Data Science 2

Training

Ondřej Týbl

Charles University, Prague

Outline

1. (Stochastic) Gradient Descent
2. Backpropagation
3. Activations
4. Optimizers
5. Regularization

(Stochastic) Gradient Descent

(Stochastic) Gradient Descent

Given a network parametrized by a perameter vector $\theta \in \Theta$ we train the network by minimization of the loss function $\mathcal{L}(\theta)$.

Unlike in the case of linear regression there is no direct formula for computation of the optimal parameter.

(Stochastic) Gradient Descent

Suppose a differentiable loss function $\mathcal{L} : \mathbb{R}^p \mapsto [0, \infty)$.

Algorithm (Gradient Descent)

Initialize $\theta_0 \in \mathbb{R}^p$.

Choose a sequence of learning rates $\{\alpha_n\}_{n=0}^{\infty}$, where $\alpha_n > 0$.

Iterate until convergence

$$\theta_{n+1} = \theta_n - \alpha_n \nabla_\theta \mathcal{L}(\theta_n), \quad n \in \mathbb{R}.$$

(Stochastic) Gradient Descent

- Learning rate too small: slow training
- Learning rate too large: jumps easily out of a *good region*

Typically converges to a local minima, unless the loss is regular enough.

Theorem (Convergence of Gradient Descent)

Suppose $\mathcal{L} : \mathbb{R}^p \mapsto \mathbb{R}$ is convex and differentiable with its gradient being Lipschitz with a Lipschitz constant $L \in (0, \infty)$. Denote its global minimum as θ^*. Assume constant learning rate $\alpha_n = \alpha \leq 1/L$. Then we have

$$|\mathcal{L}(\theta^*) - \mathcal{L}(\theta_n)| \leq \frac{1}{2\alpha n} \|\theta^* - \theta_0\|^2, \quad n \in \mathbb{N}.$$

In particular, θ_n converges to the global optimum with linear rate of convergence.

(Stochastic) Gradient Descent

Typically, we have a training dataset $x_1, \ldots, x_N \in \mathbb{R}^m$ with labels $y_1, \ldots, y_N \in \mathbb{R}^n$ and the loss is a population average over some quantity

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{j=1}^{N} l\left(y_j, f_\theta\left(x_j\right)\right), \quad \theta \in \Theta \subset \mathbb{R}^p.$$

Example: For the regression task we might take

$$l(x, y) = \|x - y\|^2, \quad x, y \in \mathbb{R}^n$$

and for the classification task we might take[1].

$$l(x, y) = -\sum_{i=1}^{n} x_i \log y_i$$

[1] Here we assume that the network output and labels represent a distribution over n categories, see previous lectures.

(Stochastic) Gradient Descent

When the population is large, evaluation of $\mathcal{L}$ at a single point is then too expensive. We can, however, estimate the population average with an unbiased estimator – the sample average.

$$\mathcal{L}(\theta) = \frac{1}{N}\sum_{j=1}^{N} l\left(y_j, f_\theta\left(x_j\right)\right) \sim \frac{1}{B}\sum_{j=1}^{B} l\left(y_j, f_\theta\left(x_j\right)\right)$$

(Stochastic) Gradient Descent

The optimization algorithm is then denoted as *Stochastic Gradient Descent* and the subset of training data of size B is called a *(mini)batch*.

The extreme case $B = 1$ leads to an estimation with a large variance of error. It corresponds to the perceptron training algorithm with learning rate 1.

(Stochastic) Gradient Descent

It is standard to use the intermediate case $1 < B << N$ so that

- the sample average has a low error variance[2]
- parallelism can be utilized: the hardware and software are ready to evaluate l in parallel for multiple examples at a time which speeds up the training as less iterations is needed to iterate over all the data[3]

[2] Later, we will see that a small batch size can improve training convergence as a form of regularization. So in fact, there is a trade-off between large and small batch sizes.
[3] Due to the nature of parallel computation it is desirable that B is a power of 2.

(Stochastic) Gradient Descent

In practice, training examples are iterated through so that each example is used in one batch exactly.

This way, $\lceil \frac{B}{N} \rceil$ iterations are made – we call these a one *epoch*.

Training examples are randomly shuffled between each epoch.

(Stochastic) Gradient Descent

Theorem (Robbins-Monro Convergence, Sec.2.4.1 in [1])

If the loss $\mathcal{L} : \mathbb{R}^m \mapsto [0, \infty)$ has a second derivative which is continuous and strongly convex having a root θ^* in the interior of its domain, then the stochastic gradient descent estimator with learning rate $\{\alpha_n\}_{n=0}^{\infty}$ such that

$$\sum_{n=0}^{\infty} \alpha_n = \infty \quad \text{and} \quad \sum_{n=0}^{\infty} \alpha_n^2 < \infty$$

converges to θ^* almost surely.

The learning rate should not decay too fast so that any point can be reached; the learning rate should decay so that the procedure converges to a specific point[4].

[4] Think of $\alpha_n \sim 1/n$.

(Stochastic) Gradient Descent

In practice the loss is much more complicated and *learning rate schedules* are used.

First, warm-up period of few epochs when the learning rate starts from zero and linearly increases to a specific value to assure a smooth transition from initial state to a region with large gradients.

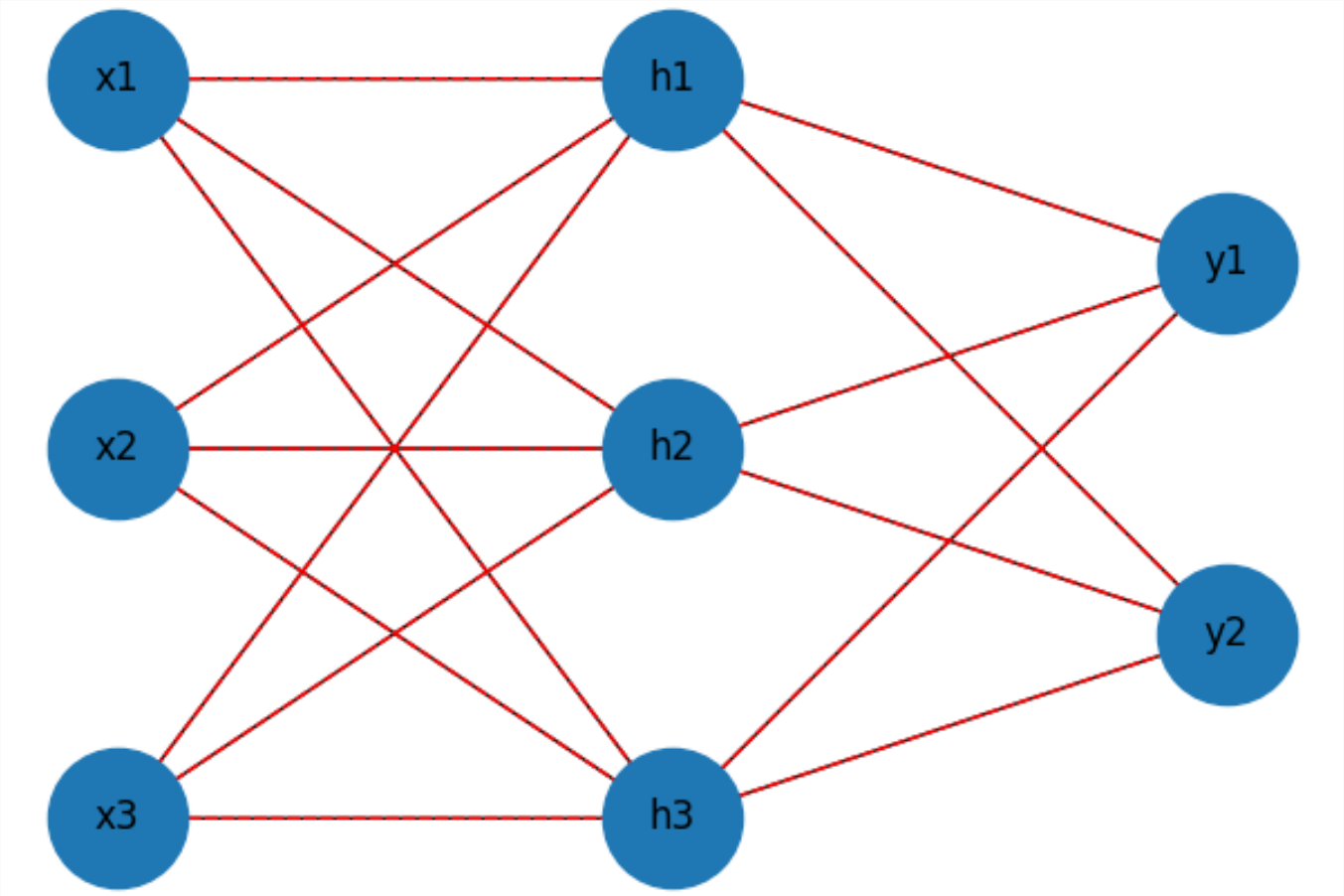
Then a scheduled decay. For example

- exponential decay $\log \alpha_n = -a \cdot n + b$,
- polynomial decay $\alpha_n \sim 1/n^a$,
- cosine decay $\alpha_n \sim \cos(a \cdot n)$,
- learning rate drop after a specific number of iterations without loss decrease.

Backpropagation

Backpropagation

Assume a network $f_\theta : \mathbb{R}^3 \mapsto \mathbb{R}^2$ for a regression task with one hidden layer. The activation function for the hidden layer being the rectified linear unit $ReLU(x) = \max(0, x)$ and the output activation function being the identity.



Backpropagation

We represent $\theta \in \mathbb{R}^{9+3+6+2}$ by $W^1 \in \mathbb{R}^{3\times 3}, w_0^1 \in \mathbb{R}^3, W^2 \in \mathbb{R}^{2\times 3}, w_0^2 \in \mathbb{R}^2$,

$$f_\theta(x) = W^2 ReLU\left(W^1 x + w_0^1\right) + w_0^2, \quad x \in \mathbb{R}^3$$

With training examples $(x_1, y_1), \ldots, (x_N, y_N) \in \mathbb{R}^5$ we minimize the loss

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{j=1}^{N} \|f_\theta(x_j) - y_j\|^2$$

by stochastic gradient descent. We therefore need to evaluate

$$\frac{\partial}{\partial \theta_i} \left(\|f_\theta(x) - y\|^2\right), \quad i = 1, \ldots, 20.$$

Backpropagation

For fixed $x \in \mathbb{R}^3$ and $y \in \mathbb{R}^2$ denote

$$Z = W^1 x + w_0^1, \quad H = ReLU\left(Z\right), \quad f_\theta\left(x\right) = W^2 H + w_0^2$$

Then, in the matrix notation

$$\frac{\partial}{\partial f_\theta\left(x\right)}\left(\left\|f_\theta\left(x\right) - y\right\|^2\right) = 2\left(f_\theta\left(x\right) - y\right) \in \mathbb{R}^{1\times 2} \tag{1}$$

and by the chain rule

$$\begin{aligned}\frac{\partial}{\partial W^2}\left(\left\|f_\theta\left(x\right) - y\right\|^2\right) &= \frac{\partial}{\partial f_\theta\left(x\right)}\left(\left\|f_\theta\left(x\right) - y\right\|^2\right)\frac{\partial}{\partial W^2}\left(f_\theta\left(x\right)\right) \\ &= 2\left(f_\theta\left(x\right) - y\right)^T H^T \in \mathbb{R}^{2\times 3}\end{aligned}$$

$$\begin{aligned}\frac{\partial}{\partial w_0^2}\left(\left\|f_\theta\left(x\right) - y\right\|^2\right) &= \frac{\partial}{\partial f_\theta\left(x\right)}\left(\left\|f_\theta\left(x\right) - y\right\|^2\right)\frac{\partial}{\partial w_0^2}\left(f_\theta\left(x\right)\right) \\ &= 2\left(f_\theta\left(x\right) - y\right) I_{2\times 2} \in \mathbb{R}^2\end{aligned}$$

Backpropagation

For W^1 and w_0^1 we continue with the chain rule.

$$\begin{aligned}\frac{\partial}{\partial W^1}\left(\|f_\theta(x)-y\|^2\right) &= \frac{\partial}{\partial f_\theta(x)}\left(\|f_\theta(x)-y\|^2\right)\frac{\partial}{\partial H}\left(f_\theta(x)\right)\frac{\partial}{\partial Z}(H)\frac{\partial}{\partial W^1}(Z)\\ &= 2\left(\left(f_\theta(x)-y\right)W^2\frac{\partial ReLU}{\partial Z}(H)\right)^T x^T \in \mathbb{R}^{3\times 3}\end{aligned}$$

$$\begin{aligned}\frac{\partial}{\partial w_0^1}\left(\|f_\theta(x)-y\|^2\right) &= \frac{\partial}{\partial f_\theta(x)}\left(\|f_\theta(x)-y\|^2\right)\frac{\partial}{\partial H}\left(f_\theta(x)\right)\frac{\partial}{\partial Z}(H)\frac{\partial}{\partial w_0^1}(Z)\\ &= 2\left(f_\theta(x)-y\right)W^2\frac{\partial ReLU}{\partial Z}(H)\, I_{3\times 3} \in \mathbb{R}^{1\times 3}\end{aligned}$$

Backpropagation

We recall that the activation function is applied component wise and therefore

$$ReLU(x) = (\max(0, x_1), \ldots, \max(0, x_m))^T, \quad x \in \mathbb{R}^m$$

and

$$\frac{\partial}{\partial Z}(H) = \frac{\partial ReLU}{\partial Z}(H) = \begin{bmatrix} \mathbf{1}_{(0,\infty)}(h_1) & 0 & 0 \\ 0 & \mathbf{1}_{(0,\infty)}(h_2) & 0 \\ 0 & 0 & \mathbf{1}_{(0,\infty)}(h_3) \end{bmatrix}, \quad h \in \mathbb{R}^3$$

Backpropagation

Backpropagation is time and memory efficient we used only

1. derivatives of the activation functions,
2. already evaluated node values,
3. matrix multiplication of 1 and 2.

Backpropagation

We distinguish

- *forward pass* that produces the network output,
- *backward pass* that computes the derivatives.

For $\theta \in \mathbb{R}^p$ the backpropagation algorithm computes all the derivatives in $O(p)$ time. So overall one forward and backward iteration scales linearly with the number of parameters.

Backpropagation

Now we modify our example to the binary classification where labels $y_j \in \{0, 1\}$. The network output[5] $f_\theta(x) \in \mathbb{R}$ represents the logits and instead of the identity we apply the sigmoid activation after the output layer. The loss is the cross entropy $\mathcal{L}(\theta) = \sum_{j=1}^{N} l\left(f_\theta\left(x_j\right), y_j\right)$ with

$$l\left(f_\theta\left(x_j\right), y_j\right) = -\left(y_j \log \frac{1}{1+e^{-f_\theta\left(x_j\right)}} + \left(1-y_j\right) \log \frac{1}{1+e^{f_\theta\left(x_j\right)}}\right).$$

Then we have

$$\frac{\partial l}{\partial f_\theta(x)}\left(f_\theta(x), y\right) = p_\theta(x) - y.$$

where we put $p_\theta(x) = \frac{1}{1+e^{-f_\theta(x)}}$ and we obtained comparable result to (1) for the regression.

[5] In contrast to the regression example, here the network output is one dimensional.

Activations

Activations

From backpropagation we know that what matters are the derivatives of the activations.

For deep networks the derivatives should be easy to compute and around 1 as we multiply in the chain rule.[6]

- too large derivatives lead to *exploding gradient* (overflow of the parameters)
- too small derivatives lead to *vanishing gradient* and slow training

$$\frac{\partial \mathcal{L}}{\partial W^1} = \frac{\partial \mathcal{L}}{\partial y_1} \frac{\partial y_1}{\partial y_2} \frac{\partial y_2}{\partial y_3} \frac{\partial y_3}{\partial y_4} \cdots$$

[6] The derivatives near to zero matter the most as we initialize so that neurons start with small values.

Activations

The sigmoid (usually in the output layer for binary classification)

$$\sigma(x) = \frac{1}{1+e^{-x}}, \quad \sigma'(x) = \sigma(x)(1-\sigma(x)), \quad \sigma'(0) = \frac{1}{4}$$

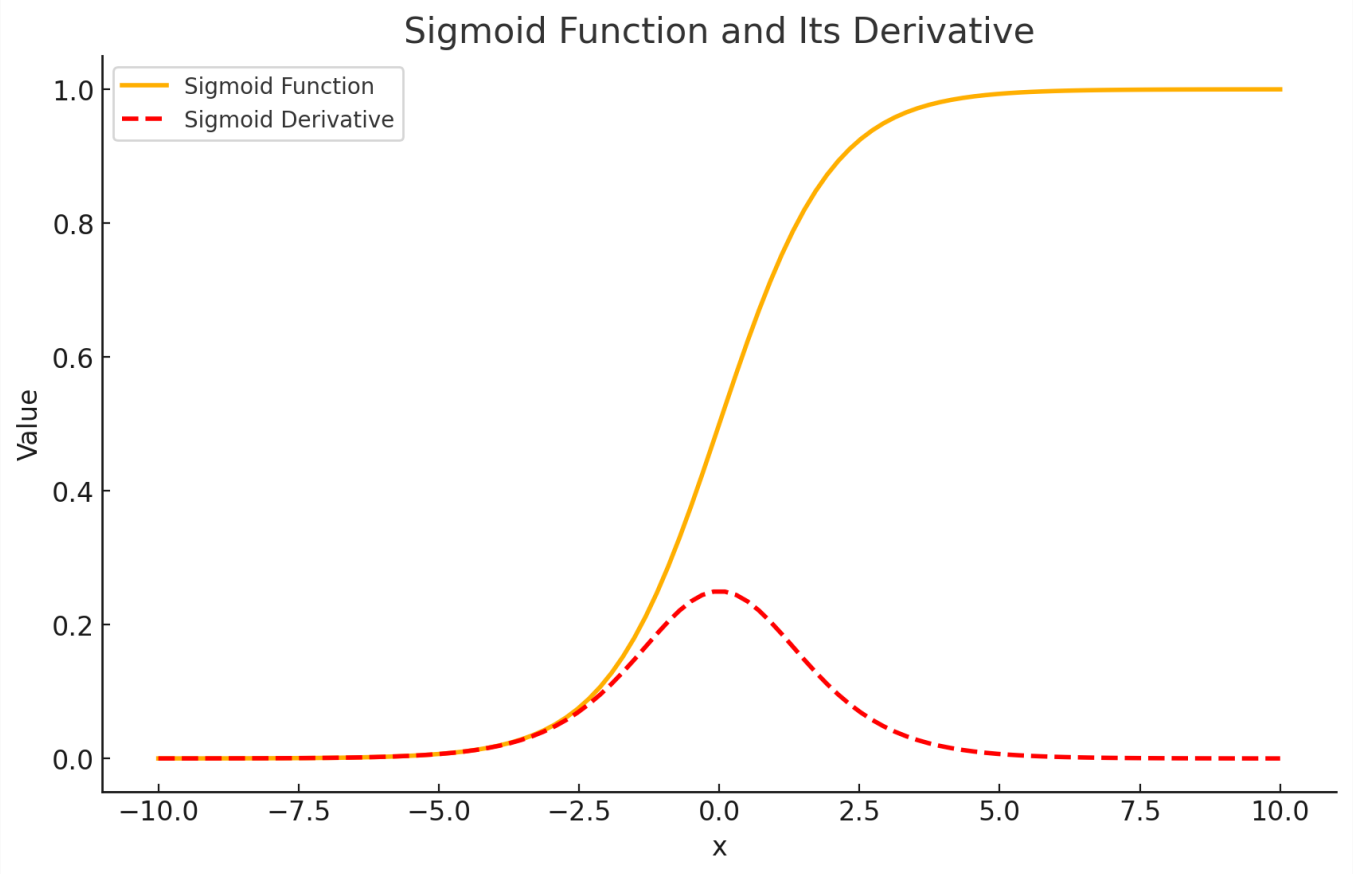


Figure 2: Sigmoid activation function with its derivative.

Activations

The hyperbolic tangent

$$\tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1}, \quad \tanh'(x) = 1 - \tanh^2(x), \quad \tanh'(0) = 1$$

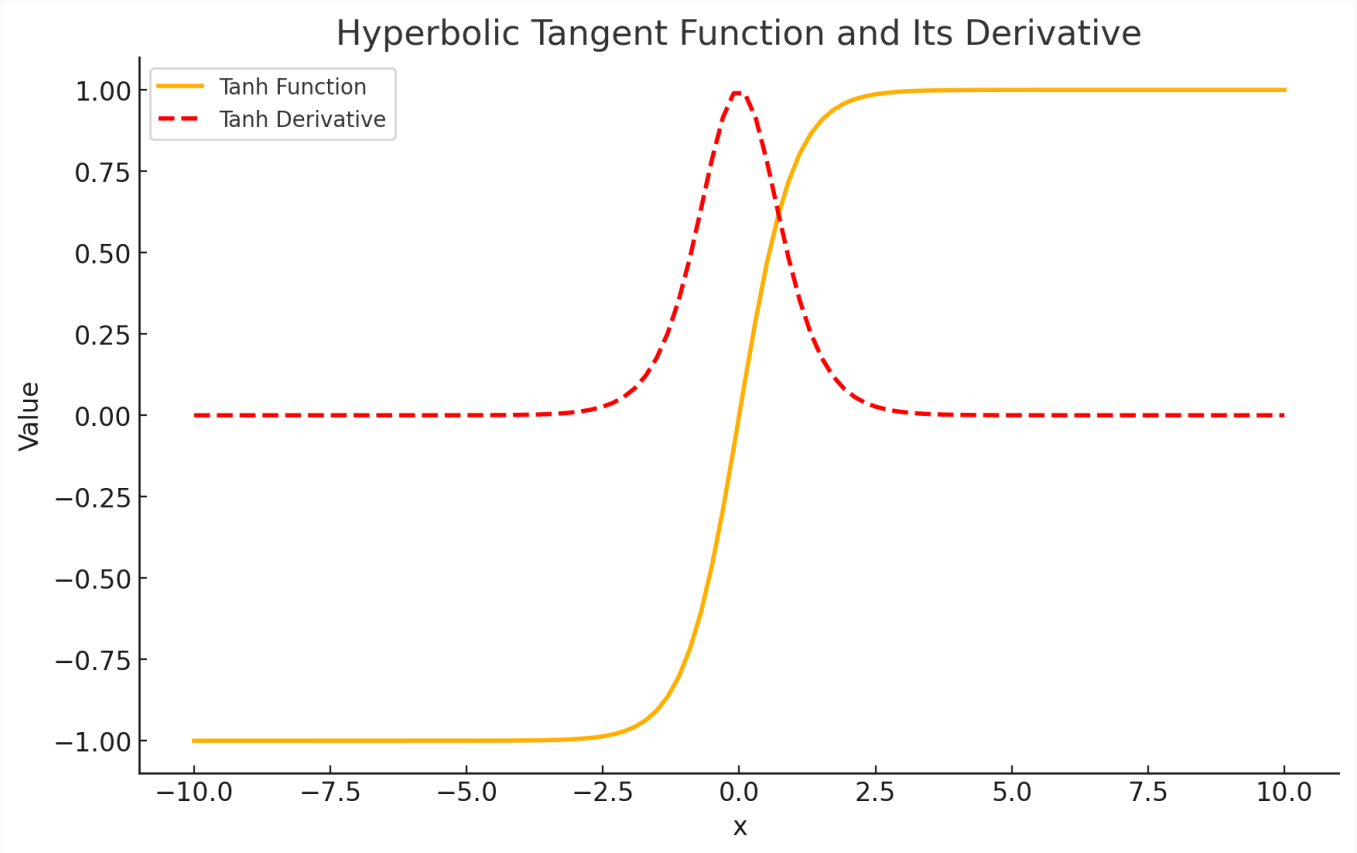


Figure 3: Hyperbolic tangent activation function with its derivative.

Activations

The hyperbolic tangent is just a rescale of the sigmoid function so that it vanishes at zero with unit derivative.

$$\tanh(x) = 2\sigma(2x) - 1$$

Activations

The rectified linear unit

$$ReLU(x) = \max(0, x), \quad ReLU'(x) = \mathbf{1}_{(0,\infty)}(x), \quad ReLU'(0) \text{ not defined}$$

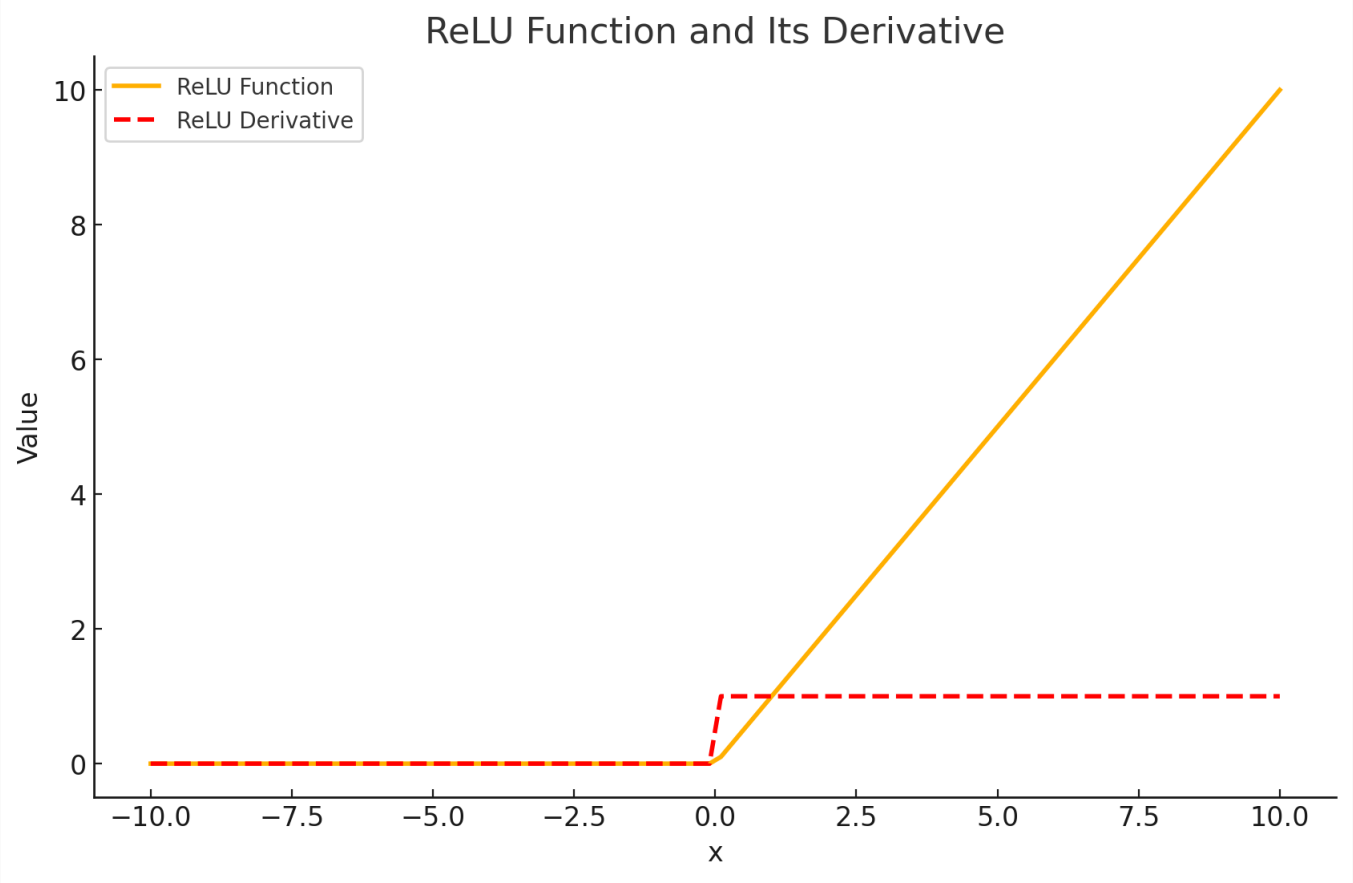


Figure 4: Rectified linear unit activation function with its derivative.

Activations

Swish function

$$\text{swish}(x) = \frac{x}{1+e^{-x}} = x\sigma(x), \quad \text{swish}'(x) = \ldots, \quad \text{swish}'(0) = \frac{1}{2}$$

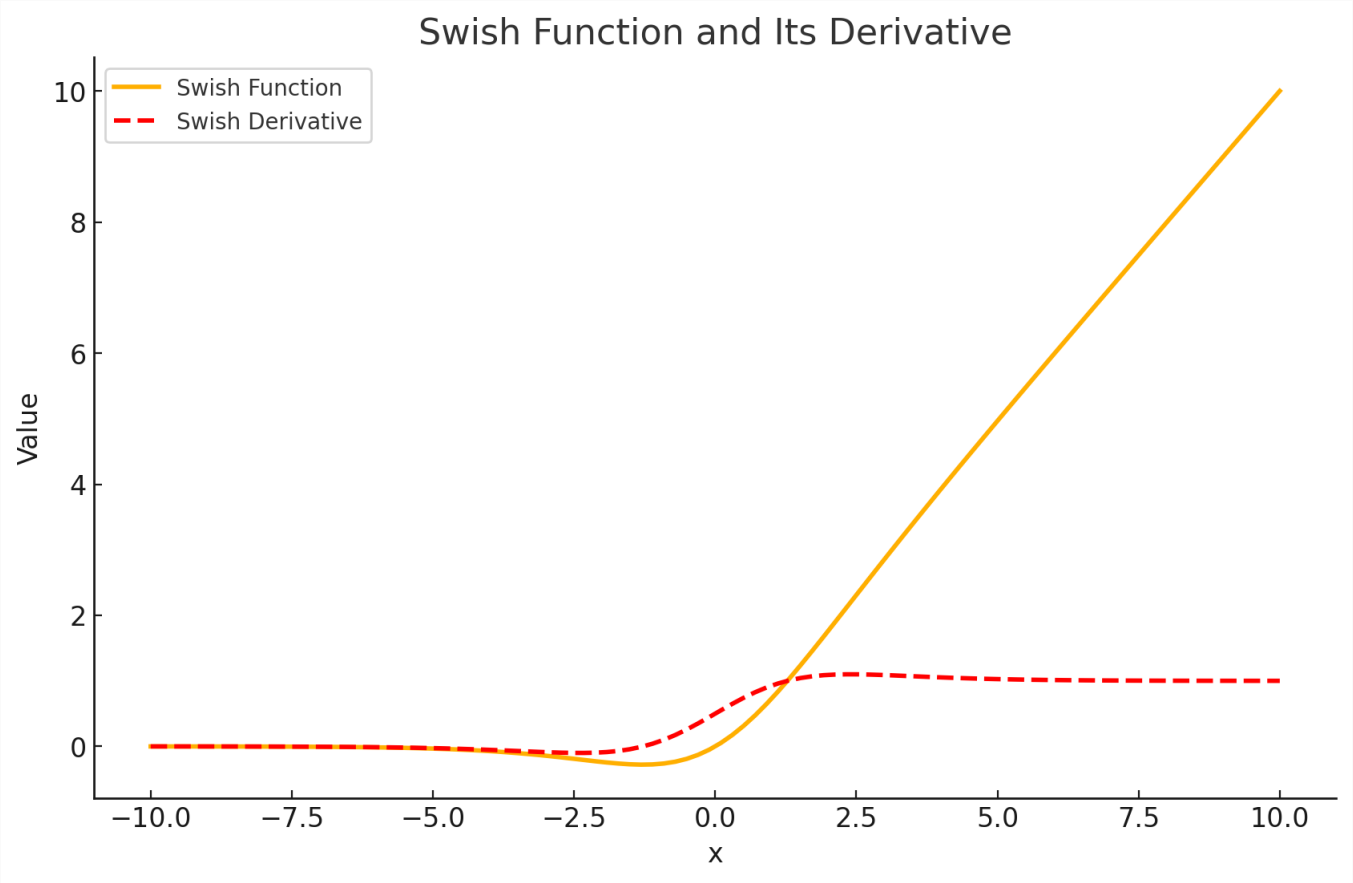


Figure 5: Swish activation function with its derivative.

Optimizers

Optimizers

Over time, researchers have come up with several tweaks to the classical stochastic gradient descent leading. Those are generally called *optimizers*.

Optimizers

Algorithm (Stochastic Gradient Descent with Momentum, link)

Initialize $\theta_0 \in \mathbb{R}^p$.

Choose a learning rate[7] $\alpha > 0$ and a momentum $\beta \in (0, 1)$[8] and set $v_0 = 0$.

Iterate until convergence

- choose a batch of labeled examples $(x_1, y_1), \ldots, (x_B, y_B) \in \mathbb{R}^m \times \mathbb{R}^n$
- update

$$v_{n+1} = \beta v_n - \alpha \nabla_\theta \left(\frac{1}{B} \sum_{j=1}^{B} l\left(y_j, f_{\theta_n}\left(x_j\right)\right) \right)$$

$$\theta_{n+1} = \theta_n + v_{n+1}$$

[7] We use a constant learning rate for simplicity.

[8] Typically, β is around 0.9.

Optimizers

Algorithm (Adaptive Moment Estimation, 2014)

Initialize $\theta_0 \in \mathbb{R}^p$.

Choose a learning rate $\alpha > 0$, momentum $\beta_1, \beta_2 \in (0, 1)$ and set $v_0 = r_0 = 0$, $\epsilon << 1$.

Iterate until convergence

- choose a batch of labeled examples $(x_1, y_1), \ldots, (x_B, y_B) \in \mathbb{R}^m \times \mathbb{R}^n$
- update

$$v_{n+1} = (1 - \beta_1^n)^{-1} \left(\beta_1 v_n + (1 - \beta_1) \nabla_\theta \left(\frac{1}{B} \sum_{j=1}^{B} l\left(y_j, f_{\theta_n}\left(x_j\right)\right) \right) \right)$$

$$r_{n+1} = (1 - \beta_2^n)^{-1} \left(\beta_2 r_n + (1 - \beta_2) \left(\nabla_\theta \left(\frac{1}{B} \sum_{j=1}^{B} l\left(y_j, f_{\theta_n}\left(x_j\right)\right) \right) \right)^2 \right)$$

$$\theta_{n+1} = \theta_n - \frac{\alpha}{\sqrt{r_n + \epsilon}} v_{n+1}$$

Optimizers

The square root, second power and division are computed element wise.

v_n is an exponential moving average of the first moment of the gradient of l

r_n is an exponential moving average of the squared norm of the gradient of l

Typically, β_1 is around 0.9 and β_2 around 0.999.

Optimizers

Many other optimizers exist. To name one we give a reference to LAMB used primarily in computer vision where large batch sizes are needed, see [2].

Regularization

Regularization

Generalization error tends to be better for networks with less over fitting.

Regularization

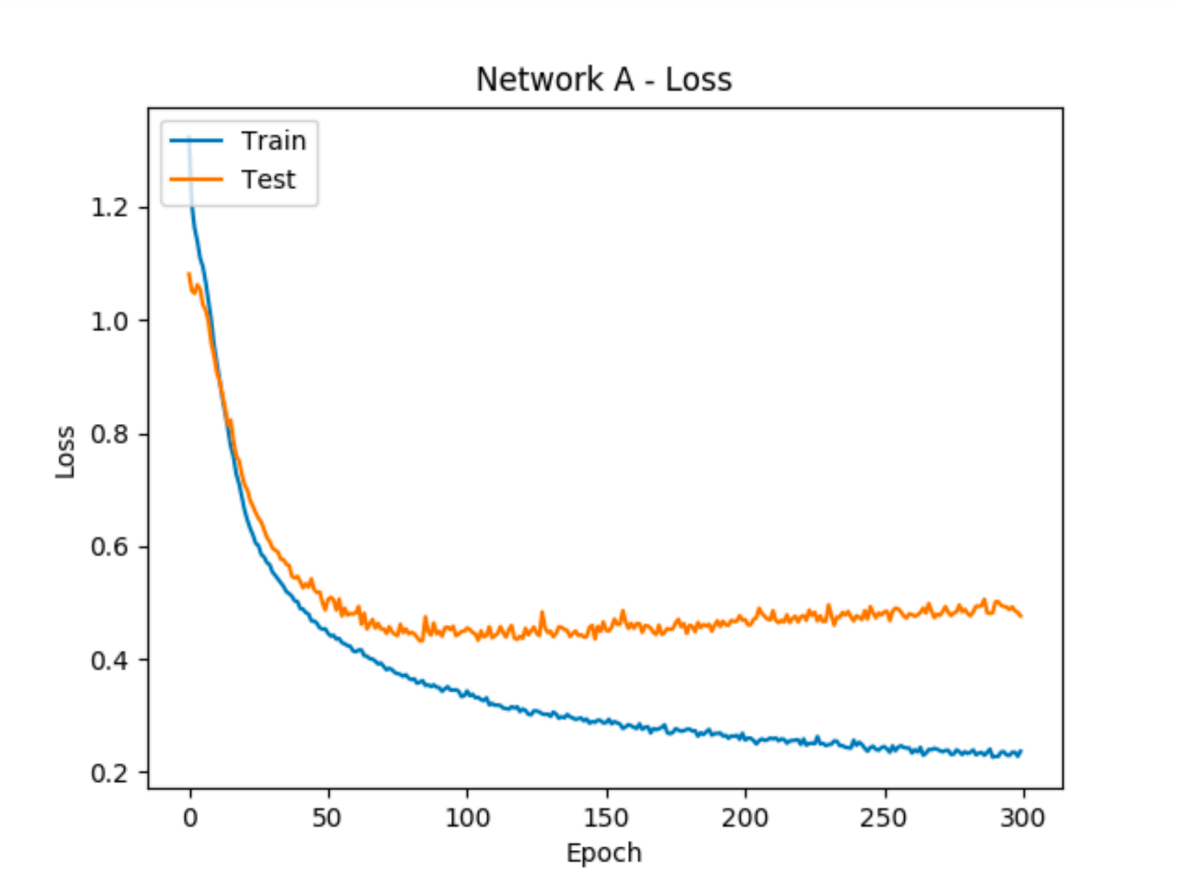
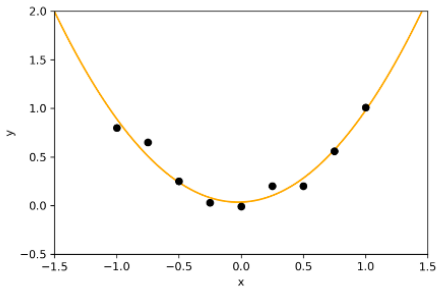


Figure 6: Example of train and test loss evolution.

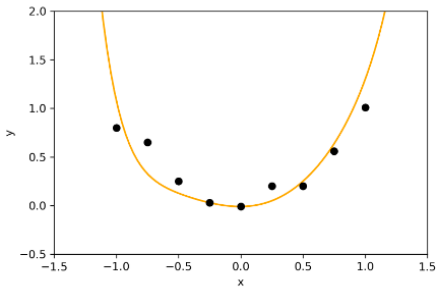
Typically, we set *early stopping* to avoid over fitting on train set.

Regularization

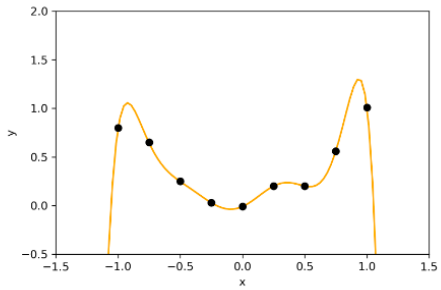
*L*1 and *L*2 regularization adds penalty terms to the cost function to discourage complex models.



(a) #params = 3
MSE = 0.006
L2 norm = 0.90
L1 norm = 0.98



(b) #params = 9
MSE = 0.035
L2 norm = 1.06
L1 norm = 2.32



(c) #params = 9
MSE = 0
L2 norm = 32.69
L1 norm = 70.03

Figure 7: Examples of polynomials fitting the data with losses and parameter norms.

Regularization

Smaller parameter norm is obtained through adjusted loss (another hyper parameter $\lambda > 0$ is introduced)

- *L1*, or *lasso*

$$\mathcal{L}_1(\theta) = \mathcal{L}(\theta) + \lambda \|\theta\|_1 = \mathcal{L}(\theta) + \lambda \left(\sum_{i=1}^{p} |\theta_i| \right)$$

- *L2* or *ridge regression*

$$\mathcal{L}_2(\theta) = \mathcal{L}(\theta) + \lambda \|\theta\|^2 = \mathcal{L}(\theta) + \lambda \left(\sum_{i=1}^{p} |\theta_i|^2 \right)$$

Regularization

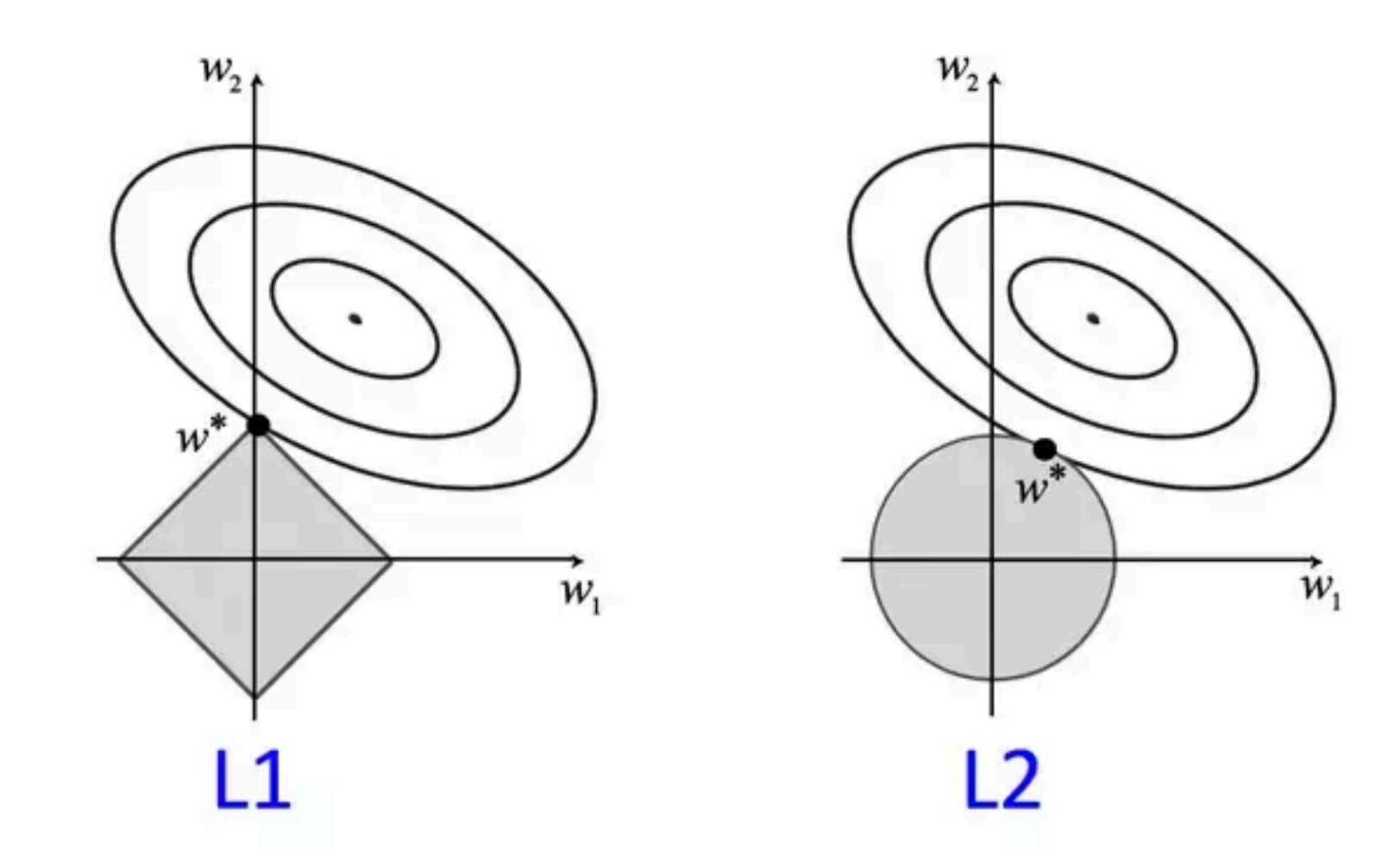


Figure 8: Example of loss landscape and unit balls in the parameter space measured either via $L1$ or $L2$ norm.

Regularization

L2 regularization is sometimes called *weight decay*. This follows from the form of the stochastic gradient descent update[9]

$$\begin{aligned}\theta_{n+1} &= \theta_n - \alpha \nabla_\theta \mathcal{L}_2(\theta_n) \\ &= \theta_n - \alpha \nabla_\theta \left(\mathcal{L}(\theta_n) + \lambda \|\theta_n\|^2 \right) \\ &= \theta_n - \alpha \nabla_\theta \mathcal{L}(\theta_n) - 2\alpha\lambda\theta_n \\ &= (1 - 2\alpha\lambda)\,\theta_n - \alpha \nabla_\theta \mathcal{L}(\theta_n)\end{aligned}$$

where $\alpha > 0$ is the learning rate and $\lambda > 0$ is the regularization hyper parameter.

[9] However, the introduction of momentum destroys this relation.

Regularization

We have seen that for the classification task[10] we have

$$\frac{\partial \mathcal{L}}{\partial f_\theta(x)} = p_\theta(x) - y, \quad \text{where} \quad p_\theta(x)_j = \frac{e^{f_\theta(x)_j}}{\sum_{i=1}^{K} e^{f_\theta(x)_i}}$$

where $y \in \mathbb{R}^K$ is one-hot encoded.

Therefore, the logit corresponding to the true class is *always* pushed up in the gradient descent iteration and all the remaining are pushed down. If we make the model *less sure* about the true class it typically generalizes better.

[10] Above shown for a binary task but works generally with K classes.

Regularization

We thus adjust the one hot encoded label distribution $y \in \mathbb{R}^K$ using [11] $\alpha \in (0, 1)$

$$(1 - \alpha)\, y + \left(\frac{\alpha}{K}, \ldots, \frac{\alpha}{K}\right)^T \in \mathbb{R}^K$$

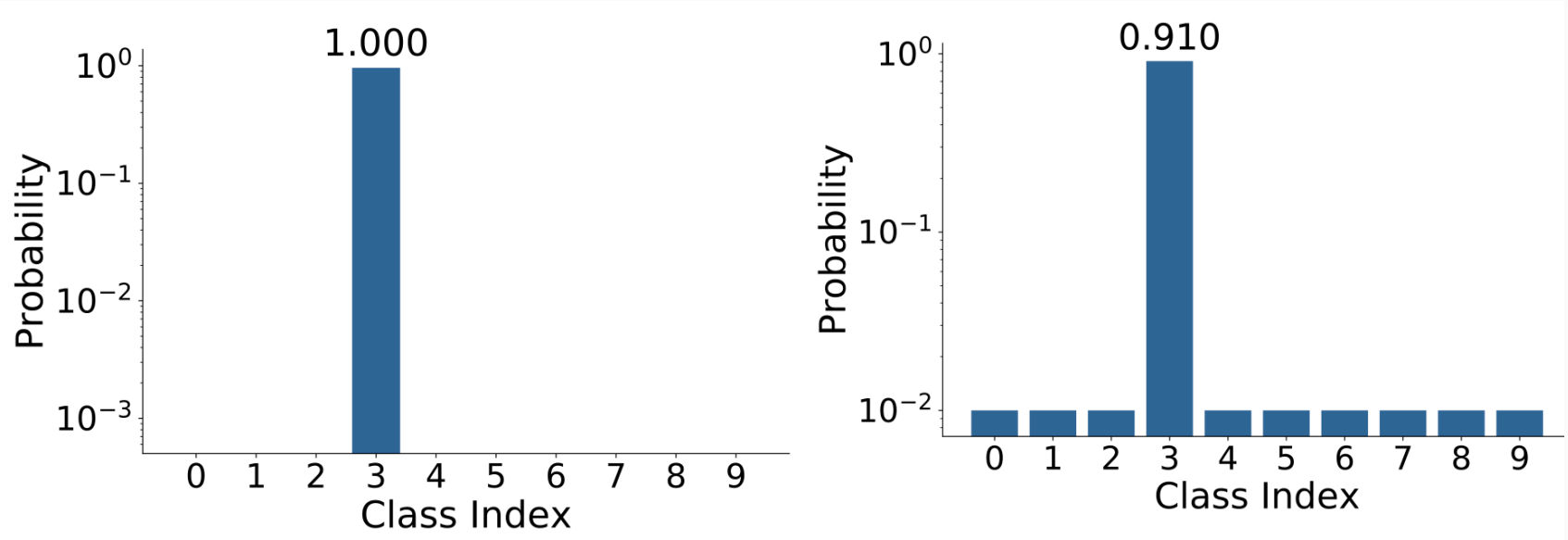


Figure 9: Examples of label distributions with 10 classes without (left) and with (right) label smoothing with parameter $\alpha = 0.1$.

[11] Typically $\alpha \leq 0.1$

Regularization

Dataset augmentation and *dropout* will be introduced in the context of convolutional neural networks later.

References i

C. M. Bishop.
Neural networks for pattern recognition.
Oxford university press, 1995.

Y. You, J. Li, S. Reddi, J. Hseu, S. Kumar, S. Bhojanapalli, X. Song, J. Demmel, K. Keutzer, and C.-J. Hsieh.
Large batch optimization for deep learning: Training bert in 76 minutes.
arXiv preprint arXiv:1904.00962, 2019.