Deep Learning (BEV033DLE)
Lecture 13 Recurrent Neural Networks

Czech Technical University in Prague

- Recurrent models
- Special cases and recurrent back propagation
- Error back propagation through time
- Gated recurrent units, GRU and LSTM networks
Recurrent networks

Recurrent models in a nutshell

- input sequence \( x = (x_1, \ldots, x_t, \ldots, x_T), x_t \in \mathbb{R}^n \). Similarly: output sequence \( y \) with elements \( y_t \) and sequence \( h \) of (hidden) states with elements \( h_t \in \mathbb{R}^d \). Often all three sequences have the same length.

- recurrent (dynamic) system with outputs

\[
\begin{align*}
  h_t &= f(x_t, h_{t-1}, w) \\
  y_t &= g(h_t, v)
\end{align*}
\]

where \( w \) and \( v \) are parameters. The model defines sequence mappings \( h = F_w(x) \) and \( y = G_v(h) \).

- loss function \( \ell(y, y') \); often locally additive \( \sum_t \ell(y_t, y'_t) \)

Training goal: given training data \( \mathcal{T} = \{(x^j, y^j) \mid j = 1, \ldots, m\} \), learn the model parameters \( w, v \) by solving

\[
\frac{1}{m} \sum_{x,y \in \mathcal{T}} \ell(y, (G_v \circ F_w)(x)) \rightarrow \min_{w,v}
\]
Recurrent networks

Incarnations of recurrent models and related tasks

- Deep neural network for classification with additional feedback connections. \( x \) - input, constant not depending on time. \( y \) - output of the network, network head, e.g. logsoftmax, \( h \) - states of all hidden layers. The loss function depends only on the last output \( y_T \).

- "infinite state automata": the output space is sufficient for keeping the history, thus \( h \) and \( y \) can be identified, i.e. \( y_t = f(x_t, y_{t-1}, w) \).

Example: landcover type monitoring for a geo-location: \( x \) - sequence of spectral satellite measurements, \( y \) - sequence of states (e.g. coniferous forest, broadleaf forest, clearcut, bark beetle degradation etc.)

- general sequence segmentation: hidden states \( h_t \) are needed for keeping track of longer past and are latent.

Examples: speech recognition, \( x \) - audio signal, \( y \) - sequence of words. NLP translation:
Learning RNNs special cases: infinite state automata

Learning RNNs is particularly simple in the case that

- $h$ and $y$ can be identified, i.e. $y_t = f(x_t, y_{t-1}, w)$ and
- the loss is locally additive $\sum_t \ell(y_t, y'_t)$

We can split the sequences $(x, y)$ from training data into triplets $(y_{t-1}, x_t, y_t)$ and train $f$ from

$$\frac{1}{m} \sum_{x, y \in \mathcal{T}} \sum_t \ell(y_t, f_w(x_t, y_{t-1})) \rightarrow \min_w$$

Neither forward nor backward propagation through the sequence are needed.

If the hidden states $h_t$ do not coincide with outputs $y_t$ and are latent, then learning becomes considerably more complicated.
Learning RNNs special cases: Recurrent backpropagation

Recall the following:

**Recurrence backpropagation:** (Almeida, 1987), (Pineda, 1987)

Learning approach for classifier/regression networks with *feedback connections*.

Denote: network input $x$, network output $y_t$ and $h_t$ denoting outputs of all hidden layers.

$$ h_t = f(x, h_{t-1}, w) \quad \text{and} \quad y_t = g(h_t, v) $$

**Assumption:** the network configuration $h_t$ converges to a fixpoint $h^*$ if we clamp its input to $x$. Computing $\nabla_v \ell$ poses no problem. What about $\nabla_w \ell$?

We have (implicit function theorem)

$$ \frac{\partial h^*}{\partial w} = \left[I - J_f(h^*)\right]^{-1} \frac{\partial f}{\partial w}, $$

where $J_f(h^*) = \frac{\partial f(x, w, h^*)}{\partial h}$ is the Jacobian of $f$ w.r.t. $h$.

Now, let us consider the gradient of the loss w.r.t. $w$.

$$ \partial_w \ell = \partial_y \ell \partial_{h^*} g \left[I - J_f(h^*)\right]^{-1} \partial_w f(x, w, h^*) $$

Applying this directly would require to compute $\left[I - J_f(h^*)\right]^{-1}$!
Learning RNNs special cases: Recurrent backpropagation

Now, introduce the (column) vector $z$

$$z = [I - J_f(h^*)]^{-1} \left( \frac{\partial \ell}{\partial y} \frac{\partial h^*}{\partial g} \right)^T$$

Multiplying both sides by $[I - J_f(h^*)]$, we get

$$z = J_f(h^*)^T z + \left( \frac{\partial \ell}{\partial y} \frac{\partial h^*}{\partial g} \right)^T.$$

This is a fixpoint equation for $z$ and can be solved by fixpoint iteration. The resulting algorithm for computing the derivative $\frac{\partial \ell}{\partial w}$ is:

- fix $x$, run the network until convergence $\rightarrow h^*$
- start from $z_0$ and iterate

$$z_i = J_f(h^*)^T z_{i-1} + \left( \frac{\partial \ell}{\partial y} \frac{\partial h^*}{\partial g} \right)^T$$

until convergence.
- Return

$$\frac{\partial \ell}{\partial w} = z^T \frac{\partial f(x,w,h^*)}{\partial h}$$
Learning RNNs general case: backpropagation through time

Assumptions:

\[ h_t = f(x_t, h_{t-1}, w) \]
\[ y_t = g(h_t, v) \]

The mappings \( f \) and \( g \) are implemented by neural networks and are differentiable w.r.t. their inputs and parameters. The loss function \( \ell(y, y') \) is differentiable.

**Example 1.** Both mappings \( f \) and \( g \) are implemented by one layer networks

\[ a_t = Wh_t + Ux_t + b \]
\[ o_t = Vh_t + c \]
\[ h_t = \tanh(a_t) \]
\[ y_t = \text{softmax}(o_t) \]
Learning RNNs general case: backpropagation through time

**Computing the gradients:** Unroll the network in time and apply backpropagation.

Let us consider the loss for a single example \((x, y^*)\) from the training data.

Computing the gradient w.r.t. \(v\) is easy (see Slide 4.). Let us consider the gradient w.r.t. \(w\)

\[
\partial_w L(y^*, y) = \sum_{t=1}^{T} \partial_w \ell(y^*_t, y_t) = \sum_{t=1}^{T} \partial_{y_t} \ell(y^*_t, y_t) \partial_{h_t} g(h_t, v) \partial_w h_t
\]

The first two terms are simple. For the last one we have the recurrent expression

\[
\partial_w h_t = \partial_w f(x_t, h_{t-1}, w) + \partial_{h_{t-1}} f(x_t, h_{t-1}, w) \partial_w h_{t-1}
\]

This gives

\[
\partial_w h_t = \partial_w f(x_t, h_{t-1}, w) + \sum_{i=1}^{t-1} \left[ \prod_{j=i+1}^{t} \partial_{h_{j-1}} f(x_j, h_{i-1}, w) \right] \partial_w f(x_i, h_{i-1}, w)
\]
Learning RNNs general case: backpropagation through time

Problems:

- backpropagation through time is computationally expensive

- Exploding/vanishing gradients: consider for simplicity the linear recurrence $h_t = Wh_{t-1}$. For $\tau$ steps we get $h_\tau = W^\tau h_0$. Suppose that we can write $W = U^{-1}\Lambda U$, where $\Lambda$ is diagonal. We get
  $$h_\tau = U^{-1}\Lambda^\tau U h_0.$$  
  Eigenvalues with magnitude less than one will decay and eigenvalues with magnitude greater than one will explode.

- We can not apply batch normalisation as simple remedy.

- We want the following model ability: events long in the past can trigger changes in conjunction with current measurements.

- skip connections?, designate special nodes in $h_t$ for keeping record of events long in the past?
RNNs with gated recurrent units

LSTM (Hochreiter, Schmidhuber, 1997), GRU (Cho et al., 2014), ...

Gated recurrent unit (simplified):

A cell consisting of a recurrent unit $h_t$ and a gate unit $u_t \in [0, 1]

\begin{align*}
    h_t &= u_{t-1}h_{t-1} + [1 - u_{t-1}]f(x_t, h_{t-1}, w) \\
    u_t &= S(x_t, h_t, v)
\end{align*}

The gate unit $u_t$ has sigmoid nonlinearity and “decides” whether to copy $h_t$ from $h_{t-1}$ or to apply the recurrence with $f$. 
RNNs with gated recurrent units

Gated recurrent unit (general):

- $h$ is a state vector
- $u$ is a vector of “update” gates
- $r$ is a vector of “reset” gates

The update equations are

$$h_t = u_{t-1} \odot h_{t-1} + [1 - u_{t-1}] \odot S(Ux_{t-1} + Wr_{t-1} \odot h_{t-1})$$

where $\odot$ denotes the element-wise product of vectors. The gate unit outputs are given by

$$u_t = S(U^u x_t + W^u h_t)$$
$$r_t = S(U^r x_t + W^r h_t)$$

LSTM cells are somewhat more complicated – they have separate “forget” and “update” gates.