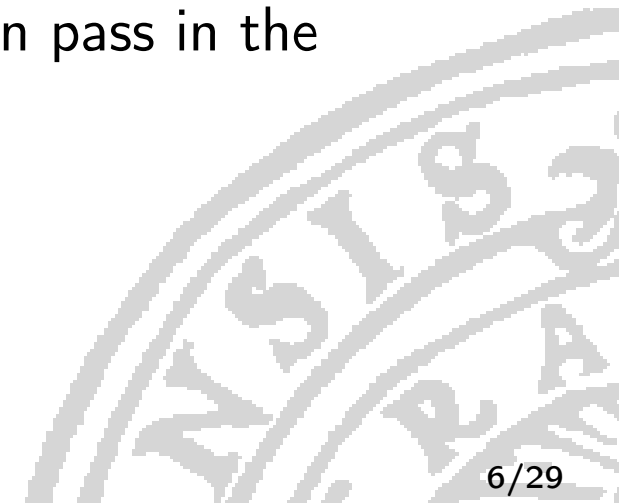
} while ($C(\boldsymbol{w}^{new}) < C(\boldsymbol{w}^{old})$)

Making gradient descent (GD) faster

Specifically for (FF) neural networks:

- stochastic GD: batch size = 1;
- batch GD: batch = training set;
- mini-batch GD: 1 < batch size < training set size;
- epoch.

Number of feed-forward pass and backpropagation pass in the above (training set size = n)?



Problems with squared error loss

EASY SETTING: one neuron, binary classification, two weights w (including bias) and one training example x.

Problems with squared error loss

EASY SETTING: one neuron, binary classification, two weights w (including bias) and one training example x.

$$C(\boldsymbol{w}) = \frac{1}{2}(y(x) - \sigma(x, \boldsymbol{w}))^2$$

$$\frac{\partial C}{\partial w_0} = (\sigma(x, \boldsymbol{w}) - y(x))\frac{\partial \sigma}{\partial \boldsymbol{z}}x$$

What happens when the neuron badly misclassify?

Problems with squared error loss

EASY SETTING: one neuron, binary classification, two weights w (including bias) and one training example x.

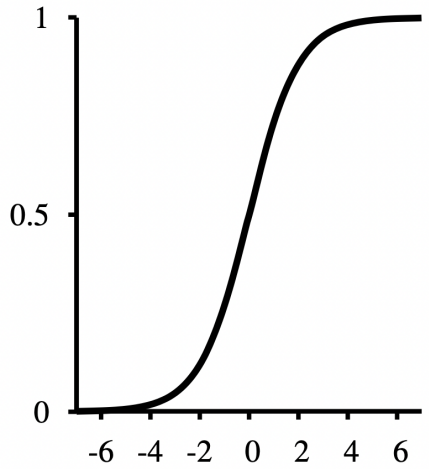
$$C(\boldsymbol{w}) = \frac{1}{2}(y(x) - \sigma(x, \boldsymbol{w}))^2$$

$$\frac{\partial C}{\partial w_0} = (\sigma(x, \boldsymbol{w}) - y(x))\frac{\partial \sigma}{\partial \boldsymbol{z}}x$$

What happens when the neuron badly misclassify?

- $|\sigma(x, \boldsymbol{w}) - y(x)| \approx 1$ which means that:
- either $\sigma(x, \boldsymbol{w}) = 0$ (and $y(x) = 1$) or $\sigma(x, \boldsymbol{w}) = 1$ (and $y(x) = 0$) ...

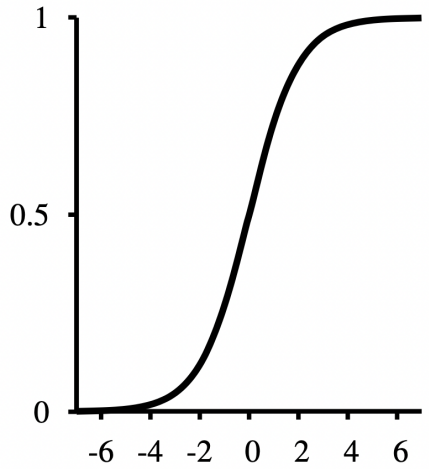
Problems with squared error loss: SLOW



When $\sigma(x, \boldsymbol{w}) = 0$ or $\sigma(x, \boldsymbol{w}) = 1$ then $\frac{\partial C}{\partial w_0}(\boldsymbol{w}) \approx 0$ therefore, recalling the update rule of GD:

$$w_0^{new} := w_0^{old} - \eta \frac{\partial C}{\partial w_0}(w^{old})$$

Problems with squared error loss: SLOW



When $\sigma(x, \boldsymbol{w}) = 0$ or $\sigma(x, \boldsymbol{w}) = 1$ then $\frac{\partial C}{\partial w_0}(\boldsymbol{w}) \approx 0$ therefore, recalling the update rule of GD:

$$w_0^{new} := w_0^{old} - \eta \frac{\partial C}{\partial w_0}(w^{old})$$

The learning is very slow at the beginning!

(The same intuition also applies to deep network.)

Problems with squared error loss: NON-CONVEX

$$C(\boldsymbol{w}) = \frac{1}{2n}\sum_{i=1}^{n} L_{sel}(\boldsymbol{x}, y, \sigma) = \frac{1}{2n}\sum_{i=1}^{n}(y(\boldsymbol{x}_i) - \frac{1}{1+e^{-(\boldsymbol{w}^T\boldsymbol{x}_i)}})^2$$

A function C is convex if it is twice differentiable and its second derivative $\frac{\partial^2 C}{\partial \boldsymbol{w}^2}$ is positive for all $\boldsymbol{w}$. A convex function has a global minimum.

C($\mathbf{w}$) is NON-CONVEX (in general): it might have local minima!

Introducing: cross-entropy loss function

Definition (Cross-entropy loss function)

Let $\boldsymbol{x}$ be an example, $y(\boldsymbol{x})$ a target function and $\sigma(\boldsymbol{x}, \boldsymbol{w})$ an hypothesis:

$$L_{ce}(\boldsymbol{x}, y, a) = \begin{cases} -log(\sigma(\boldsymbol{x}, \boldsymbol{w})) & \text{if } y(\boldsymbol{x}) = 1 \\ -log(1 - \sigma(\boldsymbol{x}, \boldsymbol{w})) & \text{if } y(\boldsymbol{x}) = 0 \end{cases}$$

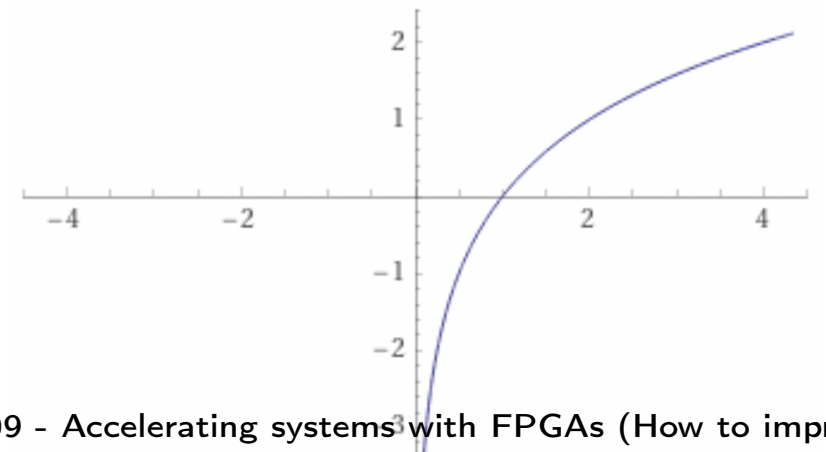
Introducing: cross-entropy loss function

Definition (Cross-entropy loss function)

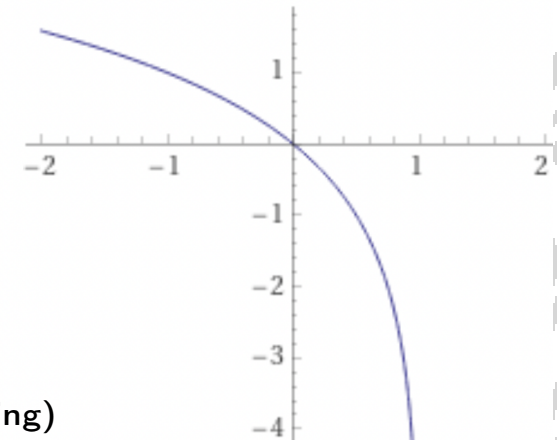
Let $\boldsymbol{x}$ be an example, $y(\boldsymbol{x})$ a target function and $\sigma(\boldsymbol{x}, \boldsymbol{w})$ an hypothesis:

$$L_{ce}(\boldsymbol{x}, y, a) = \begin{cases} -log(\sigma(\boldsymbol{x}, \boldsymbol{w})) & \text{if } y(\boldsymbol{x}) = 1 \\ -log(1 - \sigma(\boldsymbol{x}, \boldsymbol{w})) & \text{if } y(\boldsymbol{x}) = 0 \end{cases}$$

$log(z)$



$log(1 - z)$

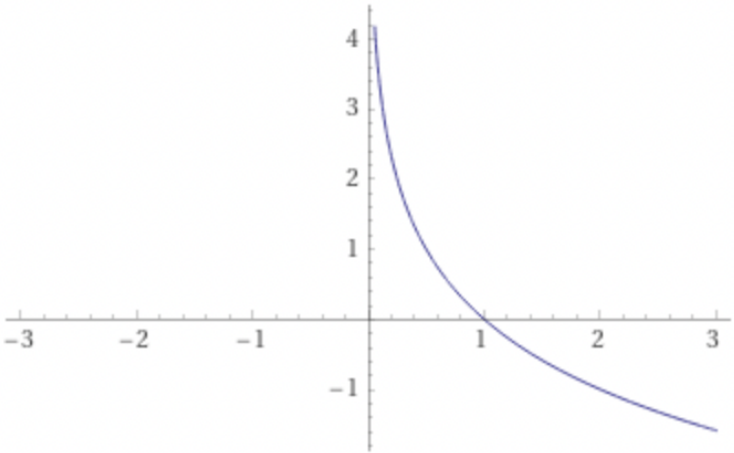


Cross-entropy loss function

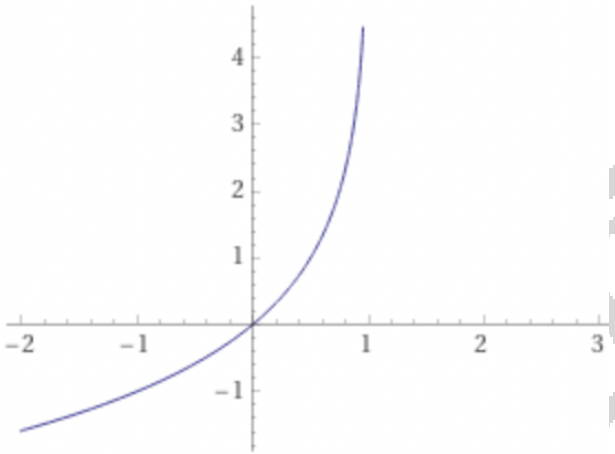
$$L_{ce}(\boldsymbol{x}, y, \sigma) = \begin{cases} -log(\sigma(\boldsymbol{x}, \boldsymbol{w})) & \text{if } y(\boldsymbol{x}) = 1 \\ -log(1 - \sigma(\boldsymbol{x}, \boldsymbol{w})) & \text{if } y(\boldsymbol{x}) = 0 \end{cases}$$

$$L_{ce}(\boldsymbol{x}, y, \sigma) = -y(\boldsymbol{x})\ log(\sigma(\boldsymbol{x}, \boldsymbol{w})) - (1 - y(\boldsymbol{x}))\ log(1 - \sigma(\boldsymbol{x}, \boldsymbol{w}))$$

$-log(z)$



$-log(1 - z)$



Good news

If we take L_{ce} as loss function, then the cost function is:

$$C(\boldsymbol{w}) = \frac{1}{n}\sum_{i=1}^{n} L_{ce}(\boldsymbol{x}_i, y, \sigma) =$$

$$-\frac{1}{n}\sum_{i=1}^{n}(y(\boldsymbol{x}_i)\ log(\sigma(\boldsymbol{x}_i, \boldsymbol{w})) + (1 - y(\boldsymbol{x}_i))\ log(1 - \sigma(\boldsymbol{x}_i, \boldsymbol{w})))$$

Good news #1

$$C(\boldsymbol{w}) = -\frac{1}{n}\sum_{i=1}^{n}(y(\boldsymbol{x}_i)\ log(\sigma(\boldsymbol{x}_i,\boldsymbol{w}))+(1-y(\boldsymbol{x}_i))\ log(1-\sigma(\boldsymbol{x}_i,\boldsymbol{w})))$$

In a NN with no hidden layers, it is convex and it always has a global minimum, regardless of the TS (with hidden layers, it is not always the case).

Good news #2

$$C(\boldsymbol{w}) = -\frac{1}{M}\sum_{i=1}^{M}(y(\boldsymbol{x}_i)\ log(\sigma(\boldsymbol{x}_i,\boldsymbol{w}))+(1-y(\boldsymbol{x}_i))\ log(1-\sigma(\boldsymbol{x}_i,\boldsymbol{w})))$$

Let's compute the partial derivative wrt w_j:

$$\frac{\partial C}{\partial w_j} = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{y(x)}{\sigma(\boldsymbol{x}_i,\boldsymbol{w})} - \frac{1-y(\boldsymbol{x}_i)}{1-\sigma(\boldsymbol{x}_i,\boldsymbol{w})}\right)\frac{\partial \sigma}{\partial w_j}$$

by doing some math we end up with:

$$\frac{\partial C}{\partial w_j} = \frac{1}{n}\sum_{i=1}^{n} x_j(\sigma(\boldsymbol{x}_i,\boldsymbol{w}) - y(\boldsymbol{x}_i))$$

Good news #2

$$C(\boldsymbol{w}) = -\frac{1}{M}\sum_{i=1}^{M}(y(\boldsymbol{x}_i)\ log(\sigma(\boldsymbol{x}_i,\boldsymbol{w}))+(1-y(\boldsymbol{x}_i))\ log(1-\sigma(\boldsymbol{x}_i,\boldsymbol{w})))$$

Let's compute the partial derivative wrt w_j:

$$\frac{\partial C}{\partial w_j} = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{y(x)}{\sigma(\boldsymbol{x}_i,\boldsymbol{w})} - \frac{1-y(\boldsymbol{x}_i)}{1-\sigma(\boldsymbol{x}_i,\boldsymbol{w})}\right)\frac{\partial \sigma}{\partial w_j}$$

by doing some math we end up with:

$$\frac{\partial C}{\partial w_j} = \frac{1}{n}\sum_{i=1}^{n} x_j(\sigma(\boldsymbol{x}_i,\boldsymbol{w}) - y(\boldsymbol{x}_i))$$

Now the update on weights depends on the error in the output $\sigma(\boldsymbol{x}_i,\boldsymbol{w}) - y(\boldsymbol{x}_i)$!

Overfitting

Problem

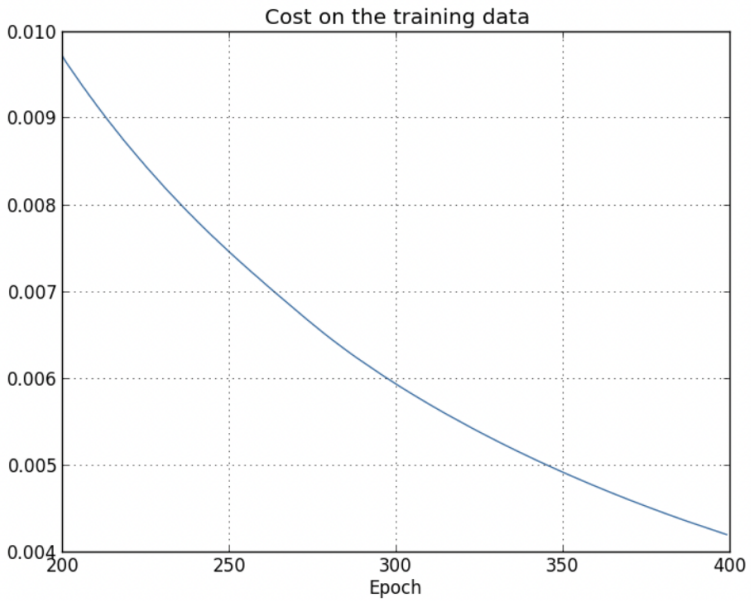
When do we stop the learning phase?

Overfitting

Problem

When do we stop the learning phase?

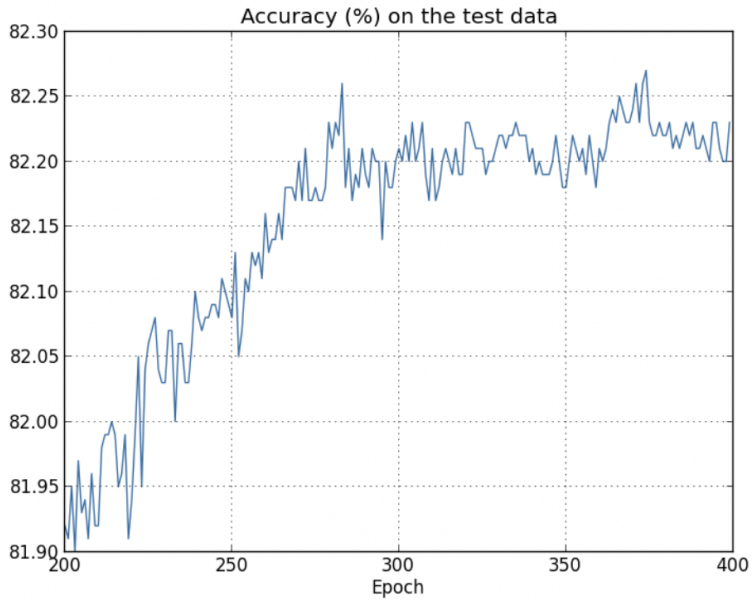
Recall: we have training set and test set. Clearly, the more we train, the less the cost C_{train} (on the training set) becomes:



30 hidden neurons, a mini-batch size of 10, 400 epochs, 1000 training images, $\eta = 0.5$.

Overfitting, cont'd

But if we look at the classification accuracy on the test set:



After around epoch 280, the model does not get better accuracy. It is overfitting the training data. That is: it is specialising in perfectly recognising the train examples, but it does not generalise on new examples.

The validation set

We can use another set, the validation set, and we use this to compute the accuracy (instead of using the training set) for choosing hyper-parameters. We stop when the accuracy does not improve (*early stopping* strategy).

Why?

The validation set

We can use another set, the validation set, and we use this to compute the accuracy (instead of using the training set) for choosing hyper-parameters. We stop when the accuracy does not improve (*early stopping* strategy).

Why?

If we used the test set, we would overfit the hyper-parameters to the test set.

Always better to have lots of data

When you have a lot of data, overfitting is never a problem!

Avoiding overfitting: regularization

Weight decay regularization:

$$C(\boldsymbol{w}) = -\frac{1}{n}\sum_{i=1}^{n}(y\ log(\sigma)) + (1-y)\ log(1-\sigma) + \frac{\lambda}{2n}\sum_{j=1}^{m} w_m^2$$

Idea:

bias towards small weights.

What happens with regularization term

$$C(\boldsymbol{w}) = C_0 + \frac{\lambda}{2n}\sum_{j=1}^{m} w_m^2$$

The partial derivatives wrt w (bias excluded) are:

$$\frac{\partial C}{\partial w} = \frac{\partial C_0}{\partial w} + \frac{\lambda}{n} w$$

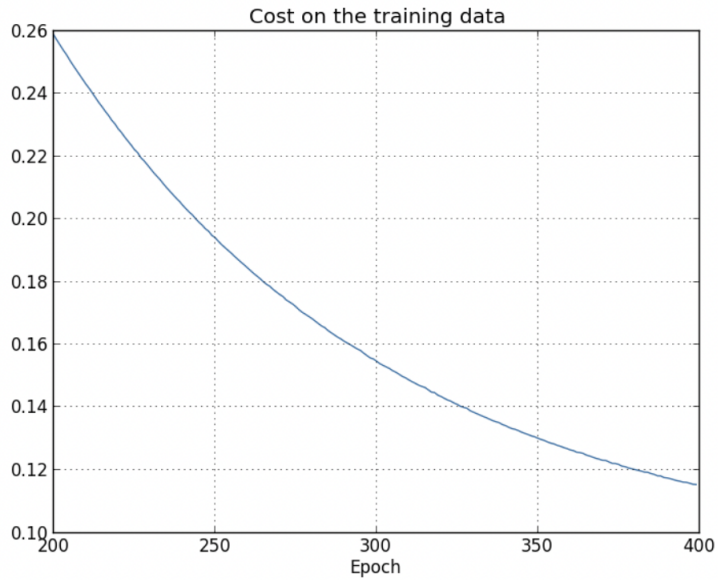
The new gradient descent updates the weights in this way:

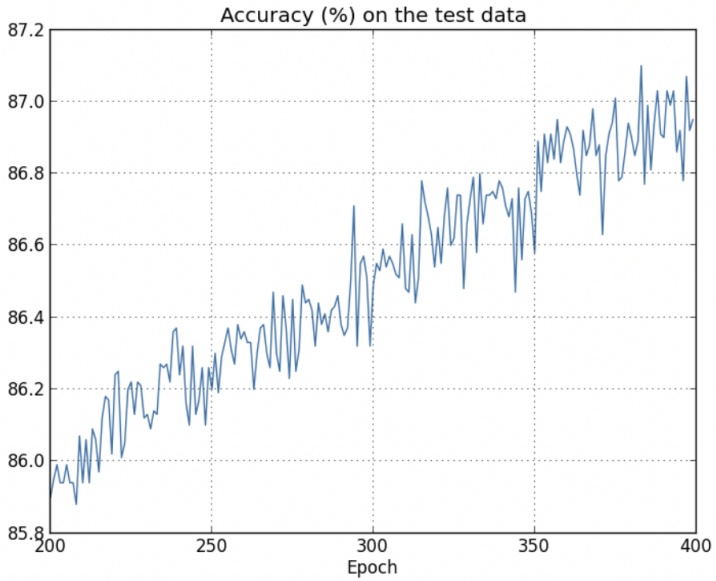
$$w_{new} = w_{old} - \eta \frac{\partial C_0}{\partial w}(\boldsymbol{w}_{old}) - \frac{\eta\lambda}{n} w_{old} = (1 - \frac{\eta\lambda}{n}) w_{old} - \eta \frac{\partial C_0}{\partial w}(\boldsymbol{w}_{old})$$

notice the weight decay, making the weight smaller.

Does it work? ($\lambda = 0.1$, TS size = 1000)

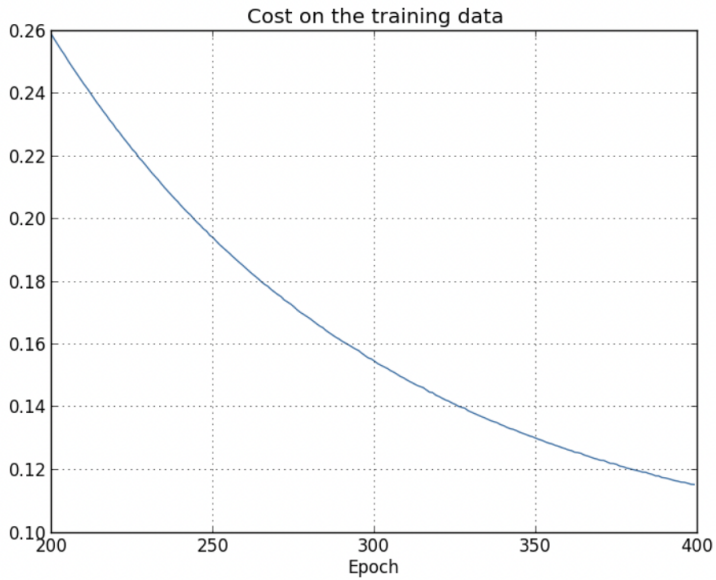
With regularization

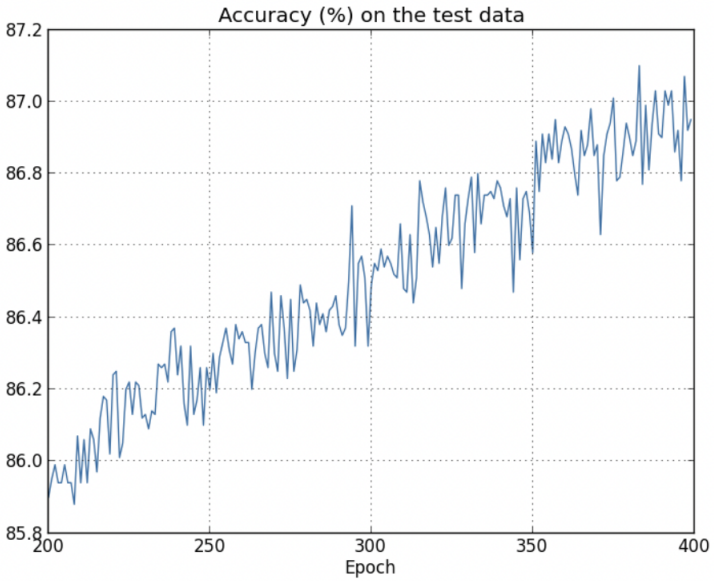




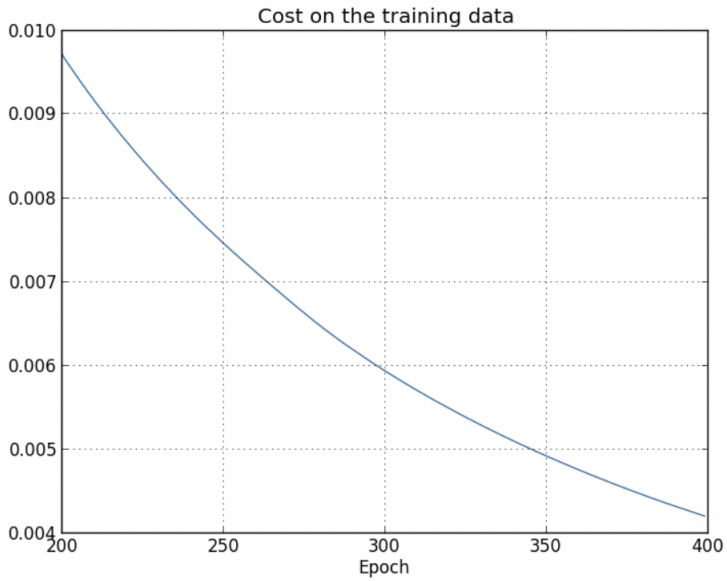
Does it work? ($\lambda = 0.1$, TS size = 1000)

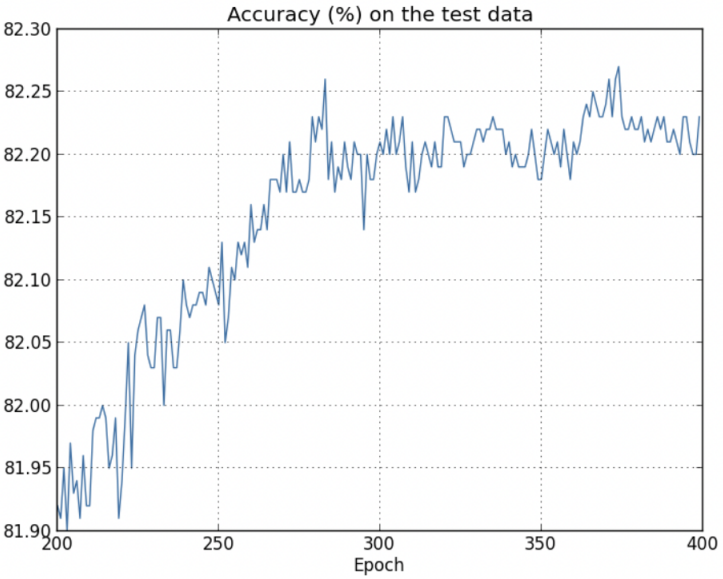
With regularization





Without regularization





Why does it work?

For empirical reasons (and kind of heuristic).

Smaller weights $\approx$ lower complexity. Occam's razor: prefer simpler hypothesis to explain a phenomenon.

Also: if weights are smaller, small changes on the inputs do not change much the outputs. And, they are resistant to noise in the training data.

Why are biases not regularized?

Again, mostly for empirical reasons, and convention.

However, having large biases:

- does not affect overfitting as much as having large weights and
- provides more flexibility, sometimes is desirable.

Importance of hyper-paramters

- 30 hidden neurons;
- mini-batch size of 10;
- 30 epochs;
- $\eta = 10$;
- $\lambda = 1000$.

Importance of hyper-paramters

- 30 hidden neurons;
- mini-batch size of 10;
- 30 epochs;
- $\eta = 10$;
- $\lambda = 1000$.

Accuracy on evaluation data: 10%!

Coping with random results

How to (more or less) scientifically address the problem of setting the hyper-paramters?
Strategies:

- try to get results fast: restrict the classification classes (and therefore training set);
- attack one hyper-parameter at a time, starting from η and monitor the training cost.
- move to λ using the accuracy on evaluation set;
- use early stopping for the number of epochs.