UE Apprentissage SIA S9

Supervise Learning

Mathieu FAUVEL

CESBIO, Université de Toulouse, CNES/CNRS/INRAe/IRD/UPS, Toulouse, FRANCE

October 16, 2023

Outline

Outline

Myself

- Contact: mathieu.fauvel@inrae.fr
- CV:

2004-2007	Ph.D. degree in Signal and Image Processing from the INP, Grenoble & the University of Iceland
2007-2008	Assistant Professor Grenoble
2008-2010	Post-doc position at INRIA - MISTIS Team
2010-2011	Assistant Professor Toulouse
2011-2018	Associate Professor at DYNAFOR & INP, Toulouse
Since 2018	Research (CRCN) at CESBIO, INRAe

- Research interests are:

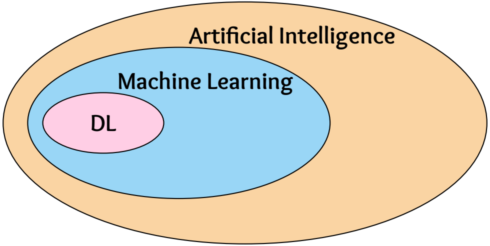
Machine learning for environmental/ecological monitoring

Course Objectives

- Learn basics of modern machine learning
- Understand how each step works
  - Data préparation
  - Model definition
  - Optimization step
- Implement various Deep Learning models in PyTorch
- Application to Computer Vision

What is Supervised (Machine) Learning ?

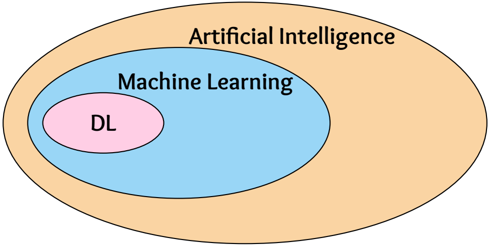
- **Artificial Intelligence**
  Perform human tasks using computers and algorithms



What is Supervised (Machine) Learning ?

- **Artificial Intelligence**
  Perform human tasks using computers and algorithms
- Mitchell, *The discipline of machine learning*:

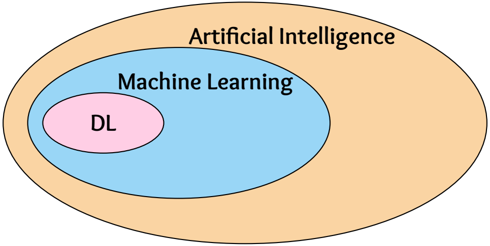
  Machine Learning is defined as the capacity of a computer program to improve its performance measure with observations



What is Supervised (Machine) Learning ?

- **Artificial Intelligence**
  Perform human tasks using computers and algorithms
- Mitchell, *The discipline of machine learning*:

  Machine Learning is defined as the capacity of a computer program to improve its performance measure with observations
- **Deep Learning**
  - Lot of data data
  - Complex model
  - High performance computing



Main Notations

Data

- Observed data $\mathbf{x} \in \mathbb{R}^d$, called *input variables* or *predictors*.
- Data to be predicted $y \in \mathbb{R}^p$, called *output variables* or *responses*.

Main Notations

Data

- Observed data $\mathbf{x} \in \mathbb{R}^d$, called *input variables* or *predictors*.
- Data to be predicted $y \in \mathbb{R}^p$, called *output variables* or *responses*.

Learning Problem

- If y are quantitative data: *regression* and $y \in \mathbb{R}^p$.
- If y are categorical data: *classification* and $y \in \{C_1, \ldots, C_C\}$ with C_i refers to the i^{th} category.

Main Notations

Data

- Observed data $\mathbf{x} \in \mathbb{R}^d$, called *input variables* or *predictors*.
- Data to be predicted $y \in \mathbb{R}^p$, called *output variables* or *responses*.

Learning Problem

- If y are quantitative data: *regression* and $y \in \mathbb{R}^p$.
- If y are categorical data: *classification* and $y \in \{C_1, \ldots, C_C\}$ with C_i refers to the i^{th} category.

Prediction function

$$\begin{aligned} f_{\boldsymbol{\theta}} : \mathbb{R}^d &\to \mathbb{R}^p \\ \mathbf{x} &\mapsto y \end{aligned}$$

Online References

- Aston Zhang et al. "Dive into Deep Learning". In: *arXiv preprint arXiv:2106.11342* (2021)
- Simon J.D. Prince. *Understanding Deep Learning*. MIT Press, 2023. URL: https://udlbook.github.io/udlbook/
- Sebastian Raschka, Yuxi (Hayden) Liu, and Vahid Mirjalili. *Machine Learning with PyTorch and Scikit-Learn*. Birmingham, UK: Packt Publishing, 2022. ISBN: 978-1801819312
- Kevin P. Murphy. *Probabilistic Machine Learning: An introduction*. MIT Press, 2022. URL: http://probml.github.io/book1
- Kevin P. Murphy. *Probabilistic Machine Learning: Advanced Topics*. MIT Press, 2023. URL: http://probml.github.io/book2

Outline

Outline

Univariate linear model

- $f(x) = wx + b$ and $\boldsymbol{\theta} = (w, b)$
- Loss function: $\ell\big(f(x_i), y_i\big) = \big(f(x_i) - y_i\big)^2$
- Objective function:

$$\mathcal{L} = \frac{1}{n}\sum_{i=1}^{n} \ell_i$$

- Gradients update: $\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^{t} - \eta \nabla_{\boldsymbol{\theta}} \mathcal{L}$

$$\frac{\partial \mathcal{L}}{\partial w} = \frac{2}{n}\sum_{i=1}^{n} x_i(wx_i + b - y_i)$$

$$\frac{\partial \mathcal{L}}{\partial b} = \frac{2}{n}\sum_{i=1}^{n} (wx_i + b - y_i)$$

Univariate linear model

- $f(x) = wx + b$ and $\boldsymbol{\theta} = (w, b)$
- Loss function: $\ell\big(f(x_i), y_i\big) = \big(f(x_i) - y_i\big)^2$
- Objective function:

$$\mathcal{L} = \frac{1}{n}\sum_{i=1}^{n} \ell_i$$

- Gradients update: $\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^{t} - \eta \nabla_{\boldsymbol{\theta}} \mathcal{L}$

$$\frac{\partial \mathcal{L}}{\partial w} = \frac{2}{n}\sum_{i=1}^{n} x_i(wx_i + b - y_i)$$

$$\frac{\partial \mathcal{L}}{\partial b} = \frac{2}{n}\sum_{i=1}^{n} (wx_i + b - y_i)$$

Work

Implement the 1D regression in pytorch of the following function:

$$f(x) = 2x - 1$$

Do the notebook:

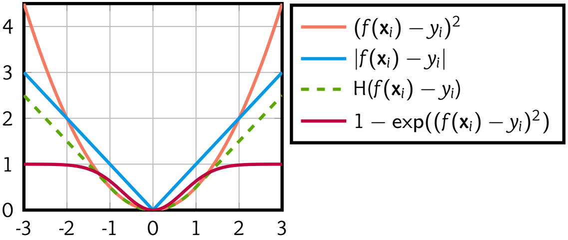
regression_1d_toy.ipynb

Multivariate linear model

- $f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$ and $\boldsymbol{\theta} = (\mathbf{w}, b)$
- Same loss and objective function and so same gradient updates ...

Multivariate linear model

- $f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$ and $\boldsymbol{\theta} = (\mathbf{w}, b)$
- Same loss and objective function and so same gradient updates ...
- We can use other loss function

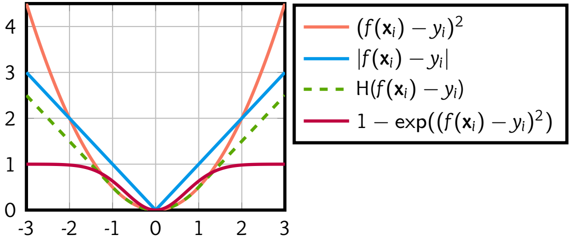


with H stands for *Hubert loss*:

$$H(f(\mathbf{x}_i) - y_i) = \begin{cases} \frac{1}{2}(f(\mathbf{x}_i - y_i))^2 & \text{for } |f(\mathbf{x}_i - y_i)| \leq \delta, \\ (|f(\mathbf{x}_i - y_i)| - 0.5)\,, & \text{otherwise.} \end{cases}$$

Multivariate linear model

- $f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$ and $\boldsymbol{\theta} = (\mathbf{w}, b)$
- Same loss and objective function and so same gradient updates ...
- We can use other loss function



with H stands for *Hubert loss*:

$$H(f(\mathbf{x}_i) - y_i) = \begin{cases} \frac{1}{2}(f(\mathbf{x}_i - y_i))^2 & \text{for } |f(\mathbf{x}_i - y_i)| \leq \delta, \\ (|f(\mathbf{x}_i - y_i)| - 0.5)\,, & \text{otherwise.} \end{cases}$$

Work

On the California housing data set

- Prepare (train/validation) data
- Define a multi. linear model
- Optimize your hyperparemeters (batch size, learning rate)
- Switch L2 to other loss function

Scaling feature for multivariate linear model with L2 loss function

- Suppose we have two features of different scale (e.g. because of different unit)

$$\mathbf{x}_1 \sim \mathcal{N}(0, 1) \text{ and } \mathbf{x}_2 \sim \mathcal{N}(10, 10)$$

- What is the impact on the gradient update ?

Scaling feature for multivariate linear model with L2 loss function

- Suppose we have two features of different scale (e.g. because of different unit)

$$\mathbf{x}_1 \sim \mathcal{N}(0,1) \text{ and } \mathbf{x}_2 \sim \mathcal{N}(10,10)$$

- What is the impact on the gradient update ?
- Noting $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_n]^\top$ and $\mathbf{y} = [y_1, \ldots, y_n]^\top$

$$\nabla_{\mathbf{w}} = -\mathbf{X}^\top \underbrace{(\mathbf{y} - \mathbf{X}\mathbf{w})}_{\mathbf{e}} \Rightarrow \qquad \nabla_{\mathbf{w}_p} = -\sum_{i=1}^{n} \mathbf{e}_i \mathbf{x}_{ip}$$

- Parameter update

$$\mathbf{w}_p^{(t+1)} = \mathbf{w}_p^{(t)} + \frac{\eta}{n} \sum_{i=1}^{n} \mathbf{e}_i^{(t)} \mathbf{x}_{ip}$$

Scaling feature for multivariate linear model with L2 loss function

- Suppose we have two features of different scale (e.g. because of different unit)

$$\mathbf{x}_1 \sim \mathcal{N}(0,1) \text{ and } \mathbf{x}_2 \sim \mathcal{N}(10,10)$$

- What is the impact on the gradient update ?
- Noting $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_n]^\top$ and $\mathbf{y} = [y_1, \ldots, y_n]^\top$

$$\nabla_{\mathbf{w}} = -\mathbf{X}^\top \underbrace{(\mathbf{y} - \mathbf{X}\mathbf{w})}_{\mathbf{e}} \Rightarrow \qquad \nabla_{\mathbf{w}_p} = -\sum_{i=1}^{n} \mathbf{e}_i \mathbf{x}_{ip}$$

- Parameter update

$$\mathbf{w}_p^{(t+1)} = \mathbf{w}_p^{(t)} + \frac{\eta}{n} \sum_{i=1}^{n} \mathbf{e}_i^{(t)} \mathbf{x}_{ip}$$

- Standardization of the input feature (and for regression the output too): scaling_feature.ipynb

$$\tilde{\mathbf{x}} = (\mathbf{x} - \boldsymbol{\mu}) \oslash \boldsymbol{\sigma}$$

Outline

Classification as a regression problem

- Predict categorical variables ("Dogs", "Cats", "Monkey" …)
- Estimation of the posterior probability

$$p(y_i = c|\mathbf{x}_i)\ \forall c \in \{1, \ldots, C\} \text{ with } p(y_i = c|\mathbf{x}_i) \geq 0 \text{ and } \sum_{c=1}^{C} p(y_i = c|\mathbf{x}_i) = 1$$

- MAP rule: $\hat{c}_i = \arg\max_c p(y_i = c|\mathbf{x}_i)$

Classification as a regression problem

- Predict categorical variables ("Dogs", "Cats", "Monkey" ...)
- Estimation of the posterior probability

$$p(y_i = c|\mathbf{x}_i)\ \forall c \in \left\{1, \ldots, C\right\} \text{ with } p(y_i = c|\mathbf{x}_i) \geq 0 \text{ and } \sum_{c=1}^{C} p(y_i = c|\mathbf{x}_i) = 1$$

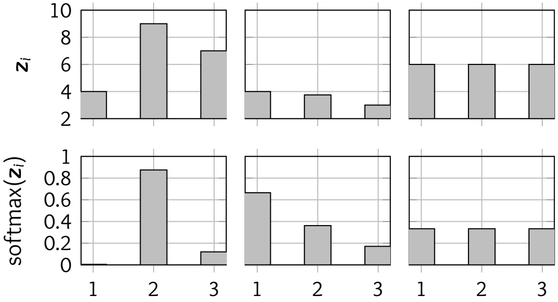
- MAP rule: $\hat{c}_i = \arg\max_c p(y_i = c|\mathbf{x}_i)$
- Softmax linear regression

$$1 = \sum_{c=1}^{C} p(y_i = c|\mathbf{x}_i) = \sum_{c=1}^{C} \exp\left\{\mathbf{w}_c^\top \mathbf{x}_i + b_c + K_i\right\} = \exp\{K_i\} \sum_{c=1}^{C} \exp\left\{\mathbf{w}_c^\top \mathbf{x}_i + b_c\right\}$$

thus

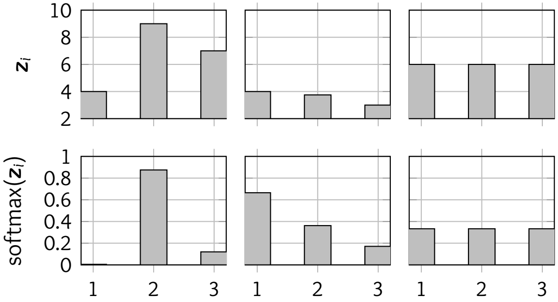
$$p(y_i = c|\mathbf{x}_i) = \frac{\exp\left\{\mathbf{w}_c^\top \mathbf{x}_i + b_c\right\}}{\sum_{c'=1}^{C} \exp\left\{\mathbf{w}_{c'}^\top \mathbf{x}_i + b_{c'}\right\}} = \frac{\exp(\mathbf{z}_{ic})}{\sum_{c'=1}^{C} \exp(\mathbf{z}_{ic'})} = \text{softmax}(\mathbf{z}_i)_c$$

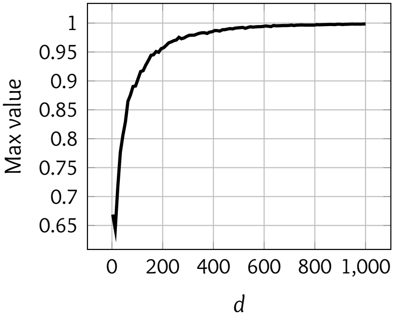
Softmax



- Translation invariant: $\mathrm{softmax}(\mathbf{z}+K)=\mathrm{softmax}(\mathbf{z})$
- Preserve ordering relation: $\arg\max_c(\mathbf{p}_i)=\arg\max_c(\mathbf{z}_i)$
- If $\mathbf{z}\in\mathbb{R}^d$ with $d\to\infty$ then $\mathrm{softmax}(\mathbf{z})\to$ "*One-hot-vector*"

Softmax





- Translation invariant: $\text{softmax}(\mathbf{z} + K) = \text{softmax}(\mathbf{z})$
- Preserve ordering relation: $\arg\max_c(\mathbf{p}_i) = \arg\max_c(\mathbf{z}_i)$
- If $\mathbf{z} \in \mathbb{R}^d$ with $d \to \infty$ then $\text{softmax}(\mathbf{z}) \to$ "*One-hot-vector*"

Cross entropy loss function

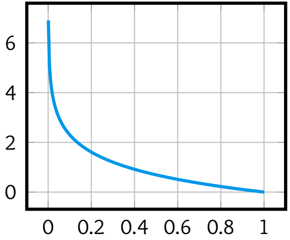
- One hot encoding: $y_i = c \rightarrow \mathbf{y}_i \in \mathbb{R}^C,\ \mathbf{y}_{ic} = 1$ if $i = c$ else 0
- Likelihood:

$$l(\mathbf{y}_i, \hat{\mathbf{y}}_i) = \prod_{c'=1}^{C} p(y_i = c' | \mathbf{x}_i)^{\mathbf{y}_{ic'}} = \frac{\exp(\mathbf{z}_{ic})}{\sum_{c'=1}^{C} \exp(\mathbf{z}_{ic'})}$$

Cross entropy loss function

- One hot encoding: $y_i = c \to \mathbf{y}_i \in \mathbb{R}^C,\ \mathbf{y}_{ic} = 1$ if $i = c$ else 0
- Likelihood:

$$l(\mathbf{y}_i, \hat{\mathbf{y}}_i) = \prod_{c'=1}^{C} p(y_i = c' | \mathbf{x}_i)^{\mathbf{y}_{ic'}} = \frac{\exp(\mathbf{z}_{ic})}{\sum_{c'=1}^{C} \exp(\mathbf{z}_{ic'})}$$



- Negative log likelihood:

$$\begin{aligned}\ell(\mathbf{y}_i, \hat{\mathbf{y}}_i) &= -\sum_{c'=1}^{C} \mathbf{y}_{ic'} \log\left\{p(y_i = c' | \mathbf{x}_i)\right\} \\ &= -\log\left\{p(y_i = c | \mathbf{x}_i)\right\} \\ &= -\left(\mathbf{w}_c^\top \mathbf{x}_i + b_c\right) + \underbrace{\log\left\{\sum_{c'=1}^{C} \exp(\mathbf{w}_{c'}^\top \mathbf{x}_i + b_{c'})\right\}}_{\text{logsumexp}(\mathbf{z}_i)}\end{aligned}$$

Work

What happens if:

- All $\mathbf{z}_{ic}$ are small ?
- One $\mathbf{z}_{ic}$ is very large ?
- All $\mathbf{z}_{ic}$ are very large ?

Multivariate linear classifier

- The model

$$\mathbf{p}_i = \text{softmax}\left(\mathbf{W}\mathbf{x}_i + \mathbf{b}\right)$$

- Cross entropy loss function
- Same comments than for regression
  - ⋆ Split train, val and test
  - ⋆ Standardizatiton
  - ⋆ Batch training
  - ⋆ Tune learning rate

Work

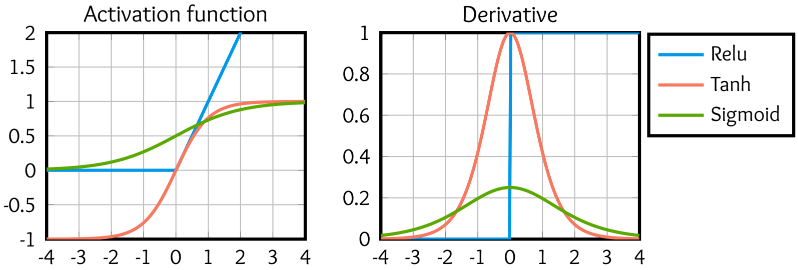
Implement a linear classifier in pytorch for the classification of Fashion MNIST data set.

Outline

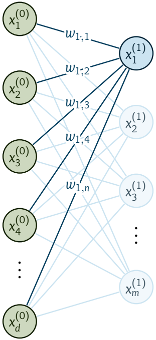
Outline

Activation function

- As discussed earlier we need non-lineraity between two layers
- Done with *activation function* between two layers
  - ⋆ Rectified Linear (ReLU): $\max(x, 0)$
  - ⋆ Sigmoid: $\dfrac{1}{1 + \exp(-x)}$
  - ⋆ Tanh: $\tanh(x)$



- $\mathbf{x}^{(1)} = \sigma\left(\mathbf{W}^{(0)}\mathbf{x}^{(0)} + \mathbf{b}^{(0)}\right)$
- $\mathbf{x}^{(2)} = \sigma\left(\mathbf{W}^{(1)}\mathbf{x}^{(1)} + \mathbf{b}^{(1)}\right)$
- $\ldots$
- $\mathbf{y} = \mathbf{W}^{H}\mathbf{x}^{H} + \mathbf{b}^{H}$



$$= \sigma\left(w_{1,0}x_0^{(0)} + w_{1,1}x_1^{(0)} + \ldots + w_{1,d}x_d^{(0)} + b_1^{(0)}\right)$$

$$= \sigma\left(\sum_{i=1}^{d} w_{1,i}x_i^{(0)} + b_1^{(0)}\right)$$

$$\begin{pmatrix} x_1^{(1)} \\ x_2^{(1)} \\ \vdots \\ x_m^{(1)} \end{pmatrix} = \sigma\left[\begin{pmatrix} w_{1,0} & w_{1,1} & \ldots & w_{1,d} \\ w_{2,0} & w_{2,1} & \ldots & w_{2,d} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m,0} & w_{m,1} & \ldots & w_{m,d} \end{pmatrix}\begin{pmatrix} x_1^{(0)} \\ x_2^{(0)} \\ \vdots \\ x_d^{(0)} \end{pmatrix} + \begin{pmatrix} b_1^{(0)} \\ b_2^{(0)} \\ \vdots \\ b_m^{(0)} \end{pmatrix}\right]$$

$$\mathbf{x}^{(1)} = \sigma\left(\mathbf{W}^{(0)}\mathbf{x}^{(0)} + \mathbf{b}^{(0)}\right)$$

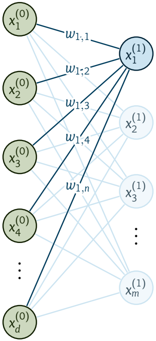
From https://tikz.net/neural_networks/

Multi Layer Perceptron (MLP)

- $\mathbf{x}^{(1)} = \sigma\left(\mathbf{W}^{(0)}\mathbf{x}^{(0)} + \mathbf{b}^{(0)}\right)$
- $\mathbf{x}^{(2)} = \sigma\left(\mathbf{W}^{(1)}\mathbf{x}^{(1)} + \mathbf{b}^{(1)}\right)$
- $\dots$
- $\mathbf{y} = \mathbf{W}^{H}\mathbf{x}^{H} + \mathbf{b}^{H}$

Work

- Why the last layer does not have non-linearity ?
- For an univariate regression problem, shows that a 2-layers MLP is a piece-wise linear function.
- Implement an MLP classifier for Fashion MNIST data set



$$= \sigma\left(w_{1,0}x_0^{(0)} + w_{1,1}x_1^{(0)} + \ldots + w_{1,d}x_d^{(0)} + b_1^{(0)}\right)$$

$$= \sigma\left(\sum_{i=1}^{d} w_{1,i}x_i^{(0)} + b_1^{(0)}\right)$$

$$\begin{pmatrix} x_1^{(1)} \\ x_2^{(1)} \\ \vdots \\ x_m^{(1)} \end{pmatrix} = \sigma\left[\begin{pmatrix} w_{1,0} & w_{1,1} & \dots & w_{1,d} \\ w_{2,0} & w_{2,1} & \dots & w_{2,d} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m,0} & w_{m,1} & \dots & w_{m,d} \end{pmatrix}\begin{pmatrix} x_1^{(0)} \\ x_2^{(0)} \\ \vdots \\ x_d^{(0)} \end{pmatrix} + \begin{pmatrix} b_1^{(0)} \\ b_2^{(0)} \\ \vdots \\ b_m^{(0)} \end{pmatrix}\right]$$

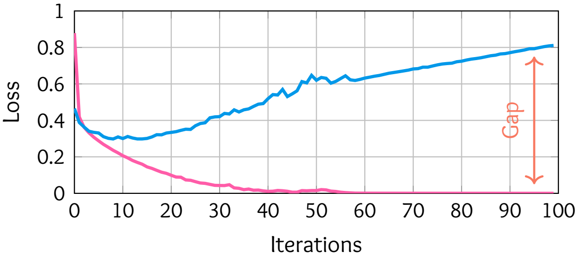
$$\mathbf{x}^{(1)} = \sigma\left(\mathbf{W}^{(0)}\mathbf{x}^{(0)} + \mathbf{b}^{(0)}\right)$$

From *https://tikz.net/neural_networks/*

Outline

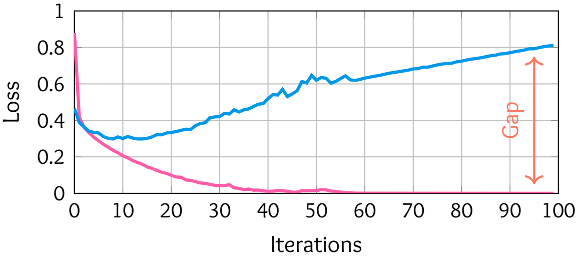
Generalization Error

- *Generalization error*: Difference between the train accuracy and val/test accuracy
- *Overfitting*: High generalization error



Generalization Error

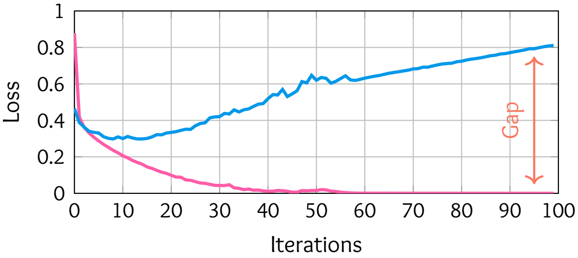
- *Generalization error*: Difference between the train accuracy and val/test accuracy
- *Overfitting*: High generalization error



Theory: favor simpler model and/or smooth model

Generalization Error

- *Generalization error*: Difference between the train accuracy and val/test accuracy
- *Overfitting*: High generalization error

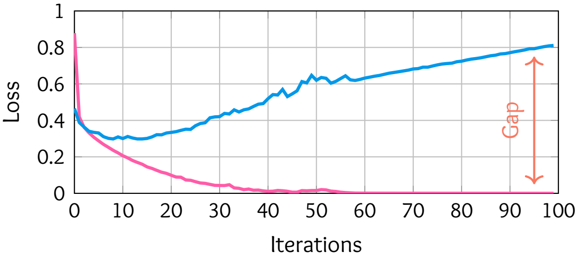


Theory: favor simpler model and/or smooth model

- Early stopping: monitor the validation loss

Generalization Error

- *Generalization error*: Difference between the train accuracy and val/test accuracy
- *Overfitting*: High generalization error



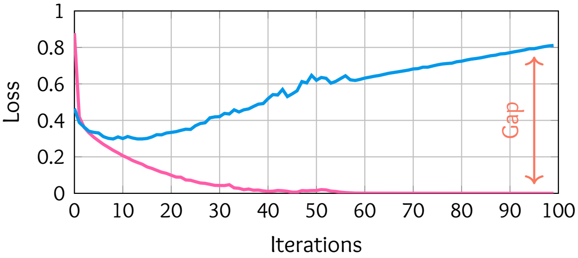
Theory: favor simpler model and/or smooth model

- Early stopping: monitor the validation loss
- Weight decay: Tikhonov regularization

$$\mathcal{L} = \frac{1}{n}\sum_{i=1}^{n} \ell_i + \frac{\lambda}{H}\sum_{h=1}^{H} \|\mathbf{W}_h\|_2^2$$

Generalization Error

- *Generalization error*: Difference between the train accuracy and val/test accuracy
- *Overfitting*: High generalization error



Theory: favor simpler model and/or smooth model

- Early stopping: monitor the validation loss
- Weight decay: Tikhonov regularization
- Noise injection: Dropout

$$\mathcal{L} = \frac{1}{n}\sum_{i=1}^{n} \ell_i + \frac{\lambda}{H}\sum_{h=1}^{H} \|\mathbf{W}_h\|_2^2$$

Dropout

- Smoothness: $f(\mathbf{x}+\epsilon) \approx f(\mathbf{x})$
- Idea: Noise injection during the learning process such as $\mathbb{E}[\tilde{\mathbf{x}}] = \mathbf{x}$
- With MLP:

$$f(\mathbf{x}) = f_H \circ f_{H-1} \circ \ldots \circ f_0(\mathbf{x})$$

and $\mathbf{x}^{(h+1)} = f_h(\mathbf{x}^{(h)}) = \sigma\left(\mathbf{W}^{(h)}\mathbf{x}^{(h)} + \mathbf{b}^{(h)}\right)$

- Dropout: remove (set to zero) some internal features with probability p

$$\mathbf{x}^{(h+1)} = f_h(\tilde{\mathbf{x}}^{(h)}) = \sigma\left(\mathbf{W}^{(h)}(\mathbf{m}^{(h)} \odot \mathbf{x}^{(h)}) + \mathbf{b}^{(h)}\right)$$

with

$$\mathbf{m}_i^{(h)} = \begin{cases} 0 & \text{with probability } p \\ 1/(1-p) & \text{with probability } 1-p \end{cases}$$

Dropout

- Smoothness: $f(\mathbf{x}+\epsilon) \approx f(\mathbf{x})$
- Idea: Noise injection during the learning process such as $\mathbb{E}[\tilde{\mathbf{x}}] = \mathbf{x}$
- With MLP:

$$f(\mathbf{x}) = f_H \circ f_{H-1} \circ \ldots \circ f_0(\mathbf{x})$$

  and $\mathbf{x}^{(h+1)} = f_h(\mathbf{x}^{(h)}) = \sigma\left(\mathbf{W}^{(h)}\mathbf{x}^{(h)} + \mathbf{b}^{(h)}\right)$

- Dropout: remove (set to zero) some internal features with probability p

$$\mathbf{x}^{(h+1)} = f_h(\tilde{\mathbf{x}}^{(h)}) = \sigma\left(\mathbf{W}^{(h)}(\mathbf{m}^{(h)} \odot \mathbf{x}^{(h)}) + \mathbf{b}^{(h)}\right)$$

  with

$$\mathbf{m}_i^{(h)} = \begin{cases} 0 & \text{with probability } p \\ 1/(1-p) & \text{with probability } 1-p \end{cases}$$

- Enable during training and disable during inference (can be used to estimate posterior distribution)

$$f(\tilde{\mathbf{x}}) = f_H \circ D_H \circ f_{H-1} \circ \ldots \circ D_1 \circ f_0(\mathbf{x})$$

To do

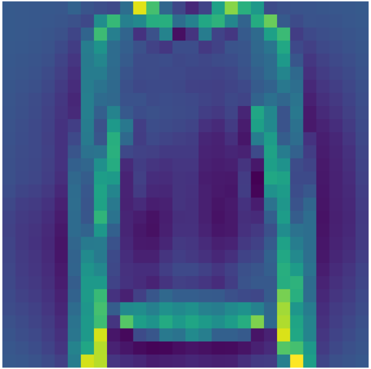
Work

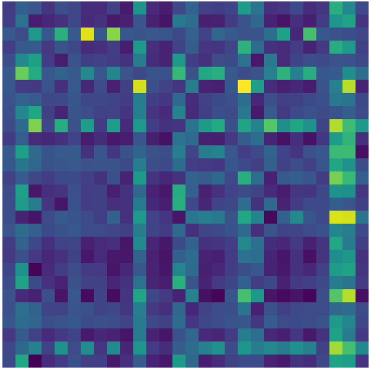
- Implement early stopping
- Implement dropout
- (Optional) Implement Tikhonov regulatization (weight decay in pytorch)

Outline

Outline

An image is not a vector of pixels





2D Convolution

- Fully connected layer:

$$x_{pq}^{(1)} = \sigma\left(\sum_{i,j=1}^{H,W} \mathbf{w}_{pqij}^{(0)} x_{ij}^{(0)} + b_{pq}^{(0)}\right)$$

- Act locally: restrict to neighborhood of pixel p, q

$$x_{pq}^{(1)} = \sigma\left(\sum_{i,j=-\Delta}^{i,j=+\Delta,} \mathbf{w}_{ij}^{(0)} x_{(p+i)(j+q)}^{(0)} + b^{(0)}\right)$$

$$\begin{pmatrix} w_{pq11} & w_{pq12} & \dots & w_{pq1j} & \dots & w_{pq1W} \\ w_{pq21} & w_{pq22} & \dots & w_{pq2j} & \dots & w_{pq2W} \\ \vdots & \vdots & & \vdots & & \vdots \\ w_{pqi1} & w_{pqi2} & \dots & w_{pqij} & \dots & w_{pqiW} \\ \vdots & \vdots & & \vdots & & \vdots \\ w_{pqH1} & w_{pqH2} & \dots & w_{pqHj} & \dots & w_{pqHW} \end{pmatrix}$$

2D Convolution

- Fully connected layer:

$$x_{pq}^{(1)} = \sigma\left(\sum_{i,j=1}^{H,W} \mathbf{w}_{pqij}^{(0)} x_{ij}^{(0)} + b_{pq}^{(0)}\right)$$

- Act locally: restrict to neighborhood of pixel p, q

$$x_{pq}^{(1)} = \sigma\left(\sum_{i,j=-\Delta}^{i,j=+\Delta,} \mathbf{w}_{ij}^{(0)} x_{(p+i)(j+q)}^{(0)} + b^{(0)}\right)$$

$$\begin{pmatrix} 0 & \dots & 0 & \dots & 0 \\ 0 & \dots & 0 & \dots & 0 \\ \vdots & w_{(i)(j-1)} & w_{(i-1)(j)} & w_{(i-1)(j+1)} & \vdots \\ 0 & w_{(i)(j-1)} & w_{(i)(j)} & w_{(i)(j+1)} & 0 \\ \vdots & w_{(i+1)(j-1)} & w_{(i+1)(j)} & w_{(i+1)(j+1)} & \vdots \\ 0 & \dots & 0 & \dots & 0 \end{pmatrix}$$

2D Convolution

- Fully connected layer:

$$x_{pq}^{(1)} = \sigma\left(\sum_{i,j=1}^{H,W} \mathbf{w}_{pqij}^{(0)} x_{ij}^{(0)} + b_{pq}^{(0)}\right)$$

- Act locally: restrict to neighborhood of pixel p, q

$$x_{pq}^{(1)} = \sigma\left(\sum_{i,j=-\Delta}^{i,j=+\Delta,} \mathbf{w}_{ij}^{(0)} x_{(p+i)(j+q)}^{(0)} + b^{(0)}\right)$$

$$\begin{pmatrix} 0 & \dots & 0 & \dots & 0 \\ 0 & \dots & 0 & \dots & 0 \\ \vdots & w_{(i)(j-1)} & w_{(i-1)(j)} & w_{(i-1)(j+1)} & \vdots \\ 0 & w_{(i)(j-1)} & w_{(i)(j)} & w_{(i)(j+1)} & 0 \\ \vdots & w_{(i+1)(j-1)} & w_{(i+1)(j)} & w_{(i+1)(j+1)} & \vdots \\ 0 & \dots & 0 & \dots & 0 \end{pmatrix}$$

2D Convolution

- Fully connected layer:

$$x^{(1)}_{pq} = \sigma\left(\sum_{i,j=1}^{H,W} \mathbf{w}^{(0)}_{pqij} x^{(0)}_{ij} + b^{(0)}_{pq}\right)$$

- Act locally: restrict to neighborhood of pixel p, q

$$x^{(1)}_{pq} = \sigma\left(\sum_{i,j=-\Delta}^{i,j=+\Delta,} \mathbf{w}^{(0)}_{ij} x^{(0)}_{(p+i)(j+q)} + b^{(0)}\right)$$

$$\begin{pmatrix}
0 & \dots & 0 & \dots & 0 \\
0 & \dots & 0 & \dots & 0 \\
\vdots & w_{(i)(j-1)} & w_{(i-1)(j)} & w_{(i-1)(j+1)} & \vdots \\
0 & w_{(i)(j-1)} & w_{(i)(j)} & w_{(i)(j+1)} & 0 \\
\vdots & w_{(i+1)(j-1)} & w_{(i+1)(j)} & w_{(i+1)(j+1)} & \vdots \\
0 & \dots & 0 & \dots & 0
\end{pmatrix}$$

Work

Show that 2D convolution can be expressed as a linear layer.

Basics on 2D Convolution 1/2

Convolution

$$
\begin{array}{|c|c|c|c|c|}
\hline
0 & 1 & 2 & 3 & 4 \\ \hline
5 & 6 & 7 & 8 & 9 \\ \hline
10 & 11 & 12 & 13 & 14 \\ \hline
15 & 16 & 17 & 18 & 19 \\ \hline
\end{array}
\;*\;
\begin{array}{|c|c|c|}
\hline
1 & 1 & 1 \\ \hline
1 & 1 & 1 \\ \hline
1 & 1 & 1 \\ \hline
\end{array}
\;=\;
\begin{array}{|c|c|c|}
\hline
54 & 63 & 72 \\ \hline
99 & 108 & 117 \\ \hline
\end{array}
$$

$$X^{(h)} \qquad\qquad K \qquad\qquad X^{(h+1)}$$

(More https://github.com/vdumoulin/conv_arithmetic/blob/master/README.md)

Basics on 2D Convolution 1/2

Convolution + Padding (1,1)

$$
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 2 & 3 & 4 & 0 \\
0 & 5 & 6 & 7 & 8 & 9 & 0 \\
0 & 10 & 11 & 12 & 13 & 14 & 0 \\
0 & 15 & 16 & 17 & 18 & 19 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
*
\begin{bmatrix}
1 & 1 & 1 \\
1 & 1 & 1 \\
1 & 1 & 1
\end{bmatrix}
=
\begin{bmatrix}
12 & 21 & 27 & 33 & 24 \\
33 & 54 & 63 & 72 & 51 \\
63 & 99 & 108 & 117 & 81 \\
52 & 81 & 87 & 93 & 64
\end{bmatrix}
$$

Basics on 2D Convolution 1/2

Convolution + Padding (1,1) + Stride (3, 2)

$$
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 2 & 3 & 4 & 0 \\
0 & 5 & 6 & 7 & 8 & 9 & 0 \\
0 & 10 & 11 & 12 & 13 & 14 & 0 \\
0 & 15 & 16 & 17 & 18 & 19 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
*
\begin{bmatrix}
1 & 1 & 1 \\
1 & 1 & 1 \\
1 & 1 & 1
\end{bmatrix}
=
\begin{bmatrix}
12 & 27 & 24 \\
52 & 87 & 64
\end{bmatrix}
$$

Basics on 2D Convolution 1/2

Convolution for multivalued images

$$\begin{bmatrix}0&1&2&3&4\\5&6&7&8&9\\10&11&12&13&14\\15&16&17&18&19\end{bmatrix} * \begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix} = \begin{bmatrix}54&63&72\\99&108&117\end{bmatrix}$$

$$+ \qquad = \begin{bmatrix}288&279&270\\243&234&225\end{bmatrix}$$

$$\begin{bmatrix}19&18&17&16&15\\14&13&12&11&10\\9&8&7&6&5\\4&3&2&1&0\end{bmatrix} * \begin{bmatrix}2&2&2\\2&2&2\\2&2&2\end{bmatrix} = \begin{bmatrix}234&216&198\\144&126&108\end{bmatrix}$$

Basics of 2D Convolution 2/2

- Multivariate input - Multivariate output

$$\mathbf{X}^{(h)} \in \mathbb{R}^{C_{(h)} \times H^{(h)} \times W^{(h)}}$$
$$\mathbf{K}^{(h)} \in \mathbb{R}^{C_{(h+1)} \times C_{(h)} \times k_H^h \times k_W^h}$$
$$\mathbf{X}^{(h+1)} \in \mathbb{R}^{C_{(h+1)} \times H^{(h+1)} \times W^{(h+1)}}$$

- Special case of 1×1 convolution: $(C_{(h+1)}, C_{(h)}, k_H, K_W) = (C_o, C_i, 1, 1)$
  - ⋆ Combine pixel across channels - no spatial operation
  - ⋆ Matrix product / linear layer in the channel dimension

$$\mathbf{X}^{(h+1)} = \text{reshape}\left\{\sigma\left(\mathbf{K}^{(h)}\, \text{reshape}\left\{\mathbf{X}^{(h)}, C_{(h)}, H^{(h)}W^{(h)}\right\} + \mathbf{b}^{(h)}\right), C_{(h+1)}, H^{(h)}, W^{(h)}\right\}$$

with reshape $\left\{\mathbf{X}^{(h)}, C_{(h)}, H^{(h)}W^{(h)}\right\} \in \mathbb{R}^{C_{(h)} \times H^{(h)}W^{(h)}}$

Pooling

- Reduce the resolution of the image $\equiv$ Use larger convolution kernel
- Downsampling with simple operator on the neighborhood:
  - Average value
  - Max value (preferred)
- After the non-linearity

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14
15	16	17	18	19

2×2 MaxPooling
Stride=2

6	8
16	18

2D CNN

Work

- Express multi-channels & multi-outputs convolution as a matrix product.
- Show that the successive convolution with two kernels is equivalent to one convolution.
- What stride size wiil you use for a max pooling with a kernel of size 2×2?
- Implement a CNN for the classification of *Fashion MNIST* data set: use one or two convolutional layers for the MLP.

Outline

Batch Normalization

- Remember scaling problem in slide 10
- Similar problem may happen to each layer **but** we cannot use the same trick: the sample mean/variance change after each batch update !

Batch Normalization

- Remember scaling problem in slide 10
- Similar problem may happen to each layer **but** we cannot use the same trick: the sample mean/variance change after each batch update !
- Apply normalization per batch: *Batch Normalization*

$$\tilde{\mathbf{x}}_B = \boldsymbol{\gamma} \odot \left[(\mathbf{x}_B - \boldsymbol{\mu}_B) \oslash \boldsymbol{\sigma}_B \right] + \boldsymbol{\beta}$$

with $\boldsymbol{\mu}_B$ and $\boldsymbol{\sigma}_B$ computed on batch B, and $\boldsymbol{\gamma}\&\boldsymbol{\beta}$ are learnable parameters

Batch Normalization

- Remember scaling problem in slide 10
- Similar problem may happen to each layer **but** we cannot use the same trick: the sample mean/variance change after each batch update !
- Apply normalization per batch: *Batch Normalization*

$$\tilde{\mathbf{x}}_B = \boldsymbol{\gamma} \odot \left[(\mathbf{x}_B - \boldsymbol{\mu}_B) \oslash \boldsymbol{\sigma}_B \right] + \boldsymbol{\beta}$$

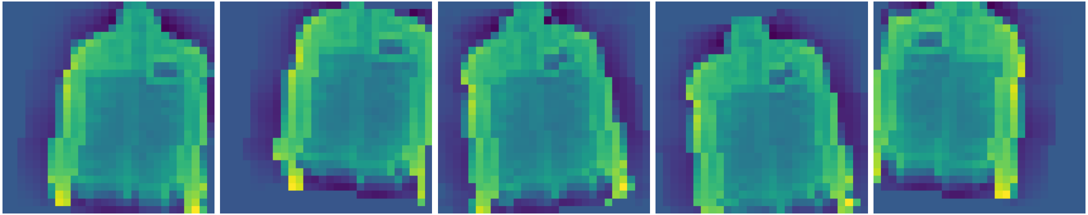
with $\boldsymbol{\mu}_B$ and $\boldsymbol{\sigma}_B$ computed on batch B, and $\boldsymbol{\gamma}\&\boldsymbol{\beta}$ are learnable parameters

Work

- What happen if the batch size is one ?
- What happen if the batch size is small ?
- What should we do in *prediction* mode ?
- Add normalization layer (2D) to your model.

Data Augmentation

- Noise injection to generate / augment data set: $(\mathbf{x}, y) \rightarrow (\tilde{\mathbf{x}}, y)$
- For image classification it is *relatively* easy:
  - ⋆ Geometric tansformation: Rotation, translation, symmetry, radiometry ...
  - ⋆ Radiometric transformation: Invert, jitter ...
  - ⋆ *Any transformation that can model the variability of the acquisition process*



Work

Implement data augmentation with Pytorch for the training data only.

Residual Network

- Deep Neural Network: $f_h(\mathbf{x}^{(h)}) := f_h(\mathbf{x}^{(h)}, \theta_h)$

$$f(\mathbf{x}) = f_H \circ f_{H-1} \circ \ldots \circ f_0(\mathbf{x})$$

Residual Network

- Deep Neural Network: $f_h(\mathbf{x}^{(h)}) := f_h(\mathbf{x}^{(h)}, \theta_h)$

$$f(\mathbf{x}) = f_H \circ f_{H-1} \circ \ldots \circ f_0(\mathbf{x})$$

- Let $f_h = g_2 \circ g_1 \circ g_0$

$$\mathbf{x} \xrightarrow{g_0} \mathbf{x}^{(1)} \xrightarrow{g_1} \mathbf{x}^{(2)} \xrightarrow{g_2} f_h(\mathbf{x})$$

$$\frac{\partial f_h(\mathbf{x})}{\partial \alpha_0} = \underrightarrow{\frac{g_2}{g_1} \circ \frac{g_1}{g_0} \circ \frac{g_0}{\alpha_0}(\mathbf{x})} \qquad \frac{\partial g(\mathbf{x})}{\partial \alpha_1} = \frac{g_2}{g_1} \circ \frac{g_1}{\alpha_1}(\mathbf{x}^{(1)}) \qquad \frac{\partial g(\mathbf{x})}{\partial \alpha_2} = \frac{g_2}{\alpha_2}(\mathbf{x}^{(2)})$$

Residual Network

- Deep Neural Network: $f_h(\mathbf{x}^{(h)}) := f_h(\mathbf{x}^{(h)}, \theta_h)$

$$f(\mathbf{x}) = f_H \circ f_{H-1} \circ \ldots \circ f_0(\mathbf{x})$$

- Let $f_h = g_2 \circ g_1 \circ g_0$ with *residual connection*

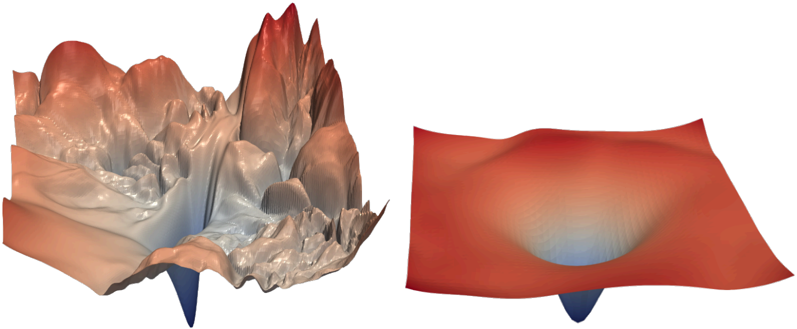
$$\mathbf{x} \xrightarrow{g_0} \oplus \to \mathbf{x}^{(1)} \xrightarrow{g_1} \oplus \to \mathbf{x}^{(2)} \xrightarrow{g_2} \oplus \to f_h(\mathbf{x})$$

$$\frac{\partial f_h(\mathbf{x})}{\partial \alpha_0} = \frac{g_2}{g_1} \circ \frac{g_1}{g_0} \circ \frac{g_0}{\alpha_0}(\mathbf{x}) + \frac{g_1}{g_0} \circ \frac{g_0}{\alpha_0}(\mathbf{x}) + \frac{g_0}{\alpha_0}(\mathbf{x})$$

  - ⋆ Improve gradient propagation
  - ⋆ Prevent vanishing/shattered gradient

- Deep Neural Network: $f_h(\mathbf{x}^{(h)}) := f_h(\mathbf{x}^{(h)}, \theta_h)$

$$f(\mathbf{x}) = f_H \circ f_{H-1} \circ \ldots \circ f_0(\mathbf{x})$$



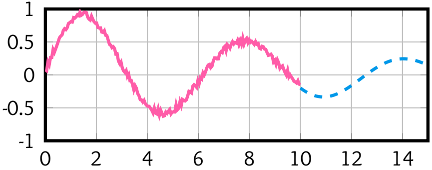
From Hao Li et al. "Visualizing the Loss Landscape of Neural Nets". In: *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. NIPS'18. Montréal, Canada: Curran Associates Inc., 2018, pp. 6391–6401

Outline

Outline

Sequential data

- Time series



- Text data

This course is really cool

Auto-regressive model

- Autogressive models: $p(\mathbf{x}_t|\mathbf{x}_{t-1}, \ldots, \mathbf{x}_1)$
- Sequence model:

$$p(\mathbf{x}_T, \mathbf{x}_{T-1}, \ldots, \mathbf{x}_1) = \prod_{t=2}^{T} p(\mathbf{x}_t|\mathbf{x}_{t-1}, \ldots, \mathbf{x}_1)p(\mathbf{x}_1)$$

- Markov model:

$$p(\mathbf{x}_T|\mathbf{x}_{T-1}, \ldots, \mathbf{x}_1) = \prod_{t=2}^{T} p(\mathbf{x}_t|\mathbf{x}_{t-1})p(\mathbf{x}_1)$$

Auto-regressive model

- Autogressive models: $p(\mathbf{x}_t|\mathbf{x}_{t-1}, \dots, \mathbf{x}_1)$
- Sequence model:

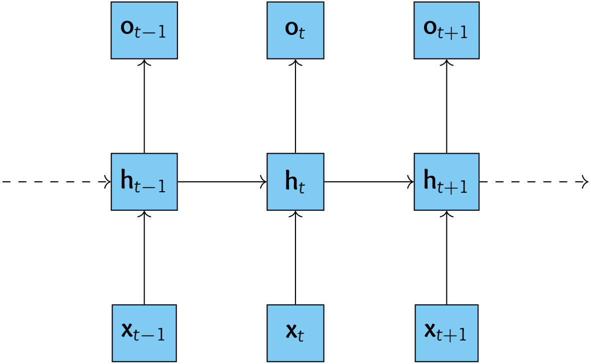
$$p(\mathbf{x}_T, \mathbf{x}_{T-1}, \dots, \mathbf{x}_1) = \prod_{t=2}^{T} p(\mathbf{x}_t|\mathbf{x}_{t-1}, \dots, \mathbf{x}_1)p(\mathbf{x}_1)$$

- Markov model:

$$p(\mathbf{x}_T|\mathbf{x}_{T-1}, \dots, \mathbf{x}_1) = \prod_{t=2}^{T} p(\mathbf{x}_t|\mathbf{x}_{t-1})p(\mathbf{x}_1)$$

- Example: $p(\mathbf{x}_t|\mathbf{x}_{t-1})$ could model
  - ⋆ The probability to get letter after a observing a specific letter (or a set of)
  - ⋆ The expected value of a signal given past values of the same signal

Latent autoregressive model



- $\mathbf{h}_t = f(\mathbf{x}_t, \mathbf{h}_{t-1}) = \sigma_h\left(\mathbf{W}_{hx}\mathbf{x}_t + \mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{b}_h\right)$
- $\mathbf{o}_t = g(\mathbf{h}_t) = \sigma_o\left(\mathbf{W}_{oh}\mathbf{h}_t + \mathbf{b}_o\right)$
- Backpropagation through time - BPTT

$$\ell = \frac{1}{T}\sum_{t=1}^{T} \ell(\mathbf{y}_t, \mathbf{o}_t)$$

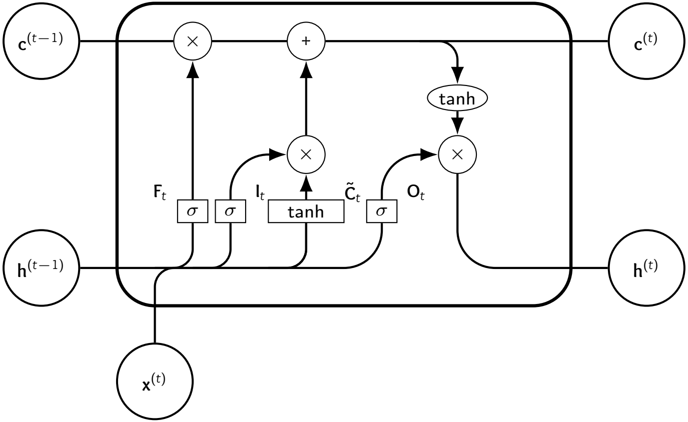
with $\mathbf{h}_0 = \mathbf{0}$

Issue with BPTT

$$
\begin{aligned}
\frac{\partial \ell}{\partial \mathbf{W}_{hh}} &= \frac{1}{T}\sum_{t=1}^{T} \frac{\partial \ell(\mathbf{y}_t, \mathbf{o}_t)}{\partial \mathbf{W}_{hh}} \\
&= \frac{1}{T}\sum_{t=1}^{T} \frac{\partial \ell(\mathbf{y}_t, \mathbf{o}_t)}{\partial \mathbf{o}_t} \frac{\partial \mathbf{o}_t}{\partial \mathbf{W}_{hh}} \\
&= \frac{1}{T}\sum_{t=1}^{T} \frac{\partial \ell(\mathbf{y}_t, \mathbf{o}_t)}{\partial \mathbf{o}_t} \frac{\partial g(\mathbf{h}_t)}{\partial \mathbf{h}_t} \left[\frac{\partial f(\mathbf{x}_t, \mathbf{h}_{t-1})}{\partial \mathbf{W}_{hh}} + \frac{f(\mathbf{x}_t, \mathbf{h}_{t-1})}{\partial \mathbf{h}_{t-1}} \frac{\partial \mathbf{h}_{t-1}}{\partial \mathbf{W}_{hh}} \right] \\
&= \frac{1}{T}\sum_{t=1}^{T} \frac{\partial \ell(\mathbf{y}_t, \mathbf{o}_t)}{\partial \mathbf{o}_t} \frac{\partial g(\mathbf{h}_t)}{\partial \mathbf{h}_t} \left[\frac{\partial f(\mathbf{x}_t, \mathbf{h}_{t-1})}{\partial \mathbf{W}_{hh}} + \sum_{p=1}^{t-1} \left(\prod_{q=p+1}^{t} \frac{\partial f(\mathbf{x}_q, \mathbf{h}_{q-1})}{\partial \mathbf{h}_{q-1}} \right) \frac{\partial f(\mathbf{x}_p, \mathbf{h}_{p-1})}{\partial \mathbf{W}_{hh}} \right]
\end{aligned}
$$

Outline

Long Short Term Memory (LSTM)



- Forget gate

$$F_t = \sigma\left(W_{fh}\mathbf{h}_{(t-1)} + W_{fx}\mathbf{x}_t + b_f\right)$$

- Input gate

$$I_t = \sigma\left(W_{ih}\mathbf{h}_{(t-1)} + W_{ix}\mathbf{x}_t + b_i\right)$$

- Input node

$$\tilde{C}_t = \tanh\left(W_{\tilde{c}h}\mathbf{h}_{(t-1)} + W_{\tilde{c}x}\mathbf{x}_t + b_{\tilde{c}}\right)$$

- Output gate

$$O_t = \sigma\left(W_{oh}\mathbf{h}_{(t-1)} + W_{ox}\mathbf{x}_t + b_o\right)$$

Good overview of LSTM: `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`

Sequence prediction with RNN 1/2

Learning (toy) problem

Learn a function f that, given a fix sized sequence of characters, predicts the most probable next one from an alphabet

$$f(\text{T h i s c o u r}) = \text{s}$$

$$f(\text{i s a w e s o m}) = \text{e}$$

Tokenization

Tokenization is the action of cutting input data into parts that can be embedded into a vector space.

```
text ="This course is awesome"
text_set = sorted(set(text))
[' ', 'T', 'a', 'c', 'e', 'h', 'i', 'm', 'o', 'r', 's', 'u', 'w']
char2int = {ch: i for i, ch in enumerate(text_set)}
{' ': 0, 'T': 1, 'a': 2, 'c': 3, 'e': 4, 'h': 5, 'i': 6, 'm': 7, 'o': 8,
↪ 'r': 9, 's': 10, 'u': 11, 'w': 12}
```

Sequence prediction with RNN 2/2

- Classification problem on sequence, x_t = token

$$\mathbf{x}_{T+1} = f(\mathbf{x_T}, \dots, \mathbf{x_0})$$

- By-product: Once we have $p(\mathbf{x}_{T+1}|\mathbf{x}_T, \dots, \mathbf{x}_0)$ we can sample random text

$$p(\mathbf{x}_{T+1}|\mathbf{x}_T, \dots, \mathbf{x}_0) \sim \text{Multinomial}$$

Work

- Implement a simple tokenizer in pytorch
- Implement a RNN that learn to predict the next character of a sequence
- Once trained, generate some sentences of varying size