Hands-on derivation of tangent linear and adjoint codes

How to integrate and train neural networks inside 4D-Var

Marcin Chrust and Sébastien Massart
marcin.chrust@ecmwf.int
European Centre for Medium-Range Weather Forecasts

Outline

- Bridging machine learning and data assimilation under the common 4D-Var framework
- Tangent linear and adjoint of a simple Multi Layer Perceptron (MLP) neural network
- General MLP neural network and its tangent linear and adjoint
- Link between the adjoint coding and the backpropagation algorithm

Bridging machine learning and data assimilation under the common 4D-Var framework

- Let the dynamical model be parameterised by a set of parameters $\mathbf{p}$ constant over the assimilation window:

$$\mathbf{x}_k = \mathcal{M}_{k:0}\left(\mathbf{p}, \mathbf{x}_0\right).$$

- The non-linear cost function is:

$$\mathcal{J}\left(\mathbf{p}, \mathbf{x}_0\right) = \frac{1}{2}\left\|\mathbf{x}_0 - \mathbf{x}_0^{\mathrm{b}}\right\|^2_{\mathbf{B}^{-1}} + \frac{1}{2}\left\|\mathbf{p} - \mathbf{p}^{\mathrm{b}}\right\|^2_{\mathbf{P}^{-1}} + \frac{1}{2}\sum_{k=0}^{L}\left\|\mathbf{y}_k - \mathcal{H}_k \circ \mathcal{M}_{k:0}\left(\mathbf{p}, \mathbf{x}_0\right)\right\|^2_{\mathbf{R}_k^{-1}}.$$

- Consider $\mathbf{p}$ to be a set of parameters (weights and biases) of a NN.

Bridging machine learning and data assimilation under the common 4D-Var framework

Let's derive the quadratic cost function by making a change of variables $(\delta\mathbf{p}, \delta\mathbf{x}_0) \triangleq \left(\mathbf{p} - \mathbf{p}^{\mathsf{i}}, \mathbf{x}_0 - \mathbf{x}_0^{\mathsf{i}}\right)$, where $\left(\mathbf{p}^{\mathsf{i}}, \mathbf{x}_0^{\mathsf{i}}\right)$ is the first guess:

$$
\begin{aligned}
\mathcal{J}(\mathbf{p}, \mathbf{x}_0) &= \mathcal{J}\left(\mathbf{p}^{\mathsf{i}} + \delta\mathbf{p}, \mathbf{x}_0^{\mathsf{i}} + \delta\mathbf{x}_0\right), \\
&= \frac{1}{2}\left\|\mathbf{x}_0^{\mathsf{i}} - \mathbf{x}_0^{\mathsf{b}} + \delta\mathbf{x}_0\right\|^2_{\mathbf{B}^{-1}} + \frac{1}{2}\left\|\mathbf{p}^{\mathsf{i}} - \mathbf{p}^{\mathsf{b}} + \delta\mathbf{p}\right\|^2_{\mathbf{P}^{-1}} \\
&\quad + \frac{1}{2}\sum_{k=0}^{L}\left\|\mathbf{y}_k - \boldsymbol{\mathcal{H}}_k \circ \boldsymbol{\mathcal{M}}_{k:0}\left(\mathbf{p}^{\mathsf{i}} + \delta\mathbf{p}, \mathbf{x}_0^{\mathsf{i}} + \delta\mathbf{x}_0\right)\right\|^2_{\mathbf{R}_k^{-1}}, \\
&\approx \frac{1}{2}\left\|\mathbf{x}_0^{\mathsf{i}} - \mathbf{x}_0^{\mathsf{b}} + \delta\mathbf{x}_0\right\|^2_{\mathbf{B}^{-1}} + \frac{1}{2}\left\|\mathbf{p}^{\mathsf{i}} - \mathbf{p}^{\mathsf{b}} + \delta\mathbf{p}\right\|^2_{\mathbf{P}^{-1}} \\
&\quad + \frac{1}{2}\sum_{k=0}^{L}\left\|\mathbf{d}_k - \mathbf{H}_k\mathbf{M}_{k:0}\begin{pmatrix}\delta\mathbf{p} \\ \delta\mathbf{x}_0\end{pmatrix}\right\|^2_{\mathbf{R}_k^{-1}}, \\
&\triangleq \widehat{\mathcal{J}}(\delta\mathbf{p}, \delta\mathbf{x}_0).
\end{aligned}
$$

where:

- $\mathbf{d}_k \triangleq \mathbf{y}_k - \boldsymbol{\mathcal{H}}_k \circ \boldsymbol{\mathcal{M}}_{k:0}\left(\mathbf{p}^{\mathsf{i}}, \mathbf{x}_0^{\mathsf{i}}\right)$,
- $\mathbf{H}_k$ is the tangent linear (TL) operator of $\boldsymbol{\mathcal{H}}_k$ taken at $\boldsymbol{\mathcal{M}}_{k:0}\left(\mathbf{p}^{\mathsf{i}}, \mathbf{x}_0^{\mathsf{i}}\right)$,
- $\mathbf{M}_{k:0}$ is the TL operator of $\boldsymbol{\mathcal{M}}_{k:0}$ taken at $\left(\mathbf{p}^{\mathsf{i}}, \mathbf{x}_0^{\mathsf{i}}\right)$
- $\mathbf{M}_{k:0}$ can be split into $\mathbf{M}^{\mathsf{x}}_{k:0}$ and $\mathbf{M}^{\mathsf{p}}_{k:0}$

Bridging machine learning and data assimilation under the common 4D-Var framework

Let's derive the gradients of the quadratic cost function with respect to $\delta\mathbf{x}_0$ and $\delta\mathbf{p}$ respectively:

$$\begin{aligned}
\nabla_{\delta\mathbf{x}_0}\widehat{\mathcal{J}}(\delta\mathbf{p},\delta\mathbf{x}_0) &= \mathbf{B}^{-1}\left(\mathbf{x}_0^{\mathsf{i}} - \mathbf{x}_0^{\mathsf{b}} + \delta\mathbf{x}_0\right) \\
&\quad - \sum_{k=0}^{L} \mathbf{M}^{\mathsf{x}}_{k:0}{}^{T}\mathbf{H}_k^T\mathbf{R}_k^{-1}\left(\mathbf{d}_k - \mathbf{H}_k\mathbf{M}_{k:0}\begin{pmatrix}\delta\mathbf{p}\\ \delta\mathbf{x}_0\end{pmatrix}\right) \\
\nabla_{\delta\mathbf{p}}\widehat{\mathcal{J}}(\delta\mathbf{p},\delta\mathbf{x}_0) &= \mathbf{P}^{-1}\left(\mathbf{p}^{\mathsf{i}} - \mathbf{p}^{\mathsf{b}} + \delta\mathbf{p}\right) \\
&\quad - \sum_{k=0}^{L} \mathbf{M}^{\mathsf{p}}_{k:0}{}^{T}\mathbf{H}_k^T\mathbf{R}_k^{-1}\left(\mathbf{d}_k - \mathbf{H}_k\mathbf{M}_{k:0}\begin{pmatrix}\delta\mathbf{p}\\ \delta\mathbf{x}_0\end{pmatrix}\right).
\end{aligned}$$

Remember the lecture yesterday on the 4DVar: we will change the loop on the observations to use $\mathbf{M}_{k:k-1}$ instead of $\mathbf{M}_{k:0}$

Bridging machine learning and data assimilation under the common 4D-Var framework

Gradient of the incremental cost function $\widehat{\mathcal{J}}$.

Input: $\delta\mathbf{p}$ and $\delta\mathbf{x}_0$

1: $\mathbf{z}_0 \leftarrow \mathbf{R}_0^{-1}\left(\mathbf{H}_0\delta\mathbf{x}_0 - \mathbf{d}_0\right)$
2: **for** $k = 1$ **to** L **do**
3: $\quad \delta\mathbf{x}_k \leftarrow \mathbf{M}_{k:k-1}\left(\delta\mathbf{p}, \delta\mathbf{x}_{k-1}\right)^{\top}$ ▷ TL of the dynamical model $\boldsymbol{\mathcal{M}}_{k:k-1}$
4: $\quad \mathbf{z}_k \leftarrow \mathbf{R}_k^{-1}\left(\mathbf{H}_k\delta\mathbf{x}_k - \mathbf{d}_k\right)$
5: **end for**

Bridging machine learning and data assimilation under the common 4D-Var framework

Gradient of the incremental cost function $\widehat{\mathcal{J}}$.

Input: $\delta \mathbf{p}$ and $\delta \mathbf{x}_0$

$\mathbf{z}_0 \leftarrow \mathbf{R}_0^{-1} \left(\mathbf{H}_0 \delta \mathbf{x}_0 - \mathbf{d}_0 \right)$

for $k = 1$ **to** L **do**

  $\delta \mathbf{x}_k \leftarrow \mathbf{M}_{k:k-1} \left(\delta \mathbf{p}, \delta \mathbf{x}_{k-1} \right)^\top$ ▷ TL of the dynamical model $\boldsymbol{\mathcal{M}}_{k:k-1}$

  $\mathbf{z}_k \leftarrow \mathbf{R}_k^{-1} \left(\mathbf{H}_k \delta \mathbf{x}_k - \mathbf{d}_k \right)$

end for

$\delta \tilde{\mathbf{x}}_L \leftarrow \mathbf{0}$ ▷ AD variable for system state

$\delta \tilde{\mathbf{p}}_L \leftarrow \mathbf{0}$ ▷ AD variable for model parameters

Bridging machine learning and data assimilation under the common 4D-Var framework

Gradient of the incremental cost function $\widehat{\mathcal{J}}$.

Input: $\delta\mathbf{p}$ and $\delta\mathbf{x}_0$

$\mathbf{z}_0 \leftarrow \mathbf{R}_0^{-1}\left(\mathbf{H}_0\delta\mathbf{x}_0 - \mathbf{d}_0\right)$

for $k = 1$ **to** L **do**

  $\delta\mathbf{x}_k \leftarrow \mathbf{M}_{k:k-1}\left(\delta\mathbf{p}, \delta\mathbf{x}_{k-1}\right)^\top$ ▷ TL of the dynamical model $\boldsymbol{\mathcal{M}}_{k:k-1}$

  $\mathbf{z}_k \leftarrow \mathbf{R}_k^{-1}\left(\mathbf{H}_k\delta\mathbf{x}_k - \mathbf{d}_k\right)$

end for

$\delta\tilde{\mathbf{x}}_L \leftarrow \mathbf{0}$ ▷ AD variable for system state

$\delta\tilde{\mathbf{p}}_L \leftarrow \mathbf{0}$ ▷ AD variable for model parameters

for $k = L$ **to** 1 **do**

  $\delta\tilde{\mathbf{x}}_k \leftarrow \mathbf{H}_k^\top\mathbf{z}_k + \delta\tilde{\mathbf{x}}_k$

  $\left(\delta\tilde{\mathbf{p}}_{k-1}, \delta\tilde{\mathbf{x}}_{k-1}\right)^\top \leftarrow \left(\mathbf{M}_{k:k-1}\right)^\top\delta\tilde{\mathbf{x}}_k + \left(\delta\tilde{\mathbf{p}}_k, \mathbf{0}\right)^\top$ ▷ AD of the dyn. model $\boldsymbol{\mathcal{M}}_{k:k-1}$

end for

$\delta\tilde{\mathbf{x}}_0 \leftarrow \mathbf{H}_0^\top\mathbf{z}_0 + \delta\tilde{\mathbf{x}}_0$

$\delta\tilde{\mathbf{x}}_0 \leftarrow \mathbf{B}^{-1}\left(\mathbf{x}_0^{\mathsf{i}} - \mathbf{x}_0^{\mathsf{b}} + \delta\mathbf{x}_0\right) + \delta\tilde{\mathbf{x}}_0$

$\delta\tilde{\mathbf{p}} \leftarrow \mathbf{P}^{-1}\left(\mathbf{p}^{\mathsf{i}} - \mathbf{p}^{\mathsf{b}} + \delta\mathbf{p}\right) + \delta\tilde{\mathbf{p}}_0$

Output: $\nabla_{\delta\mathbf{p}}\widehat{\mathcal{J}} = \delta\tilde{\mathbf{p}}$ and $\nabla_{\delta\mathbf{x}_0}\widehat{\mathcal{J}} = \delta\tilde{\mathbf{x}}_0$

Tangent linear and adjoint of a simple Multi Layer Perceptron (MLP) neural network

Let's consider that our state vector $\mathbf{x}$ is a scalar x and that the dynamical model $\mathcal{M}$ is a Multi Layer Perceptron with a single hidden layer:

$$x_{k+1} = \mathcal{M}_{k+1:k}\left(\mathbf{p}, x_k\right) = f^1(w^1 x_k + b^1).$$

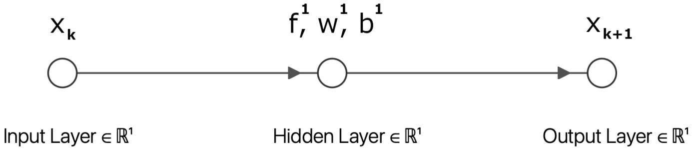

Figure: Simple multi-layer perceptron (MLP).

Tangent linear and adjoint of a simple Multi Layer Perceptron (MLP) neural network

$$x_{k+1} = \mathcal{M}_{k+1:k}\left(\mathbf{p}, x_k\right) = f^1(w^1 x_k + b^1).$$

NL:

Input: x_k, w^1, b^1

$a^0 = x_k$

$z^1 = w^1 a^0 + b^1$

$a^1 = f^1(z^1)$

$x_{k+1} = a^1$

Output: x_{k+1}

we will use $(f^1)'|_{z^1}$ as the derivative of f^1 with regards to z^1

TL:

AD:

Tangent linear and adjoint of a simple Multi Layer Perceptron (MLP) neural network

$$x_{k+1} = \mathcal{M}_{k+1:k}(\mathbf{p}, x_k) = f^1(w^1 x_k + b^1).$$

NL:

Input: x_k, w^1, b^1

$$a^0 = x_k$$
$$z^1 = w^1 a^0 + b^1$$
$$a^1 = f^1(z^1)$$
$$x_{k+1} = a^1$$

Output: x_{k+1}

we will use $(f^1)'|_{z^1}$ as the derivative of f^1 with regards to z^1

TL:

Input: $\delta x_k, \delta w^1, \delta b^1, x_k, w^1, b^1$

$$\delta a^0 = \delta x_k$$
$$\delta z^1 = w^1 \delta a^0 + a^0 \delta w^1 + \delta b^1$$
$$\delta a^1 = (f^1)'|_{z^1} \delta z^1$$
$$\delta x_{k+1} = \delta a^1$$

Output: δx_{k+1}

Note: we need to recompute z^1 inside the TL/AD for $(f^1)'|_{z^1}$ (not shown)

AD:

Tangent linear and adjoint of a simple Multi Layer Perceptron (MLP) neural network

$$x_{k+1} = \mathcal{M}_{k+1:k}(\mathbf{p}, x_k) = f^1(w^1 x_k + b^1).$$

NL:

Input: x_k, w^1, b^1

$$a^0 = x_k$$
$$z^1 = w^1 a^0 + b^1$$
$$a^1 = f^1(z^1)$$
$$x_{k+1} = a^1$$

Output: x_{k+1}

we will use $(f^1)'|_{z^1}$ as the derivative of f^1 with regards to z^1

TL:

Input: δx_k, δw^1, δb^1, x_k, w^1, b^1

$$\delta a^0 = \delta x_k$$
$$\delta z^1 = w^1 \delta a^0 + a^0 \delta w^1 + \delta b^1$$
$$\delta a^1 = (f^1)'|_{z^1} \delta z^1$$
$$\delta x_{k+1} = \delta a^1$$

Output: δx_{k+1}

Note: we need to recompute z^1 inside the TL/AD for $(f^1)'|_{z^1}$ (not shown)

AD:

Input: $\delta \tilde{x}_{k+1}$, x_k, w^1, b^1

$$\delta \tilde{a}^1 = \delta \tilde{a}^0 = \delta \tilde{z}^1 = 0$$
$$\delta \tilde{a}^1 = \delta \tilde{a}^1 + \delta \tilde{x}_{k+1}$$
$$\delta \tilde{x}_{k+1} = 0$$
$$\delta \tilde{z}^1 = \delta \tilde{z}^1 + (f^1)'|_{z^1} \delta \tilde{a}^1$$
$$\delta \tilde{a}^1 = 0$$
$$\delta \tilde{a}^0 = \delta a^0 + w^1 \delta \tilde{z}^1$$
$$\delta \tilde{w}^1 = \delta \tilde{w}^1 + a^0 \delta \tilde{z}^1$$
$$\delta \tilde{b}^1 = \delta \tilde{b}^1 + \delta \tilde{z}^1$$
$$\delta \tilde{z}^1 = 0$$
$$\delta \tilde{x}_k = \delta \tilde{x}_k + \delta \tilde{a}^0$$
$$\delta \tilde{a}^0 = 0$$

Output: $\delta \tilde{x}_k, \delta \tilde{w}^1, \delta \tilde{b}^1$

General MLP neural network and its tangent linear and adjoint

Let's consider now that our state vector $\mathbf{x}$ is a vector of dimension n and that the model $\mathcal{M}$ is a Multi Layer Perceptron with L layers:

$$\mathbf{x}_{k+1} = \mathcal{M}_{k+1:k}\left(\mathbf{p}, \mathbf{x}_k\right) = \mathbf{f^L}\left(\mathbf{W^L f^{L-1}}\left(\mathbf{W^{L-1}}\right) \ldots \mathbf{f^2}\left(\mathbf{W^2 f^1}\left(\mathbf{W^1 a^0}\right)\right) \ldots\right)$$

where $\mathbf{a^0} = \begin{bmatrix} \mathbf{x}_k \\ 1 \end{bmatrix} = \mathbf{P}\mathbf{x}_k$ and $\mathbf{W^i} = \begin{bmatrix} w^i_{1,1} & \ldots & w^i_{1,n_{i-1}} & b^i_1 \\ \vdots & \ddots & \vdots & \vdots \\ w^i_{n_i,1} & \ldots & w^i_{n_i,n_{i-1}} & b^i_{n_i} \end{bmatrix}$

General MLP neural network and its tangent linear and adjoint

$$\mathbf{x}_{k+1} = \boldsymbol{\mathcal{M}}_{k+1:k}\left(\mathbf{p}, \mathbf{x}_k\right) = \mathbf{f^L}\left(\mathbf{W^L}\mathbf{f^{L-1}}\left(\mathbf{W^{L-1}}\right)\ldots\mathbf{f^2}\left(\mathbf{W^2}\mathbf{f^1}\left(\mathbf{W^1}\mathbf{a^0}\right)\right)\ldots\right)$$

TASK: write NL, TL and ADJ pseudo-code

- write NL, TL and ADJ pseudo-code
- follow the notation where z^l is the input to the $l-th$ hidden layer and a^l is the output of the $l-th$ hidden layer
- make use of a loop structure

General MLP neural network and its tangent linear and adjoint

$$\mathbf{x}_{k+1} = \mathcal{M}_{k+1:k}\left(\mathbf{p}, \mathbf{x}_k\right) = \mathbf{f}^{\mathbf{L}}\left(\mathbf{W}^{\mathbf{L}}\mathbf{f}^{\mathbf{L}-\mathbf{1}}\left(\mathbf{W}^{\mathbf{L}-\mathbf{1}}\right)\ldots\mathbf{f}^{\mathbf{2}}\left(\mathbf{W}^{\mathbf{2}}\mathbf{f}^{\mathbf{1}}\left(\mathbf{W}^{\mathbf{1}}\mathbf{a}^{\mathbf{0}}\right)\right)\ldots\right)$$

NL:

Input: $\mathbf{x}_k, \mathbf{W}^i$

$\mathbf{a}^0 = \mathbf{P}\mathbf{x}_k$

for $k = 1$ to L **do**

$\mathbf{z}^i = \mathbf{W}^i\mathbf{a}^{i-1}$

$\mathbf{a}^i = \mathbf{f}^i(\mathbf{z}^i)$

end for

$\mathbf{x}_{k+1} = \mathbf{a}^L$

Output: $\mathbf{x}_{k+1}$

TL:

AD:

General MLP neural network and its tangent linear and adjoint

$$\mathbf{x}_{k+1} = \mathcal{M}_{k+1:k}\left(\mathbf{p}, \mathbf{x}_k\right) = \mathbf{f}^{\mathbf{L}}\left(\mathbf{W}^{\mathbf{L}}\mathbf{f}^{\mathbf{L-1}}\left(\mathbf{W}^{\mathbf{L-1}}\right)\ldots\mathbf{f}^{\mathbf{2}}\left(\mathbf{W}^{\mathbf{2}}\mathbf{f}^{\mathbf{1}}\left(\mathbf{W}^{\mathbf{1}}\mathbf{a}^{\mathbf{0}}\right)\right)\ldots\right)$$

NL:

Input: $\mathbf{x}_k, \mathbf{W}^i$
$\mathbf{a}^0 = \mathbf{P}\mathbf{x}_k$
for $k = 1$ to L **do**
$\mathbf{z}^i = \mathbf{W}^i\mathbf{a}^{i-1}$
$\mathbf{a}^i = \mathbf{f}^i(\mathbf{z}^i)$
end for
$\mathbf{x}_{k+1} = \mathbf{a}^L$
Output: $\mathbf{x}_{k+1}$

TL:

Input: $\delta\mathbf{x}_k, \delta\mathbf{W}^{\mathbf{i}}, \mathbf{x}_k$, W^i
$\delta\mathbf{a}^0 = \mathbf{P}\delta\mathbf{x}_k$
for $k = 1$ to L **do**
$\delta\mathbf{z}^i = \mathbf{W}^i\delta\mathbf{a}^{i-1} + \mathbf{a}^{i-1}\delta\mathbf{W}^i$
$\delta\mathbf{a}^i = (\mathbf{f}^i)'|_{\mathbf{z}^i}\delta\mathbf{z}^i$
end for
$\delta\mathbf{x}_{k+1} = \delta\mathbf{a}^L$
Output: $\delta\mathbf{x}_{k+1}$

AD:

General MLP neural network and its tangent linear and adjoint

$$\mathbf{x}_{k+1} = \mathcal{M}_{k+1:k}(\mathbf{p}, \mathbf{x}_k) = \mathbf{f^L}\left(\mathbf{W^L f^{L-1}}\left(\mathbf{W^{L-1}}\right)\dots \mathbf{f^2}\left(\mathbf{W^2 f^1}\left(\mathbf{W^1 a^0}\right)\right)\dots\right)$$

NL:

Input: $\mathbf{x}_k, \mathbf{W}^i$
$\mathbf{a}^0 = \mathbf{P}\mathbf{x}_k$
for $k = 1$ to L **do**
$\mathbf{z}^i = \mathbf{W}^i \mathbf{a}^{i-1}$
$\mathbf{a}^i = \mathbf{f}^i(\mathbf{z}^i)$
end for
$\mathbf{x}_{k+1} = \mathbf{a}^L$
Output: $\mathbf{x}_{k+1}$

TL:

Input: $\delta\mathbf{x}_k, \delta\mathbf{W^i}, \mathbf{x}_k, W^i$
$\delta\mathbf{a}^0 = \mathbf{P}\delta\mathbf{x}_k$
for $k = 1$ to L **do**
$\delta\mathbf{z}^i = \mathbf{W}^i \delta\mathbf{a}^{i-1} + \mathbf{a}^{i-1}\delta\mathbf{W}^i$
$\delta\mathbf{a}^i = (\mathbf{f}^i)'|_{\mathbf{z}^i}\delta\mathbf{z}^i$
end for
$\delta\mathbf{x}_{k+1} = \delta\mathbf{a}^L$
Output: $\delta\mathbf{x}_{k+1}$

AD:

Input: $\delta\tilde{\mathbf{x}}_{k+1}, \mathbf{x}_k, W^i$
$\delta\tilde{\mathbf{a}}^L = \delta\tilde{\mathbf{a}}^L + \delta\tilde{\mathbf{x}}_{k+1}$
$\delta\tilde{\mathbf{x}}_{k+1} = 0$
for $k = L$ to 1 **do**
$\delta\tilde{\mathbf{z}}^i = \delta\tilde{\mathbf{z}}^i + (\mathbf{f}^i)'|_{\mathbf{z}^i}\delta\tilde{\mathbf{a}}^i$
$\delta\tilde{\mathbf{a}}^i = 0$
$\delta\tilde{\mathbf{a}}^{i-1} = \delta\mathbf{a}^{i-1} + \mathbf{W}^{iT}\delta\tilde{\mathbf{z}}^i$
$\delta\tilde{\mathbf{W}}^i = \delta\tilde{\mathbf{W}}^i + \mathbf{a}^{i-1,T}\delta\tilde{\mathbf{z}}^i$
$\delta\tilde{\mathbf{z}}^i = 0$
end for
$\delta\tilde{\mathbf{x}}_k = \delta\tilde{\mathbf{x}}_k + P^T\delta\tilde{\mathbf{a}}^0$
$\delta\tilde{\mathbf{a}}^0 = 0$
Output: $\delta\tilde{\mathbf{a}}^0, \delta\tilde{\mathbf{W}}^i$

Link between the adjoint coding and the backpropagation algorithm

$$\mathbf{x}_{k+1} = \mathcal{M}_{k+1:k}\left(\mathbf{p}, \mathbf{x}_k\right) = \mathbf{f^L}\left(\mathbf{W^L}\mathbf{f^{L-1}}\left(\mathbf{W^{L-1}}\right)\cdots\mathbf{f^2}\left(\mathbf{W^2}\mathbf{f^1}\left(\mathbf{W^1}\mathbf{a^0}\right)\right)\cdots\right)$$

Let's take the derivative of the neural network w.r.t. the input $\mathbf{x_k}$ and its parameters $\mathbf{W^i}$ of the i-th layer:

$$\frac{\mathbf{df^L}(\cdots)}{\mathbf{dx_k}} = \frac{\mathbf{df^L}}{\mathbf{da^L}} \cdot \frac{\mathbf{da^L}}{\mathbf{dz^L}} \cdot \frac{\mathbf{dz^L}}{\mathbf{da^{L-1}}} \cdot \frac{\mathbf{da^{L-1}}}{\mathbf{dz^{L-1}}} \cdot \ldots \cdot \frac{\mathbf{da^1}}{\mathbf{dz^1}} \cdot \frac{\partial \mathbf{z^1}}{\partial \mathbf{a_0}} \cdot \frac{\mathbf{da^0}}{\mathbf{dx_k}}$$

$$\frac{\mathbf{df^L}(\cdots)}{\mathbf{dW^i}} = \frac{\mathbf{df^L}}{\mathbf{da^L}} \cdot \frac{\mathbf{da^L}}{\mathbf{dz^L}} \cdot \frac{\mathbf{dz^L}}{\mathbf{da^{L-1}}} \cdot \frac{\mathbf{da^{L-1}}}{\mathbf{dz^{L-1}}} \cdot \ldots \cdot \frac{\mathbf{da^i}}{\mathbf{dz^i}} \cdot \frac{\partial \mathbf{z^i}}{\partial \mathbf{W^i}}$$

And evaluate the derivatives:

$$\frac{\mathbf{df^L}(\cdots)}{\mathbf{dx_k}} = (\mathbf{f}^L)'|_{\mathbf{z}^L} \cdot \mathbf{W^L} \cdot (\mathbf{f}^{L-1})'|_{\mathbf{z}^{L-1}} \cdot \mathbf{W^{L-1}} \cdot \ldots \cdot (\mathbf{f}^1)'|_{\mathbf{z}^1} \cdot \mathbf{W^1} \cdot \mathbf{P}$$

$$\frac{\mathbf{df^L}(\cdots)}{\mathbf{dW^i}} = (\mathbf{f}^L)'|_{\mathbf{z}^L} \cdot \mathbf{W^L} \cdot (\mathbf{f}^{L-1})'|_{\mathbf{z}^{L-1}} \cdot \mathbf{W^{L-1}} \cdot \ldots \cdot (\mathbf{f}^i)'|_{\mathbf{z}^i} \cdot \mathbf{W^i} \cdot \mathbf{a^{i-1}}$$

Link between the adjoint coding and the backpropagation algorithm

Note that we can write the Tangent Linear model as:

$$\delta \mathbf{x}_{k+1} = M_{k+1:k}\left(\delta \mathbf{p}, \delta \mathbf{x}_k\right) = \frac{\mathbf{df^L}\left(\cdots\right)}{\mathbf{dx_k}} \delta \mathbf{x_k} + \sum_{i=1}^{L} \frac{\mathbf{df^L}\left(\cdots\right)}{\mathbf{dW^i}} \delta \mathbf{W^i}$$

Recall our TL model derived using the line by line approach:

TL:

Input: $\delta \mathbf{x}_k, \delta \mathbf{W^i}$

$\delta \mathbf{a}^0 = \mathbf{P} \delta \mathbf{x}_k$

for $k = 1$ to L **do**

$\delta \mathbf{z}^i = \mathbf{W}^i \delta \mathbf{a}^{i-1} + \mathbf{a}^{i-1} \delta \mathbf{W}^i$

$\delta \mathbf{a}^i = (\mathbf{f}^i)' |_{\mathbf{z}^i} \delta \mathbf{z}^i$

end for

$\delta \mathbf{x}_{k+1} = \delta \mathbf{a}^L$

Link between the adjoint coding and the backpropagation algorithm

Recall that the gradient is the transpose of the derivative:

$$\begin{aligned}
\nabla_{\mathbf{x_k}} \mathbf{f^L}(\cdots) &= \mathbf{P}^T \cdot \left(\mathbf{W^1}\right)^T \cdot (\mathbf{f}^1)'|_{\mathbf{z}^1} \cdot \ldots \cdot \left(\mathbf{W^{L-1}}\right)^T \cdot (\mathbf{f}^{L-1})'|_{\mathbf{z}^{L-1}} \cdot \left(\mathbf{W^L}\right)^T \cdot (\mathbf{f}^L)'|_{\mathbf{z}^L} \\
\nabla_{\mathbf{W^i}} \mathbf{f^L}(\cdots) &= \left(\mathbf{a^{i-1}}\right)^T \left(\mathbf{W^i}\right)^T \cdot (\mathbf{f}^i)'|_{\mathbf{z}^i} \cdot \ldots \cdot \left(\mathbf{W^{L-1}}\right)^T \cdot (\mathbf{f}^{L-1})'|_{\mathbf{z}^{L-1}} \cdot \left(\mathbf{W^L}\right)^T \cdot (\mathbf{f}^L)'|_{\mathbf{z}^L} \\
&= \left(\mathbf{a^{i-1}}\right)^T \delta_{\mathbf{i}}
\end{aligned}$$

Note that $\delta_{\mathbf{i}}$ can be computed recursively going from the end to the beginning:

$$\begin{aligned}
\delta_{\mathbf{L}} &= (\mathbf{f}^L)'|_{\mathbf{z}^L} \\
\delta_{\mathbf{i-1}} &= (\mathbf{f}^{i-1})'|_{\mathbf{z}^{i-1}} \cdot \left(\mathbf{W^i}\right)^T \cdot \delta_{\mathbf{i}}
\end{aligned}$$

This allows for an efficient computation of the gradient of the neural network and is known as the **backpropagation** algorithm.

Link between the adjoint coding and the backpropagation algorithm

Finally note that we can write the adjoint model as:

$$\delta\tilde{\mathbf{x}}_{\mathbf{k}} = \nabla_{\mathbf{x_k}}\mathbf{f^L}(\cdots)\,\delta\tilde{\mathbf{x}}_{\mathbf{k+1}}$$
$$\delta\tilde{\mathbf{W}}_{\mathbf{i}} = \nabla_{\mathbf{W^i}}\mathbf{f^L}(\cdots)\,\delta\tilde{\mathbf{x}}_{\mathbf{k+1}}$$

where

$$\begin{aligned}\nabla_{\mathbf{x_k}}\mathbf{f^L}(\cdots) = & \mathbf{P}^T\cdot\left(\mathbf{W^1}\right)^T\cdot(\mathbf{f}^1)'|_{\mathbf{z}^1}\cdot\ldots\cdot\\ & \left(\mathbf{W^{L-1}}\right)^T\cdot(\mathbf{f}^{L-1})'|_{\mathbf{z}^{L-1}}\cdot\\ & \left(\mathbf{W^L}\right)^T\cdot(\mathbf{f}^L)'|_{\mathbf{z}^L}\end{aligned}$$

$$\begin{aligned}\nabla_{\mathbf{W^i}}\mathbf{f^L}(\cdots) = & \left(\mathbf{a^{i-1}}\right)^T\left(\mathbf{W^i}\right)^T\cdot(\mathbf{f}^i)'|_{\mathbf{z}^i}\cdot\ldots\cdot\\ & \left(\mathbf{W^{L-1}}\right)^T\cdot(\mathbf{f}^{L-1})'|_{\mathbf{z}^{L-1}}\cdot\\ & \left(\mathbf{W^L}\right)^T\cdot(\mathbf{f}^L)'|_{\mathbf{z}^L}\end{aligned}$$

Recall our implementation of the adjoint code: AD:

Input: $\delta\tilde{\mathbf{x}}_{k+1}$
- $\delta\tilde{\mathbf{a}}^L = \delta\tilde{\mathbf{a}}^L + \delta\tilde{\mathbf{x}}_{k+1}$
- $\delta\tilde{\mathbf{x}}_{k+1} = 0$
- **for** $k = L$ to 1 **do**
  - $\delta\tilde{\mathbf{z}}^i = \delta\tilde{\mathbf{z}}^i + (\mathbf{f}^i)'|_{\mathbf{z}^i}\delta\tilde{\mathbf{a}}^i$
  - $\delta\tilde{\mathbf{a}}^i = 0$
  - $\delta\tilde{\mathbf{a}}^{i-1} = \delta\mathbf{a}^{i-1} + \mathbf{W}^{iT}\delta\tilde{\mathbf{z}}^i$
  - $\delta\tilde{\mathbf{W}}^i = \delta\tilde{\mathbf{W}}^i + \mathbf{a}^{i-1,T}\delta\tilde{\mathbf{z}}^i$
  - $\delta\tilde{\mathbf{z}}^i = 0$
- **end for**
- $\delta\tilde{\mathbf{x}}_k = \delta\tilde{\mathbf{x}}_k + P^T\delta\tilde{\mathbf{a}}^0$
- $\delta\tilde{\mathbf{a}}^0 = 0$

Can you verify that we have effectively implemented the **backpropagation** algorithm?