Chapter ML:IV (continued)

IV. Neural Networks

- ☐ Perceptron Learning
- ☐ Multilayer Perceptron Basics
- ☐ Multilayer Perceptron with Two Layers
- ☐ Multilayer Perceptron at Arbitrary Depth
- ☐ Advanced MLPs
- ☐ Automatic Gradient Computation

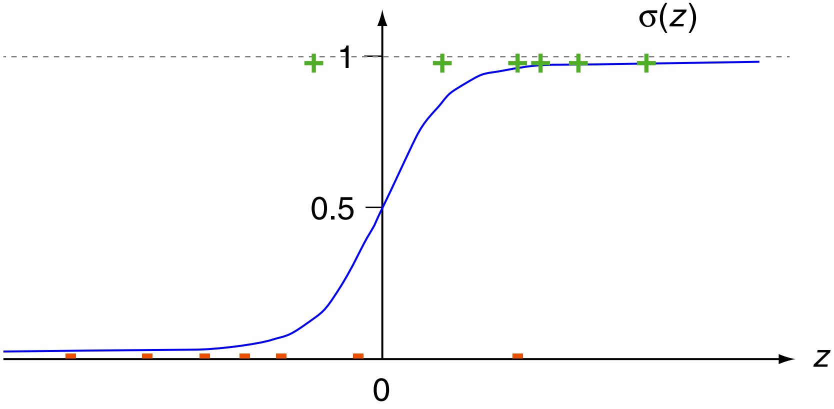
Advanced MLPs

Output Normalization: Softmax

For two classes ($k = 2$), the scalar sigmoid output $\sigma(z)$ determines both class probabilities for $\mathbf{x}$:

- $p(1 \mid \mathbf{x}) := \sigma(z)$
- $p(0 \mid \mathbf{x}) := 1 - \sigma(z)$

z is the dot product of the final layer's weights with the previous layer's output. I.e., for networks with one active layer $z = \mathbf{w}^T\mathbf{x}$; for d active layers $z = \mathbf{w}_d^T\,\mathbf{y}^{h_{d-1}}$.



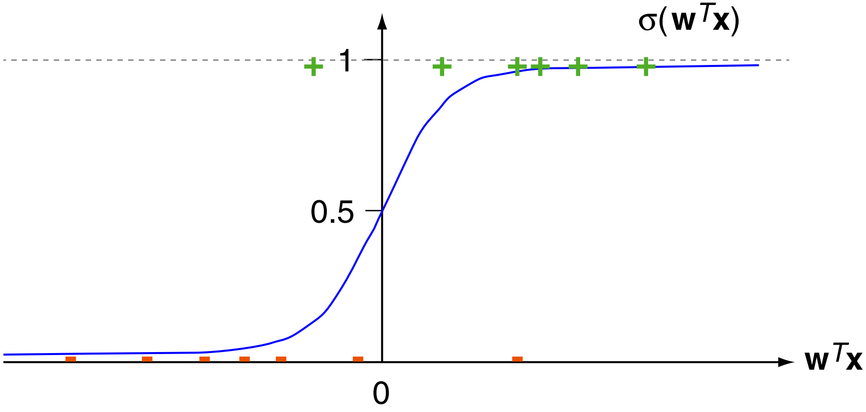
Advanced MLPs

Output Normalization: Softmax (continued)

For two classes ($k = 2$), the scalar sigmoid output $\sigma(z)$ determines both class probabilities for $\mathbf{x}$:

- ❑ $p(1 \mid \mathbf{x}) := \sigma(z)$
- ❑ $p(0 \mid \mathbf{x}) := 1 - \sigma(z)$

z is the dot product of the final layer's weights with the previous layer's output. I.e., for networks with one active layer $z = \mathbf{w}^T\mathbf{x}$; for d active layers $z = \mathbf{w}_d^T\,\mathbf{y}^{h_{d-1}}$.



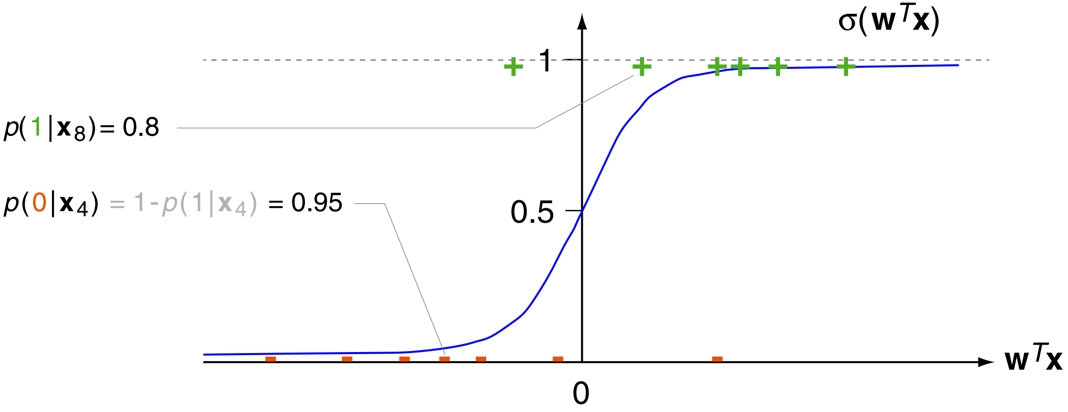
Advanced MLPs

Output Normalization: Softmax (continued)

For two classes ($k = 2$), the scalar sigmoid output $\sigma(z)$ determines both class probabilities for $\mathbf{x}$:

- ❑ $p(1 \mid \mathbf{x}) := \sigma(z)$
- ❑ $p(0 \mid \mathbf{x}) := 1 - \sigma(z)$

z is the dot product of the final layer's weights with the previous layer's output. I.e., for networks with one active layer $z = \mathbf{w}^T\mathbf{x}$; for d active layers $z = \mathbf{w}_d^T\, \mathbf{y}^{h_{d-1}}$.



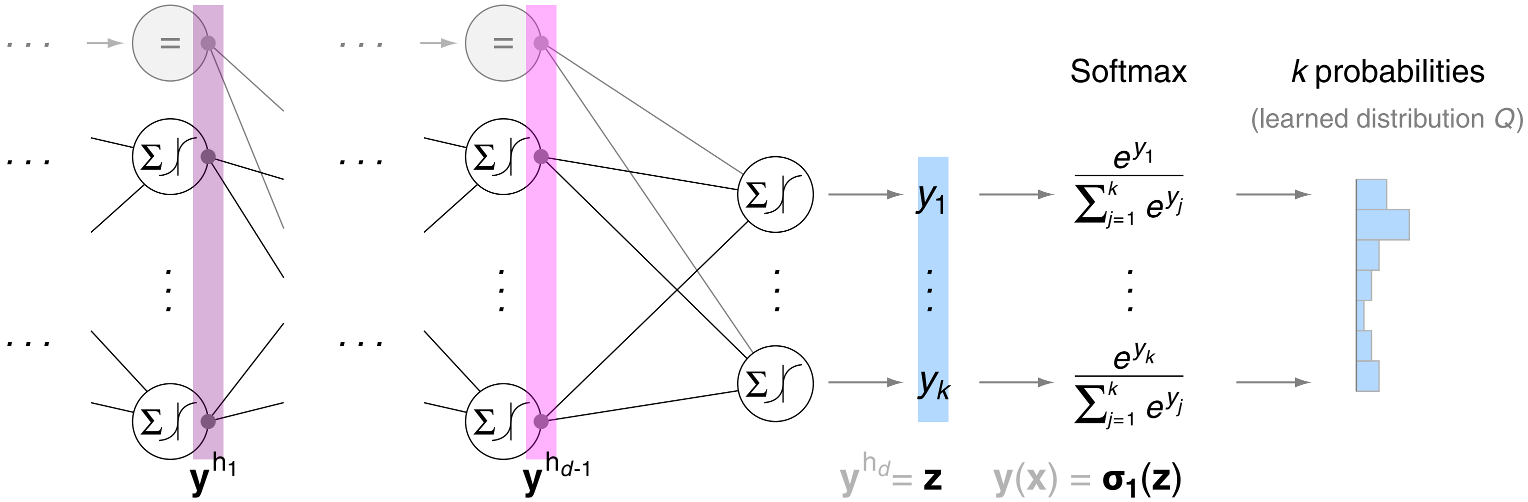
Advanced MLPs

Output Normalization: Softmax (continued)

The softmax function $\boldsymbol{\sigma}_1 : \mathbf{R}^k \to \Delta^{k-1}$, $\Delta^{k-1} \subset \mathbf{R}^k$, generalizes the logistic (sigmoid) function to k dimensions or k exclusive classes [Wikipedia] :

$$\boldsymbol{\sigma}_1(\mathbf{z})|_i = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}$$

Multi-layer perceptron for k classes:



[cross-entropy loss]

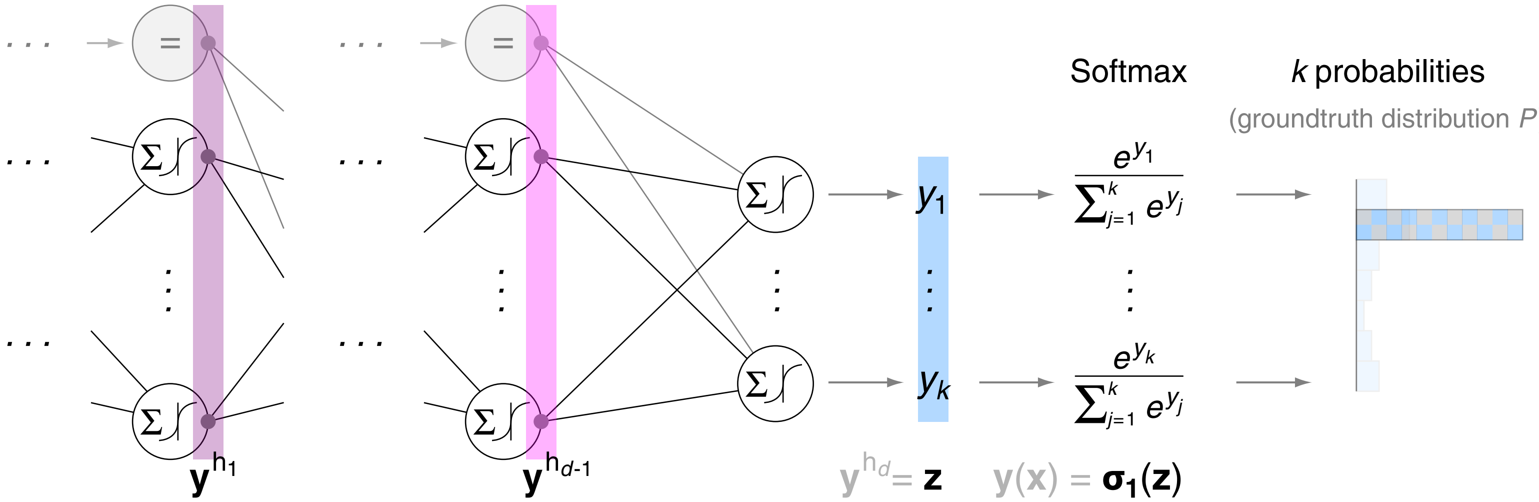
Advanced MLPs

Output Normalization: Softmax (continued)

The softmax function $\boldsymbol{\sigma}_1 : \mathbf{R}^k \to \Delta^{k-1}$, $\Delta^{k-1} \subset \mathbf{R}^k$, generalizes the logistic (sigmoid) function to k dimensions or k exclusive classes [Wikipedia] :

$$\boldsymbol{\sigma}_1(\mathbf{z})\big|_i = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}$$

Multi-layer perceptron for k classes:



[cross-entropy loss]

Remarks:

- The standard $k{-}1$-simplex, denoted as Δ^{k-1}, contains all k-tuples with non-negative elements that sum to 1:

$$\Delta^{k-1} = \left\{(p_1, \ldots, p_k) \in \mathbf{R}^k : \textstyle\sum_{i=1}^{k} p_i = 1 \text{ and } p_i \geq 0 \text{ for all } i\right\}$$

- The softmax function ensures Axiom I (positivity) and Axiom II (unitarity) of Kolmogorov.

- The single output in the two-class setting, the class 1 probability $\sigma(z)$, can be rewritten as softmax vector that comprises both class probabilities:

$$\mathbf{x} \;\rightarrow\; \begin{bmatrix} p(1 \mid \mathbf{x}) \\ p(0 \mid \mathbf{x}) \end{bmatrix} = \begin{bmatrix} \sigma(z) \\ 1{-}\sigma(z) \end{bmatrix} = \begin{bmatrix} \frac{1}{1+e^{-z}} \\ \sigma(-z) \end{bmatrix} = \begin{bmatrix} \frac{e^z}{1+e^z} \\ \frac{1}{1+e^z} \end{bmatrix} = \begin{bmatrix} \frac{e^z}{e^0+e^z} \\ \frac{e^0}{e^0+e^z} \end{bmatrix} = \boldsymbol{\sigma}_1(\tbinom{z}{0})$$

Advanced MLPs

Loss Function: Cross-Entropy

Definition 2 (Cross Entropy)

Let $\mathbf{C}$ be a random variable with distribution P and a finite number of realizations C. Let Q be another distribution of $\mathbf{C}$. Then, the cross entropy of distribution Q relative to the distribution P, denoted as $H(P, Q)$, is defined as follows:

$$H(P, Q) = -\sum_{c \in C} P(\mathbf{C}{=}c) \cdot \log\big(Q(\mathbf{C}{=}c)\big)$$

Advanced MLPs

Loss Function: Cross-Entropy (continued)

Definition 2 (Cross Entropy)

Let *C* be a random variable with distribution P and a finite number of realizations C. Let Q be another distribution of *C*. Then, the cross entropy of distribution Q relative to the distribution P, denoted as $H(P, Q)$, is defined as follows:

$$H(P, Q) = -\sum_{c \in C} P(\mathbf{C}{=}c) \cdot \log\big(Q(\mathbf{C}{=}c)\big)$$

- The cross entropy $H(P, Q)$ is the average number of *total* bits to represent an event *C*$=c$ under the distribution Q instead of under the distribution P.

- The relative entropy, also called Kullback-Leibler divergence, is the average number of *additional* bits to represent an event under Q instead of under P.

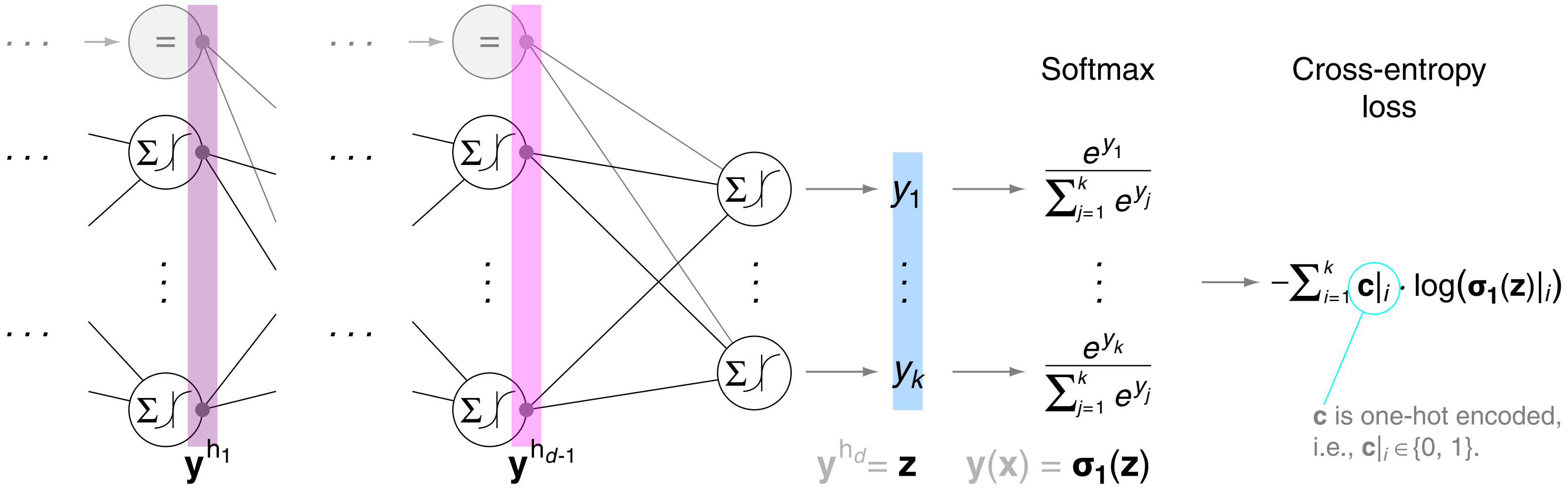
Advanced MLPs

Loss Function: Cross-Entropy (continued)

Definition 2 (Cross Entropy)

Let C be a random variable with distribution P and a finite number of realizations C. Let Q be another distribution of C. Then, the cross entropy of distribution Q relative to the distribution P, denoted as $H(P, Q)$, is defined as follows:

$$H(P,Q) = -\sum_{c \in C} P(C{=}c) \cdot \log\big(Q(C{=}c)\big)$$



$\mathbf{c}$ is one-hot encoded, i.e., $\mathbf{c}|_i \in \{0, 1\}$.

[softmax]

Advanced MLPs

Cross-Entropy in Classification Settings

[logistic loss: definition, derivation]

$$H(P,Q) = -\sum_{c\in C} P(\mathsf{C}{=}c)\cdot \log\big(Q(\mathsf{C}{=}c)\big)$$

- Random variable C denotes a class.
- Realizations of C: $C = \{c_1, \ldots, c_k\}$.
- P, Q define distributions of C.

$$H(p,q) = -\sum_{c\in C} p(c)\cdot \log\big(q(c)\big)$$

- Probability functions p, q related to P, Q.
- Class labels $C = \{c_1, \ldots, c_k\}$.

$$l_\sigma(z) = -c\cdot \log\big(\sigma(z)\big) - (1-c)\cdot \log\big(1-\sigma(z)\big)$$

- Two classes encoded as c, $c \in \{0,1\}$.
- Example with groundtruth $(\mathbf{x}, c) \in D$.
- Classifier output $\sigma(z)$, $z = y(\mathbf{x})$.

$$l_{\boldsymbol{\sigma}_1}(\mathbf{z}) = -\sum_{i=1}^{k} \mathbf{c}|_i \cdot \log\big(\boldsymbol{\sigma}_1(\mathbf{z})|_i\big)$$

- k classes, hot-encoded as $\mathbf{c}^T$, $\mathbf{c}^T \in \{(1,0,\ldots,0),\ldots,(0,\ldots,0,1)\}$.
- Example with groundtruth $(\mathbf{x}, \mathbf{c}) \in D$.
- Classifier output $\boldsymbol{\sigma}_1(\mathbf{z})$, $\mathbf{z} = \mathbf{y}(\mathbf{x})$.

Advanced MLPs

Cross-Entropy in Classification Settings (continued)

[logistic loss: definition, derivation]

$$H(P,Q) = -\sum_{c \in C} P(\mathsf{C}{=}c) \cdot \log\big(Q(\mathsf{C}{=}c)\big)$$

- Random variable C denotes a class.
- Realizations of C: $C = \{c_1, \ldots, c_k\}$.
- P, Q define distributions of C.

$$H(p,q) = -\sum_{c \in C} p(c) \cdot \log\big(q(c)\big)$$

- Probability functions p, q related to P, Q.
- Class labels $C = \{c_1, \ldots, c_k\}$.

$$l_\sigma(z) = -c \cdot \log\big(\sigma(z)\big) - (1{-}c) \cdot \log\big(1{-}\sigma(z)\big)$$

- Two classes encoded as c, $c \in \{0,1\}$.
- Example with groundtruth $(\mathbf{x}, c) \in D$.
- Classifier output $\sigma(z)$, $z = y(\mathbf{x})$.

$$l_{\boldsymbol{\sigma}_1}(\mathbf{z}) = -\sum_{i=1}^{k} \mathbf{c}|_i \cdot \log\big(\boldsymbol{\sigma}_1(\mathbf{z})|_i\big)$$

- k classes, hot-encoded as $\mathbf{c}^T$, $\mathbf{c}^T \in \{(1,0,\ldots,0),\ldots,(0,\ldots,0,1)\}$.
- Example with groundtruth $(\mathbf{x}, \mathbf{c}) \in D$.
- Classifier output $\boldsymbol{\sigma}_1(\mathbf{z})$, $\mathbf{z} = \mathbf{y}(\mathbf{x})$.

Advanced MLPs

Cross-Entropy in Classification Settings (continued)

[logistic loss: definition, derivation]

$$H(P,Q) = -\sum_{c\in C} P(\mathsf{C}{=}c)\cdot \text{log}\left(Q(\mathsf{C}{=}c)\right)$$

- Random variable C denotes a class.
- Realizations of C: $C = \{c_1,\ldots,c_k\}$.
- P, Q define distributions of C.

$$H(p,q) = -\sum_{c\in C} p(c)\cdot \text{log}\left(q(c)\right)$$

- Probability functions p, q related to P, Q.
- Class labels $C = \{c_1,\ldots,c_k\}$.

$$l_\sigma(z) = -c\cdot \text{log}\left(\sigma(z)\right) - (1{-}c)\cdot \text{log}\left(1{-}\sigma(z)\right)$$

- Two classes encoded as c, $c\in\{0,1\}$.
- Example with groundtruth $(\mathbf{x},c)\in D$.
- Classifier output $\sigma(z)$, $z = y(\mathbf{x})$.

$$l_{\boldsymbol{\sigma}_1}(\mathbf{z}) = -\sum_{i=1}^{k} \mathbf{c}|_i\cdot \text{log}\left(\boldsymbol{\sigma}_1(\mathbf{z})|_i\right)$$

- k classes, hot-encoded as $\mathbf{c}^T$, $\mathbf{c}^T\in\{(1,0,\ldots,0),\ldots,(0,\ldots,0,1)\}$.
- Example with groundtruth $(\mathbf{x},\mathbf{c})\in D$.
- Classifier output $\boldsymbol{\sigma}_1(\mathbf{z})$, $\mathbf{z} = \mathbf{y}(\mathbf{x})$.

Remarks:

- We have already encountered the cross-entropy loss function in logistic regression, under the name logistic loss (function). Other synonyms are logarithmic loss and log loss (function).
- Note that c (in the two-class setting) or $\mathbf{c}|_i$ (in the general case) are either 0 or 1, and that they can be interpreted as probability for the occurrence of the respective class; a similar argument applies to the functions $\sigma()$ and $\boldsymbol{\sigma}_1()$, which are interpreted as class probabilities as well.

  Under this interpretation, the logistic loss can be rewritten as cross entropy (and vice versa):

$$
\begin{aligned}
l_\sigma(z) &= -c \cdot \log(\sigma(z)) - (1-c) \cdot \log(1-\sigma(z)) \\
&= -(c \cdot \log(\sigma(z)) + (1-c) \cdot \log(1-\sigma(z))) \\
&= -(p(c_1) \cdot \log(q(c_1)) + p(c_2) \cdot \log(q(c_2))) \\
&= -\sum_{c \in C} p(c) \cdot \log(q(c)) \ = \ H(p,q)
\end{aligned}
$$

  Similarly, the cross-entry loss in the MLP illustration is written as logistic loss.

- $\mathbf{c}|_i$ denotes the projection operator, which returns the ith vector component (dimension) of $\mathbf{c}$, $\mathbf{c} = (c_1, \ldots, c_k)$.
- If not stated otherwise, log means $\log_2$.

Advanced MLPs

Activation Function: Rectified Linear Unit (ReLU)

[*TODO*]

Advanced MLPs

Regularization: Dropout

[*TODO*]

Advanced MLPs

Learning Rate Adaptation: Momentum

Momentum principle: a weight adaptation in iteration t considers the adaptation in iteration $t{-}1$:

$$\begin{aligned}
\underline{\Delta W^{\mathsf{o}}(t)} &= \eta \cdot (\boldsymbol{\delta}^{\mathsf{o}} \otimes \mathbf{y}^{\mathsf{h}}(\mathbf{x})|_{1,\ldots,l}) && + \alpha \cdot \Delta W^{\mathsf{o}}(t{-}1) \\
\underline{\Delta W^{\mathsf{h}}(t)} &= \eta \cdot (\boldsymbol{\delta}^{\mathsf{h}} \otimes \mathbf{x}) && + \alpha \cdot \Delta W^{\mathsf{h}}(t{-}1)
\end{aligned}$$

$$\begin{aligned}
\underline{\Delta W^{\mathsf{h}_s}(t)} &= \eta \cdot (\boldsymbol{\delta}^{\mathsf{h}_s} \otimes \mathbf{y}^{\mathsf{h}_{s-1}}(\mathbf{x})|_{1,\ldots,l_{s-1}}) + \alpha \cdot \Delta W^{\mathsf{h}_s}(t{-}1), \quad s = d, d{-}1, \ldots, 2 \\
\underline{\Delta W^{\mathsf{h}_1}(t)} &= \eta \cdot (\boldsymbol{\delta}^{\mathsf{h}_1} \otimes \mathbf{x}) \qquad\qquad\qquad\quad + \alpha \cdot \Delta W^{\mathsf{h}_1}(t{-}1)
\end{aligned}$$

The term α, $0 \leq \alpha < 1$, is called “momentum”.

Advanced MLPs

Learning Rate Adaptation: Momentum (continued)

Momentum principle: a weight adaptation in iteration t considers the adaptation in iteration $t{-}1$:

$$
\begin{aligned}
\underline{\Delta W^{\mathsf{o}}(t)} &= \eta \cdot (\boldsymbol{\delta}^{\mathsf{o}} \otimes \mathbf{y}^{\mathsf{h}}(\mathbf{x})|_{1,\ldots,l}) &&+ \alpha \cdot \Delta W^{\mathsf{o}}(t{-}1) \\
\underline{\Delta W^{\mathsf{h}}(t)} &= \eta \cdot (\boldsymbol{\delta}^{\mathsf{h}} \otimes \mathbf{x}) &&+ \alpha \cdot \Delta W^{\mathsf{h}}(t{-}1)
\end{aligned}
$$

$$
\begin{aligned}
\underline{\Delta W^{\mathsf{h}_s}(t)} &= \eta \cdot (\boldsymbol{\delta}^{\mathsf{h}_s} \otimes \mathbf{y}^{\mathsf{h}_{s-1}}(\mathbf{x})|_{1,\ldots,l_{s-1}}) &&+ \alpha \cdot \Delta W^{\mathsf{h}_s}(t{-}1), \quad s = d, d{-}1, \ldots, 2 \\
\underline{\Delta W^{\mathsf{h}_1}(t)} &= \eta \cdot (\boldsymbol{\delta}^{\mathsf{h}_1} \otimes \mathbf{x}) &&+ \alpha \cdot \Delta W^{\mathsf{h}_1}(t{-}1)
\end{aligned}
$$

The term α, $0 \leq \alpha < 1$, is called “momentum”.

Effects:

- ❑ Due the “adaptation inertia” local minima can be overcome.
- ❑ If the direction of the descent does not change, the adaptation increment and, as a consequence, the speed of convergence is increased.

Remarks:

- Recap. The symbol »$\otimes$« denotes the dyadic product, also called outer product or tensor product. The dyadic product takes two vectors and returns a second order tensor, called a dyadic in this context: $\mathbf{v} \otimes \mathbf{w} \equiv \mathbf{v}\mathbf{w}^T$. [Wikipedia]

Chapter ML:IV (continued)

IV. Neural Networks

Automatic Gradient Computation

The IGD Algorithm

Algorithm:	$\text{IGD}_{\text{MLP}^*}$	IGD for the d-layer MLP with arbitrary model and objective functions.
Input:	D	Multiset of examples $(\mathbf{x}, \mathbf{c})$ with $\mathbf{x} \in \mathbf{R}^p$, $\mathbf{c} \in \{0,1\}^k$.
	η, $l()$, $R()$, λ	Learning rate, loss and regularization functions and parameters.
Output:	$W^{\mathsf{h}_1}, \ldots, W^{\mathsf{h}_d}$	Weight matrices of the d layers. (= hypothesis)

```
 1.  FOR $s = 1$ TO $d$ DO *initialize_random_weights*$(W^{\mathsf{h}_s})$ ENDDO, $t = 0$
 2.  REPEAT
 3.    $t = t + 1$
 4.    FOREACH $(\mathbf{x}, \mathbf{c}) \in D$ DO
 5.      $\mathbf{y}^{\mathsf{h}_1}(\mathbf{x}) = \begin{pmatrix} 1 \\ \mathbf{tanh}(W^{\mathsf{h}_1}\,\mathbf{x}) \end{pmatrix}$ // forward propagation; $\mathbf{x}$ is extended by $x_0 = 1$
         FOR $s = 2$ TO $d{-}1$ DO $\mathbf{y}^{\mathsf{h}_s}(\mathbf{x}) = \begin{pmatrix} 1 \\ \text{ReLU}(W^{\mathsf{h}_s}\,\mathbf{y}^{\mathsf{h}_{s-1}}(\mathbf{x})) \end{pmatrix}$ ENDDO
         $\mathbf{y}(\mathbf{x}) = \boldsymbol{\sigma}_1(W^{\mathsf{h}_d}\,\mathbf{y}^{\mathsf{h}_{d-1}}(\mathbf{x}))$
 6.      $\boldsymbol{\delta} = \mathbf{c} - \mathbf{y}(\mathbf{x})$
 7a.     $\ell(\mathbf{w}) = l(\boldsymbol{\delta}) + \frac{\lambda}{n} R(\mathbf{w})$ // backpropagation (Steps 7a+7b)
         $\nabla\ell(\mathbf{w}) = \text{autodiff}(\ell(), \mathbf{w})$
 7b.     FOR $s = 1$ TO $d$ DO $\Delta W^{\mathsf{h}_s} = \eta \cdot \nabla^{\mathsf{h}_s}\ell(\mathbf{w})$ ENDDO
 8.      FOR $s = 1$ TO $d$ DO $W^{\mathsf{h}_s} = W^{\mathsf{h}_s} + \Delta W^{\mathsf{h}_s}$ ENDDO
 9.    ENDDO
10.  UNTIL(*convergence*$(D, \mathbf{y}(\,\cdot\,), t)$)
11.  *return*$(W^{\mathsf{h}_1}, \ldots, W^{\mathsf{h}_d})$
```

Automatic Gradient Computation

The IGD Algorithm (continued)

Algorithm:	$\text{IGD}_{\text{MLP}^*}$	IGD for the d-layer MLP with arbitrary model and objective functions.
Input:	D	Multiset of examples $(\mathbf{x}, \mathbf{c})$ with $\mathbf{x} \in \mathbf{R}^p$, $\mathbf{c} \in \{0,1\}^k$.
	η, $l()$, $R()$, λ	Learning rate, loss and regularization functions and parameters.
Output:	$W^{\text{h}_1}, \ldots, W^{\text{h}_d}$	Weight matrices of the d layers. (= hypothesis)

```
 1.  FOR $s = 1$ TO $d$ DO initialize_random_weights($W^{\text{h}_s}$) ENDDO, $t = 0$
 2.  REPEAT
 3.    $t = t + 1$
 4.    FOREACH $(\mathbf{x}, \mathbf{c}) \in D$ DO
 5.      $\mathbf{y}^{\text{h}_1}(\mathbf{x}) = \begin{pmatrix} 1 \\ \mathbf{tanh}(W^{\text{h}_1}\mathbf{x}) \end{pmatrix}$  // forward propagation; $\mathbf{x}$ is extended by $x_0 = 1$
         FOR $s = 2$ TO $d{-}1$ DO $\mathbf{y}^{\text{h}_s}(\mathbf{x}) = \begin{pmatrix} 1 \\ \text{ReLU}(W^{\text{h}_s}\mathbf{y}^{\text{h}_{s-1}}(\mathbf{x})) \end{pmatrix}$ ENDDO
         $\mathbf{y}(\mathbf{x}) = \boldsymbol{\sigma}_1(W^{\text{h}_d}\mathbf{y}^{\text{h}_{d-1}}(\mathbf{x}))$
 6.      $\boldsymbol{\delta} = \mathbf{c} - \mathbf{y}(\mathbf{x})$
7a.      $\ell(\mathbf{w}) = l(\boldsymbol{\delta}) + \frac{\lambda}{n}R(\mathbf{w})$  // backpropagation (Steps 7a+7b)
         $\nabla\ell(\mathbf{w}) = \text{autodiff}(\ell(), \mathbf{w})$
7b.      FOR $s = 1$ TO $d$ DO $\Delta W^{\text{h}_s} = \eta \cdot \nabla^{\text{h}_s}\ell(\mathbf{w})$ ENDDO
 8.      FOR $s = 1$ TO $d$ DO $W^{\text{h}_s} = W^{\text{h}_s} + \Delta W^{\text{h}_s}$ ENDDO
 9.    ENDDO
10.  UNTIL(convergence($D, \mathbf{y}(\,\cdot\,), t$))
11.  return($W^{\text{h}_1}, \ldots, W^{\text{h}_d}$)
```

Automatic Gradient Computation

The IGD Algorithm (continued)

Algorithm:	$\text{IGD}_{\text{MLP}^*}$	IGD for the d-layer MLP with arbitrary model and objective functions.
Input:	D	Multiset of examples $(\mathbf{x}, \mathbf{c})$ with $\mathbf{x} \in \mathbf{R}^p$, $\mathbf{c} \in \{0,1\}^k$.
	η, $l()$, $R()$, λ	Learning rate, loss and regularization functions and parameters.
Output:	$W^{\mathsf{h}_1}, \ldots, W^{\mathsf{h}_d}$	Weight matrices of the d layers. (= hypothesis)

```
FOR $s = 1$ TO $d$ DO initialize_random_weights($W^{\mathsf{h}_s}$) ENDDO, $t = 0$
REPEAT
   $t = t + 1$
   FOREACH $(\mathbf{x}, \mathbf{c}) \in D$ DO
                          Model function evaluation.
                          Calculation of residual vector.
                          Calculation of derivative of the loss.

                          Parameter vector update $\widehat{=}$ one gradient step down.
   ENDDO
UNTIL(convergence($D, \mathbf{y}(\cdot), t$))
return($W^{\mathsf{h}_1}, \ldots, W^{\mathsf{h}_d}$)
```

Automatic Gradient Computation

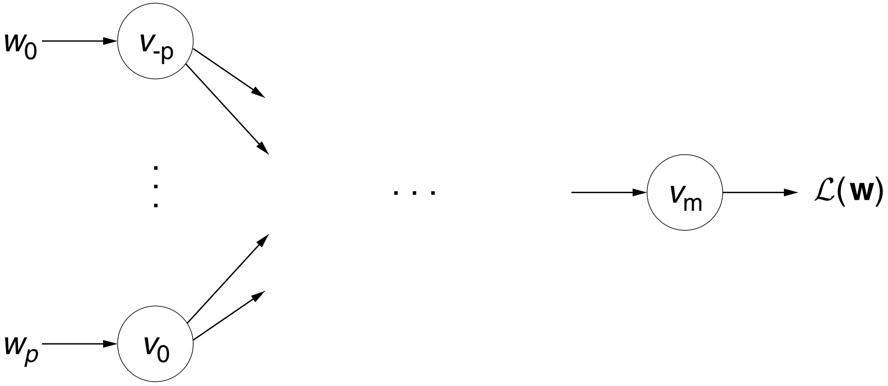
Reverse-Mode Automatic Differentiation in Computational Graphs

Reverse-mode AD corresponds to a generalized backpropagation algorithm.

Let $\mathcal{L}(w_1, \ldots, w_p)$ be the function to be differentiated.

- Consider $\mathcal{L}$ as a computational graph of elementary operations, assigning each intermediate result to a variable v_i with $-p \leq i \leq m$

  (naming convention: $v_{-p\ldots0}$ for inputs, $v_{1\ldots m-1}$ for intermediate variables, $v_m \equiv \mathcal{L}$ for the output)

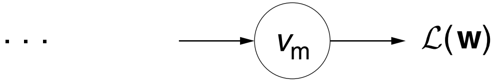


Automatic Gradient Computation

Reverse-Mode Automatic Differentiation in Computational Graphs (continued)

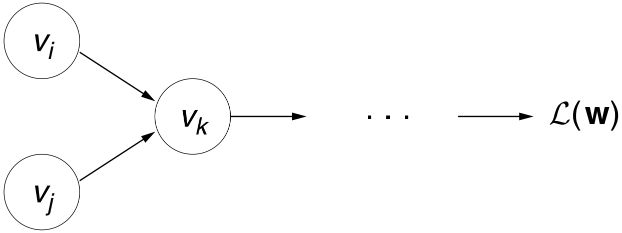
For each intermediate variable v_i, an adjoint value $\nabla^{v_i}\mathcal{L} \equiv \frac{\partial \mathcal{L}}{\partial v_i}$ is computed based on its descendants in the computation graph.

(1)



$$\nabla^{v_m}\mathcal{L} \equiv \frac{\partial \mathcal{L}}{\partial v_m} = \frac{\partial v_m}{\partial v_m} = 1$$

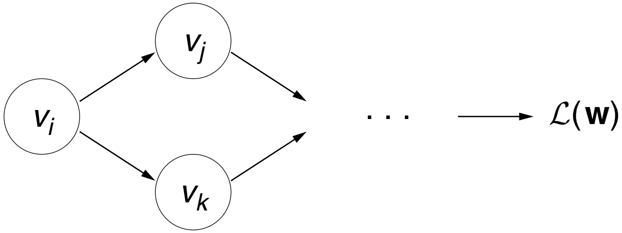
(2)



$$\nabla^{v_i}\mathcal{L} \equiv \frac{\partial \mathcal{L}}{\partial v_i} = \frac{\partial \mathcal{L}}{\partial v_k} \cdot \frac{\partial v_k}{\partial v_i} = \nabla^{v_k}\mathcal{L} \cdot \frac{\partial v_k}{\partial v_i}$$

$$\nabla^{v_j}\mathcal{L} \equiv \frac{\partial \mathcal{L}}{\partial v_j} = \frac{\partial \mathcal{L}}{\partial v_k} \cdot \frac{\partial v_k}{\partial v_j} = \nabla^{v_k}\mathcal{L} \cdot \frac{\partial v_k}{\partial v_j}$$

(3)



$$\nabla^{v_i}\mathcal{L} = \nabla^{v_j}\mathcal{L} \cdot \frac{\partial v_j}{\partial v_i} + \nabla^{v_k}\mathcal{L} \cdot \frac{\partial v_k}{\partial v_i}$$

Remarks:

- Adjoints are computed in reverse, starting from $\nabla^{v_m}\mathcal{L}$.
- For any step $v_j = g(\ldots, v_i, \ldots)$ in the graph, the local gradients $\frac{\partial g}{\partial v_i}$ must be computable.

Automatic Gradient Computation

Autodiff Example: Setting

Consider the RSS loss for a simple logistic regression model and a very small dataset.

Dataset: $D = \{((1, 1.5)^T_, 0), ((1.5, -1)^T_, 1)\}$

Model function: $y(x) = \sigma(\mathbf{w}^T\mathbf{x})$

Loss function: $\mathcal{L}(\mathbf{w}) = L_2(\mathbf{w}) = \sum_{(\mathbf{x},c)\in D} (c - y(\mathbf{x}))^2$

$\mathcal{L}(\mathbf{w})$ is the objective function to be minimized, and hence what we want to compute the derivative of; everything except $\mathbf{w}$ is held constant.

Given the setting above, we can rewrite $\mathcal{L}$ as:

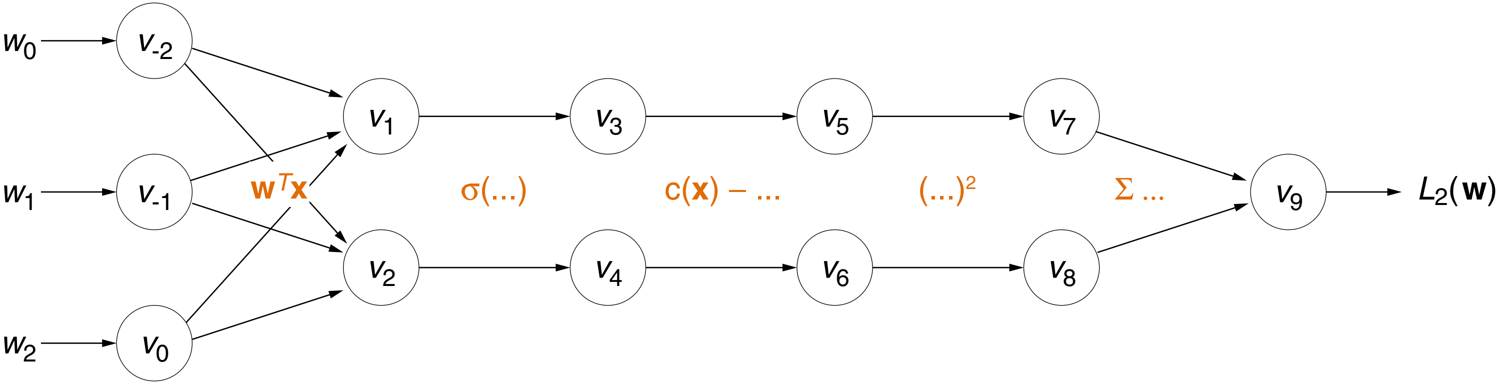
$$\begin{aligned} \mathcal{L}(\mathbf{w}) &= (c_1 - \sigma(\mathbf{w}^T\mathbf{x}_1))^2 + (c_2 - \sigma(\mathbf{w}^T\mathbf{x}_2))^2 \\ &= (-\sigma(w_0 + w_1 + 1.5w_2))^2 + (1 - \sigma(w_0 + 1.5w_1 - w_2))^2 \end{aligned}$$

Using reverse-mode automatic differentiation, we'll simultaneously evaluate the loss and its derivative at $\mathbf{w} = (-1, 1.5, 0.5)^T$.

Automatic Gradient Computation

Autodiff Example: Computational Graph

$$\mathcal{L}(\mathbf{w}) = \underbrace{\overbrace{\underbrace{(-\overbrace{\sigma(\underbrace{w_0+w_1+1.5w_2}_{v_1})}^{v_3})}_{v_5}{}^2}^{v_7} + \overbrace{\underbrace{(1-\overbrace{\sigma(\underbrace{w_0+1.5w_1-w_2}_{v_2})}^{v_4})}_{v_6}{}^2}^{v_8}}_{v_9}$$



Automatic Gradient Computation

Autodiff Example: Forward and Reverse Trace

$$\mathcal{L}(\mathbf{w}) = \underbrace{\overbrace{(\underbrace{-\overbrace{\sigma(\underbrace{w_0 + w_1 + 1.5w_2}_{v_1})}^{v_3}}_{v_5})^2}^{v_7} + \overbrace{(\underbrace{1 - \overbrace{\sigma(\underbrace{w_0 + 1.5w_1 - w_2}_{v_2})}^{v_4}}_{v_6})^2}^{v_8}}_{v_9}$$

at

$$\mathbf{w} = (-1, 1.5, 0.5)^T$$

Forward primal trace		Reverse adjoint trace
$v_0 = w_0$	$= -1$	
$v_{-1} = w_1$	$= 1.5$	
$v_{-2} = w_2$	$= 0.5$	
$v_1 = v_0 + v_{-1} + 1.5 \cdot v_{-2}$	$= 1.25$	
$v_2 = v_0 + 1.5 \cdot v_{-1} - v_{-2}$	$= 0.75$	
$v_3 = \sigma(v_1)$	$= 0.78$	
$v_4 = \sigma(v_2)$	$= 0.68$	
$v_5 = 0 - v_3$	$= -0.78$	
$v_6 = 1 - v_4$	$= 0.32$	
$v_7 = v_5^2$	$= 0.61$	
$v_8 = v_6^2$	$= 0.1$	
$v_9 = v_7 + v_8$	$= 0.71$	
$\mathcal{L} = v_9$	$= 0.71$	

Automatic Gradient Computation

Autodiff Example: Forward and Reverse Trace (continued)

$$\mathcal{L}(\mathbf{w}) = \underbrace{\overbrace{(\underbrace{-\overbrace{\sigma(\underbrace{w_0 + w_1 + 1.5w_2}_{v_1})}^{v_3}}_{v_5})^2}^{v_7} + \overbrace{(\underbrace{1 - \overbrace{\sigma(\underbrace{w_0 + 1.5w_1 - w_2}_{v_2})}^{v_4}}_{v_6})^2}^{v_8}}_{v_9} \quad \text{at} \quad \mathbf{w} = (-1, 1.5, 0.5)^T$$

Forward primal trace		Reverse adjoint trace	
$v_0 = w_0$	$= -1$		
$v_{-1} = w_1$	$= 1.5$		
$v_{-2} = w_2$	$= 0.5$		
$v_1 = v_0 + v_{-1} + 1.5 \cdot v_{-2}$	$= 1.25$		
$v_2 = v_0 + 1.5 \cdot v_{-1} - v_{-2}$	$= 0.75$		
$v_3 = \sigma(v_1)$	$= 0.78$		
$v_4 = \sigma(v_2)$	$= 0.68$		
$v_5 = 0 - v_3$	$= -0.78$		
$v_6 = 1 - v_4$	$= 0.32$		
$v_7 = v_5^2$	$= 0.61$		
$v_8 = v_6^2$	$= 0.1$		
$v_9 = v_7 + v_8$	$= 0.71$		
$\mathcal{L} = v_9$	$= 0.71$	$\nabla^{v_9}\mathcal{L} = \frac{\partial \mathcal{L}}{\partial v_9}$	$= 1$

Automatic Gradient Computation

Autodiff Example: Forward and Reverse Trace (continued)

$$\mathcal{L}(\mathbf{w}) = \underbrace{\overbrace{(\underbrace{-\overbrace{\sigma(\underbrace{w_0+w_1+1.5w_2}_{v_1})}^{v_3}}_{v_5})^2}^{v_7} + \overbrace{(\underbrace{1-\overbrace{\sigma(\underbrace{w_0+1.5w_1-w_2}_{v_2})}^{v_4}}_{v_6})^2}^{v_8}}_{v_9} \qquad \text{at} \qquad \mathbf{w} = (-1, 1.5, 0.5)^T$$

Forward primal trace		Reverse adjoint trace	
$v_0 = w_0$	$= -1$		
$v_{-1} = w_1$	$= 1.5$		
$v_{-2} = w_2$	$= 0.5$		
$v_1 = v_0 + v_{-1} + 1.5 \cdot v_{-2}$	$= 1.25$		
$v_2 = v_0 + 1.5 \cdot v_{-1} - v_{-2}$	$= 0.75$		
$v_3 = \sigma(v_1)$	$= 0.78$		
$v_4 = \sigma(v_2)$	$= 0.68$		
$v_5 = 0 - v_3$	$= -0.78$		
$v_6 = 1 - v_4$	$= 0.32$		
$v_7 = v_5^2$	$= 0.61$		
$v_8 = v_6^2$	$= 0.1$		
$v_9 = v_7 + v_8$	$= 0.71$	$\nabla^{v_8}\mathcal{L} = \nabla^{v_9}\mathcal{L} \cdot \frac{\partial v_9}{\partial v_8} = 1 \cdot 1$	$= 1$
		$\nabla^{v_7}\mathcal{L} = \nabla^{v_9}\mathcal{L} \cdot \frac{\partial v_9}{\partial v_7} = 1 \cdot 1$	$= 1$
$\mathcal{L} = v_9$	$= 0.71$	$\nabla^{v_9}\mathcal{L} = \frac{\partial \mathcal{L}}{\partial v_9}$	$= 1$

Automatic Gradient Computation

Autodiff Example: Forward and Reverse Trace (continued)

$$\mathcal{L}(\mathbf{w}) = \underbrace{\overbrace{(\underbrace{-\overbrace{\sigma(\underbrace{w_0 + w_1 + 1.5w_2}_{v_1})}^{v_3}}_{v_5})^2}^{v_7} + \overbrace{(\underbrace{1 - \overbrace{\sigma(\underbrace{w_0 + 1.5w_1 - w_2}_{v_2})}^{v_4}}_{v_6})^2}^{v_8}}_{v_9} \quad \text{at} \quad \mathbf{w} = (-1, 1.5, 0.5)^T$$

Forward primal trace		Reverse adjoint trace	
$v_0 = w_0$	$= -1$		
$v_{-1} = w_1$	$= 1.5$		
$v_{-2} = w_2$	$= 0.5$		
$v_1 = v_0 + v_{-1} + 1.5 \cdot v_{-2}$	$= 1.25$		
$v_2 = v_0 + 1.5 \cdot v_{-1} - v_{-2}$	$= 0.75$		
$v_3 = \sigma(v_1)$	$= 0.78$		
$v_4 = \sigma(v_2)$	$= 0.68$		
$v_5 = 0 - v_3$	$= -0.78$	$\nabla^{v_3}\mathcal{L} = \nabla^{v_5}\mathcal{L} \cdot (-1)$	$= 1.55$
$v_6 = 1 - v_4$	$= 0.32$	$\nabla^{v_4}\mathcal{L} = \nabla^{v_6}\mathcal{L} \cdot (-1)$	$= -0.64$
$v_7 = v_5^2$	$= 0.61$	$\nabla^{v_5}\mathcal{L} = \nabla^{v_7}\mathcal{L} \cdot \frac{\partial v_7}{\partial v_5} = \nabla^{v_7}\mathcal{L} \cdot 2v_5$	$= -1.55$
$v_8 = v_6^2$	$= 0.1$	$\nabla^{v_6}\mathcal{L} = \nabla^{v_8}\mathcal{L} \cdot \frac{\partial v_8}{\partial v_6} = \nabla^{v_8}\mathcal{L} \cdot 2v_6$	$= 0.64$
$v_9 = v_7 + v_8$	$= 0.71$	$\nabla^{v_8}\mathcal{L} = \nabla^{v_9}\mathcal{L} \cdot \frac{\partial v_9}{\partial v_8} = 1 \cdot 1$	$= 1$
		$\nabla^{v_7}\mathcal{L} = \nabla^{v_9}\mathcal{L} \cdot \frac{\partial v_9}{\partial v_7} = 1 \cdot 1$	$= 1$
$\mathcal{L} = v_9$	$= 0.71$	$\nabla^{v_9}\mathcal{L} = \frac{\partial \mathcal{L}}{\partial v_9}$	$= 1$

Automatic Gradient Computation

Autodiff Example: Forward and Reverse Trace (continued)

$$\mathcal{L}(\mathbf{w}) = \underbrace{\overbrace{(\underbrace{-\overbrace{\sigma(\underbrace{w_0 + w_1 + 1.5w_2}_{v_1})}^{v_3}}_{v_5})^2}^{v_7} + \overbrace{(\underbrace{1 - \overbrace{\sigma(\underbrace{w_0 + 1.5w_1 - w_2}_{v_2})}^{v_4}}_{v_6})^2}^{v_8}}_{v_9}$$

at $\mathbf{w} = (-1, 1.5, 0.5)^T$

Forward primal trace		Reverse adjoint trace	
$v_0 = w_0$	$= -1$		
$v_{-1} = w_1$	$= 1.5$		
$v_{-2} = w_2$	$= 0.5$		
$v_1 = v_0 + v_{-1} + 1.5 \cdot v_{-2}$	$= 1.25$		
$v_2 = v_0 + 1.5 \cdot v_{-1} - v_{-2}$	$= 0.75$		
$v_3 = \sigma(v_1)$	$= 0.78$	$\nabla^{v_1}\mathcal{L} = \nabla^{v_3}\mathcal{L} \cdot \sigma(v_1) \cdot (1 - \sigma(v_1))$	$= 0.27$
$v_4 = \sigma(v_2)$	$= 0.68$	$\nabla^{v_2}\mathcal{L} = \nabla^{v_4}\mathcal{L} \cdot \sigma(v_2) \cdot (1 - \sigma(v_2))$	$= -0.14$
$v_5 = 0 - v_3$	$= -0.78$	$\nabla^{v_3}\mathcal{L} = \nabla^{v_5}\mathcal{L} \cdot (-1)$	$= 1.55$
$v_6 = 1 - v_4$	$= 0.32$	$\nabla^{v_4}\mathcal{L} = \nabla^{v_6}\mathcal{L} \cdot (-1)$	$= -0.64$
$v_7 = v_5^2$	$= 0.61$	$\nabla^{v_5}\mathcal{L} = \nabla^{v_7}\mathcal{L} \cdot \frac{\partial v_7}{\partial v_5} = \nabla^{v_7}\mathcal{L} \cdot 2v_5$	$= -1.55$
$v_8 = v_6^2$	$= 0.1$	$\nabla^{v_6}\mathcal{L} = \nabla^{v_8}\mathcal{L} \cdot \frac{\partial v_8}{\partial v_6} = \nabla^{v_8}\mathcal{L} \cdot 2v_6$	$= 0.64$
$v_9 = v_7 + v_8$	$= 0.71$	$\nabla^{v_8}\mathcal{L} = \nabla^{v_9}\mathcal{L} \cdot \frac{\partial v_9}{\partial v_8} = 1 \cdot 1$	$= 1$
		$\nabla^{v_7}\mathcal{L} = \nabla^{v_9}\mathcal{L} \cdot \frac{\partial v_9}{\partial v_7} = 1 \cdot 1$	$= 1$
$\mathcal{L} = v_9$	$= 0.71$	$\nabla^{v_9}\mathcal{L} = \frac{\partial \mathcal{L}}{\partial v_9}$	$= 1$

Automatic Gradient Computation

Autodiff Example: Forward and Reverse Trace (continued)

$$\mathcal{L}(\mathbf{w}) = \underbrace{\overbrace{(\underbrace{-\overbrace{\sigma(\underbrace{w_0 + w_1 + 1.5w_2}_{v_1})}^{v_3}}_{v_5})^2}^{v_7} + \overbrace{(\underbrace{1 - \overbrace{\sigma(\underbrace{w_0 + 1.5w_1 - w_2}_{v_2})}^{v_4}}_{v_6})^2}^{v_8}}_{v_9} \quad \text{at} \quad \mathbf{w} = (-1, 1.5, 0.5)^T$$

Forward primal trace	Reverse adjoint trace
$v_0 = w_0 \quad = -1$	
$v_{-1} = w_1 \quad = 1.5$	
$v_{-2} = w_2 \quad = 0.5$	
$v_1 = v_0 + v_{-1} + 1.5 \cdot v_{-2} \quad = 1.25$	$\nabla^{v_{-2}}\mathcal{L} = \nabla^{v_{-2}}\mathcal{L} + \nabla^{v_1}\mathcal{L} \cdot 1.5 \quad = 0.54$
	$\nabla^{v_{-1}}\mathcal{L} = \nabla^{v_{-1}}\mathcal{L} + \nabla^{v_1}\mathcal{L} \quad = 0.06$
	$\nabla^{v_0}\mathcal{L} = \nabla^{v_0}\mathcal{L} + \nabla^{v_1}\mathcal{L} \quad = 0.13$
$v_2 = v_0 + 1.5 \cdot v_{-1} - v_{-2} \quad = 0.75$	$\nabla^{v_{-2}}\mathcal{L} = \nabla^{v_2}\mathcal{L} \cdot (-1) \quad = 0.14$
	$\nabla^{v_{-1}}\mathcal{L} = \nabla^{v_2}\mathcal{L} \cdot 1.5 \quad = -0.28$
	$\nabla^{v_0}\mathcal{L} = \nabla^{v_2}\mathcal{L} \quad = -0.14$
$v_3 = \sigma(v_1) \quad = 0.78$	$\nabla^{v_1}\mathcal{L} = \nabla^{v_3}\mathcal{L} \cdot \sigma(v_1) \cdot (1 - \sigma(v_1)) \quad = 0.27$
$v_4 = \sigma(v_2) \quad = 0.68$	$\nabla^{v_2}\mathcal{L} = \nabla^{v_4}\mathcal{L} \cdot \sigma(v_2) \cdot (1 - \sigma(v_2)) \quad = -0.14$
$v_5 = 0 - v_3 \quad = -0.78$	$\nabla^{v_3}\mathcal{L} = \nabla^{v_5}\mathcal{L} \cdot (-1) \quad = 1.55$
$v_6 = 1 - v_4 \quad = 0.32$	$\nabla^{v_4}\mathcal{L} = \nabla^{v_6}\mathcal{L} \cdot (-1) \quad = -0.64$
$v_7 = v_5^2 \quad = 0.61$	$\nabla^{v_5}\mathcal{L} = \nabla^{v_7}\mathcal{L} \cdot \frac{\partial v_7}{\partial v_5} = \nabla^{v_7}\mathcal{L} \cdot 2v_5 \quad = -1.55$
$v_8 = v_6^2 \quad = 0.1$	$\nabla^{v_6}\mathcal{L} = \nabla^{v_8}\mathcal{L} \cdot \frac{\partial v_8}{\partial v_6} = \nabla^{v_8}\mathcal{L} \cdot 2v_6 \quad = 0.64$
$v_9 = v_7 + v_8 \quad = 0.71$	$\nabla^{v_8}\mathcal{L} = \nabla^{v_9}\mathcal{L} \cdot \frac{\partial v_9}{\partial v_8} = 1 \cdot 1 \quad = 1$
	$\nabla^{v_7}\mathcal{L} = \nabla^{v_9}\mathcal{L} \cdot \frac{\partial v_9}{\partial v_7} = 1 \cdot 1 \quad = 1$
$\mathcal{L} = v_9 \quad = 0.71$	$\nabla^{v_9}\mathcal{L} = \frac{\partial \mathcal{L}}{\partial v_9} \quad = 1$

Automatic Gradient Computation

Autodiff Example: Forward and Reverse Trace (continued)

$$\mathcal{L}(\mathbf{w}) = \underbrace{\overbrace{(\underbrace{-\overbrace{\sigma(\underbrace{w_0 + w_1 + 1.5w_2}_{v_1})}^{v_3}}_{v_5})^2}^{v_7} + \overbrace{(\underbrace{1 - \overbrace{\sigma(\underbrace{w_0 + 1.5w_1 - w_2}_{v_2})}^{v_4}}_{v_6})^2}^{v_8}}_{v_9} \qquad \text{at} \qquad \mathbf{w} = (-1, 1.5, 0.5)^T$$

Forward primal trace		Reverse adjoint trace	
$v_0 = w_0$	$= -1$	$\nabla^{w_0}\mathcal{L} = \frac{\partial \mathcal{L}}{\partial w_0} = \nabla^{v_0}\mathcal{L}$	$= \mathbf{0.13}$
$v_{-1} = w_1$	$= 1.5$	$\nabla^{w_1}\mathcal{L} = \frac{\partial \mathcal{L}}{\partial w_1} = \nabla^{v_{-1}}\mathcal{L}$	$= \mathbf{0.06}$
$v_{-2} = w_2$	$= 0.5$	$\nabla^{w_2}\mathcal{L} = \frac{\partial \mathcal{L}}{\partial w_2} = \nabla^{v_{-2}}\mathcal{L}$	$= \mathbf{0.54}$
$v_1 = v_0 + v_{-1} + 1.5 \cdot v_{-2}$	$= 1.25$	$\nabla^{v_{-2}}\mathcal{L} = \nabla^{v_{-2}}\mathcal{L} + \nabla^{v_1}\mathcal{L} \cdot 1.5$	$= 0.54$
		$\nabla^{v_{-1}}\mathcal{L} = \nabla^{v_{-1}}\mathcal{L} + \nabla^{v_1}\mathcal{L}$	$= 0.06$
		$\nabla^{v_0}\mathcal{L} = \nabla^{v_0}\mathcal{L} + \nabla^{v_1}\mathcal{L}$	$= 0.13$
$v_2 = v_0 + 1.5 \cdot v_{-1} - v_{-2}$	$= 0.75$	$\nabla^{v_{-2}}\mathcal{L} = \nabla^{v_2}\mathcal{L} \cdot (-1)$	$= 0.14$
		$\nabla^{v_{-1}}\mathcal{L} = \nabla^{v_2}\mathcal{L} \cdot 1.5$	$= -0.28$
		$\nabla^{v_0}\mathcal{L} = \nabla^{v_2}\mathcal{L}$	$= -0.14$
$v_3 = \sigma(v_1)$	$= 0.78$	$\nabla^{v_1}\mathcal{L} = \nabla^{v_3}\mathcal{L} \cdot \sigma(v_1) \cdot (1 - \sigma(v_1))$	$= 0.27$
$v_4 = \sigma(v_2)$	$= 0.68$	$\nabla^{v_2}\mathcal{L} = \nabla^{v_4}\mathcal{L} \cdot \sigma(v_2) \cdot (1 - \sigma(v_2))$	$= -0.14$
$v_5 = 0 - v_3$	$= -0.78$	$\nabla^{v_3}\mathcal{L} = \nabla^{v_5}\mathcal{L} \cdot (-1)$	$= 1.55$
$v_6 = 1 - v_4$	$= 0.32$	$\nabla^{v_4}\mathcal{L} = \nabla^{v_6}\mathcal{L} \cdot (-1)$	$= -0.64$
$v_7 = v_5^2$	$= 0.61$	$\nabla^{v_5}\mathcal{L} = \nabla^{v_7}\mathcal{L} \cdot \frac{\partial v_7}{\partial v_5} = \nabla^{v_7}\mathcal{L} \cdot 2v_5$	$= -1.55$
$v_8 = v_6^2$	$= 0.1$	$\nabla^{v_6}\mathcal{L} = \nabla^{v_8}\mathcal{L} \cdot \frac{\partial v_8}{\partial v_6} = \nabla^{v_8}\mathcal{L} \cdot 2v_6$	$= 0.64$
$v_9 = v_7 + v_8$	$= 0.71$	$\nabla^{v_8}\mathcal{L} = \nabla^{v_9}\mathcal{L} \cdot \frac{\partial v_9}{\partial v_8} = 1 \cdot 1$	$= 1$
		$\nabla^{v_7}\mathcal{L} = \nabla^{v_9}\mathcal{L} \cdot \frac{\partial v_9}{\partial v_7} = 1 \cdot 1$	$= 1$
$\mathcal{L} = v_9$	$= 0.71$	$\nabla^{v_9}\mathcal{L} = \frac{\partial \mathcal{L}}{\partial v_9}$	$= 1$

Remarks:

- For brevity, in the example, we assumed that the derivative $\frac{\partial}{\partial z}\sigma(z) = \sigma(z) \cdot (1 - \sigma(z))$ is already known. We could also have decomposed $\sigma(z) = \frac{1}{1+\exp(-z)}$ into e.g., $v_1 = -z$, $v_2 = \exp(v_1)$, $v_3 = 1 + v_2$, $v_4 = \frac{1}{v_3}$. In this case, only the four atomic derivatives would need to be known.
- The function to be automatically differentiated need not have a closed-form representation; it only has to be composed of computable and differentiable atomic steps. Thus, AD can also compute derivatives for various algorithms that may take different branches depending on the input.

Automatic Gradient Computation

Reverse-mode Autodiff Algorithm for Scalar-valued Functions

Algorithm:	autodiff	Reverse-mode automatic differentiation
Input:	$f : \mathbf{R}^p \to \mathbf{R}$	Function to differentiate.
	$(w_1, \ldots, w_p)^T$	Point at which the gradient should be evaluated
Output:	$(\bar{w}_1, \ldots, \bar{w}_p)^T$	Gradient of f at the point $(w_1, \ldots, w_p)^T$.

```
1. $\bar{w}_i = 0$ for $i$ in $1 \ldots p$                                   // initialize gradients
2. $v_1, \ldots, v_k$ = *operands*$(f)$
3. $\frac{\partial f}{\partial v_1}, \ldots, \frac{\partial f}{\partial v_k}$ = *gradients*$(f)$     // gradient of $f$ wrt. its immediate operands
4. FOREACH $j = 1, \ldots, k$ DO
5.   IF $v_j \in \{w_1, \ldots, w_p\}$ THEN
6.     $\bar{v}_j$ += $\frac{\partial f}{\partial v_j}$
7.   ELSE
8.     $(\bar{w}_1, \ldots, \bar{w}_p)^T$ += $\frac{\partial f}{\partial v_j} \cdot$ *autodiff*$(v_j, (w_1, \ldots, w_p)^T)$
9. RETURN $(\bar{w}_1, \ldots, \bar{w}_p)^T$
```

Remarks:

- There exists also a forward mode of automatic differentiation. One key difference is in the runtime complexity; for a function $f : \mathbf{R}^n \rightarrow \mathbf{R}^m$, to compute all $n \cdot m$ partial derivatives in the Jacobian matrix requires $O(n)$ iterations in forward mode and $O(m)$ iterations in reverse mode. Reverse mode is usually preferred in machine learning, where we typically have $m = 1$ (a scalar loss), and n arbitrarily large (e.g., billions of parameters of a deep neural network). See also [Baydin et al., 2018].