Deep learning

5.6. Architecture choice and training protocol

François Fleuret

https://fleuret.org/dlc/

Jan 1, 2021

idiap RESEARCH INSTITUTE
UNIVERSITÉ DE GENÈVE
EPFL

Choosing the network structure is a difficult exercise. **There is no silver bullet.**

- Re-use something "well known, that works", or at least start from there,

Choosing the network structure is a difficult exercise. **There is no silver bullet.**

- Re-use something "well known, that works", or at least start from there,
- split feature extraction / inference (although this is debatable),

Choosing the network structure is a difficult exercise. **There is no silver bullet.**

- Re-use something "well known, that works", or at least start from there,
- split feature extraction / inference (although this is debatable),
- modulate the capacity until it overfits a small subset, but does not overfit / underfit the full set,
- capacity increases with more layers, more channels, larger receptive fields, or more units,
- regularization to reduce the capacity or induce sparsity,

Choosing the network structure is a difficult exercise. **There is no silver bullet.**

- Re-use something "well known, that works", or at least start from there,
- split feature extraction / inference (although this is debatable),
- modulate the capacity until it overfits a small subset, but does not overfit / underfit the full set,
- capacity increases with more layers, more channels, larger receptive fields, or more units,
- regularization to reduce the capacity or induce sparsity,
- identify common paths for siamese-like,
- identify what path(s) or sub-parts need more/less capacity,

Choosing the network structure is a difficult exercise. **There is no silver bullet.**

- Re-use something "well known, that works", or at least start from there,
- split feature extraction / inference (although this is debatable),
- modulate the capacity until it overfits a small subset, but does not overfit / underfit the full set,
- capacity increases with more layers, more channels, larger receptive fields, or more units,
- regularization to reduce the capacity or induce sparsity,
- identify common paths for siamese-like,
- identify what path(s) or sub-parts need more/less capacity,
- use prior knowledge about the "scale of meaningful context" to size filters / combinations of filters (e.g. knowing the size of objects in a scene, the max duration of a sound snippet that matters),

Choosing the network structure is a difficult exercise. **There is no silver bullet.**

- Re-use something "well known, that works", or at least start from there,
- split feature extraction / inference (although this is debatable),
- modulate the capacity until it overfits a small subset, but does not overfit / underfit the full set,
- capacity increases with more layers, more channels, larger receptive fields, or more units,
- regularization to reduce the capacity or induce sparsity,
- identify common paths for siamese-like,
- identify what path(s) or sub-parts need more/less capacity,
- use prior knowledge about the "scale of meaningful context" to size filters / combinations of filters (e.g. knowing the size of objects in a scene, the max duration of a sound snippet that matters),
- grid-search all the variations that come to mind (and hopefully have farms of GPUs to do so).

Choosing the network structure is a difficult exercise. **There is no silver bullet.**

- Re-use something "well known, that works", or at least start from there,
- split feature extraction / inference (although this is debatable),
- modulate the capacity until it overfits a small subset, but does not overfit / underfit the full set,
- capacity increases with more layers, more channels, larger receptive fields, or more units,
- regularization to reduce the capacity or induce sparsity,
- identify common paths for siamese-like,
- identify what path(s) or sub-parts need more/less capacity,
- use prior knowledge about the "scale of meaningful context" to size filters / combinations of filters (e.g. knowing the size of objects in a scene, the max duration of a sound snippet that matters),
- grid-search all the variations that come to mind (and hopefully have farms of GPUs to do so).

We will re-visit this list with additional regularization / normalization methods.

Regarding the learning rate, for training to succeed it has to

- reduce the loss quickly $\Rightarrow$ large learning rate,
- not be trapped in a bad minimum $\Rightarrow$ large learning rate,

Regarding the learning rate, for training to succeed it has to

- reduce the loss quickly $\Rightarrow$ large learning rate,
- not be trapped in a bad minimum $\Rightarrow$ large learning rate,
- not bounce around in narrow valleys $\Rightarrow$ small learning rate, and
- not oscillate around a minimum $\Rightarrow$ small learning rate.

Regarding the learning rate, for training to succeed it has to

- reduce the loss quickly $\Rightarrow$ large learning rate,
- not be trapped in a bad minimum $\Rightarrow$ large learning rate,
- not bounce around in narrow valleys $\Rightarrow$ small learning rate, and
- not oscillate around a minimum $\Rightarrow$ small learning rate.

These constraints lead to a general policy of using **a larger step size first, and a smaller one in the end.**

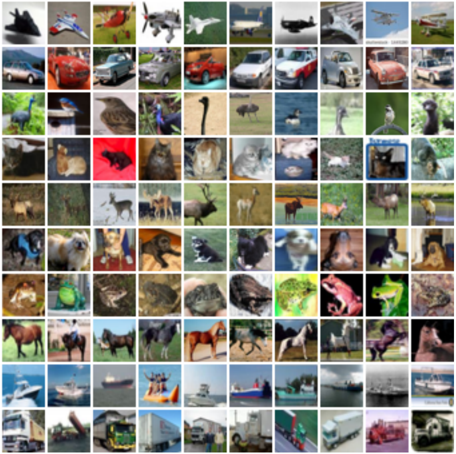
Regarding the learning rate, for training to succeed it has to

- reduce the loss quickly $\Rightarrow$ large learning rate,
- not be trapped in a bad minimum $\Rightarrow$ large learning rate,
- not bounce around in narrow valleys $\Rightarrow$ small learning rate, and
- not oscillate around a minimum $\Rightarrow$ small learning rate.

These constraints lead to a general policy of using **a larger step size first, and a smaller one in the end.**

The practical strategy is to look at the losses and error rates across epochs and pick a learning rate and learning rate adaptation. For instance by reducing it at discrete pre-defined steps, or with a geometric decay.

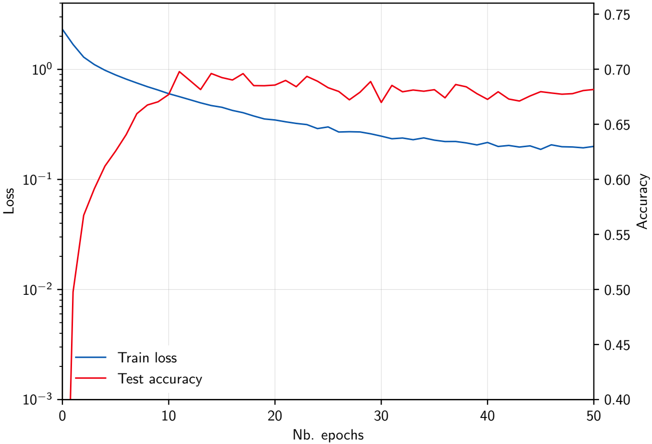
CIFAR10 data-set



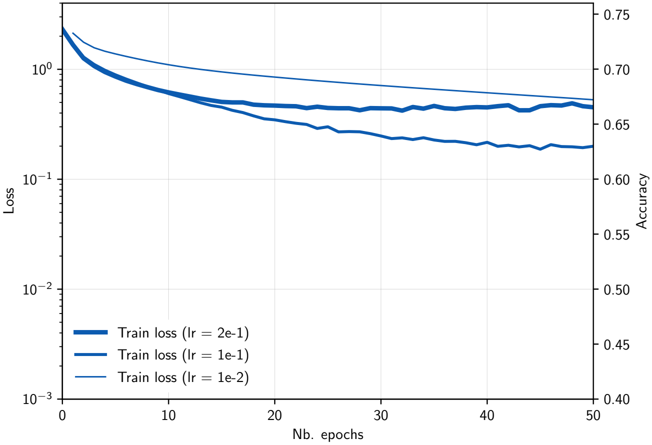
32×32 color images, $50,000$ train samples, $10,000$ test samples.

(Krizhevsky, 2009, chap. 3)

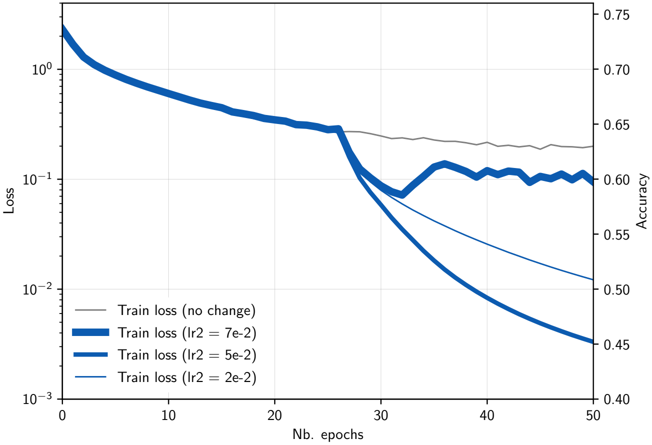
Small convnet on CIFAR10, cross-entropy, batch size 100, $\eta = 1e - 1$.



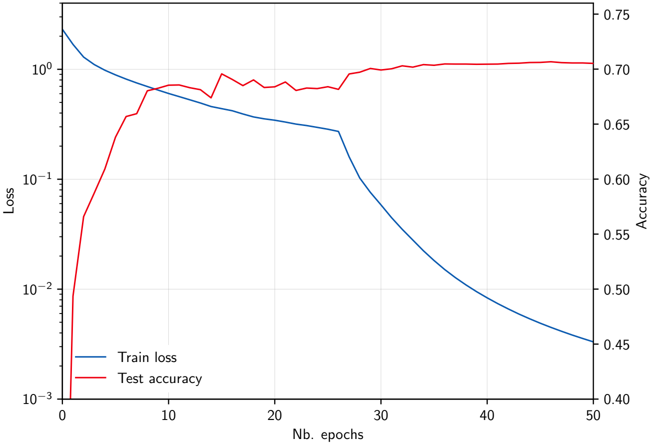
Small convnet on CIFAR10, cross-entropy, batch size 100



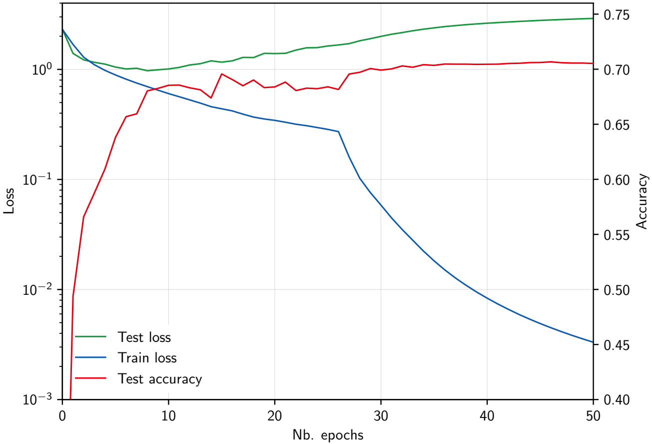
Using $\eta = 1e-1$ for 25 epochs, then reducing it.



Using $\eta = 1e-1$ for 25 epochs, then $\eta = 5e-2$.



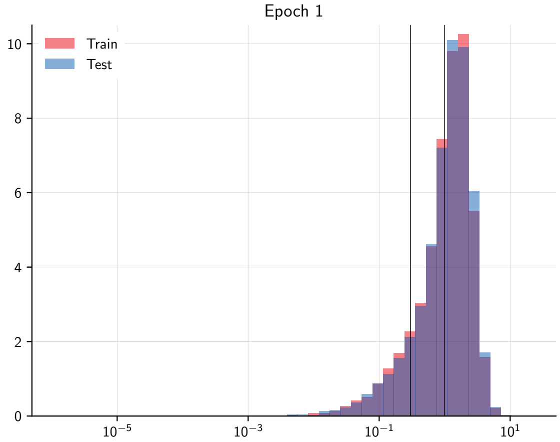
While the test error still goes down, the test loss may increase, as it gets even worse on misclassified examples, and decreases less on the ones getting fixed.



We can plot the train and test distributions of the per-sample loss

$$\ell = -\log\left(\frac{\exp(f_Y(X;w))}{\sum_k \exp(f_k(X;w))}\right)$$

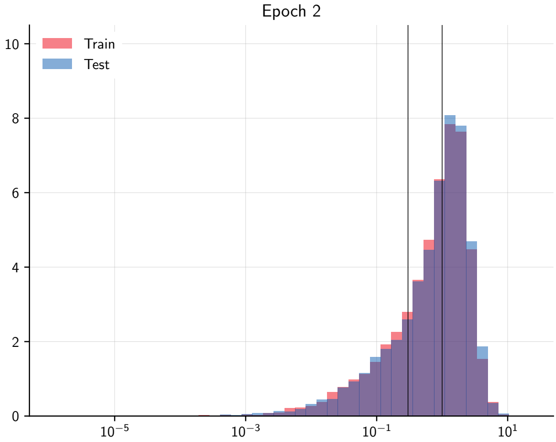
through epochs to visualize the over-fitting.



We can plot the train and test distributions of the per-sample loss

$$\ell = -\log\left(\frac{\exp(f_Y(X;w))}{\sum_k \exp(f_k(X;w))}\right)$$

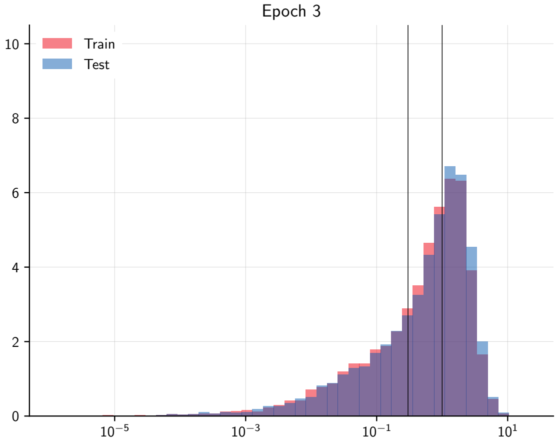
through epochs to visualize the over-fitting.



We can plot the train and test distributions of the per-sample loss

$$\ell = -\log\left(\frac{\exp(f_Y(X;w))}{\sum_k \exp(f_k(X;w))}\right)$$

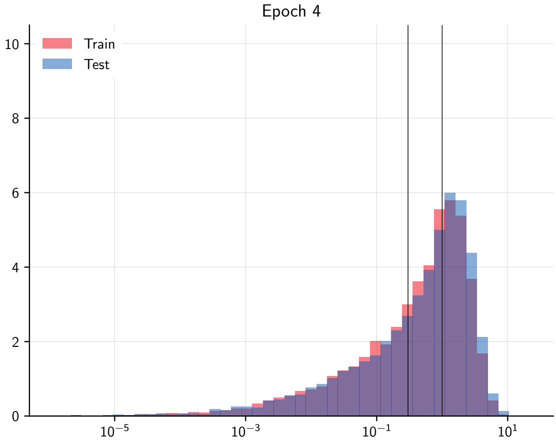
through epochs to visualize the over-fitting.



We can plot the train and test distributions of the per-sample loss

$$\ell = -\log\left(\frac{\exp(f_Y(X;w))}{\sum_k \exp(f_k(X;w))}\right)$$

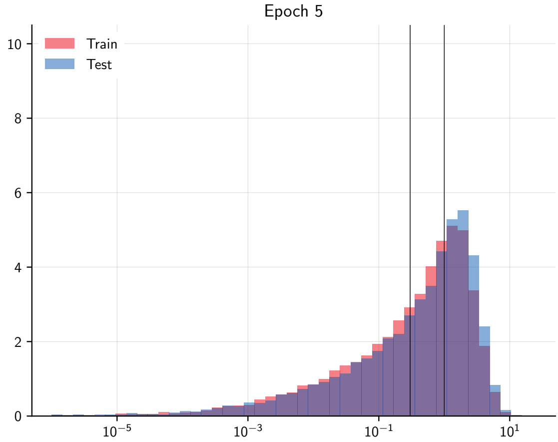
through epochs to visualize the over-fitting.



We can plot the train and test distributions of the per-sample loss

$$\ell = -\log\left(\frac{\exp(f_Y(X;w))}{\sum_k \exp(f_k(X;w))}\right)$$

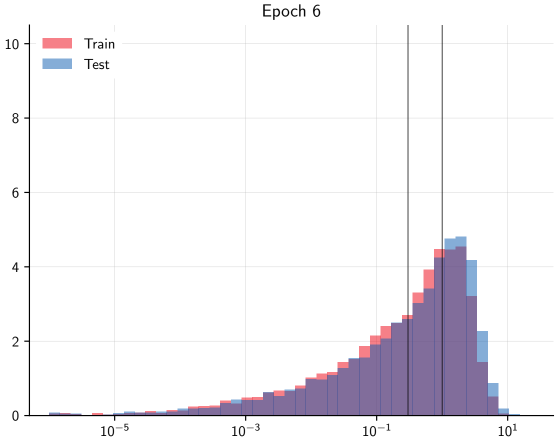
through epochs to visualize the over-fitting.



We can plot the train and test distributions of the per-sample loss

$$\ell = -\log\left(\frac{\exp(f_Y(X;w))}{\sum_k \exp(f_k(X;w))}\right)$$

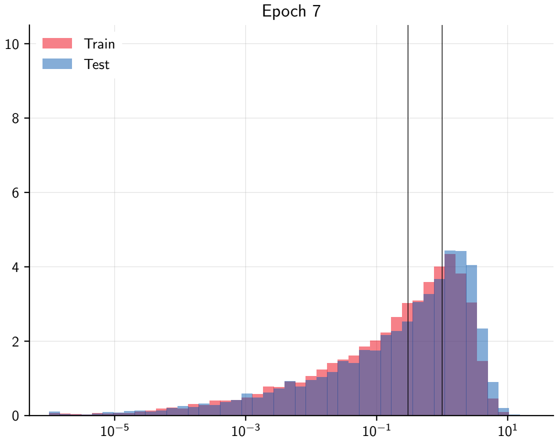
through epochs to visualize the over-fitting.



We can plot the train and test distributions of the per-sample loss

$$\ell = -\log\left(\frac{\exp(f_Y(X;w))}{\sum_k \exp(f_k(X;w))}\right)$$

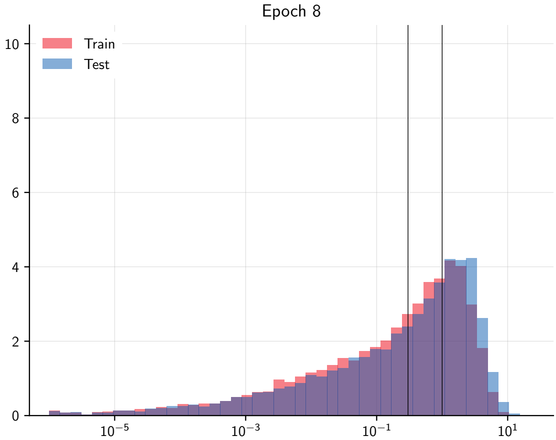
through epochs to visualize the over-fitting.



We can plot the train and test distributions of the per-sample loss

$$\ell = -\log\left(\frac{\exp(f_Y(X;w))}{\sum_k \exp(f_k(X;w))}\right)$$

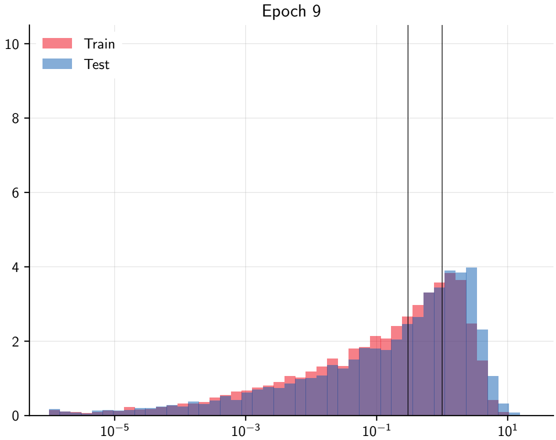
through epochs to visualize the over-fitting.



We can plot the train and test distributions of the per-sample loss

$$\ell = -\log \left(\frac{\exp(f_Y(X; w))}{\sum_k \exp(f_k(X; w))} \right)$$

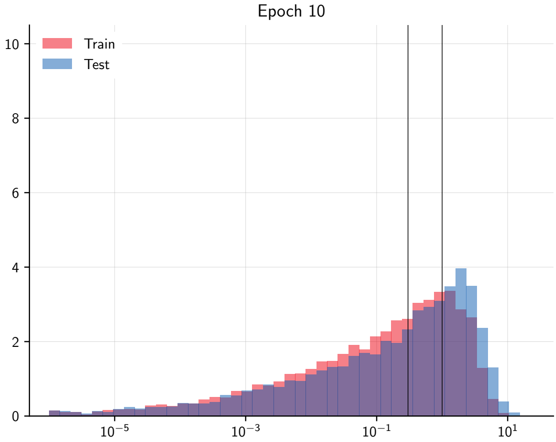
through epochs to visualize the over-fitting.



We can plot the train and test distributions of the per-sample loss

$$\ell = -\log \left(\frac{\exp(f_Y(X;w))}{\sum_k \exp(f_k(X;w))} \right)$$

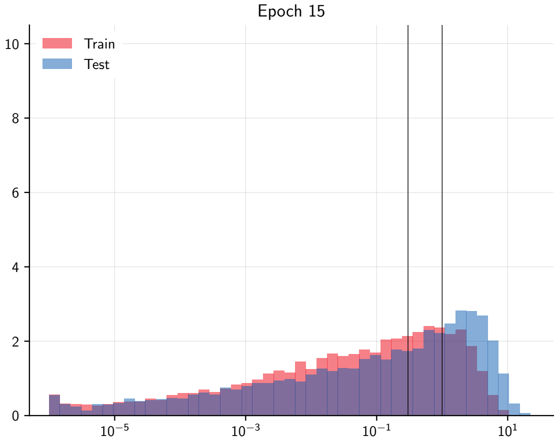
through epochs to visualize the over-fitting.



We can plot the train and test distributions of the per-sample loss

$$\ell = -\log\left(\frac{\exp(f_Y(X;w))}{\sum_k \exp(f_k(X;w))}\right)$$

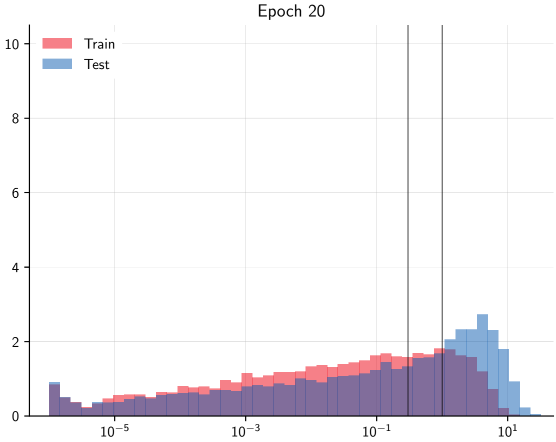
through epochs to visualize the over-fitting.



We can plot the train and test distributions of the per-sample loss

$$\ell = -\log\left(\frac{\exp(f_Y(X;w))}{\sum_k \exp(f_k(X;w))}\right)$$

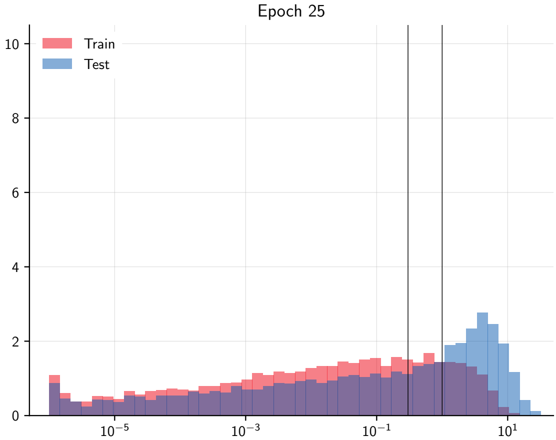
through epochs to visualize the over-fitting.



We can plot the train and test distributions of the per-sample loss

$$\ell = -\log\left(\frac{\exp(f_Y(X;w))}{\sum_k \exp(f_k(X;w))}\right)$$

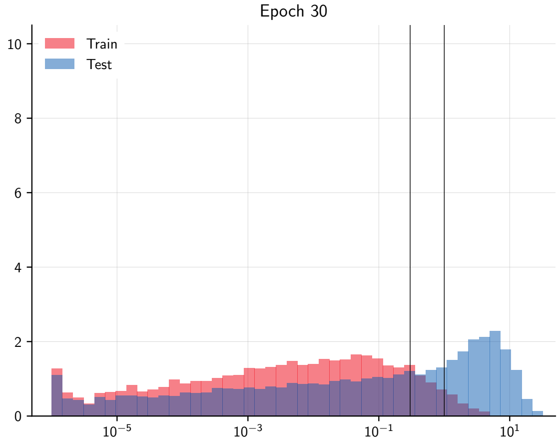
through epochs to visualize the over-fitting.



We can plot the train and test distributions of the per-sample loss

$$\ell = -\log\left(\frac{\exp(f_Y(X;w))}{\sum_k \exp(f_k(X;w))}\right)$$

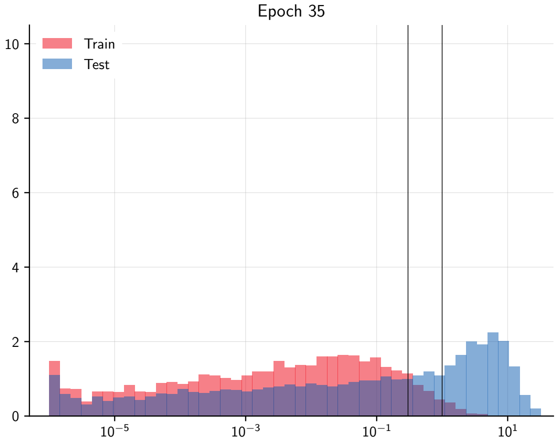
through epochs to visualize the over-fitting.



We can plot the train and test distributions of the per-sample loss

$$\ell = -\log\left(\frac{\exp(f_Y(X;w))}{\sum_k \exp(f_k(X;w))}\right)$$

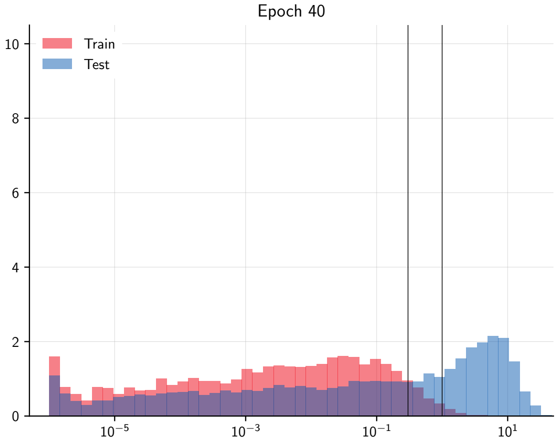
through epochs to visualize the over-fitting.



We can plot the train and test distributions of the per-sample loss

$$\ell = -\log\left(\frac{\exp(f_Y(X; w))}{\sum_k \exp(f_k(X; w))}\right)$$

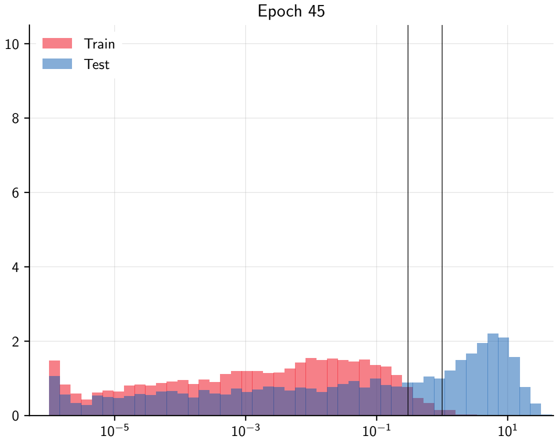
through epochs to visualize the over-fitting.



We can plot the train and test distributions of the per-sample loss

$$\ell = -\log\left(\frac{\exp(f_Y(X;w))}{\sum_k \exp(f_k(X;w))}\right)$$

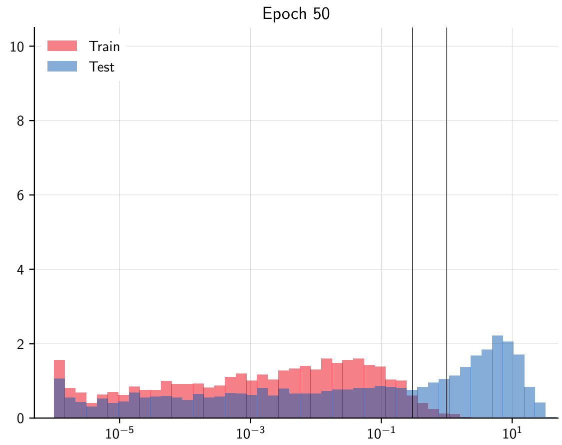
through epochs to visualize the over-fitting.



We can plot the train and test distributions of the per-sample loss

$$\ell = -\log\left(\frac{\exp(f_Y(X; w))}{\sum_k \exp(f_k(X; w))}\right)$$

through epochs to visualize the over-fitting.



The end

References

A. Krizhevsky. **Learning multiple layers of features from tiny images**. Master's thesis, Department of Computer Science, University of Toronto, 2009.