COMP90051

Workshop Week 04

About the Workshops

- 7 sessions in total
  - Tue 12:00-13:00 AH211
  - Tue 12:00-13:00 AH108 *
  - Tue 13:00-14:00 AH210
  - Tue 16:15-17:15 AH109
  - Tue 17:15-18:15 AH236 *
  - Tue 18:15-19:15 AH236 *
  - Fri 14:15-15:15 AH211

About the Workshops

- Homepage
  - https://trevorcohn.github.io/comp90051-2017/workshops

- Solutions will be released on next Friday (a week later).

Syllabus

1	Introduction; Probability theory	Probabilistic models; Parameter fitting	
2	Linear regression; Intro to regularization	Logistic regression; Basis expansion	
3	Optimization; Regularization	Perceptron	←
4	Backpropagation	CNNs; Auto-encoders	
5	Hard-margin SVMs	Soft-margin SVMs	
6	Additional topics	Kernel methods	
7	Unsupervised learning	Unsupervised learning	
8	Dimensionality reduction; Principal component analysis	Multidimensional scaling; Spectral clustering	
9	Bayesian fundamentals	Bayesian inference with conjugate priors	
10	PGMs, fundamentals	Conditional independence	
11	PGMs, inference	Belief propagation	
12	Statistical inference; Apps	Subject review	

Outline

- Review the lecture, background knowledge, etc.
  - Supervised learning as an optimization problem
  - Perceptron update rule & loss function
  - Logistic regression
    - Predict function
    - Log loss (a.k.a. cross entropy)
- Notebook tasks
  - Task 1: Logistic regression
  - Task 2: Perceptron classifier

Outline

- Review the lecture, background knowledge, etc.
  - Supervised learning as an optimization problem
  - Perceptron update rule & loss function
  - Logistic regression
    - Predict function
    - Log loss (a.k.a. cross entropy)
- Notebook tasks
  - Task 1: Logistic regression
  - Task 2: Perceptron classifier

Supervised learning as an optimization problem

- Dataset
  - Preprocessing / normalization / feature selection
  - Split into train*/test, where test served as the held-out set
    - Split train* into train/validation (or k folds train/validation, CV)
- Model / Objective function
  - Parameters, solved either analytically or by an optimizer
    - Solved on the training set (or training sets in k folds, CV)
  - Hyper-parameters, e.g. regularization parameter
    - Selected on the validation set (or validation sets in k folds, CV)
- Evaluation on the held-out set

Supervised learning as an optimization problem

- Dataset
  - Preprocessing / normalization / feature selection
  - Split into train*/test, where test served as the held-out set
- Model / Objective function
  - Parameters, solved either analytically or by an optimizer
    - Solved on the training set*
- Evaluation on the held-out set

How to solve an optimization problem?

- Analytic solution → solve it by a formula
  - $\min_{\boldsymbol{w}} \|\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}\|_2^2 + \lambda\|\boldsymbol{w}\|_2^2,\ \lambda \geq 0 \quad \rightarrow \quad \boldsymbol{w}^* = (\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I})^{-1}\boldsymbol{X}^T\boldsymbol{y}$
- Iterative methods → solve it by an optimization algorithm
  - To minimize an objective
    - Coordinate descent
    - Gradient-based optimization algorithms (optimizers)
      - Simplest: gradient descent & stochastic gradient descent
      - BFGS (in `4a_logistic_regression.ipynb`)
      - Many more in packages…
  - To maximize an objective
    - Convert it to an equivalent minimization problem

- Linear regression has an analytic solution (with L2)
- Perceptron has its own update rule
  - can be interpreted as using stochastic gradient descent (with an appropriate loss function defined)
- Gradient-based optimizers can be used for
  - Linear regression
  - Support vector machines
  - Logistic regression, neural networks
- Deep neural networks are usually constructed and optimized using special packages…

Gradient-based optimizers in TensorFlow™

- `tf.train.Optimizer`
- `tf.train.GradientDescentOptimizer`
- `tf.train.AdadeltaOptimizer`
- `tf.train.AdagradOptimizer`
- `tf.train.AdagradDAOptimizer`
- `tf.train.MomentumOptimizer`
- `tf.train.AdamOptimizer`
- `tf.train.FtrlOptimizer`
- `tf.train.ProximalGradientDescentOptimizer`
- `tf.train.ProximalAdagradOptimizer`
- `tf.train.RMSPropOptimizer`

https://www.tensorflow.org/api_guides/python/train

Gradient-based optimizers in PYTORCH

class `torch.optim.Optimizer`(*params, defaults*)[source]

class `torch.optim.Adadelta`(*params, lr=1.0, rho=0.9, eps=1e-06, weight_decay=0*)[source]

class `torch.optim.Adagrad`(*params, lr=0.01, lr_decay=0, weight_decay=0*)[source]

class `torch.optim.Adam`(*params, lr=0.001, betas=(0.9, 0.999), eps=1e-08, weight_decay=0*)[source]

class `torch.optim.Adamax`(*params, lr=0.002, betas=(0.9, 0.999), eps=1e-08, weight_decay=0*)[source]

class `torch.optim.ASGD`(*params, lr=0.01, lambd=0.0001, alpha=0.75, t0=1000000.0, weight_decay=0*)[source]

class `torch.optim.LBFGS`(*params, lr=1, max_iter=20, max_eval=None, tolerance_grad=1e-05, tolerance_change=1e-09, history_size=100, line_search_fn=None*)[source]

class `torch.optim.RMSprop`(*params, lr=0.01, alpha=0.99, eps=1e-08, weight_decay=0, momentum=0, centered=False*)[source]

class `torch.optim.Rprop`(*params, lr=0.01, etas=(0.5, 1.2), step_sizes=(1e-06, 50)*)[source]

class `torch.optim.SGD`(*params, lr=<object object>, momentum=0, dampening=0, weight_decay=0, nesterov=False*)[source]

http://pytorch.org/docs/master/optim.html#algorithms

Outline

- Review the lecture, background knowledge, etc.
  - Supervised learning as an optimization problem
  - Perceptron update rule & loss function
  - Logistic regression
    - Predict function
    - Log loss (a.k.a. cross entropy)

- Notebook tasks
  - Task 1: Logistic regression
  - Task 2: Perceptron classifier

Perceptron update rule (2-D points)

- Data points
  - $\boldsymbol{x}_1 = [1 \quad x_{1,1} \quad x_{1,2}] \rightarrow y_1 = +1$, $\boldsymbol{x}_2 = [1 \quad x_{2,1} \quad x_{2,2}] \rightarrow y_2 = +1$
  - $\boldsymbol{x}_3 = [1 \quad x_{3,1} \quad x_{3,2}] \rightarrow y_3 = -1$, $\boldsymbol{x}_4 = [1 \quad x_{4,1} \quad x_{4,2}] \rightarrow y_4 = -1$
- Model parameters
  - $\boldsymbol{w} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix}$
- Decision function
  - $f(\boldsymbol{x}; \boldsymbol{w}) = \boldsymbol{x}\boldsymbol{w} = x_0 w_0 + x_1 w_1 + x_2 w_2$
  - Predict $+1$ if $f(\boldsymbol{x}; \boldsymbol{w}) > 0$, predict -1 if $f(\boldsymbol{x}; \boldsymbol{w}) < 0$

Perceptron update rule (2-D points)

- Iterate over all the points
  - Current point is $\boldsymbol{x}_i$, and its label is y_i
  - Call decision function $s = f(\boldsymbol{x}_i; \boldsymbol{w}) = \boldsymbol{x}_i \boldsymbol{w}$
  - Predict $\hat{y}_i$ as $+1$ if $s > 0$, as -1 if $s < 0$
  - If $y_i \neq \hat{y}_i$
    - If $y_i = +1$, $\boldsymbol{w}^{new} = \boldsymbol{w}^{old} + \boldsymbol{x}_i^T$, or $\begin{cases} w_0^{new} = w_0^{old} + x_{i,0} \\ w_1^{new} = w_1^{old} + x_{i,1} \\ w_2^{new} = w_2^{old} + x_{i,2} \end{cases}$
    - If $y_i = -1$, $\boldsymbol{w}^{new} = \boldsymbol{w}^{old} - \boldsymbol{x}_i^T$, or $\begin{cases} w_0^{new} = w_0^{old} - x_{i,0} \\ w_1^{new} = w_1^{old} - x_{i,1} \\ w_2^{new} = w_2^{old} - x_{i,2} \end{cases}$

Suppose $\boldsymbol{x}_1$ is misclassified

- This means $y_1 = +1$, $\hat{y}_1 = -1$, $s = f\left(\boldsymbol{x}_1; \boldsymbol{w}^{old}\right) = \boldsymbol{x}_1 \boldsymbol{w}^{old} < 0$
- Apply the update rule:
  - $\boldsymbol{w}^{new} = \boldsymbol{w}^{old} + \boldsymbol{x}_1^T$
- How does s change?
  - $s^{new} = \boldsymbol{x}_1 \boldsymbol{w}^{new} = \boldsymbol{x}_1\left(\boldsymbol{w}^{old} + \boldsymbol{x}_1^T\right) = \boldsymbol{x}_1 \boldsymbol{w}^{old} + \boldsymbol{x}_1 \boldsymbol{x}_1^T > s^{old}$
- We hope $s > 0$ (because $y_1 = +1$)
- After updating $\boldsymbol{w}$, $s^{new} > s^{old}$, s^{new} may still < 0
- But it improves a bit (at least for $\boldsymbol{x}_1$)

Suppose $\boldsymbol{x}_3$ is misclassified

- This means $y_3 = -1$, $\hat{y}_3 = +1$, $s = f(\boldsymbol{x}_3; \boldsymbol{w}^{old}) = \boldsymbol{x}_3\boldsymbol{w}^{old} > 0$
- Apply the update rule:
  - $\boldsymbol{w}^{new} = \boldsymbol{w}^{old} - \boldsymbol{x}_3^T$
- How does s change?
  - $s^{new} = \boldsymbol{x}_3\boldsymbol{w}^{new} = \boldsymbol{x}_3(\boldsymbol{w}^{old} - \boldsymbol{x}_3^T) = \boldsymbol{x}_3\boldsymbol{w}^{old} - \boldsymbol{x}_3\boldsymbol{x}_3^T < s^{old}$
- We hope $s < 0$ (because $y_3 = -1$)
- After updating $\boldsymbol{w}$, $s^{new} < s^{old}$, s^{new} may still > 0
- But it improves a bit (at least for $\boldsymbol{x}_3$)

A uniform update rule

- For misclassified positive instance $\boldsymbol{x}$, $y = +1$, hope $s > 0$
  - $\boldsymbol{w}^{new} = \boldsymbol{w}^{old} + \boldsymbol{x}^T$ to increase $s = f(\boldsymbol{x}; \boldsymbol{w}) = \boldsymbol{x}\boldsymbol{w}$
- For misclassified negative instance $\boldsymbol{x}$, $y = -1$, hope $s < 0$
  - $\boldsymbol{w}^{new} = \boldsymbol{w}^{old} - \boldsymbol{x}^T$ to decrease $s = f(\boldsymbol{x}; \boldsymbol{w}) = \boldsymbol{x}\boldsymbol{w}$

- A uniform update rule:
- For misclassified instance $\boldsymbol{x}$, $\boldsymbol{w}^{new} = \boldsymbol{w}^{old} + y\boldsymbol{x}^T$ so that
  - $s = f(\boldsymbol{x}; \boldsymbol{w}) = \boldsymbol{x}\boldsymbol{w}$ is increased if $y = +1$
  - $s = f(\boldsymbol{x}; \boldsymbol{w}) = \boldsymbol{x}\boldsymbol{w}$ is decreased if $y = -1$

Perceptron update rule (2-D points)

- Iterate over all the points
  - Current point is $\boldsymbol{x}_i$, and its label is y_i
  - Call decision function $s = f(\boldsymbol{x}_i; \boldsymbol{w}) = \boldsymbol{x}_i \boldsymbol{w}$
  - Predict $\hat{y}_i$ as $+1$ if $s > 0$, as -1 if $s < 0$
  - If $y_i \neq \hat{y}_i$
    - $\boldsymbol{w}^{new} = \boldsymbol{w}^{old} + y\boldsymbol{x}_i^T$, or $\begin{cases} w_0^{new} = w_0^{old} + yx_{i,0} \\ w_1^{new} = w_1^{old} + yx_{i,1} \\ w_2^{new} = w_2^{old} + yx_{i,2} \end{cases}$

Perceptron update rule (2-D points)

- Iterate over all the points
  - Current point is $\boldsymbol{x}_i$, and its label is y_i
  - Call decision function $s = f(\boldsymbol{x}_i; \boldsymbol{w}) = \boldsymbol{x}_i \boldsymbol{w}$
  - Predict $\hat{y}_i$ as +1 if $s > 0$, as −1 if $s < 0$
  - If $y_i \neq \hat{y}_i \Leftrightarrow y_i, \hat{y}_i = -1, +1 \text{ or } +1, -1 \Leftrightarrow y_i \hat{y}_i < 0 \Leftrightarrow y_i s < 0$
    - $\boldsymbol{w}^{new} = \boldsymbol{w}^{old} + y\boldsymbol{x}_i^T$, or $\begin{cases} w_0^{new} = w_0^{old} + yx_{i,0} \\ w_1^{new} = w_1^{old} + yx_{i,1} \\ w_2^{new} = w_2^{old} + yx_{i,2} \end{cases}$

Perceptron update rule (2-D points)

- Iterate over all the points
  - Current point is $\boldsymbol{x}_i$, and its label is y_i
  - If misclassified $\Leftrightarrow y f(\boldsymbol{x}_i; \boldsymbol{w}) = y \boldsymbol{x}_i \boldsymbol{w} < 0$
    - $\boldsymbol{w}^{new} = \boldsymbol{w}^{old} + y\boldsymbol{x}_i^T$, or $\begin{cases} w_0^{new} = w_0^{old} + yx_{i,0} \\ w_1^{new} = w_1^{old} + yx_{i,1} \\ w_2^{new} = w_2^{old} + yx_{i,2} \end{cases}$

- The update rule for implementation
- We can further define a loss function, but only for theoretic analysis.

Define a loss function

- For misclassified instance $\boldsymbol{x}$, $\boldsymbol{w}^{new} = \boldsymbol{w}^{old} + y\boldsymbol{x}^T$ so that
  - $s = f(\boldsymbol{x}; \boldsymbol{w}) = \boldsymbol{x}\boldsymbol{w}$ is increased if $y = +1$
  - $s = f(\boldsymbol{x}; \boldsymbol{w}) = \boldsymbol{x}\boldsymbol{w}$ is decreased if $y = -1$

- Loss functions should be minimized
- Define $L(f(\boldsymbol{x}; \boldsymbol{w}), y) = -yf(\boldsymbol{x}; \boldsymbol{w})$ for misclassified $\boldsymbol{x}$
  - $-yf(\boldsymbol{x}; \boldsymbol{w}) = -\boldsymbol{x}\boldsymbol{w}$ is decreased if $y = +1$
  - $-yf(\boldsymbol{x}; \boldsymbol{w}) = +\boldsymbol{x}\boldsymbol{w}$ is decreased if $y = -1$
- So $L(f(\boldsymbol{x}; \boldsymbol{w}), y) = -yf(\boldsymbol{x}; \boldsymbol{w})$ decreases after updating $\boldsymbol{w}$

Gradient of the loss function

- $L(f(\boldsymbol{x};\boldsymbol{w}),y) = -yf(\boldsymbol{x};\boldsymbol{w}) = -y(x_0 w_0 + x_1 w_1 + x_2 w_2)$

$$\frac{\partial L}{\partial w_0} = -yx_0 \qquad \frac{\partial L}{\partial w_1} = -yx_1 \qquad \frac{\partial L}{\partial w_2} = -yx_2$$

- So

$$\frac{\partial L}{\partial \boldsymbol{w}} = \begin{bmatrix} \frac{\partial L}{\partial w_0} \\ \frac{\partial L}{\partial w_1} \\ \frac{\partial L}{\partial w_2} \end{bmatrix} = -y \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = -y\boldsymbol{x}^T$$

The uniform update rule

- For misclassified instance $\boldsymbol{x}$, $\boldsymbol{w}^{new} = \boldsymbol{w}^{old} + y\boldsymbol{x}^T$

- For misclassified instance $\boldsymbol{x}$, the loss function is defined as $L(f(\boldsymbol{x};\boldsymbol{w}), y) = -yf(\boldsymbol{x};\boldsymbol{w})$

$$\frac{\partial L}{\partial \boldsymbol{w}} = -y\boldsymbol{x}^T$$

- So the update rule can be written as

$$\boldsymbol{w}^{new} = \boldsymbol{w}^{old} + y\boldsymbol{x}^T = \boldsymbol{w}^{old} - \frac{\partial L}{\partial \boldsymbol{w}}$$

Perceptron update rule (2-D points)

- Define $L(f(\boldsymbol{x};\boldsymbol{w}),y) = -yf(\boldsymbol{x};\boldsymbol{w})$ for misclassified instances

- Iterate over all the points
  - Current point is $\boldsymbol{x}_i$, and its label is y_i
  - If misclassified $\Leftrightarrow yf(\boldsymbol{x}_i;\boldsymbol{w}) = y\boldsymbol{x}_i\boldsymbol{w} < 0 \Leftrightarrow L(f(\boldsymbol{x};\boldsymbol{w}),y) > 0$

$$\boldsymbol{w}^{new} = \boldsymbol{w}^{old} - \frac{\partial L}{\partial \boldsymbol{w}}$$

Perceptron → stochastic gradient descent

- Define

$$L(f(\boldsymbol{x};\boldsymbol{w}),y) = \begin{cases} -yf(\boldsymbol{x};\boldsymbol{w}) & \text{if } L(f(\boldsymbol{x};\boldsymbol{w}),y) > 0 \Leftrightarrow \text{misclassified} \\ 0 & otherwise \end{cases}$$

- Then

$$\frac{\partial L}{\partial \boldsymbol{w}} = \begin{cases} -y\boldsymbol{x}^T & \text{if } L(f(\boldsymbol{x};\boldsymbol{w}),y) > 0 \Leftrightarrow \text{misclassified} \\ \boldsymbol{0} & otherwise \end{cases}$$

- Iterate over all the points
  - Current point is $\boldsymbol{x}_i$, and its label is y_i
  - Apply the update rule

$$\boldsymbol{w}^{new} = \boldsymbol{w}^{old} - \frac{\partial L}{\partial \boldsymbol{w}}$$

Outline

- Review the lecture, background knowledge, etc.
  - Supervised learning as an optimization problem
  - Perceptron update rule & loss function
  - Logistic regression
    - Predict function
    - Log loss (a.k.a. cross entropy)
- Notebook tasks
  - Task 1: Logistic regression
  - Task 2: Perceptron classifier

Logistic regression (2-D points, 3 classes)

- Data points
  - $\boldsymbol{x}_1 = [1 \quad x_{1,1} \quad x_{1,2}] \rightarrow y_1 = 0$, $\boldsymbol{x}_2 = [1 \quad x_{2,1} \quad x_{2,2}] \rightarrow y_2 = 1$
  - $\boldsymbol{x}_3 = [1 \quad x_{3,1} \quad x_{3,2}] \rightarrow y_3 = 2$, $\boldsymbol{x}_4 = [1 \quad x_{4,1} \quad x_{4,2}] \rightarrow y_4 = 2$
- Model parameters
  - $\boldsymbol{W} = [\boldsymbol{w}_0 \quad \boldsymbol{w}_1 \quad \boldsymbol{w}_2] = \begin{bmatrix} w_{0,0} & w_{0,1} & w_{0,2} \\ w_{1,0} & w_{1,1} & w_{1,2} \\ w_{2,0} & w_{2,1} & w_{2,2} \end{bmatrix}$
- Decision function, predict $\boldsymbol{x}$ as class j if s_j is the largest.
  - $f(\boldsymbol{x}; \boldsymbol{w}) = \boldsymbol{x}\boldsymbol{W} = \begin{bmatrix} \boldsymbol{x}\boldsymbol{w}_0 \\ \boldsymbol{x}\boldsymbol{w}_1 \\ \boldsymbol{x}\boldsymbol{w}_2 \end{bmatrix}^T = \begin{bmatrix} x_0 w_{0,0} + x_1 w_{1,0} + x_2 w_{2,0} \\ x_0 w_{0,1} + x_1 w_{1,1} + x_2 w_{2,1} \\ x_0 w_{0,2} + x_1 w_{1,2} + x_2 w_{2,2} \end{bmatrix}^T = \begin{bmatrix} s_0 \\ s_1 \\ s_2 \end{bmatrix}^T$

Logistic regression (2-D points, 3 classes)

- Model parameters
  - $$\boldsymbol{W} = [\boldsymbol{w}_0 \quad \boldsymbol{w}_1 \quad \boldsymbol{w}_2] = \begin{bmatrix} w_{0,0} & w_{0,1} & w_{0,2} \\ w_{1,0} & w_{1,1} & w_{1,2} \\ w_{2,0} & w_{2,1} & w_{2,2} \end{bmatrix}$$
- Decision function, predict $\boldsymbol{x}$ as class j if s_j is the largest.
  - $$f(\boldsymbol{x}; \boldsymbol{w}) = \boldsymbol{x}\boldsymbol{W} = \begin{bmatrix} \boldsymbol{x}\boldsymbol{w}_0 \\ \boldsymbol{x}\boldsymbol{w}_1 \\ \boldsymbol{x}\boldsymbol{w}_2 \end{bmatrix}^T = \begin{bmatrix} x_0 w_{0,0} + x_1 w_{1,0} + x_2 w_{2,0} \\ x_0 w_{0,1} + x_1 w_{1,1} + x_2 w_{2,1} \\ x_0 w_{0,2} + x_1 w_{1,2} + x_2 w_{2,2} \end{bmatrix}^T = \begin{bmatrix} s_0 \\ s_1 \\ s_2 \end{bmatrix}^T$$
- Output the distribution
  - $$p(y|\boldsymbol{x}; \boldsymbol{W}) = \frac{1}{e^{s_0} + e^{s_1} + e^{s_2}} [e^{s_0} \quad e^{s_1} \quad e^{s_2}] \propto [e^{s_0} \quad e^{s_1} \quad e^{s_2}]$$

Logistic regression (2-D points, 3 classes)

- Decision function, predict $\boldsymbol{x}$ as class j if s_j is the largest.
  - $$f(\boldsymbol{x};\boldsymbol{w}) = \boldsymbol{x}\boldsymbol{W} = \begin{bmatrix} \boldsymbol{x}\boldsymbol{w}_0 \\ \boldsymbol{x}\boldsymbol{w}_1 \\ \boldsymbol{x}\boldsymbol{w}_2 \end{bmatrix}^T = \begin{bmatrix} x_0 w_{0,0} + x_1 w_{1,0} + x_2 w_{2,0} \\ x_0 w_{0,1} + x_1 w_{1,1} + x_2 w_{2,1} \\ x_0 w_{0,2} + x_1 w_{1,2} + x_2 w_{2,2} \end{bmatrix}^T = \begin{bmatrix} s_0 \\ s_1 \\ s_2 \end{bmatrix}^T$$
- Output the distribution
  - $$p(y|\boldsymbol{x};\boldsymbol{W}) = \frac{1}{e^{s_0}+e^{s_1}+e^{s_2}}\begin{bmatrix} e^{s_0} & e^{s_1} & e^{s_2} \end{bmatrix} \propto \begin{bmatrix} e^{s_0} & e^{s_1} & e^{s_2} \end{bmatrix}$$
- The log-loss

$$L(\boldsymbol{x}_i, y_i; \boldsymbol{W}) = -\log p(y = y_i | \boldsymbol{x} = \boldsymbol{x}_i; \boldsymbol{W}) = -\log \frac{e^{s_{y_i}}}{e^{s_0} + e^{s_1} + e^{s_2}}$$

$$= -s_{y_i} + \log(e^{s_0} + e^{s_1} + e^{s_2})$$

Log-loss for each data point & mean loss

i	y_i	$p(y=j \mid \boldsymbol{x}=\boldsymbol{x}_i, \boldsymbol{W})$			log-loss
		$j=0$	$j=1$	$j=2$	
1	0	0.6	0.3	0.1	
2	1	0.2	0.7	0.1	
3	2	0.5	0.3	0.2	
4	2	0.3	0.3	0.4	

Log-loss for each data point & mean loss

i	y_i	$p(y=j\|\boldsymbol{x}=\boldsymbol{x}_i,\boldsymbol{W})$			log-loss
		$j=0$	$j=1$	$j=2$	
1	0	0.6	0.3	0.1	$-\log 0.6$
2	1	0.2	0.7	0.1	$-\log 0.7$
3	2	0.5	0.3	0.2	$-\log 0.2$
4	2	0.3	0.3	0.4	$-\log 0.4$

- Mean loss (without regularization)

$$L = \frac{1}{4}\sum_{i=1}^{4} -\log p(y = y_i|\boldsymbol{x} = \boldsymbol{x}_i; \boldsymbol{W})$$

$$= -\frac{1}{4}(\log 0.6 + \log 0.7 + \log 0.2 + \log 0.4)$$

Log-loss for each data point & mean loss

i	y_i	$p(y = j \vert \boldsymbol{x} = \boldsymbol{x}_i, \boldsymbol{W})$			$\downarrow$ log-loss
		$j = 0$	$j = 1$	$j = 2$	
1	0	0.6 ↑	0.3	0.1	$-\log 0.6$
2	1	0.2	0.7 ↑	0.1	$-\log 0.7$
3	2	0.5	0.3	0.2 ↑	$-\log 0.2$
4	2	0.3	0.3	0.4 ↑	$-\log 0.4$

- Mean loss (without regularization)

$$\downarrow L = \frac{1}{4}\sum_{i=1}^{4} -\log p(y = y_i | \boldsymbol{x} = \boldsymbol{x}_i; \boldsymbol{W})$$

$$= -\frac{1}{4}(\log 0.6 + \log 0.7 + \log 0.2 + \log 0.4)$$

Outline

- Review the lecture, background knowledge, etc.
  - Supervised learning as an optimization problem
  - Perceptron update rule & loss function
  - Logistic regression
    - Predict function
    - Log loss (a.k.a. cross entropy)

- Notebook tasks
  - Task 1: Logistic regression
  - Task 2: Perceptron classifier