Optimization for Deep Learning

Vincent Lepetit

September 12, 2021

- Loss functions;
- Generalization and Overfitting;
- Optimization for Deep Learning;
- Back-Propagation;
- Automatic Differentiation;
- Optimization Algorithms;
- Optimization Techniques.

- Loss functions;
- Generalization and Overfitting;
- Optimization for Deep Learning;
- Back-Propagation;
- Automatic Differentiation;
- Optimization Algorithms;
- Optimization Techniques.

How Can We Find the Parameters of a Deep Network?

Example of a two-layer network:

$$\begin{cases} \mathbf{h}(\mathbf{x}) = g(\mathbf{W}\mathbf{x} + \mathbf{b}) \\ \mathbf{o}(\mathbf{x}) = \mathbf{W}_2\mathbf{h}(\mathbf{x}) + \mathbf{b}_2 \end{cases}$$

In the case of supervised learning:

- Consider a training set made of pairs of input and expected output $(\mathbf{x}_i, \mathbf{e}_i)$;
- Introduce a loss function with a minimum $\hat{\Theta}$ that corresponds to good values for the parameters Θ:

$$\hat{\Theta} = \underset{\Theta}{\operatorname{argmin}} \ \mathcal{L}(\Theta)\,, \tag{1}$$

with $\Theta = [\mathbf{W}, \mathbf{b}, \mathbf{W}_2, \mathbf{b}_2]$.

- How do we define the loss function?
- How do we find a (good) minimum?

Loss Function

The loss function can be virtually any function that is differentiable.

Basic loss functions correspond to the cross-entropy between the expected values and the predicted ones:

$$\mathcal{L}(\Theta) = -\log \prod_{i=1}^{N} p_{\text{model}}(\mathbf{e}_i \mid \mathbf{o}(\mathbf{x}_i; \Theta)) \,, \tag{2}$$

where

- $\{(\mathbf{x}_i, \mathbf{e}_i)\}_{i=1..N}$ is the training set;
- $\mathbf{o}(\mathbf{x}_i; \Theta)$ is the output of the network with parameters Θ for input $\mathbf{x}_i$;
- $p_{\text{model}}(\mathbf{e} \mid \mathbf{o}(\mathbf{x}; \Theta))$: How we compute the probability for a value $\mathbf{e}$ given the output $\mathbf{o}(\mathbf{x}; \Theta)$ of the network.

(the cross-entropy can also be seen here as the negative log-likelihood of the training set given the predictions of the network)

Regression Problems

If we take: $p_{\mathsf{model}}(\mathbf{e} \mid \mathbf{o}(\mathbf{x};\Theta)) = \mathcal{N}(\mathbf{e};\, \mathbf{o}(\mathbf{x};\Theta), \sigma\mathbf{I})$:

$$\begin{aligned}\mathcal{L}(\Theta) &= -\log \textstyle\prod_{i=1}^{N} p_{\mathsf{model}}(\mathbf{e}_i \mid \mathbf{o}(\mathbf{x}_i;\Theta)) \\ &= k_1 \textstyle\sum_{i=1}^{N} \|\mathbf{e}_i - \mathbf{o}(\mathbf{x}_i;\Theta)\|^2 + k_2\,.\end{aligned} \tag{3}$$

(because $\mathcal{N}(\mathbf{e};\, \mathbf{o}, \sigma\mathbf{I}) = c_1 \exp(-\|\mathbf{e} - \mathbf{o}\|^2 / c_2)$)

If we ignore constants k_1 and k_2 that do not influence minimum $\hat{\Theta}$:

$$\mathcal{L}(\Theta) = \sum_{i=1}^{N} \|\mathbf{e}_i - \mathbf{o}(\mathbf{x}_i;\Theta)\|^2\,, \tag{4}$$

which is simply the mean squared error (MSE).

Classification Problems

Training set: $\{(\mathbf{x}_i, c_i)\}_{i=1..N}$, where $c_i = [1;C]$ is the expected class index, among C possible classes.

In this case, the output $\mathbf{o}(\mathbf{x}_i;\Theta)$ of the network is taken to be a C-vector: $\mathbf{o}(\mathbf{x}_i;\Theta) \in \mathbb{R}^C$.

Let's introduce vector $\mathbf{d} = \mathrm{softmax}(\mathbf{o})$:

$$\mathbf{d}_i = \frac{\exp(\mathbf{o}_i)}{\sum_{j=1}^{C} \exp(\mathbf{o}_j)} \,. \tag{5}$$

- The $\mathrm{softmax}$ operation is a soft version of the operation "maximum value of $\mathbf{o}$ is changed to 1, the other values are changed to 0".
- The $\mathrm{softmax}$ operation transforms $\mathbf{o}$ into a distribution $\mathbf{d}$ over the C classes:

$$\begin{aligned} &\forall i\, \mathbf{d}_i \in [0,1]\,; \\ &\sum_{i=1}^{C} \mathbf{d}_i = 1\,. \end{aligned} \tag{6}$$

Classification Problems (2)

Training set: $\{(\mathbf{x}_i, c_i)\}_{i=1..N}$, where $c_i = [1; C]$ is the expected class index, among C possible classes.

Take: $p_{\mathsf{model}}(c \mid \mathbf{o}(\mathbf{x}; \Theta)) = \mathbf{d}_c$ with $\mathbf{d} = \mathrm{softmax}(\mathbf{o}(\mathbf{x}; \Theta))$.

Loss function:

$$\begin{aligned} \mathcal{L}(\Theta) &= -\log \prod_{i=1}^{N} p_{\mathsf{model}}(c_i \mid \mathbf{o}(\mathbf{x}_i; \Theta)) \\ &= \sum_{i=1}^{N} -\log \mathrm{softmax}(\mathbf{o}(\mathbf{x}_i; \Theta))_{c_i} \end{aligned} \tag{7}$$

Be Careful with Predicted Distributions!

$p_{\text{model}}(c \mid \mathbf{o}(\mathbf{x};\Theta)) = \mathbf{d}_c$ with $\mathbf{d} = \mathrm{softmax}(\mathbf{o}(\mathbf{x};\Theta))$.

$\mathbf{d}$ is not a reliable distribution. For example, if the input $\mathbf{x}$ does not belong to any class, we would expect a uniform distribution for $\mathbf{d}$, which is not the case in practice.

M. Hein, M. Andriushchenko, and J. Bitterwolf. "Why ReLU networks yield high-confidence predictions far away from the training data and how to mitigate the problem". In: *CVPR*. 2019.

- Loss functions;
- Generalization and Overfitting;

Generalization

We want to perform well on future, unseen data.

We minimize (in the case of MSE):

$$\sum_{i=1}^{N} \|\mathbf{e}_i - \mathbf{o}(\mathbf{x}_i;\Theta)\|^2 \,, \tag{8}$$

on the training data, but what we would like to minimize is

$$\sum_{(\mathbf{x},\mathbf{e})\in\mathcal{T}_T} \|\mathbf{e} - \mathbf{o}(\mathbf{x};\Theta)\|^2 \,, \tag{9}$$

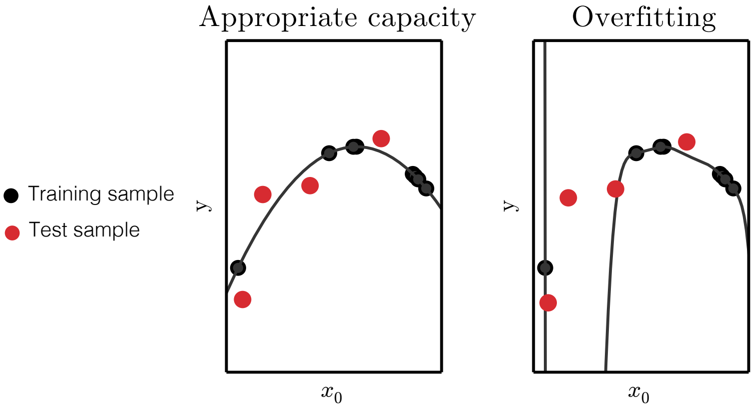
where $\mathcal{T}_T$ is the set of any other data.

In practice, a test set $\mathcal{T}_T$ is used to evaluate the performance of the network. ($\mathcal{T}_T$ should not be used during training!)

Generalization and Overfitting

The loss function can be small on the training data, but large on the test data:

$\rightarrow$ Overfitting.

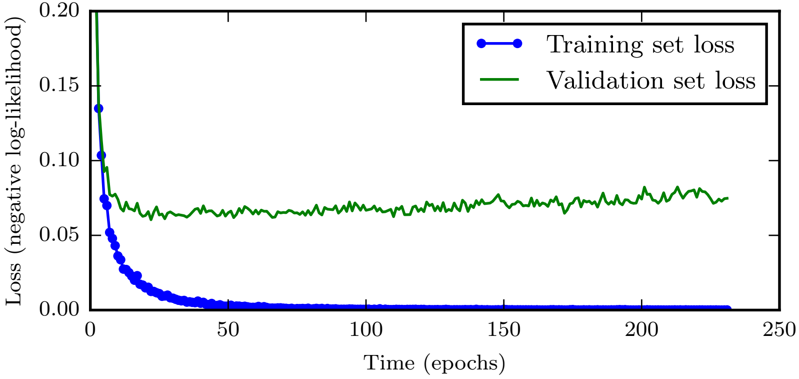


Fitting a polynomial of degree 2

Fitting a polynomial of degree 9

Preventing Overfitting

Use a *validation set*: Monitor the loss on the validation set during training, stop when it starts increasing:



Regularization

For example:

$$\begin{cases} \mathbf{h}(\mathbf{x}) = g(\mathbf{W}\mathbf{x} + \mathbf{b}) \\ \mathbf{o}(\mathbf{x}) = \mathbf{W}_2^\top \mathbf{h}(\mathbf{x}) + \mathbf{b}_2 \end{cases}$$

$$\begin{aligned} &\mathcal{L}(\Theta) = \textstyle\sum_{i=1}^N \|\mathbf{e}_i - \mathbf{o}(\mathbf{x}_i;\Theta)\| \\ &\rightarrow \mathcal{L}(\Theta) = \textstyle\sum_{i=1}^N \|\mathbf{e}_i - \mathbf{o}(\mathbf{x}_i;\Theta)\| + \lambda\|\mathbf{W}\|^2 \,. \end{aligned} \tag{10}$$

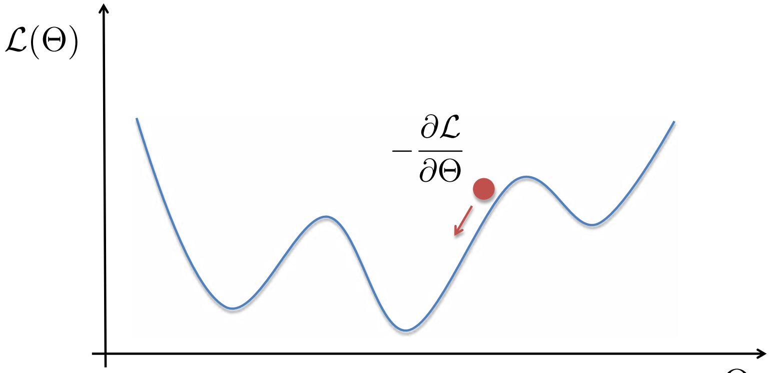
See also Slides on optimization techniques.

- Loss functions;
- Generalization and Overfitting;
- Optimization for Deep Learning;

Optimization for Deep Models

Variants of Gradient Descent:

$$\Theta \leftarrow \Theta - \lambda \frac{\partial \mathcal{L}(\Theta)}{\partial \Theta} \,. \tag{11}$$



Gradient Descent

Variants of Gradient Descent:

$$\Theta \leftarrow \Theta - \lambda \frac{\partial \mathcal{L}(\Theta)}{\partial \Theta} \, . \tag{12}$$

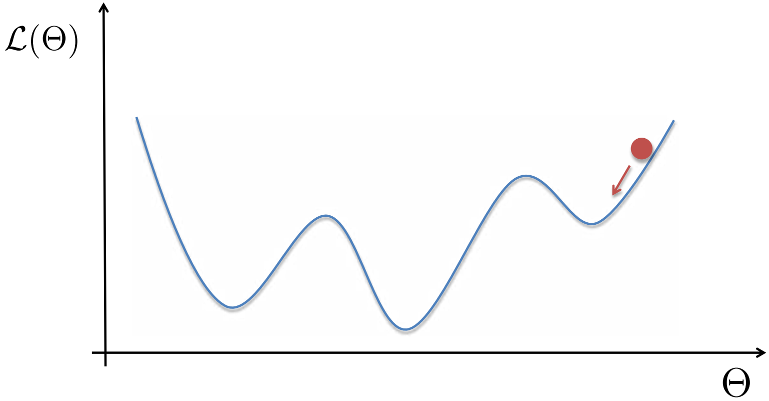
More sophisticated algorithms require computing the Hessian (or an approximation), and/or its inverse (or an approximation).

Many gradient descent iterations can be performed when these algorithms perform a single iteration.

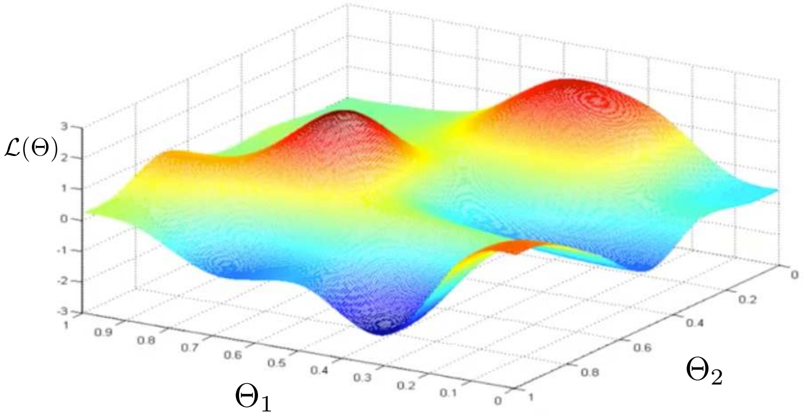
$\rightarrow$ Gradient descent still converges faster.

Léon Bottou. "Large-Scale Machine Learning with Stochastic Gradient Descent". In: *International Conference on Computational Statistics*. 2010.

Local Minima?



A Function in 2D



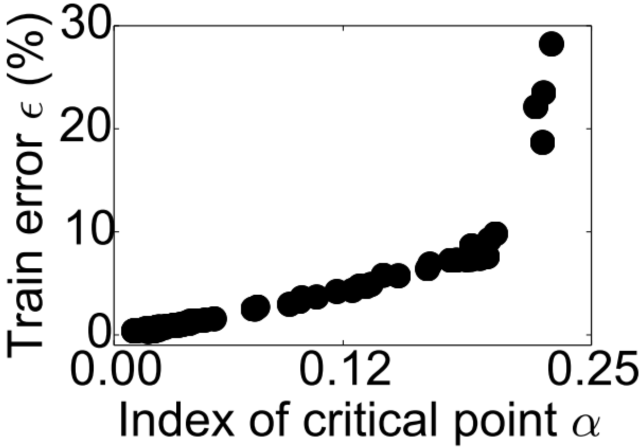
Not Many Local Minima in Large Spaces

For Θ^* to be a local minimum, we need:

- $\left\| \frac{\partial \mathcal{L}}{\partial \Theta}(\Theta^*) \right\| = 0$, and
- all the eigenvalues of $\left(\frac{\partial^2 \mathcal{L}}{\partial \Theta^2}(\Theta^*) \right)$ are positive.

For random functions in n dimensions, the probability for all the eigenvalues to be all negative is $1/n$.

"Almost Local Minima" Seem to Be Good Minima!



α: Proportion of negative eigenvalues.

I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. 2014.

- Loss functions;
- Generalization and Overfitting;
- Optimization for Deep Learning;
- Computing the gradient efficiently: Back-Propagation;

Computing the Gradient of the Loss Function: Example

Case of a 3-layer network, and least-squares loss:

$$\mathcal{L}(\Theta) = \sum_{(\mathbf{x}_0,\mathbf{e})\in\mathcal{T}} L\left(o(\mathbf{x}_0;\Theta),\mathbf{e}\right). \tag{13}$$

$$\begin{cases} \bar{\mathbf{x}}_1 = \mathbf{W}_1\mathbf{x}_0 \\ \mathbf{x}_1 = g(\bar{\mathbf{x}}_1) \\ \bar{\mathbf{x}}_2 = \mathbf{W}_2\mathbf{x}_1 \\ \mathbf{x}_2 = g(\bar{\mathbf{x}}_2) \\ \mathbf{x}_3 = \mathbf{W}_3\mathbf{x}_2 = o(\mathbf{x}_0;\Theta) \\ L(o(\mathbf{x}_0;\Theta),\mathbf{e}) = \frac{1}{2}\|\mathbf{x}_3-\mathbf{e}\|^2 \end{cases} \tag{14}$$

$$\Theta = \left[\mathbf{W}_1,\mathbf{W}_2,\mathbf{W}_3\right]. \tag{15}$$

We would like to compute:

$$\left(\frac{\partial\mathcal{L}}{\partial\Theta}\right) \text{ i.e. } \left(\frac{\partial L}{\partial\mathbf{W}_1}\right)(o(\mathbf{x}_0;\Theta),\mathbf{e}) \text{ , } \left(\frac{\partial L}{\partial\mathbf{W}_2}\right)(..) \text{ , and } \left(\frac{\partial L}{\partial\mathbf{W}_3}\right)(..) \text{ .} \tag{16}$$

Computing $\left(\frac{\partial L}{\partial \mathbf{W}_3}\right)$

$$\begin{aligned} L &= \frac{1}{2}\|\mathbf{x}_3 - \mathbf{e}\|^2 \\ &= \frac{1}{2}\sum_i (\mathbf{x}_3^i - \mathbf{e}^i)^2 \\ &= \frac{1}{2}\sum_i \left(\sum_j \mathbf{W}_3^{ij}\mathbf{x}_2^j - \mathbf{e}^i\right)^2 \text{ because } \mathbf{x}_3^i = \sum_j \mathbf{W}_3^{ij}\mathbf{x}_2^j\,. \end{aligned} \tag{17}$$

Computing $\left(\frac{\partial L}{\partial \mathbf{W}_3}\right)$ (2)

$$L \quad = \frac{1}{2}\sum_i \left(\sum_j \mathbf{W}_3^{ij}\mathbf{x}_2^j - \mathbf{e}^i\right)^2 \tag{18}$$

Thus:

$$\frac{\partial L}{\partial \mathbf{W}_3^{kl}} = \sum_i \left(\sum_j \mathbf{W}_3^{ij}\mathbf{x}_2^j - \mathbf{e}^i\right)\frac{\partial}{\partial \mathbf{W}_3^{kl}}\left(\sum_j \mathbf{W}_3^{ij}\mathbf{x}_2^j - \mathbf{e}^i\right) \tag{19}$$

We have:

$$\frac{\partial}{\partial \mathbf{W}_3^{kl}}\left(\sum_j \mathbf{W}_3^{ij}\mathbf{x}_2^j - \mathbf{e}^i\right) = \left(\sum_j \frac{\partial \mathbf{W}_3^{ij}}{\partial \mathbf{W}_3^{kl}}\mathbf{x}_2^j\right) \tag{20}$$

$$\frac{\partial \mathbf{W}_3^{ij}}{\partial \mathbf{W}_3^{kl}} = \begin{cases} 1 \text{ if } i = k \text{ and } j = l \text{ , and} \\ 0 \text{ otherwise.} \end{cases} \tag{21}$$

Computing $\left(\frac{\partial L}{\partial \mathbf{W}_3}\right)$ (3)

We have:

$$\frac{\partial}{\partial \mathbf{W}_3^{kl}}\left(\sum_j \mathbf{W}_3^{ij}\mathbf{x}_2^j - \mathbf{e}^i\right) = \left(\sum_j \frac{\partial \mathbf{W}_3^{ij}}{\partial \mathbf{W}_3^{kl}}\mathbf{x}_2^j\right) \tag{22}$$

$$\rightarrow \frac{\partial \mathbf{W}_3^{ij}}{\partial \mathbf{W}_3^{kl}} = \begin{cases} 1 \text{ if } i = k \text{ and } j = l \text{ , and} \\ 0 \text{ otherwise.} \end{cases} \tag{23}$$

$$\rightarrow \left(\sum_j \frac{\partial \mathbf{W}_3^{ij}}{\partial \mathbf{W}_3^{kl}}\mathbf{x}_2^j\right) = \begin{cases} \mathbf{x}_2^l & \text{if } k = i\text{, and} \\ 0 & \text{otherwise} \end{cases} \tag{24}$$

Computing $\left(\frac{\partial L}{\partial \mathbf{W}_3}\right)$ (4)

$$\frac{\partial L}{\partial \mathbf{W}_3^{kl}} = \sum_i \left(\sum_j \mathbf{W}_3^{ij}\mathbf{x}_2^j - \mathbf{e}^i\right) \frac{\partial}{\partial \mathbf{W}_3^{kl}} \left(\sum_j \mathbf{W}_3^{ij}\mathbf{x}_2^j - \mathbf{e}^i\right) \tag{25}$$

$$\left(\sum_j \mathbf{W}_3^{ij}\mathbf{x}_2^j - \mathbf{e}^i\right) = \mathbf{x}_3^i - \mathbf{e}^i \tag{26}$$

$$\left(\sum_j \frac{\partial \mathbf{W}_3^{ij}}{\partial \mathbf{W}_3^{kl}}\mathbf{x}_2^j\right) = \begin{cases} \mathbf{x}_2^l & \text{if } k = i, \text{ and} \\ 0 & \text{otherwise} \end{cases} \tag{27}$$

$$\rightarrow \frac{\partial L}{\partial \mathbf{W}_3^{kl}} = (\mathbf{x}_3^k - \mathbf{e}^k)\mathbf{x}_2^l \tag{28}$$

Computing $\left(\frac{\partial L}{\partial \mathbf{W}_3}\right)$ (5)

$$\frac{\partial L}{\partial \mathbf{W}_3^{kl}} = (\mathbf{x}_3^k - \mathbf{e}^k)\mathbf{x}_2^l \tag{29}$$

$$\rightarrow \frac{\partial L}{\partial \mathbf{W}_3} = (\mathbf{x}_3 - \mathbf{e})\mathbf{x}_2^\top = \boldsymbol{\delta}_3\mathbf{x}_2^\top \tag{30}$$

Computing $\left(\frac{\partial L}{\partial \mathbf{W}_2}\right)$

$$L \quad = \frac{1}{2}\sum_i \left(\sum_j \mathbf{W}_3^{ij}\mathbf{x}_2^j - \mathbf{e}^i\right)^2 \tag{31}$$

$$\frac{\partial L}{\partial \mathbf{W}_2^{kl}} = \sum_i \left(\sum_j \mathbf{W}_3^{ij}\mathbf{x}_2^j - \mathbf{e}^i\right) \frac{\partial}{\partial \mathbf{W}_2^{kl}} \left(\sum_j \mathbf{W}_3^{ij}\mathbf{x}_2^j - \mathbf{e}^i\right) \tag{32}$$

$$\frac{\partial}{\partial \mathbf{W}_2^{kl}} \left(\sum_j \mathbf{W}_3^{ij}\mathbf{x}_2^j - \mathbf{e}^i\right) = \left(\sum_j \mathbf{W}_3^{ij} \frac{\partial \mathbf{x}_2^j}{\partial \mathbf{W}_2^{kl}}\right) \tag{33}$$

$$\frac{\partial \mathbf{x}_2^j}{\partial \mathbf{W}_2^{kl}}? \tag{34}$$

Computing $\left(\frac{\partial L}{\partial \mathbf{W}_2}\right)$ (2)

$$\begin{cases} \bar{\mathbf{x}}_2 = \mathbf{W}_2 \mathbf{x}_1 \\ \mathbf{x}_2 = g(\bar{\mathbf{x}}_2) \end{cases} \tag{35}$$

$$\begin{aligned} \frac{\partial \mathbf{x}_2^j}{\partial \mathbf{W}_2^{kl}} &= \frac{\partial}{\partial \mathbf{W}_2^{kl}} \left(g(\bar{\mathbf{x}}_2^j) \right) \\ &= g'(\bar{\mathbf{x}}_2^j) \frac{\partial \bar{\mathbf{x}}_2^j}{\partial \mathbf{W}_2^{kl}} \\ &= g'(\bar{\mathbf{x}}_2^j) \frac{\partial}{\partial \mathbf{W}_2^{kl}} \left(\sum_m \mathbf{W}_2^{jm} \mathbf{x}_1^m \right) \end{aligned} \tag{36}$$

$$\frac{\partial}{\partial \mathbf{W}_2^{kl}} \left(\sum_m \mathbf{W}_2^{jm} \mathbf{x}_1^m \right) = \begin{cases} \mathbf{x}_1^l & \text{if } k = j \text{, and} \\ 0 & \text{otherwise} \end{cases} \tag{37}$$

Computing $\left(\frac{\partial L}{\partial \mathbf{W}_2}\right)$ (3)

$$\frac{\partial \mathbf{x}_2^j}{\partial \mathbf{W}_2^{kl}} = \begin{cases} g'(\bar{\mathbf{x}}_2^m)\mathbf{x}_2^{jl} & \text{if } k = j\text{, and} \\ 0 & \text{otherwise.} \end{cases} \tag{38}$$

$$\left(\sum_j \mathbf{W}_3^{ij} \frac{\partial \mathbf{x}_2^j}{\partial \mathbf{W}_2^{kl}}\right) = \mathbf{W}_3^{ik} g'(\bar{\mathbf{x}}_2^k)\mathbf{x}_1^l \tag{39}$$

Computing $\left(\frac{\partial L}{\partial \mathbf{W}_2}\right)$ (4)

$$\frac{\partial L}{\partial \mathbf{W}_2^{kl}} = \sum_i \left(\sum_j \mathbf{W}_3^{ij}\mathbf{x}_2^j - \mathbf{e}^i\right) \frac{\partial}{\partial \mathbf{W}_2^{kl}} \left(\sum_j \mathbf{W}_3^{ij}\mathbf{x}_2^j - \mathbf{e}^i\right) \tag{40}$$

$$\left(\sum_j \mathbf{W}_3^{ij}\mathbf{x}_2^j - \mathbf{e}^i\right) = (\mathbf{x}_3^i - \mathbf{e}^i) \tag{41}$$

$$\frac{\partial}{\partial \mathbf{W}_2^{kl}} \left(\sum_j \mathbf{W}_3^{ij}\mathbf{x}_2^j - \mathbf{e}^i\right) = \mathbf{W}_3^{ik} g'(\bar{\mathbf{x}}_2^k)\mathbf{x}_1^l \tag{42}$$

$$\rightarrow \frac{\partial L}{\partial \mathbf{W}_2^{kl}} = \sum_i \left(\mathbf{x}_3^i - \mathbf{e}^i\right) \mathbf{W}_3^{ik} g'(\bar{\mathbf{x}}_2^k)\mathbf{x}_1^l \tag{43}$$

Computing $\left(\frac{\partial L}{\partial \mathbf{W}_2}\right)$ (5)

$$\frac{\partial L}{\partial \mathbf{W}_2^{kl}} = \sum_i \left(\mathbf{x}_3^i - \mathbf{e}^i\right) \mathbf{W}_3^{ik} g'(\bar{\mathbf{x}}_2^k)\mathbf{x}_1^l \tag{44}$$

$$\begin{aligned} & \frac{\partial L}{\partial \mathbf{W}_2} && = \left(\mathbf{W}_3^\top (\mathbf{x}_3 - \mathbf{e}) \odot g'(\bar{\mathbf{x}}_2)\right) \mathbf{x}_1^\top \\ \rightarrow & && = \left(\mathbf{W}_3^\top \boldsymbol{\delta}_3 \odot g'(\bar{\mathbf{x}}_2)\right) \mathbf{x}_1^\top \\ & && = \boldsymbol{\delta}_2 \mathbf{x}_1^\top \end{aligned} \tag{45}$$

Computing $\left(\frac{\partial L}{\partial \mathbf{W}_1}\right)$

$$\begin{aligned}\frac{\partial L}{\partial \mathbf{W}_1} &= \left(\mathbf{W}_2^\top \boldsymbol{\delta}_2 \odot g'(\bar{\mathbf{x}}_1)\right)\mathbf{x}_0^\top \\ &= \boldsymbol{\delta}_1 \mathbf{x}_0^\top\end{aligned} \tag{46}$$

Back-Propagation

$$\left\downarrow\begin{aligned}
\bar{\mathbf{x}}_1 &= \mathbf{W}_1\mathbf{x}_0 \\
\mathbf{x}_1 &= g(\bar{\mathbf{x}}_1) \\
\bar{\mathbf{x}}_2 &= \mathbf{W}_2\mathbf{x}_1 \\
\mathbf{x}_2 &= g(\bar{\mathbf{x}}_2) \\
\mathbf{x}_3 &= \mathbf{W}_3\mathbf{x}_2 = o(\mathbf{x}_0;\Theta) \\
L(o(\mathbf{x}_0;\Theta),\mathbf{e}) &= \tfrac{1}{2}\|\mathbf{x}_3-\mathbf{e}\|^2
\end{aligned}\right.
\qquad
\left\uparrow\begin{aligned}
\frac{\partial L}{\partial \mathbf{W}_1} &= \boldsymbol{\delta}_1\mathbf{x}_0^\top \\
\boldsymbol{\delta}_1 &= \mathbf{W}_2^\top\boldsymbol{\delta}_2 \odot g'(\bar{x}_1) \\
\frac{\partial L}{\partial \mathbf{W}_2} &= \boldsymbol{\delta}_2\mathbf{x}_1^\top \\
\boldsymbol{\delta}_2 &= \mathbf{W}_3^\top\boldsymbol{\delta}_3 \odot g'(\bar{x}_2) \\
\frac{\partial L}{\partial \mathbf{W}_3} &= \boldsymbol{\delta}_3\mathbf{x}_2^\top \\
\boldsymbol{\delta}_3 &= \mathbf{x}_3-\mathbf{e}
\end{aligned}\right. \tag{47}$$

Juergen Schmidhuber. "Deep Learning in Neural Networks: An Overview". In: *arXiv Preprint* (2014).

Vanishing or Exploding Gradients

$$
\begin{aligned}
& \frac{\partial L}{\partial \mathbf{W}_1} \\
= \quad & \boldsymbol{\delta}_1 \mathbf{x}_0^\top \\
= \quad & \left(\mathbf{W}_2^\top \boldsymbol{\delta}_2 \odot g'(\bar{x}_1) \right) \mathbf{x}_0 \\
= \quad & \left(\mathbf{W}_2^\top \left(\mathbf{W}_3^\top \boldsymbol{\delta}_3 \odot g'(\bar{x}_2) \right) \odot g'(\bar{x}_1) \right) \mathbf{x}_0 \\
= & \left(\mathbf{W}_2^\top \left(\mathbf{W}_3^\top (\mathbf{x}_3 - \mathbf{e}) \odot g'(\bar{x}_2) \right) \odot g'(\bar{x}_1) \right) \mathbf{x}_0
\end{aligned}
\tag{48}
$$

Vanishing or exploding gradients happen if the coefficients of the $\mathbf{W}_i$, the $\mathbf{x}_i$, the $g'(\bar{x}_i)$ become much larger or much smaller than 1:

- Loss functions;
- Generalization and Overfitting;
- Optimization for Deep Learning;
- Back-Propagation;
- Automatic Differentiation;

Automatic Differentiation

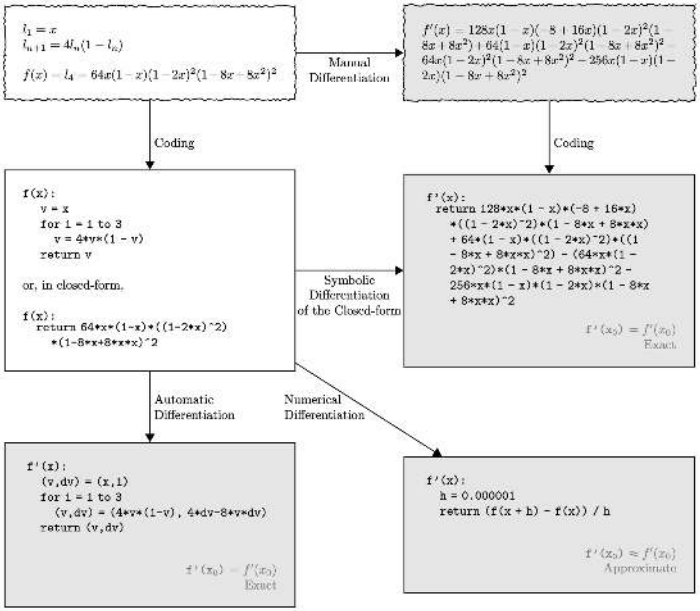
How do we compute the derivatives in practice?

- manually working out them, and coding them;
- numerical differentiation using finite difference approximation;
- symbolic differentiation (exists in Mathematica, Mapple, etc.);
- *automatic differentiation* (autodiff).

Implemented in PyTorch, and in the `tf.GradientTape` API in TensorFlow 2.

A. G. Baydin et al. “Automatic Differentiation in Machine Learning: A Survey”. In: *JMLR* (2018).

Automatic Differentiation



$$l_1 = x$$
$$l_{n+1} = 4l_n(1 - l_n)$$
$$f(x) = l_4 = 64x(1-x)(1-2x)^2(1-8x+8x^2)^2$$

$$f'(x) = 128x(1-x)(-8+16x)(1-2x)^2(1-8x+8x^2) + 64(1-x)(1-2x)^2(1-8x+8x^2)^2 - 64x(1-2x)^2(1-8x+8x^2)^2 - 256x(1-x)(1-2x)(1-8x+8x^2)^2$$

```
f(x):
   v = x
   for i = 1 to 3
     v = 4*v*(1 - v)
   return v
```

or, in closed-form,

```
f(x):
  return 64*x*(1-x)*((1-2*x)^2)
    *(1-8*x+8*x*x)^2
```

```
f'(x):
  return 128*x*(1 - x)*(-8 + 16*x)
    *((1 - 2*x)^2)*(1 - 8*x + 8*x*x)
    + 64*(1 - x)*((1 - 2*x)^2)*((1
    - 8*x + 8*x*x)^2) - (64*x*(1 -
    2*x)^2)*(1 - 8*x + 8*x*x)^2 -
    256*x*(1 - x)*(1 - 2*x)*(1 - 8*x
    + 8*x*x)^2
```

f'(x₀) = $f'(x_0)$
Exact

```
f'(x):
  (v,dv) = (x,1)
  for i = 1 to 3
    (v,dv) = (4*v*(1-v), 4*dv-8*v*dv)
  return (v,dv)
```

f'(x₀) = $f'(x_0)$
Exact

```
f'(x):
  h = 0.000001
  return (f(x + h) - f(x)) / h
```

f'(x₀) ≈ $f'(x_0)$
Approximate

Manual Differentiation
Coding
Coding
Symbolic Differentiation of the Closed-form
Automatic Differentiation
Numerical Differentiation

Automatic Differentiation

- Autodiff can differentiate computer program functions (even with branching, loops, and recursion).
- This is because a program function executes a sequence of elementary arithmetic operations (addition, subtraction, multiplication, division, etc.) and elementary functions (exp, log, sin, cos, etc.). By applying the chain rule repeatedly to these operations, derivatives of arbitrary order can be computed automatically.
- Accurate evaluation of derivatives at machine precision with only a small constant factor of overhead.

Case of Symbolic Differentiation

Consider for example:

$$\begin{aligned} h(x) &= f(x)g(x) \\ \frac{d}{dx}(h(x)) &= \left(\frac{d}{dx}f(x)\right)g(x) + f(x)\left(\frac{d}{dx}g(x)\right) \end{aligned} \tag{49}$$

Waste operations when computing both $h(x)$ and $\frac{d}{dx}(h(x))$.

Basis of autodiff: Apply symbolic differentiation at the elementary operation level and keep intermediate numerical results, in lockstep with the evaluation of the main function.

Example

Consider $f(x_1, x_2) = \ln(x_1) + x_1 x_2 - \sin(x_2)$.

Computation graph for $f(x_1, x_2)$:

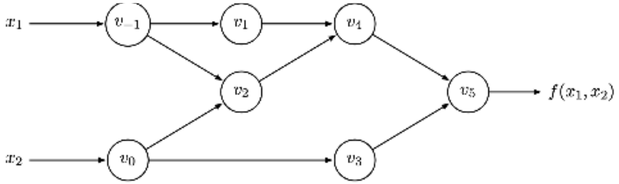
$$\begin{tikzcd}
x_1 \arrow[r] & v_{-1} \arrow[r] \arrow[dr] & v_1 \arrow[r] & v_4 \arrow[dr] & & \\
& & v_2 \arrow[ur] & & v_5 \arrow[r] & f(x_1, x_2) \\
x_2 \arrow[r] & v_0 \arrow[ur] \arrow[rr] & & v_3 \arrow[ur] & &
\end{tikzcd}$$

with

$$\begin{aligned}
v_{-1} &= x_1 \\
v_0 &= x_2 \\
v_1 &= \ln v{-}1 \\
v_2 &= v_{-1} v_0 \\
v_3 &= \sin v_0 \\
v_4 &= v_1 + v_2 \\
v_5 &= v_4 - v_3 \\
f(x_1, x_2) &= v_5
\end{aligned} \tag{50}$$

Tangent Variables

$$f(x_1, x_2) = \ln(x_1) + x_1 x_2 - \sin(x_2).$$



For computing the derivative of f with respect to x_1, introduce the tangent variables:

$$\dot{v}_i = \frac{\partial v_i}{\partial x_1}\,. \tag{51}$$

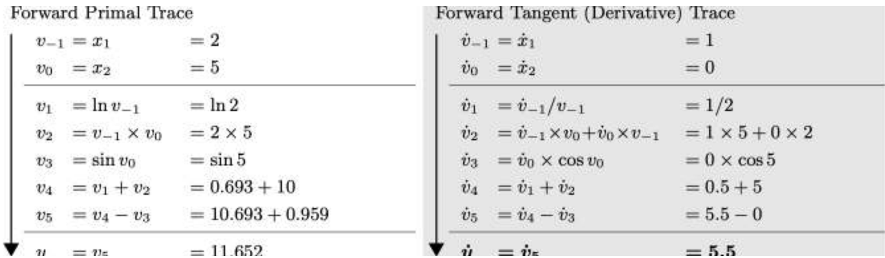
Derivative of f with Respect to x_1

For computing the derivative of f with respect to x_1, introduce the tangent variables:

$$\dot{v}_i = \frac{\partial v_i}{\partial x_1} \,. \tag{52}$$

First, set $\dot{v}_{-1} = \dot{x}_1 \leftarrow 1$ and $\dot{v}_0 = \dot{x}_2 \leftarrow 0$.

Then, compute $\dot{v}_5 = \frac{\partial f}{\partial x_1}$:



Forward Primal Trace

v_{-1}	$= x_1$	$= 2$
v_0	$= x_2$	$= 5$
v_1	$= \ln v_{-1}$	$= \ln 2$
v_2	$= v_{-1} \times v_0$	$= 2 \times 5$
v_3	$= \sin v_0$	$= \sin 5$
v_4	$= v_1 + v_2$	$= 0.693 + 10$
v_5	$= v_4 - v_3$	$= 10.693 + 0.959$
y	$= v_5$	$= 11.652$

Forward Tangent (Derivative) Trace

$\dot{v}_{-1}$	$= \dot{x}_1$	$= 1$
$\dot{v}_0$	$= \dot{x}_2$	$= 0$
$\dot{v}_1$	$= \dot{v}_{-1}/v_{-1}$	$= 1/2$
$\dot{v}_2$	$= \dot{v}_{-1} \times v_0 + \dot{v}_0 \times v_{-1}$	$= 1 \times 5 + 0 \times 2$
$\dot{v}_3$	$= \dot{v}_0 \times \cos v_0$	$= 0 \times \cos 5$
$\dot{v}_4$	$= \dot{v}_1 + \dot{v}_2$	$= 0.5 + 5$
$\dot{v}_5$	$= \dot{v}_4 - \dot{v}_3$	$= 5.5 - 0$
$\dot{y}$	$= \dot{v}_5$	$= 5.5$

Dual Numbers

Defined as $v + \dot{v}\epsilon$, with $\epsilon \neq 0$ and $\epsilon^2 = 0$.

Then:

$$
\begin{aligned}
&(v + \dot{v}\epsilon) + (u + \dot{u}\epsilon) = (v + u) + (\dot{v} + \dot{u})\epsilon\,; \\
&(v + \dot{v}\epsilon)(u + \dot{u}\epsilon) = (vu) + (v\dot{u} + \dot{v}u)\epsilon\,; \\
&f(v + \dot{v}\epsilon) = f(v) + f'(v)\dot{v}\epsilon \text{ (* - from Taylor expansion);} \\
&f(g(v + \dot{v}\epsilon)) = f(g(v)) + f'(g(v))g'(v)\dot{v}\epsilon \text{ (by applying * twice .)}
\end{aligned}
\tag{53}
$$

$\rightarrow$ We can compute the derivative of a function $f(x)$ by computing:

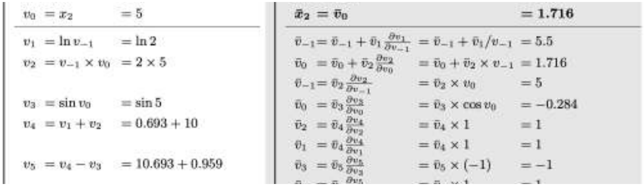
$$
\left.\frac{df(x)}{dx}\right|_{x=v} = [\text{coefficient of } \epsilon](f(v + 1\epsilon))\,. \tag{54}
$$

Reverse Accumulation Mode

The previous method is not efficient in the case of $f : \mathbb{R}^n \to \mathbb{R}^m$ with $n \gg m$ (as usually in Machine Learning).

Introduce 'adjoint':

$$\bar{v_i} = \frac{\partial y_j}{\partial v_i} \, . \tag{55}$$



v_0	$= x_2$	$= 5$
v_1	$= \ln v_{-1}$	$= \ln 2$
v_2	$= v_{-1} \times v_0$	$= 2 \times 5$
v_3	$= \sin v_0$	$= \sin 5$
v_4	$= v_1 + v_2$	$= 0.693 + 10$
v_5	$= v_4 - v_3$	$= 10.693 + 0.959$

$\bar{\boldsymbol{x}}_2$	$= \bar{v}_0$		$= \mathbf{1.716}$
$\bar{v}_{-1}$	$= \bar{v}_{-1} + \bar{v}_1 \frac{\partial v_1}{\partial v_{-1}}$	$= \bar{v}_{-1} + \bar{v}_1 / v_{-1}$	$= 5.5$
$\bar{v}_0$	$= \bar{v}_0 + \bar{v}_2 \frac{\partial v_2}{\partial v_0}$	$= \bar{v}_0 + \bar{v}_2 \times v_{-1}$	$= 1.716$
$\bar{v}_{-1}$	$= \bar{v}_2 \frac{\partial v_2}{\partial v_{-1}}$	$= \bar{v}_2 \times v_0$	$= 5$
$\bar{v}_0$	$= \bar{v}_3 \frac{\partial v_3}{\partial v_0}$	$= \bar{v}_3 \times \cos v_0$	$= -0.284$
$\bar{v}_2$	$= \bar{v}_4 \frac{\partial v_4}{\partial v_2}$	$= \bar{v}_4 \times 1$	$= 1$
$\bar{v}_1$	$= \bar{v}_4 \frac{\partial v_4}{\partial v_1}$	$= \bar{v}_4 \times 1$	$= 1$
$\bar{v}_3$	$= \bar{v}_5 \frac{\partial v_5}{\partial v_3}$	$= \bar{v}_5 \times (-1)$	$= -1$

The 2 derivatives can be computed in 1 pass, instead of 2 for the forward mode.

- Loss functions;
- Generalization and Overfitting;
- Optimization for Deep Learning;
- Back-Propagation;
- Automatic Differentiation;
- Optimization Algorithms;

Stochastic Gradient Descent (SGD)

Loss function:

$$\mathcal{L}(\Theta) = \sum_i L\left(o(\mathbf{x}_i; \Theta), \mathbf{e}_i\right) . \tag{56}$$

$$\Theta^{t+1} \leftarrow \Theta^t - \eta_t G(\Theta^t) , \tag{57}$$

where

$$G(\Theta^t) = \frac{\partial}{\partial \Theta} L\left(o(\mathbf{x}_{i_t}; \Theta_t), \mathbf{e}_{i_t}\right) . \tag{58}$$

Can converge despite the fact that $\mathcal{L}(\Theta)$ is not convex.

Stochastic Gradient Descent (SGD)

$$\Theta^{t+1} \leftarrow \Theta^t - \eta_t G(\Theta^t)\,. \tag{59}$$

Converges if the network is sufficiently overparameterized:

Theorem. Let $(\mathbf{x}_i, e_i)_{1\le i\le n}$ be a training set satisfying $\min_{i,j:i\neq j} \|\mathbf{x}_i - \mathbf{x}_j\|_2 > \delta > 0$. Consider fitting the data using a feed-forward neural network with ReLU activations. Denote by D (resp. W) the depth (resp. width) of the network. Suppose that the neural network is sufficiently over-parametrized, i.e.,

$$W \gg \text{polynomial}(n, D, \frac{1}{\delta})\,. \tag{60}$$

Then, with high probability, running SGD with *some random initialization* and properly chosen step sizes η_t yields $\mathcal{L}(\Theta^t) < \epsilon$ in $t \propto \log \frac{1}{\epsilon}$.

Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song. “A convergence theory for deep learning via over- parameterization”. In: *arXiv Preprint*. 2018.

Mini-Batch SGD

Loss function:

$$\mathcal{L}(\Theta) = \sum_i L\left(o(\mathbf{x}_i; \Theta), \mathbf{e}_i\right) . \tag{61}$$

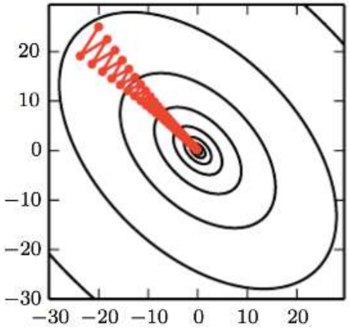
$$\Theta^{t+1} \leftarrow \Theta^t - \eta_t G(\Theta^t) , \tag{62}$$

where

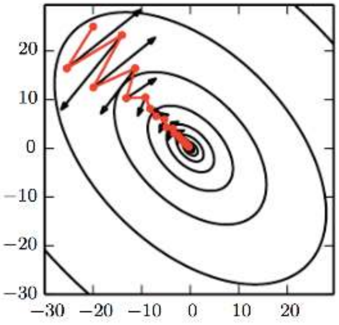
$$G(\Theta^t) = \frac{1}{K} \sum_{k=1}^{K} \frac{\partial}{\partial \Theta} L\Big(o(\mathbf{x}_{i_t^k}; \Theta_t), \mathbf{e}_{i_t^k}\Big) . \tag{63}$$

Momentum-Based SGD

$$\begin{cases} \mathbf{g}^t \leftarrow \rho \mathbf{g}^{t-1} + (1-\rho) G(\Theta^t) \\ \Delta\Theta^t \leftarrow -\eta_t \mathbf{g}^t \\ \Theta^{t+1} \leftarrow \Theta^t + \Delta\Theta^t \end{cases} \tag{64}$$



without momentum



with momentum

SGD with Adaptive Learning Rates

Preconditioning:

$$\Theta^{t+1} \leftarrow \Theta^{t} - \eta_t \mathbf{P}_t^{-1} G(\Theta^t)\,, \tag{65}$$

AdaGrad: Parameters with largest partial derivatives should have a rapid decrease.

$$\mathbf{P}_t = \left\{ \operatorname{diag}\left(\sum_{j=0}^{t} G(\Theta^t) G(\Theta^t)^\top \right) \right\}^{1/2} . \tag{66}$$

RMSProp: Introduces momentum when computing the preconditioner.

$$\mathbf{P}_t = \left\{ \operatorname{diag}\left(\alpha \mathbf{P}_{t-1} + \sum_{j=0}^{t} G(\Theta^t) G(\Theta^t)^\top \right) \right\}^{1/2} . \tag{67}$$

AdaGrad

AdaGrad: Parameters with largest partial derivatives should have a rapid decrease.

$$\begin{cases} \mathbf{g}^t \leftarrow G(\Theta^t) \\ \mathbf{r}^t \leftarrow \mathbf{r}^{t-1} + \mathbf{g}^t \odot \mathbf{g}^t \\ \Delta\Theta^t \leftarrow -\frac{\lambda}{\delta + \sqrt{\mathbf{r}^t}} \odot \mathbf{g}^t \\ \Theta^{t+1} \leftarrow \Theta^t + \Delta\Theta^t \end{cases} \tag{68}$$

John Duchi, Elad Hazan, and Yoram Singer. "Adaptive subgradient methods for online learning and stochastic optimization". In: *JMLR* (2011).

RMSProp

RMSProp: Introduces momentum.

Replace

$$\mathbf{r}^t \leftarrow \mathbf{r}^{t-1} + \mathbf{g}^t \odot \mathbf{g}^t \tag{69}$$

by

$$\mathbf{r}^t \leftarrow \rho \mathbf{r}^{t-1} + (1-\rho)\mathbf{g}^t \odot \mathbf{g}^t \tag{70}$$

$\rightarrow$ Much less influenced by first iterations.Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. “Neural Networks for Machine Learning”. In: *Lecture 6a - Overview of mini-batch gradient descent*. 2012.

Adam

$$\begin{cases} \mathbf{g}^t \leftarrow G(\Theta^t) \\ \mathbf{s}^t \leftarrow \rho_1 \mathbf{s}^{t-1} + (1-\rho_1)\mathbf{g}^t \\ \mathbf{r}^t \leftarrow \rho_2 \mathbf{r}^{t-1} + (1-\rho_2)\mathbf{g}^t \odot \mathbf{g}^t \\ \hat{\mathbf{s}}^t \leftarrow \frac{\mathbf{s}^t}{1-(\rho_1)^t} \\ \hat{\mathbf{r}}^t \leftarrow \frac{\mathbf{r}^t}{1-(\rho_2)^t} \\ \Delta\Theta^t \leftarrow -\frac{\lambda}{\delta+\sqrt{\hat{\mathbf{r}}^t}} \odot \hat{\mathbf{s}}^t \\ \Theta^{t+1} \leftarrow \Theta^t + \Delta\Theta^t \end{cases} \tag{71}$$

with $\epsilon \approx 0.001$, $\rho_1 \approx 0.9$, $\rho_2 \approx 0.999$.

Convergence proof on convex problems, but also works well on deep learning problems.

Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *ICLR*. 2015.

Adam

$\mathbf{s}^t$ is the gradient with momentum:

$$\mathbf{s}^t \leftarrow \rho_1 \mathbf{s}^{t-1} + (1-\rho_1)\mathbf{g}^t \,. \tag{72}$$

$\mathbf{r}^t$ accumulates the gradient norm (with momentum), as in RMSProp:

$$\mathbf{r}^t \leftarrow \rho_2 \mathbf{r}^{t-1} + (1-\rho_2)\mathbf{g}^t \odot \mathbf{g}^t \,. \tag{73}$$

$\hat{s}^t$ is smaller than $\mathbf{s}^t$ but converges towards it ($\rho_1 \approx 0.9$):

$$\hat{\mathbf{s}}^t \leftarrow \frac{\mathbf{s}^t}{1-(\rho_1)^t} \,. \tag{74}$$

$\hat{r}^t$ is smaller than $\mathbf{r}^t$ but converges towards it ($\rho_2 \approx 0.999$):

$$\hat{\mathbf{r}}^t \leftarrow \frac{\mathbf{r}^t}{1-(\rho_2)^t} \,. \tag{75}$$

Update as in RMSProp:

$$\begin{cases} \Delta\Theta^t \leftarrow -\frac{\lambda}{\delta+\sqrt{\hat{\mathbf{r}}^t}} \odot \hat{\mathbf{s}}^t \\ \Theta^{t+1} \leftarrow \Theta^t + \Delta\Theta^t \end{cases} \tag{76}$$

- Loss functions;
- Generalization and Overfitting;
- Optimization for Deep Learning;
- Back-Propagation;
- Automatic Differentiation;
- Optimization Algorithms;
- Optimization Techniques

Optimization Techniques

Xavier Initialization

$$\mathbf{W}_{i,j} \sim \mathsf{Uniform}\left(-\sqrt{\frac{6}{m+n}}, \sqrt{\frac{6}{m+n}}\right), \tag{77}$$

with m the number of inputs and n the number of output of matrix $\mathbf{W}$.

Balances between all the layers to have the same activation variance, and the same gradient variance.

Biases $(\mathbf{b})$ usually initialized to 0.

X. Glorot and Y. Bengio. "Understanding the Difficulty of Training Deep Feedforward Neural Networks". In: *International Conference on Artificial Intelligence and Statistics*. 2010.

Batch Normalization

Normalize distribution of each input feature in each layer across each minibatch to $\mathcal{N}(0, 1)$:

$$\begin{aligned} \mu &\leftarrow \frac{1}{m}\sum_{i=1}^{m}\bar{\mathbf{x}}^i \\ \sigma^2 &\leftarrow \frac{1}{m}\sum_{i=1}^{m}(\bar{\mathbf{x}}^i - \mu)^2 \\ \bar{\mathbf{x}}^i &\leftarrow \frac{\bar{\mathbf{x}}^i - \mu}{\sqrt{\sigma^2 + \epsilon}} \end{aligned} \tag{78}$$

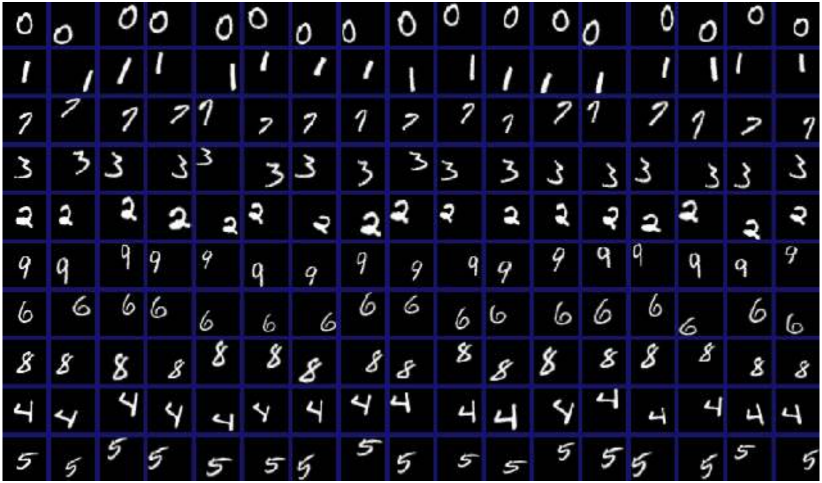
More resilient to parameter scaling.

Prevents exploding or vanishing gradients.

Sergey Ioffe and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *arXiv Preprint.* 2015.

Data Augmentation

Transform the original data, for example apply geometric transformations.



DropOut

Bagging/Ensemble methods: Averaging different models, as different models will usually not make all the same errors on the test set (AdaBoost, Random Forests, etc.).

$$\tilde{o}(\mathbf{x}) = \frac{1}{N}\sum_{i}^{N} o_i(\mathbf{x})\,, \tag{79}$$

where the o_i are different models (classifiers, regressors,..), and $\tilde{o}$ the final model.

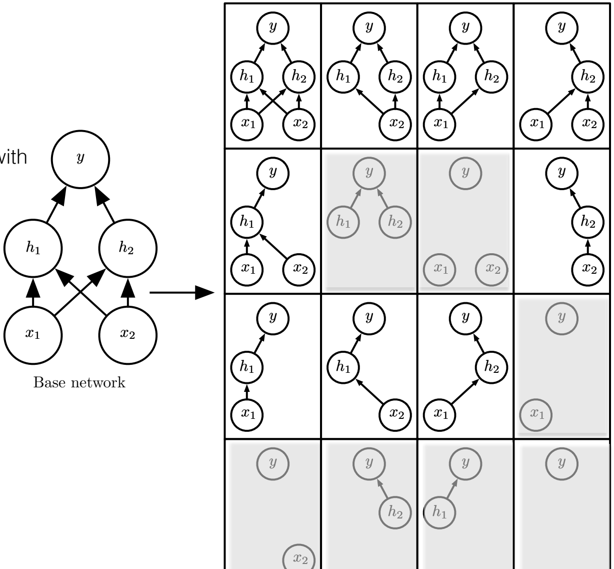
DropOut is an ensemble method that does not need to build the models explicitly.

Nitish Srivastava et al. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. In: *JMLR* (2014).

DropOut

Considers all the networks that can be formed by removing units from a network:

with

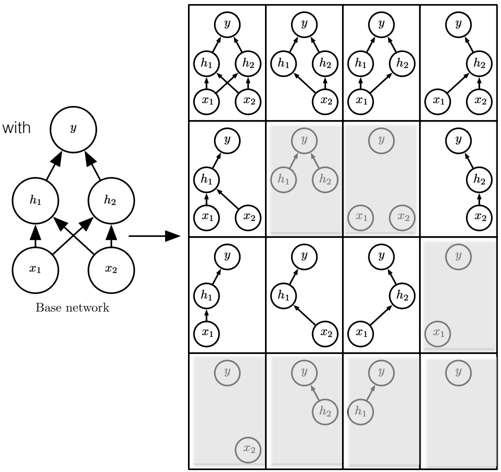


DropOut

At each optimization iteration: random binary masks on the units to consider.

The probability p to remove a unit is a metaparameter.

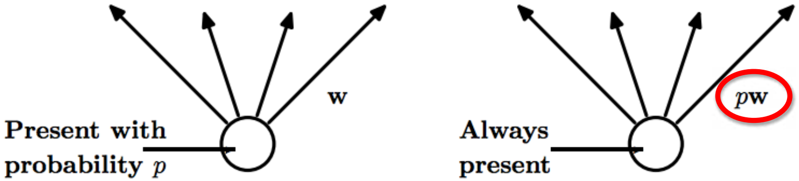
with



Base network

Ensemble of subnetworks

DropOut: Algorithm



(a) At training time (b) At test time

DropOut: Justification

Exact inference:

$$\tilde{o}(\mathbf{x}) = \frac{1}{N}\sum_{i}^{N} o(\mathbf{x}; \mu_i)\,, \tag{80}$$

where the o_i are different models (e.g. classifiers), $\tilde{o}$ the final model, and μ_i is the binary mask.

This is however intractable. DropOut provides an approximation (that works well in practice).

DropOut is exact in the case of linear classification:

$$o(\mathbf{x}) = \mathrm{softmax}(\mathbf{W}\mathbf{x} + \mathbf{b})\,, \tag{81}$$

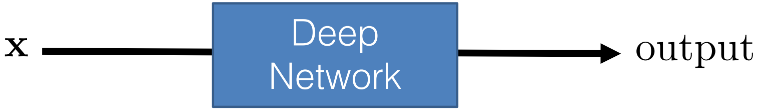
and when using the geometric mean (instead of the arithmetic mean) to average the models):

$$\tilde{o}(\mathbf{x}) = \sqrt[2^n]{\prod_{\mu \in \{0,1\}^n} o(\mathbf{x}; \mu)}\,, \tag{82}$$

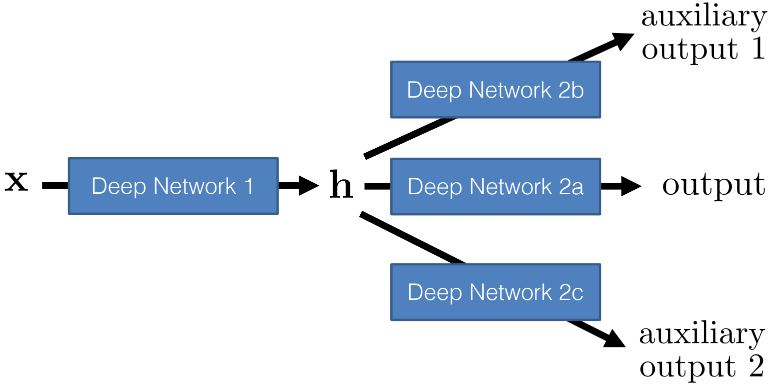
where:

$$o(\mathbf{x}; \mu) = \mathrm{softmax}(\mathbf{W}(\mu \odot \mathbf{x}) + \mathbf{b})\,. \tag{83}$$

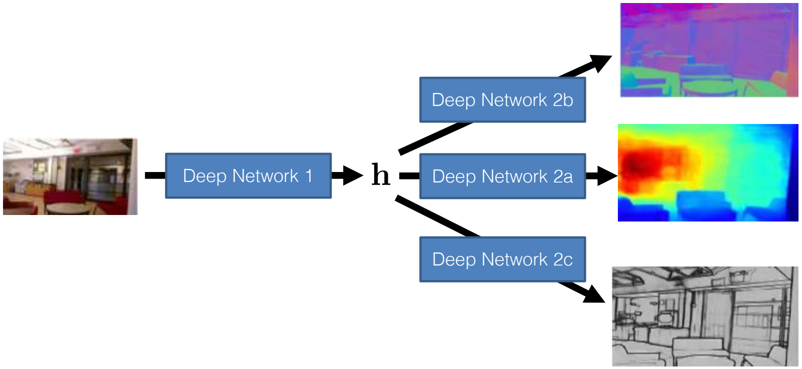
Multi-Task Learning



Multi-Task Learning



Multi-Task Learning



See alsoAmir R. Zamir et al. "Taskonomy: Disentangling Task Transfer Learning". In: *CVPR*. 2018.