Course progress

- Previously
  - PCA: dimensionality reduction (simple unsupervised learning)
  - OLS, ridge regression, conjugate gradient
  - Convex optimization, linear programming, Lagrange multipliers, duality and minimax games
  - Sparse regression (LASSO); NMF, Sparse PCA,
  - Gradient descent, dual ascent, dual decomposition, augmented Lagrangians, ADMM
  - Random sampling and randomized QR and SVD factorizations
  - Compressed sensing and matrix completion
  - DFT and FFT, shift invariant and circulant matrices/2D Fourier transform/filters
  - Graphs and their matrix representation, clustering
  - *Stochastic gradient descent, classification models and neural networks, CNNs, Backprop, hyperparameters*
- Today: RNNs, Vanishing gradient, LSTM, GRU

NN models

- Fully connected NN
- Learning images
  - CNN
- Learning sequences
  - *RNN, LSTM, GRU*
- Learning graphs
  - GNN

NN models

- For the labelled data $(v_1, y_1) \dots (v_n, y_n)$ for $v_i \in \mathbb{R}^m$ and labels y_i
$$L(x) = \frac{1}{n} \sum_{i=1}^{n} \ell_i(x)$$
- the loss function can be e.g., cross-entropy loss for k-category classification
$$\ell_i(x) = \sum_{j=1}^{k} y_{j,i} \log F_j(x, v_i)$$
where $F(x, v)$ is given by the neural network and x represent all of its parameters.

Gradient computation

- By the univariate chain rule
$$\frac{\partial}{\partial x_j}\ell = \frac{\partial}{\partial F}\ell(F)\frac{\partial F}{\partial x_j}$$
- Therefore
$$\nabla \ell(x) = \frac{\partial}{\partial F}\ell(F(x))DF(x)$$
where $\ell(x) := \ell_i(x)$ also depends on i-th true label y_i and $F(x) := F(x, v_i)$ also depends on the i-th feature vector v_i
- But since we're not differentiating with respect to y_i or v_i, we just treat them as parameters.

Chain rule

- The chain rules in the multivariate setting is $D(f \circ g) = Df \circ Dg$, so
$$D_A F = \frac{\partial}{\partial A}[R(Av + b)] = \frac{\partial R}{\partial u} \circ \frac{\partial}{\partial A}(Av + b)$$
for $u = Av + b$
- Here $\frac{\partial R}{\partial u}$ is just the Jacobian matrix
$$\begin{bmatrix} \frac{\partial R_1}{\partial u_1} & \cdots & \frac{\partial R_1}{\partial u_n} \\ \vdots & \cdots & \vdots \\ \frac{\partial R_m}{\partial u_1} & \cdots & \frac{\partial R_m}{\partial u_n} \end{bmatrix}$$
- The derivative w/r/t the matrix A is an order 3 tensor given by
$$\frac{\partial R_i}{\partial A_{jk}} = \sum_t \frac{\partial R_i}{\partial u_t} \frac{\partial u_t}{\partial A_{jk}} = \sum_t \frac{\partial R_i}{\partial u_t} v_k \delta_{tj} = \frac{\partial R_i}{\partial u_j} v_k$$
- In this fashion can automatically compute all the derivatives going backwards (called autodiff or backprop)

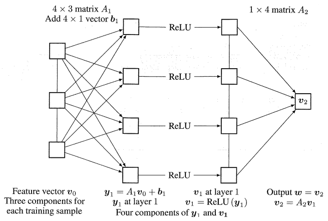
Chain rule

- For the fully connected NN

$$F = v_2 = A_2 v_1 + b_2 = A_2(R(A_1 v_0 + b_1)) + b_2$$

- By the chain rule

$$\frac{\partial F}{\partial (A_1)_{jk}} = A_2 \frac{\partial R}{\partial A_{jk}} = A_2 \frac{\partial R}{\partial u_j} v_k$$



4×3 matrix A_1
Add 4×1 vector $\boldsymbol{b}_1$
1×4 matrix A_2
ReLU
ReLU
ReLU
ReLU
$\boldsymbol{v}_2$
Feature vector $\boldsymbol{v}_0$
Three components for each training sample
$\boldsymbol{y}_1 = A_1\boldsymbol{v}_0 + \boldsymbol{b}_1$
$\boldsymbol{y}_1$ at layer 1
$\boldsymbol{v}_1$ at layer 1
$\boldsymbol{v}_1 = \text{ReLU}(\boldsymbol{y}_1)$
Four components of $\boldsymbol{y}_1$ and $\boldsymbol{v}_1$
Output $\boldsymbol{w} = \boldsymbol{v}_2$
$\boldsymbol{v}_2 = A_2\boldsymbol{v}_1$

Figure: Fig VII.2 from [1]

RNN

- Train to predict the next element in the sequence

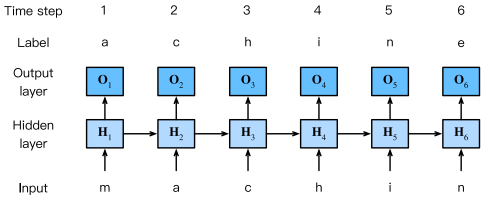
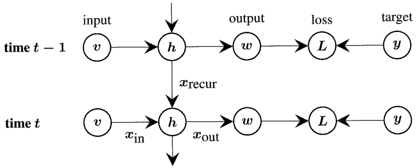


Figure: Fig 8.4.2 from [4]

RNN

- For sequences,
$$v = (v_1, ..., v^T)$$
like text or speech, want the predictions at t to depend on predictions at $t-1$ and earlier
- RNN is given by
$$\begin{aligned} h_t &= \sigma(Av_t + Bh_{t-1} + b) \\ &= \sigma(Av_t + B\sigma(Av_{t-1} + Bh_{t-2} + b) + b) \\ F_t(v) = w_t &= \text{softmax}(Ch_t) \end{aligned}$$
where the matrices A, B and C are shared across time.
- Use the latent variable h_t to approximate the history
$$p(w_t | v_{t-1}, ..., v_1) \approx p(w_t | h_{t-1})$$



Deep RNN

- Stack the hidden layers

$$h_t^l = R(A^l v_t^l + B^l h_{t-1})$$

where the matrices A^l, B^l and V^l are shared across time but not layers.

RNN

- Train to predict the next element in the sequence
- Since the prediction $F_{t,i}$ is a function of the parameters [we can use the cross-entropy loss for example]

$$L_T(A,B,C,b) = \sum_{j \in sequences} \sum_{t=1}^{T} \ell(F_{t,j}(A,B,C,b), y_{t,j})$$

$$= [- \sum_{j \in sequences} \sum_{t=1}^{T} \sum_{i \in categories} y_{t,j,i} \log(F_{t,j,i})]$$

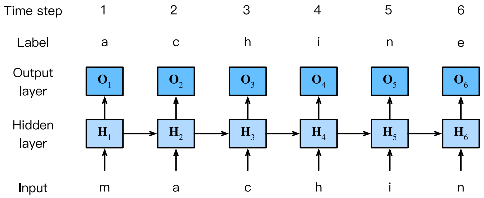


Figure: Fig 8.4.2 from [4]

RNN

- Let's fix the sequence and omit indexing by j.
- Since h_T depends on h_{T-1}, and h_{T-1} depends on h_{T-2} etc.

$$\begin{aligned}\frac{\partial}{\partial B}\ell(F_T) &= \frac{\partial \ell}{\partial F_T}\sum_{t=1}^{T}\frac{\partial F_T}{\partial h_t}\frac{\partial h_t}{\partial [B]^t} \\ &= \frac{\partial \ell}{\partial F_T}\frac{\partial F_T}{\partial h_T}\sum_{t=1}^{T}\Big(\prod_{s=t+1}^{T}\frac{\partial h_s}{\partial h_{s-1}}\Big)\frac{\partial h_t}{\partial [B]^t}\end{aligned}$$

where

$$\frac{\partial h_s}{\partial h_{s-1}} = \mathrm{diag}(\sigma'(a_t))B$$

and

$$a_t = Av_t + Bh_{t-1} + b$$

RNN

- Based on the previous slide
$$\frac{\partial F_T}{\partial h_t} = \frac{\partial F_T}{\partial h_T} \sum_{t=1}^{T} \Big(\prod_{s=t+1}^{T} \frac{\partial h_s}{\partial h_{s-1}} \Big)$$
where
$$\frac{\partial h_s}{\partial h_{s-1}} = \mathrm{diag}(\sigma'(a_s))B$$
- Often, the nonlinear activation functions have derivatives between 0 and 1, e.g.
$$0 < tanh'(x) \leq 1$$
- Thus,
$$\left\| \frac{\partial h_s}{\partial h_{s-1}} \right\| \leq \max_i \sigma'(a_s)_i \|B\| \leq \|B\|$$
and therefore
$$\left\| \frac{\partial F_T}{\partial h_t} \right\| = \left\| \frac{\partial F_T}{\partial h_T} \right\| \Big(\prod_{s=t+1}^{T} \left\| \frac{\partial h_s}{\partial h_{s-1}} \right\| \Big) \leq \left\| \frac{\partial F_T}{\partial h_T} \right\| \|B\|^{T-t}$$

Vanishing gradient

- Therefore,

$$\left\|\frac{\partial}{\partial B}\ell(F_T)\right\| \leq \left\|\frac{\partial \ell}{\partial F_T}\right\|\left\|\frac{\partial F_T}{\partial h_T}\right\|\sum_{t=1}^{T}\|B\|^{T-t}\left\|\frac{\partial h_t}{\partial [B]^t}\right\|$$

- If $\|B\| < 1$ during the training process, the gradient decreases
- You don't know whether this is because the learning has finished, or you can't propagate because of the vanishing gradient

Exploding gradient

- Actually less of a problem
- Won't confuse with the learning being finished
- Can be solved by gradient clipping - simply rescaled the gradient when it's norm exceeds a certain threshold
- Based on the practical observation that in SGD learning, you can't trust the norm of the gradient anyway (the direction of gradient is more reliable)

RNN

- Add linear shortcut connections

$$h_t = \sigma(Av_t + Bh_{t-1} + b) + \sum_{\tau=1}^{t-1} g_\tau [h_\tau]^t$$

- Then

$$\frac{\partial h_t}{\partial h_{t'}} = \prod_{s=1}^{t-t'} \frac{\partial h_{t-s+1}}{\partial h_{t-s}} + \sum_{\tau=1}^{t-1} \frac{\partial [h_\tau]^t}{\partial h_{t'}} \frac{\partial g_\tau [h_\tau]^t}{\partial [h_\tau]^t}$$

- This linear connection prevents vanishing gradient, but requires learning $g_\tau' s$, which can be computationally expensive for long sequences.

RNN+sparse update gates

- Instead of identity mappings, use

$$h_t = u_t \odot \sigma(Av_t + Bh_{t-1} + b) + (1 - u_t) \odot h_{t-1}$$

where the sparse update gates are

$$u_t = \sigma_u(A_u v_t + B_u h_{t-1} + b_u) \in [0, 1]^d$$

- Part of the learning is when to use sparse connections.

LSTM

- The above ideas led to LSTMs:

$$h_t = o_t \odot c_t$$

- the output gate determines how much of the previous states and the inputs (cell gate c^t) are exposed.

$$c_t = f_t \odot c_{t-1} + i_t \odot \tanh(A_c v_t + B_c c_{t-1} + b_c)$$

- the forget gate f_t is determines how much of the previous state is propagated linearly.

$$f_t = \sigma(A_f v_t + B_f c_{t-1} + V_f h_{t-1} + b_f)$$

- Parameters (weights) are “long-term memory”
- Previous outputs are “short-term memory”

LSTM

- Connections go back far in time
- Linear connections avoids vanishing gradients
- Minimize exploding gradients by sparsity
- h encodes the entire history and leads to embedding of sequences in $\mathbb{R}^d$

GRU

- Connections go back far in time
- Sparse connections essentially compress history in h_t
- Instead compute

$$h_t = \sum_{t'=1}^{t} w_{t'}(x_{t'}, x_t, t, t') \odot \hat{h}_{t'}(x_{t'}, t')$$

where $w_{t'}$ and $\hat{h}_{t'}$ depends on the position rather than history.

$$w_{t'}(x_{t'}, x_t, t, t') = \frac{\exp(Q(x_t, t)K(x_{t'}, t'))}{\sum_{t'=1}^{t} \exp Q(x_t, t)K(x_{t'}, t'))}$$

- Q is “query” and K is “key”

Next steps

- GNNs
- Kernel methods
- Bandits, RL
- Project presentations/Final quiz