CSC321 Neural Networks and Machine Learning

Lecture 3

January 22, 2020

Agenda

First hour:

- Multi-class classification
- Feature Mapping

Second hour:

- k-Nearest Neighbours
- Generalization

Announcement

- Homework 2 is due tomorrow
- Project 1 is due next week
- Homework 1 is graded
  - Very well done!
  - Solutions are on **Quercus**
  - Remark request due Jan 28th, 9pm on **Markus**

Review

1. In a supervised learning setup, what does $x_j^{(i)}$ represent?
2. What is the shape of the vector $\frac{\partial \mathcal{E}}{\partial \mathbf{w}}$?
3. What does α represent in the gradient descent step: $\mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{\partial \mathcal{E}}{\partial \mathbf{w}}$?
4. What can happen if α is too large? Too small?
5. What is the **batch size**? What happens if it is too large? Too small?

Classification

Classification Setup

- Data: $(x^{(1)}, t^{(1)})$, $(x^{(2)}, t^{(2)})$, ... $(x^{(N)}, t^{(N)})$
- The $x^{(i)}$ are called *inputs*
- The $t^{(i)}$ are called *targets*

In classification, the $t^{(i)}$ are discrete.

In binary classification, we used the labels $t \in \{0, 1\}$. Training examples with

- $t = 1$ is called a **positive example**
- $t = 0$ is called a **negative example** (sorry)

Multi-class classification

Instead of there being two targets (pass/fail, cancer/not cancer, before/after 2000), we have $K > 2$ targets.

Example:

- Beatles ($K = 4$):
  - John Lennon, Paul McCartney, George Harrison, Ringo Starr
- Pets ($K =$ something large):
  - cat, dog, hamster, parrot, python, ...

Representing the targets

We use a **one-hot vector** to represent the target:

$$\mathbf{t} = (0, 0, ..., 1, ..., 0)$$

This vector contains $K - 1$ zeros, and a single 1 somewhere.

Each *index* (column) in the vector represents one of the classes.

Representing the prediction

The prediction $\mathbf{y}$ will also be a vector. Like in logistic regression there will be a linear part, and an activation function.

Linear part: $\mathbf{z} = \mathbf{W}^T\mathbf{x} + \mathbf{b}$

So far, this is like having K separate logistic regression models, one for each element of the one-hot vector.

Q: What are the shapes of $\mathbf{z}$, $\mathbf{W}$, $\mathbf{x}$ and $\mathbf{b}$?

Activation Function

Instead of using a *sigmoid* function, we instead use a **softmax activation** function:

$$y_k = \text{softmax}(z_1, ..., z_K)_k = \frac{e^{z_k}}{\sum_{m=1}^{K} e^{z_m}}$$

The predictions y_k is now a **probability distribution** over the classes!

Why softmax?

- Softmax is like the multi-class equivalent of sigmoid
- Softmax is a continuous analog of the “argmax” function
- If one of the z_k is much larger than the other, then the softmax will be approximately the argmax, in the one-hot encoding

Cross-Entropy Loss

The cross-entropy loss naturally generalizes to the multi-class case:

$$\begin{aligned}\mathcal{L}(\mathbf{y}, \mathbf{t}) &= -\sum_{k=1}^{K} t_k \log(y_k) \\ &= -\mathbf{t}^T \log(\mathbf{y})\end{aligned}$$

Recall that only one of the t_k is going to be 1, and the rest are 0.

Summary

Hypothesis	$\mathbf{y} = \text{softmax}(\mathbf{W}^T\mathbf{x} + \mathbf{b})$
Loss Function	$\mathcal{L}(\mathbf{y}, \mathbf{t}) = -\mathbf{t}^T \log(\mathbf{y})$
Optimization Problem	$\min_{\mathbf{W},\mathbf{b}} \mathcal{E}(\mathbf{W}, \mathbf{b})$
Gradient Descent	$\mathbf{W} \leftarrow \mathbf{W} - \alpha \frac{\partial \mathcal{E}}{\partial \mathbf{W}}, \mathbf{b} \leftarrow \mathbf{b} - \alpha \frac{\partial \mathcal{E}}{\partial \mathbf{b}}$

Example: Beatle Recognition

Given a 100x100 pixel colour image of a face of a Beatle, identify the Beatle

Four possible labels:

- John Lennon
- Paul McCartney
- George Harrison
- Ringo Starr

Aside: Representing an image

This is what John Lennon looks like to a computer:

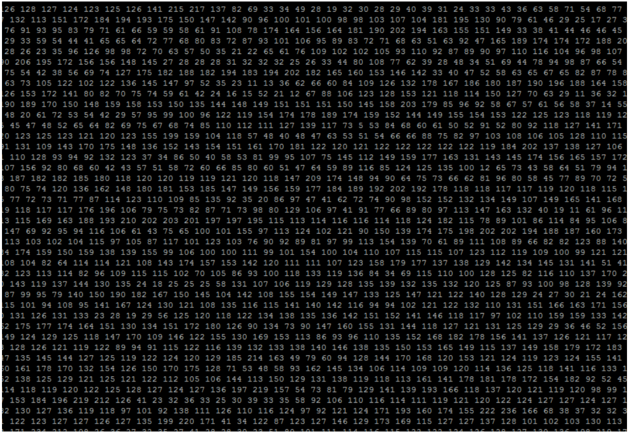
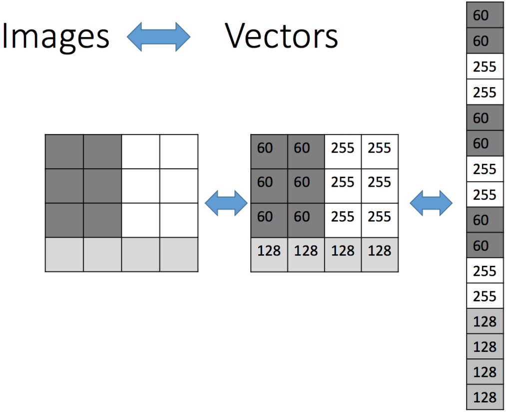


Image as a vector of features



Features and Targets

Each of our input images are 100x100 pixels

$\mathbf{y} = \text{softmax}(\mathbf{W}^T\mathbf{x} + \mathbf{b})$

Q: What will be the length of our input (feature) vectors $\mathbf{x}$?

Q: What will be the length of our one-hot targets $\mathbf{t}$?

Q: What are the shapes of $\mathbf{W}$ and $\mathbf{b}$?

Q: How many (scalar) parameters are in our model, in total?

Feature Mapping

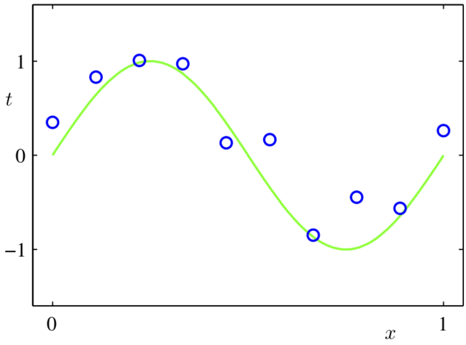
Computing New Features

In homework 2, we saw an example where computing **new features** could make a more powerful model.

- $x^{(1)} = -1,\ t^{(1)} = 1$
- $x^{(2)} = 1,\ t^{(2)} = 0$
- $x^{(3)} = 3,\ t^{(3)} = 1$

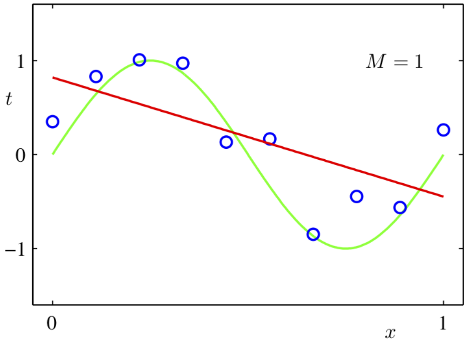
Example

Suppose we want to model the following data (from Bishop 2006):



Q: Will the model $y = wx + b$ fit the data well?

Linear Regression



Polynomial Feature mapping

One option to build a more powerful model is to fit a low-degree polynomial:

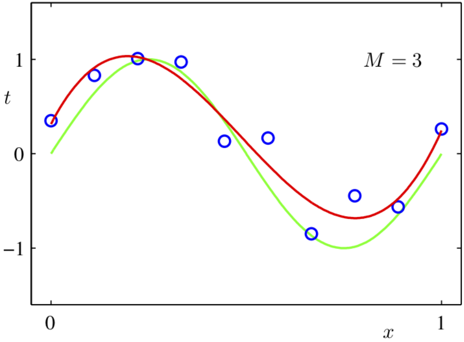
$$y = w_3 x^3 + w_2 x^2 + w_1 x + w_0$$

The above model can still be framed as linear regression, by taking

$$\mathbf{x} = \begin{bmatrix} 1 \\ x \\ x^2 \\ x^3 \end{bmatrix}$$

And we can find w_1, w_2, w_3 and $w_0 = b$ in the usual way.

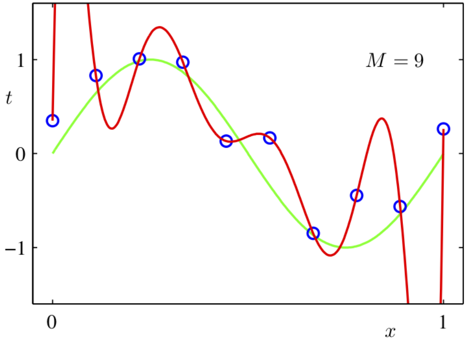
Fitting a degree 3 polynomial



Better fit!

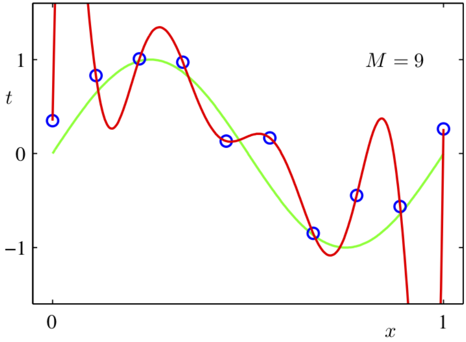
Higher degree polynomials

What about using an even higher degree polynomial?



Higher degree polynomials

What about using an even higher degree polynomial?



- This model fits the training data very well (cost $= 0$)
- ... but we don't expect this model to **generalize** to new data generated in the same way
- More parameter = more powerful model = more prone to **overfitting**

Feature Mapping in General

Computing the right features is very important. For example, if you want to predict whether someone will click on an ad for a machine learning book, you could compute:

- Last ad that they clicked on
- Last ad that they clicked on related to a book
- Time of day that this person is active
- How often this person clicks on ads

Q: How do you determine which features to include?

K-Nearest Neighbours

Same Example: Beatle Recognition

Given a 100x100 pixel colour image of a face of a Beatle, identify the Beatle

Four possible labels:

- John Lennon
- Paul McCartney
- George Harrison
- Ringo Starr

Linear Classification

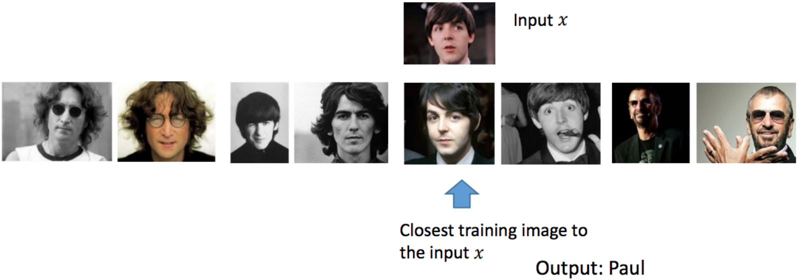
We already saw today that we can frame the problem as a logistic regression problem

$$\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$$
$$\mathbf{t} = \text{softmax}(\mathbf{z})$$

Another approach: 1-nearest neighbour

For a new image $\mathbf{x}$ for which we want to make a prediction:

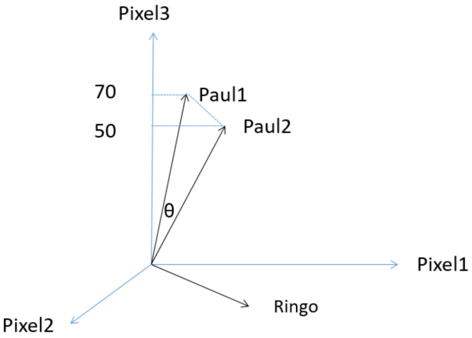
- Find the training photo/vector $\mathbf{x}^{(i)}$ that is "closest" to $\mathbf{x}$
- Output the prediction $\mathbf{y} = \mathbf{t}^{(i)}$



Are two images "close"?

Determining "closeness" using vector representations **a** and **b** of images

- Euclidean distance: $||\mathbf{a}-\mathbf{b}|| = \sqrt{\sum_i (a_i - b_i)^2} = \sqrt{(\mathbf{a}-\mathbf{b})^\mathbf{T} \cdot (\mathbf{a}-\mathbf{b})}$
- Cosine distance: $\cos(\theta_{ab}) = \frac{\mathbf{a}\cdot\mathbf{b}}{||\mathbf{a}||||\mathbf{b}||}$



Distances Measure Choice

Which distance measure makes sense?

Depends on the *invariance* that you want:

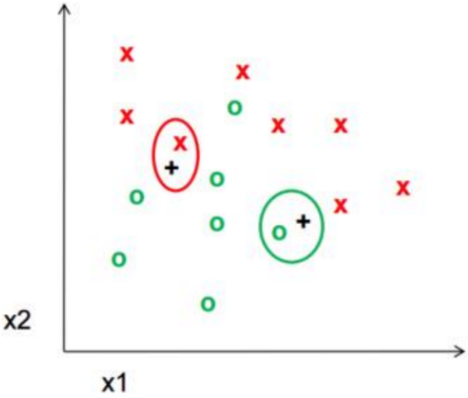
- Cosine distance is **scale invariant**: $dist(\mathbf{a}, \mathbf{b}) = dist(m\mathbf{a}, k\mathbf{b})$ for scalars m and K
- Euclidean distance is **shift invariant**: $dist(\mathbf{a}, \mathbf{b}) = dist(\mathbf{a} + \mathbf{c}, \mathbf{b} + \mathbf{c})$ for a vector $\mathbf{c}$

We'll use Euclidean distance in the next few slides, and cosine distance in project 1.

Example: 1-nearest neighbour

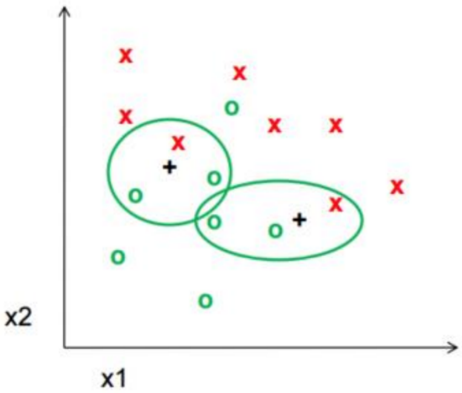
Task:

- Classify the new examples "+"
- Labels for the training set are GREEN and RED
- Choose Euclidean distance

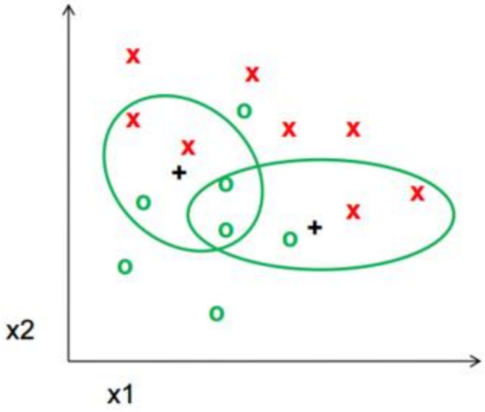


Example: 3-nearest neighbour

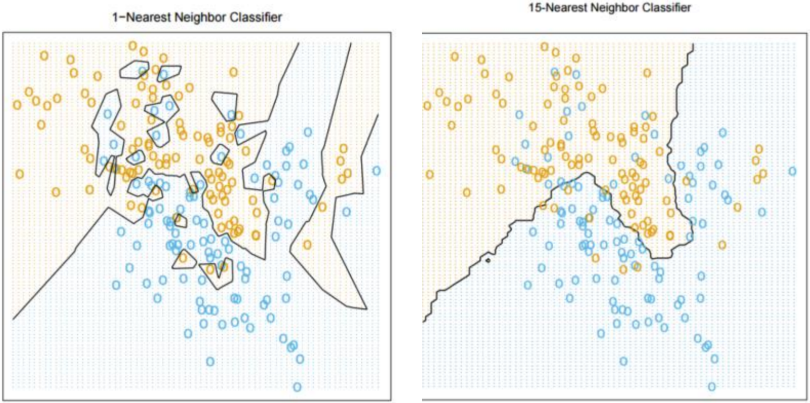
What if we use a larger set of neighbours?



Example: 5-nearest neighbour



Choice of k



- If k is too small, then our model might be too "noisy"
  - Small change in x often changes the prediction
  - Model is prone to *overfitting*
- If k is too large, then our model might be too "simple"
  - Extreme example: k = size of training set

kNN vs Linear Models

These two families of models are *very* different!

- Linear models have linear **decision boundaries**, and kNN models have arbitrary decision boundaries
- Linear models have **parameters** (weights) that we choose via solving an optimization problem
- The k-Nearest Neighbour model requires the entire training data to be available to make predictions

Remaining questions

- How do we choose k?
- How do we choose between different models?
- How do we know how well a model will perform on new data?

Generalization

Questions

- How do we choose k?
- How do we choose which features to include?
- How do we choose between different models?
- **How do we know how well a model will perform on new data?**

The Training Set

The training set is used

- to determine the value of the **parameters**
- (in kNN) to make predictions

The model's prediction accuracy over the training set is called the **training accuracy**.

Q: Can we use the **training accuracy** to estimate how well a model will perform on new data?

The Training Set

The training set is used

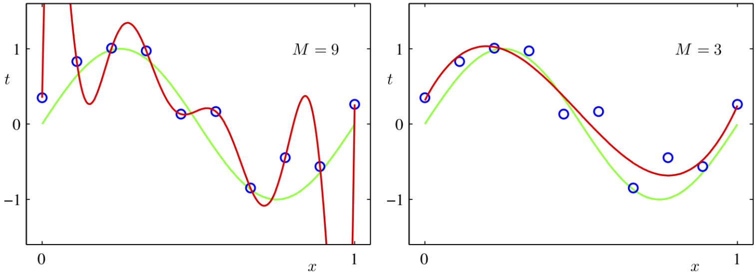
- to determine the value of the **parameters**
- (in kNN) to make predictions

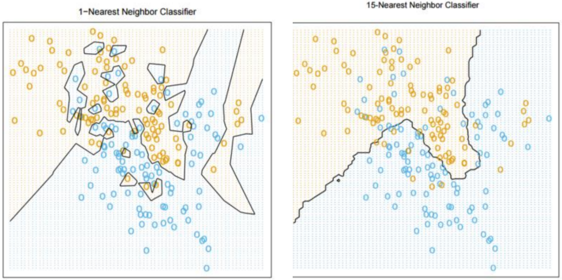
The model's prediction accuracy over the training set is called the **training accuracy**.

Q: Can we use the **training accuracy** to estimate how well a model will perform on new data?

- No! It is possible for a model to fit well to the training set, but fail to *generalize*
- We want to know how well the model performs on *new data* that we didn't already use to optimize the model

Poor Generalization





Overfitting and Underfitting

Underfitting:

- The model is simple and doesn't fit the data
- The model does not capture *discriminative* features of the data

Overfitting:

- The model is too complex and does not generalize
- The model captures information about patterns in training set that happened by chance
  - e.g. Ringo happens to be always wearing a red shirt in the training set
  - Model learns: high red pixel content => predict Ringo

Preventing Overfitting

- Use a larger training set (expensive, often not feasible)
- Use a smaller network (requires starting over, might underfit)
- Other techniques (we'll explore later)

The Test Set

We set aside a **test set** of labelled examples.

The model's prediction accuracy over the test set is called the **test accuracy**.

The purpose of the test set is to give us a good estimate of how well a model will perform on new data.

Q: In general, will the test accuracy be *higher* or *lower* than the training accuracy?

Model Choices

But what about decisions like:

- Which k to use?
- Which model to use?

Q: Why can't we use the test set to determine which model we should deploy?

Model Choices

But what about decisions like:

- Which k to use?
- Which model to use?

Q: Why can't we use the test set to determine which model we should deploy?

- If we use the test set to make modeling decisions, then we will overestimate how well our model will perform on new data!
- We are "cheating" by "looking at the test"

The Validation set

We therefore need a third set of labeled data called the **validation set**

The model's prediction accuracy over the validation set is called the **validation accuracy**.

This dataset is used to:

- Make decisions about models that is **not continuous** and can't be optimized via gradient descent
- Example: choose k, choose which features x_j to use, choose α, ...
  - These model settings are called **hyperparameters**
- The validation set is used to optimize **hyperparameters**

Splitting the data set

Example split:

- 60% Training
- 20% Validation
- 20% Test

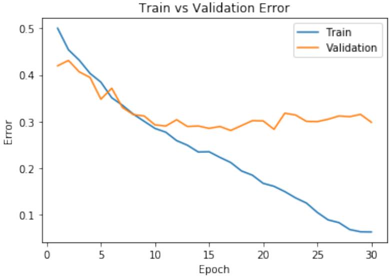
The actual split depends on the amount of data that you have.

If you have more data, you can get a way with a smaller % validation and set.

Detecting Overfitting

Learning curve:

- **x-axis**: epochs or iterations
- **y-axis**: cost, error, or accuracy



Q: In which epochs is the model overfitting? Underfitting?

Q: Why don't we plot the test accuracy plot?

Strategies to Preventing Overfitting

- Collect more data: always the best first thing to try
- Use a simpler model: doesn't work well in practice
- Early-stopping: stop training before training accuracy convergences
  - In practice, save (or **checkpoint**) the weights after every E epochs. Use the weights that produce the highest validation accuracy
- Use a training strategy that reduces overfitting:
  - Example: weight decay

Weight Decay Idea

Idea: Penalize **large weights**, by adding a term (e.g. $\sum_k w_k^2$) to the cost function

Q: Why is it not ideal to have large (absolute value) weights?

Weight Decay Idea

Idea: Penalize **large weights**, by adding a term (e.g. $\sum_k w_k^2$) to the cost function

Q: Why is it not ideal to have large (absolute value) weights?

Because large weights mean that the prediction relies **a lot** on the content of one pixel (or one feature)

Weight Decay

- L^1 regularization: add a term $\sum_{j=1}^{D} |w_j|$ to the cost function
  - Mathematically, this term encourages weights to be exactly 0
- L^2 regularization: add a term $\sum_{j=1}^{D} w_j^2$ to the cost function
  - Mathematically, in each iteration the weight is pushed towards 0
- Combination of L^1 and L^2 regularization: add a term $\sum_{j=1}^{D} |w_j| + w_j^2$ to the cost function

Example: Weight Decay for Regression

Cost function:

$$\mathcal{E}(\mathbf{w}, b) = \frac{1}{2N} \sum_i ((\mathbf{w}\mathbf{x}^{(i)} + b) - t^{(i)})^2$$

Cost function with weight decay:

$$\mathcal{E}_{WD}(\mathbf{w}, b) = \frac{1}{2N} \sum_i ((\mathbf{w}\mathbf{x}^{(i)} + b) - t^{(i)})^2 + \lambda \sum_j w_j^2$$

Weight Decay Nomanclature

$$\mathcal{E}_{WD}(\mathbf{w}, b) = \frac{1}{2N} \sum_i ((\mathbf{w}\mathbf{x}^{(i)} + b) - t^{(i)})^2 + \lambda \sum_j w_j^2$$

$$\frac{\partial \mathcal{E}_{WD}}{\partial w_j} = \frac{\partial \mathcal{E}}{\partial w_j} + \lambda 2 w_j$$

So the gradient descent update rule becomes:

$$w_j \leftarrow w_j - \alpha \left(\frac{\partial \mathcal{E}}{\partial w_j} + 2\lambda w_j \right)$$

Project 1 sklearn weight decay

- In project 1, you may get different answers from sklearn's logistic regression
  - Because of *stochastic* gradient descent
  - Because sklearn's logistic regression applies weight decay by default
- See https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegr

Reference

Some of these slides are based on the works of:

- Michael Guerzhoy
- Derek Hoiem
- Friedman, Hastie and Tibshirani
- Roger Grosse
- Bishop