1. Normalization in Deep Learning

Normalization is carried out in the input layer to normalize the raw data. However, for deep neural nets, the activation values of the middle layer nodes can get very large, causing the same problem as our example. This necessitates some type of normalization in the hidden layers as well [1], as shown in Figure 1. Let x be that tensor consisting of a batch of N images. Each of these images has C feature maps or channels with height H and weight W. Therefore, $x \in^{N \times C \times H \times W}$ is a four-dimensional tensor.

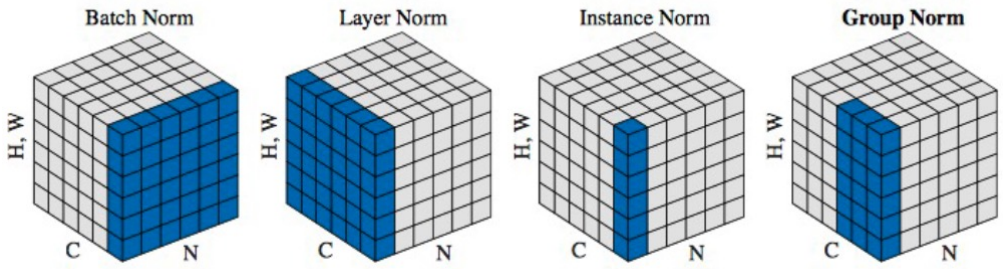

Figure 1. An overview of the normalization methods.

1.1. Batch Normalization.

BN operates on one channel (one feature map) over all the training samples in the mini-batch. BN normalizes its activation values using the mean (μ_c) and variance (σ_c^2) of the mini-batch and introduces two learnable parameters, γ and β. The network can learn these parameters using a Gradient Descent algorithm.

$$\mu_c = \frac{1}{NHW} \sum_{n=1}^{N} \sum_{w=1}^{W} \sum_{h=1}^{H} x_{ncwh} \tag{1}$$

$$\sigma_c^2 = \frac{1}{NHW} \sum_{n=1}^{N} \sum_{w=1}^{W} \sum_{h=1}^{H} (x_{ncwh} - \mu_c)^2 \tag{2}$$

We add the ϵ in computing $\hat{x}_{ncwh}$ to the denominator to stabilize the training in case the variance of the mini-batch is zero.

$$\hat{x}_{ncwh} = \frac{x_{ncwh} - \mu_c}{\sqrt{\sigma_c^2 + \epsilon}} \tag{3}$$

Finally,

$$y_{ncwh} = \gamma \hat{x}_{ncwh} + \beta, \tag{4}$$

where γ and β parameters are the scaling and bias operators. This transformation preserves the network's capacity by ensuring that the normalization layer can produce the identity function. Both normalization methods can accelerate training and make the network converge faster. Plus, BN enables larger training rates and improves the generalization of the neural network. BN depends on the batch size in that, to obtain a statistically more accurate mean and variance, the batch size needs to be large. In addition, since BN is dependent on the batch size, it can't be applied at test time the same way as the training time. The reason is that in the test phase, we usually have one example to process. Therefore, the mean and variance can't be computed like training. Instead, during test time, BN utilizes moving average and variance to perform inference.

1.2. **Instance Normalization.** Instance normalization was first coined in the StyleNet paper [2]. Instance normalization tells us that it operates on a single sample. To clarify how IN works, let's consider sample feature maps that constitute an input tensor to the IN layer. In instance normalization, we consider one training sample and feature map and take the mean and variance over its spatial locations (W and H).
To perform instance normalization for a single instance x_{nc}, we need to compute the mean and variance:

$$\mu_{nc} = \frac{1}{HW} \sum_{w=1}^{W} \sum_{h=1}^{H} x_{ncwh} \tag{5}$$

$$\sigma_{nc}^2 = \frac{1}{HW} \sum_{w=1}^{W} \sum_{h=1}^{H} (x_{ncwh} - \mu_{nc})^2. \tag{6}$$

Then, using the obtained mean and variance, each spatial dimension is normalized:

$$\hat{x}_{ncwh} = \frac{x_{ncwh} - \mu_{nc}}{\sqrt{\sigma_{nc}^2 + \epsilon}}, \tag{7}$$

where ϵ is a small value added for more stable training. In the end, using two scaling and bias parameters, we linearly transform the result.
While BN transforms the whole mini-batch of samples, IN does that to a single training sample. A small batch size setting is where IN performs better than BN. IN doesn't depend on the batch size, so its implementation is kept the same for both training and testing.

1.3. **Layer Normalization.** In BN, the statistics are computed across the batch and the spatial dims. In contrast, in Layer Normalization (LN), the statistics (mean and variance) are computed across all channels and spatial dims. Thus, the statistics are independent of the batch. This layer was initially introduced to handle vectors (mostly the RNN outputs).
Nearly nobody spoke about LN until the Transformers paper came out. We can take the mean across the spatial dimension and across all channels, as illustrated below:

$$\mu_n = \frac{1}{CHW} \sum_{c=1}^{C} \sum_{w=1}^{W} \sum_{h=1}^{H} x_{ncwh} \tag{8}$$

$$\sigma_n^2 = \frac{1}{CHW} \sum_{c=1}^{C} \sum_{w=1}^{W} \sum_{h=1}^{H} (x_{ncwh} - \mu_n)^2. \tag{9}$$

Then, using the obtained mean and variance, each spatial dimension is normalized:

$$\hat{x}_{ncwh} = \frac{x_{ncwh} - \mu_n}{\sqrt{\sigma_n^2 + \epsilon}}, \tag{10}$$

where ϵ is a small value added for more stable training. In the end, using two scaling and bias parameters, we linearly transform the result.

1.4. **Group Normalization.** Group normalization (GN) divides the channels into groups and computes the first-order statistics within each group. As a result, GN's computation is independent of batch sizes, and its accuracy is more stable than BN in a wide range of batch sizes.
For $groups = number$ of channels we get instance normalization, while for $groups = 1$ the method is reduced to layer normalization.

$$\mu_i = \frac{1}{m} \sum_{k \in \mathcal{S}_i} x_k, \tag{11}$$

$$\sigma_i^2 = \frac{1}{m}\sum_{k\in\mathcal{S}_i}(x_k - \mu_{i]})^2, \tag{12}$$

where

$$\mathcal{S}_i = \{k|k_N = i_N, \lfloor\frac{k_C}{C/G}\rfloor = \lfloor\frac{i_C}{C/G}\rfloor\}. \tag{13}$$

Note that G is the number of groups, which is a hyper-parameter. C/G is the number of channels per group. Thus, GN computes μ and σ along the (H, W) axes and along a group of C/G channels.

1.5. **Adaptive Instance Normalization.** Normalization and style transfer are closely related. In Instance Normalization, what if γ and β are injected from the feature statistics of another image y? In this way, we will be able to model any arbitrary style by just giving our desired feature image mean as β and variance as γ from style image y.
Adaptive Instance Normalization (AdaIN) receives an input image x (content) and a style input y, and simply aligns the channel-wise mean and variance of x to match those of y. Mathematically:

$$AdaIN(x, y) = \sigma(y)\frac{x - \mu(x)}{\sigma(x)} + \mu(y) \tag{14}$$

References

[1] Nikolas Adaloglou. In-layer normalization techniques for training very deep neural networks. *https://theaisummer.com/*, 2020.

[2] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Improved texture networks: Maximizing quality and diversity in feed-forward stylization and texture synthesis. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6924–6932, 2017.

7.6.1 Injecting Noise at the Input

Some classical regularization techniques can be derived in terms of training on noisy inputs. [5]. Let us consider a regression setting, where we are interested in learning a model $\hat{y}(\boldsymbol{x})$ that maps a set of features $\boldsymbol{x}$ to a scalar. The cost function we will use is the least-squares error between the model prediction $\hat{y}(\boldsymbol{x})$ and the true value y:

$$J = \mathbb{E}_{p(x,y)} \left[(\hat{y}(\boldsymbol{x}) - \boldsymbol{y})^2 \right] , \tag{7.15}$$

where we are given a dataset of m input / output pairs $\{(\boldsymbol{x}^{(1)}, y^{(1)}), \ldots, (\boldsymbol{x}^{(m)}, y^{(m)})\}$. Our training objective is to minimize the loss function, which in this case is given by the least-squares error between the model prediction $\hat{\boldsymbol{y}}(\boldsymbol{x})$ and the true label $\boldsymbol{y}$ (where $\boldsymbol{y}(\boldsymbol{x}) = [y^{(1)}(\boldsymbol{x}), \ldots, y^{(m)}(\boldsymbol{x})]$).

Now consider that with each input presentation to the model we also include a random perturbation $\boldsymbol{\epsilon} \sim (\boldsymbol{0}, \nu \boldsymbol{I})$, so that the error function becomes

$$\begin{aligned}
\tilde{J}_{\boldsymbol{x}} &= \mathbb{E}_{p(\boldsymbol{x},y,\boldsymbol{\epsilon})} \left[(\hat{y}(\boldsymbol{x} + \boldsymbol{\epsilon}) - y)^2 \right] \\
&= \mathbb{E}_{p(\boldsymbol{x},y,\boldsymbol{\epsilon})} \left[\hat{y}^2(\boldsymbol{x} + \boldsymbol{\epsilon}) - 2y\hat{y}(\boldsymbol{x} + \boldsymbol{\epsilon}) + y^2 \right] \\
&= \mathbb{E}_{p(\boldsymbol{x},y,\boldsymbol{\epsilon})} \left[\hat{y}^2(\boldsymbol{x} + \boldsymbol{\epsilon}) \right] - 2\mathbb{E}_{p(\boldsymbol{x},y,\boldsymbol{\epsilon})} \left[y\hat{y}(\boldsymbol{x} + \boldsymbol{\epsilon}) \right] + \mathbb{E}_{p(\boldsymbol{x},y,\boldsymbol{\epsilon})} \left[y^2 \right]
\end{aligned} \tag{7.16}$$

Assuming small noise, we can consider the taylor series expansion of $\hat{y}(\boldsymbol{x} + \boldsymbol{\epsilon})$ around $\hat{y}(\boldsymbol{x})$.

$$\hat{y}(\boldsymbol{x} + \boldsymbol{\epsilon}) = \hat{y}(\boldsymbol{x}) + \boldsymbol{\epsilon}^\top \nabla_{\boldsymbol{x}} \hat{y}(\boldsymbol{x}) + \frac{1}{2} \boldsymbol{\epsilon}^\top \nabla_{\boldsymbol{x}}^2 \hat{y}(\boldsymbol{x}) \boldsymbol{\epsilon} + O(\boldsymbol{\epsilon}^3) \tag{7.17}$$

Substituting this approximation for $\hat{y}(\boldsymbol{x}+\boldsymbol{\epsilon})$ into the objective function (Eq. 7.16) and using the fact that $\mathbb{E}_{p(\boldsymbol{\epsilon})}[\boldsymbol{\epsilon}] = 0$ and that $\mathbb{E}_{p(\boldsymbol{\epsilon})}[\boldsymbol{\epsilon}\boldsymbol{\epsilon}^\top] = \nu \boldsymbol{I}$ to simplify [6], we get:

$$\begin{aligned}
\tilde{J}_{\boldsymbol{x}} \approx\ & \mathbb{E}_{p(\boldsymbol{x},y,\boldsymbol{\epsilon})} \left[\left(\hat{y}(\boldsymbol{x}) + \boldsymbol{\epsilon}^\top \nabla_{\boldsymbol{x}} \hat{y}(\boldsymbol{x}) + \frac{1}{2} \boldsymbol{\epsilon}^\top \nabla_{\boldsymbol{x}}^2 \hat{y}(\boldsymbol{x}) \boldsymbol{\epsilon} \right)^2 \right] \\
& - 2\mathbb{E}_{p(\boldsymbol{x},y,\boldsymbol{\epsilon})} \left[y\hat{y}(\boldsymbol{x}) + y\boldsymbol{\epsilon}^\top \nabla_{\boldsymbol{x}} y\hat{y}(\boldsymbol{x}) + \frac{1}{2} y\boldsymbol{\epsilon}^\top \nabla_{\boldsymbol{x}}^2 \hat{y}(\boldsymbol{x}) \boldsymbol{\epsilon} \right] + \mathbb{E}_{p(\boldsymbol{x},y,\boldsymbol{\epsilon})} \left[y^2 \right] \\
=\ & \mathbb{E}_{p(\boldsymbol{x},y,\boldsymbol{\epsilon})} \left[(\hat{y}(\boldsymbol{x}) - y)^2 \right] + \mathbb{E}_{p(\boldsymbol{x},y,\boldsymbol{\epsilon})} \left[\hat{y}(\boldsymbol{x}) \boldsymbol{\epsilon}^\top \nabla_{\boldsymbol{x}}^2 \hat{y}(\boldsymbol{x}) \boldsymbol{\epsilon} + \left(\boldsymbol{\epsilon}^\top \nabla_{\boldsymbol{x}} \hat{y}(\boldsymbol{x}) \right)^2 + O(\boldsymbol{\epsilon}^3) \right] \\
& - 2\mathbb{E}_{p(\boldsymbol{x},y,\boldsymbol{\epsilon})} \left[\frac{1}{2} y\boldsymbol{\epsilon}^\top \nabla_{\boldsymbol{x}}^2 \hat{y}(\boldsymbol{x}) \boldsymbol{\epsilon} \right] \\
=\ & J + \nu \mathbb{E}_{p(\boldsymbol{x},y)} \left[(\hat{y}(\boldsymbol{x}) - y) \nabla_{\boldsymbol{x}}^2 \hat{y}(\boldsymbol{x}) \right] + \nu \mathbb{E}_{p(\boldsymbol{x},y)} \left[\| \nabla_{\boldsymbol{x}} \hat{y}(\boldsymbol{x}) \|^2 \right]
\end{aligned} \tag{7.18}$$

[5] The analysis in this section is mainly based on that in Bishop (1995a,b)

[6] In this derivation we have used two properties of the trace operator: (1) that a scalar is equal to its trace; (2) that, for a square matrix AB, $\mathrm{Tr}(AB) = \mathrm{Tr}(BA)$. These are discussed in Sec. 2.10.

If we consider minimizing this objective function, by taking the functional gradient of $\hat{y}(\boldsymbol{x})$ and setting the result to zero, we can see that

$$\hat{y}(\boldsymbol{x}) = \mathbb{E}_{p(y|\boldsymbol{x})}[y] + O(\nu).$$

This implies that the expectation in the second last term in Eq. 7.18, $\mathbb{E}_{p(\boldsymbol{x},y)}\left[(\hat{y}(\boldsymbol{x}) - y)\nabla^2_{\boldsymbol{x}}\hat{y}(\boldsymbol{x})\right.$ reduces to $O(\nu)$ because the expectation of the difference $(\hat{y}(\boldsymbol{x}) - y)$ is reduces to $O(\nu)$.

This leaves us with the objective function of the form

$$\tilde{J}_{\boldsymbol{x}} = \mathbb{E}_{p(\boldsymbol{x},y)}\left[(\hat{y}(\boldsymbol{x}) - y)^2\right] + \nu\mathbb{E}_{p(\boldsymbol{x},y)}\left[\|\nabla_{\boldsymbol{x}}\hat{y}(\boldsymbol{x})\|^2\right] + O(\nu^2).$$

For small ν, the minimization of J with added noise on the input (with covariance $\nu\boldsymbol{I}$) is equivalent to minimization of J with an additional *regularization* term given by $\nu\mathbb{E}_{p(\boldsymbol{x},y)}\left[\|\nabla_{\boldsymbol{x}}\hat{y}(\boldsymbol{x})\|^2\right]$.

Considering the behavior of this regularization term, we note that it has the effect of penalizing large gradients of the function $\hat{y}(\boldsymbol{x})$. That is, it has the effect of reducing the *sensitivity* of the output of the network with respect to small variations in its input $\boldsymbol{x}$. We can interpret this as attempting to build in some local robustness into the model and thereby promote generalization. We note also that for linear networks, this regularization term reduces to simple weight decay (as discussed in Sec. 7.2.1).

7.6.2 Injecting Noise at the Weights

Rather than injecting noise as part of the input, one could also consider adding noise directly to the model parameters. As we shall see, this can also be interpreted as equivalent (under some assumptions) to a more traditional form of regularization. Adding noise to the weights has been shown to be a effective regularization strategy in the context of recurrent neural networks[7] Jim *et al.* (1996); Graves (2011b). In the following, we will present an analysis of the effect of weight noise on a standard feedforward neural network (as introduced in Chapter 6).

As we did in the last section, we again consider the regression setting, where we wish to train a function $\hat{y}(\boldsymbol{x})$ that maps a set of features $\boldsymbol{x}$ to a scalar using the least-squares cost function between the model predictions $\hat{y}(\boldsymbol{x})$ and the true values y:

$$J = \mathbb{E}_{p(x,y)}\left[(\hat{y}(\boldsymbol{x}) - \boldsymbol{y})^2\right]. \tag{7.19}$$

We again assume we are given a dataset of m input / output pairs $\{(\boldsymbol{x}^{(1)}, y^{(1)}), \ldots, (\boldsymbol{x}^{(m)}, y^{(m)})\}$

We now assume that with each input presentation we also include a random perturbation $\boldsymbol{\epsilon_W} \sim (\boldsymbol{0}, \eta\boldsymbol{I})$ of the network weights. Let us imagine that we have

[7] Recurrent neural networks will be discussed in detail in Chapter 10

a standard L-layer MLP, we denote the perturbed model as $\hat{y}_{\boldsymbol{\epsilon W}}(\boldsymbol{x})$. Despite the injection of noise, we are still interested in minimizing the squared error of the output of the network. The objective function thus becomes:

$$\begin{aligned}\tilde{J}_{\boldsymbol{W}} &= \mathbb{E}_{p(\boldsymbol{x},y,\boldsymbol{\epsilon W})}\left[(\hat{y}_{\boldsymbol{\epsilon W}} - y)^2\right] \\ &= \mathbb{E}_{p(\boldsymbol{x},y,\boldsymbol{\epsilon W})}\left[\hat{y}^2_{\boldsymbol{\epsilon W}}(\boldsymbol{x}) - 2y\hat{y}_{\boldsymbol{\epsilon W}}(\boldsymbol{x}) + y^2\right] \end{aligned} \tag{7.20}$$

Assuming small noise, we can consider the taylor series expansion of $\hat{y}_{\boldsymbol{\epsilon W}}(\boldsymbol{x})$ around the unperturbed function $\hat{y}(\boldsymbol{x})$.

$$\hat{y}_{\boldsymbol{\epsilon W}}(\boldsymbol{x}) = \hat{y}(\boldsymbol{x}) + \boldsymbol{\epsilon}_{\boldsymbol{W}}^\top \nabla_{\boldsymbol{W}} \hat{y}(\boldsymbol{x}) + \frac{1}{2}\boldsymbol{\epsilon}_{\boldsymbol{W}}^\top \nabla^2_{\boldsymbol{W}} \hat{y}(\boldsymbol{x})\boldsymbol{\epsilon W} + O(\boldsymbol{\epsilon}_{\boldsymbol{W}}^3) \tag{7.21}$$

From here, we follow the same basic strategy that was laid-out in the pervious section in analyzing the effect of adding noise to the input. That is, we substitute the taylor series expansion of $\hat{y}_{\boldsymbol{\epsilon W}}(\boldsymbol{x})$ into the objective function in Eq. 7.20.

$$\begin{aligned}\tilde{J}_{\boldsymbol{W}} \approx\ & \mathbb{E}_{p(\boldsymbol{x},y,\boldsymbol{\epsilon W})}\left[\left(\hat{y}(\boldsymbol{x}) + \boldsymbol{\epsilon}_{\boldsymbol{W}}^\top \nabla_{\boldsymbol{W}} \hat{y}(\boldsymbol{x}) + \frac{1}{2}\boldsymbol{\epsilon}_{\boldsymbol{W}}^\top \nabla^2_{\boldsymbol{W}}\, \hat{y}(\boldsymbol{x})\boldsymbol{\epsilon W}\right)^2\right] \\ & \mathbb{E}_{p(\boldsymbol{x},y,\boldsymbol{\epsilon W})}\left[-2y\left(\hat{y}(\boldsymbol{x}) + \boldsymbol{\epsilon}_{\boldsymbol{W}}^\top \nabla_{\boldsymbol{W}} \hat{y}(\boldsymbol{x}) + \frac{1}{2}\boldsymbol{\epsilon}_{\boldsymbol{W}}^\top\, \nabla^2_{\boldsymbol{W}} \hat{y}(\boldsymbol{x})\boldsymbol{\epsilon W}\right)\right] + \mathbb{E}_{p(\boldsymbol{x},y,\boldsymbol{\epsilon W})}\left[y^2\right] \\ =\ & \mathbb{E}_{p(\boldsymbol{x},y,\boldsymbol{\epsilon W})}\left[(\hat{y}(\boldsymbol{x}) - y)^2\right] - 2\mathbb{E}_{p(\boldsymbol{x},y,\boldsymbol{\epsilon W})}\left[\frac{1}{2}y\boldsymbol{\epsilon}_{\boldsymbol{W}}^\top \nabla^2_{\boldsymbol{W}} \hat{y}(\boldsymbol{x})\boldsymbol{\epsilon}\right] \\ & + \mathbb{E}_{p(\boldsymbol{x},y,\boldsymbol{\epsilon W})}\left[\hat{y}(\boldsymbol{x})\boldsymbol{\epsilon}_{\boldsymbol{W}}^\top \nabla^2_{\boldsymbol{W}}\, \hat{y}(\boldsymbol{x})\boldsymbol{\epsilon W} + \left(\boldsymbol{\epsilon}_{\boldsymbol{W}}^\top \nabla_{\boldsymbol{W}} \hat{y}(\boldsymbol{x})\right)^2 + O(\boldsymbol{\epsilon}_{\boldsymbol{W}}^3)\right]. \end{aligned} \tag{7.22}$$

$$\tag{7.23}$$

Where we have used the fact that $\mathbb{E}_{\boldsymbol{\epsilon W})}\boldsymbol{\epsilon}_{\boldsymbol{W}} = \mathbf{0}$ to drop terms that are linear in $\boldsymbol{\epsilon}_{\boldsymbol{W}}$. Incorporating the assumption that $\mathbb{E}_{\boldsymbol{\epsilon}_{\boldsymbol{W}})}\boldsymbol{\epsilon}^2_{\boldsymbol{W}} = \eta\boldsymbol{I}$, we have:

$$\tilde{J_{\boldsymbol{W}}} \approx J + \nu\mathbb{E}_{p(\boldsymbol{x},y)}\left[(\hat{y}(\boldsymbol{x}) - y)\nabla^2_{\boldsymbol{W}} \hat{y}(\boldsymbol{x})\right] + \nu\mathbb{E}_{p(\boldsymbol{x},y)}\left[\|\nabla_{\boldsymbol{W}}\, \hat{y}(\boldsymbol{x})\|^2\right] \tag{7.24}$$

Again, if we consider minimizing this objective function, we can see that the optimal value of $\hat{y}(\boldsymbol{x})$ is:

$$\hat{y}(\boldsymbol{x}) = \mathbb{E}_{p(y|\boldsymbol{x})}[y] + O(\eta),$$

implying that the expectation in the middle term in Eq. 7.24, $\mathbb{E}_{p(\boldsymbol{x},y)}\left[(\hat{y}(\boldsymbol{x}) - y)\nabla^2_{\boldsymbol{W}}\, \hat{y}(\boldsymbol{x})\right]$, reduces to $O(\eta)$ because the expectation of the difference $(\hat{y}(\boldsymbol{x}) - y)$ is reduces to $O(\eta)$.

This leaves us with the objective function of the form

$$\tilde{J}_{\boldsymbol{W}} = \mathbb{E}_{p(\boldsymbol{x},y)}\ \ (\hat{y}(\boldsymbol{x}) - y)^2\ \ + \eta\mathbb{E}_{\,p(\boldsymbol{x},y)}\ \|\nabla_{\boldsymbol{W}}\hat{y}(\boldsymbol{x})\|^2\ \ + O(\eta^2\,).$$

$$\left[\qquad\qquad\right] \qquad\qquad \left[\qquad\qquad\right]$$

For small η, the minimization of J with added weight noise (with covariance $\eta \boldsymbol{I}$) is equivalent to minimization of J with an additional *regularization* term: $\eta \mathbb{E}_{p(\boldsymbol{x},y)} \left[\|\nabla_{\boldsymbol{W}} \hat{y}(\boldsymbol{x})\|^2 \right]$. This form of regularization encourages the parameters to go to regions of parameter space that have relatively small gradients. In other words, it pushes the model into regions where the model is relatively insensitive to small variations in the weights. Regularization strategies with this kind of behaviour have been considered before TODO restore this broken citation when it is fixed In the simplified case of linear regression (where, for instance, $\hat{(y)}(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x} + b$), this regularization term collapses into $\eta \mathbb{E}_{p(\boldsymbol{x})} \left[\|\boldsymbol{x}\|^2 \right]$, which is not a function of parameters and therefore does not contribute to the gradient of $\tilde{J}_{\boldsymbol{W}}$ w.r.t the model parameters.

7.7 Early Stopping as a Form of Regularization

When training large models with high capacity, we often observe that training error decreases steadily over time, but validation set error begins to rise again. See Fig. 7.3 for an example of this behavior. This behavior occurs very reliably.

This means we can obtain a model with better validation set error (and thus, hopefully better test set error) by returning to the parameter setting at the point in time with the lowest validation set error. Instead of running our optimization algorithm until we reach a (local) minimum, we run it until the error on the validation set has not improved for some amount of time. Every time the error on the validation set improves, we store a copy of the model parameters. When the training algorithm terminates, we return these parameters, rather than the latest parameters. This procedure is specified more formally in Alg. 7.1.

This strategy is known as *early stopping*. It is probably the most commonly used form of regularization in deep learning. Its popularity is due both to its effectiveness and its simplicity.

One way to think of early stopping is as a very efficient hyperparameter selection algorithm. In this view, the number of training steps is just another hyperparameter. We can see in Fig. 7.3 that this hyperparameter has a U-shaped validation set performance curve, just like most other model capacity control parameters. In this case, we are controlling the effective capacity of the model by determining how many steps it can take to fit the training set precisely. Most of the time, setting hyperparameters requires an expensive guess and check process, where we must set a hyperparameter at the start of training, then run training for several steps to see its effect. The "training time" hyperparameter is unique in that by definition a single run of training tries out many values of the hyperparameter. The only significant cost to choosing this hyperparameter automatically via early stopping is running the validation set evaluation periodically during