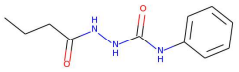
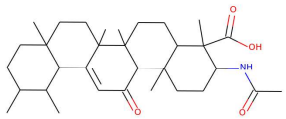
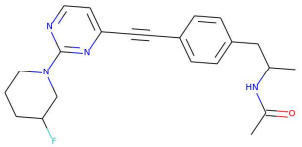
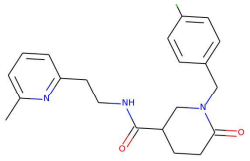
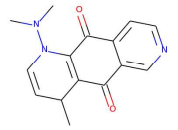
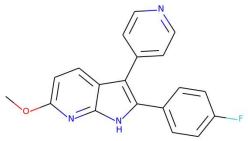
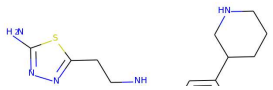
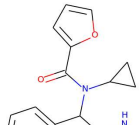
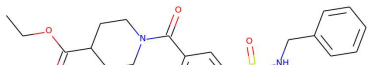
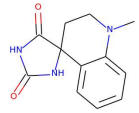
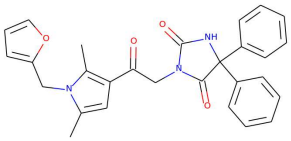
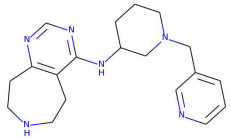
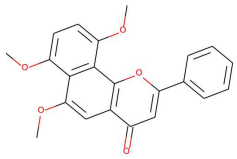
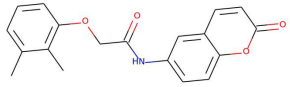
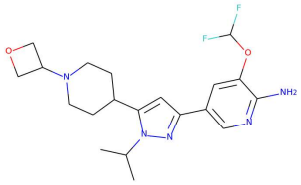
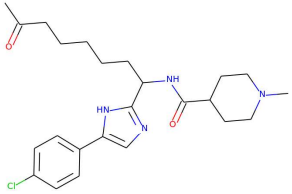
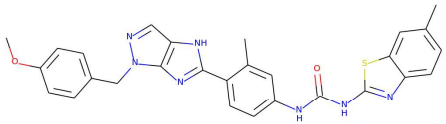
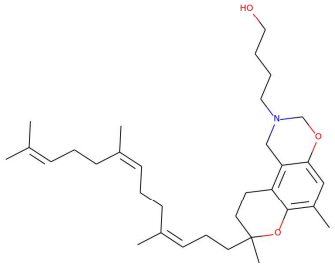
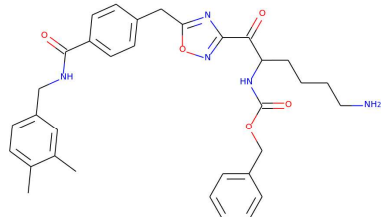











Machine Learning

Krzysztof Maziarz
Microsoft Research

Plan

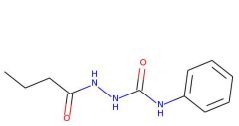
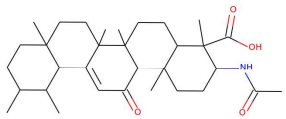
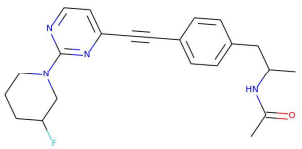
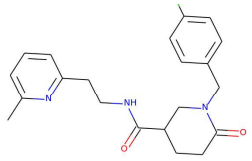
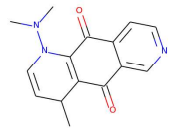
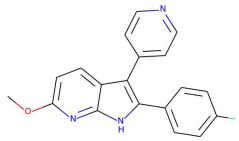
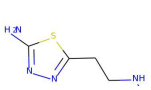
1. Co to jest machine learning?

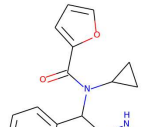
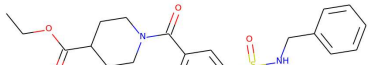
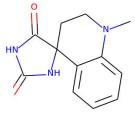
Plan

1. Co to jest machine learning?
2. Przykłady zastosowań
   - …głównie rzeczy, które jeszcze kilka lat temu nie były możliwe!
   - trochę o tym, czym sam zajmuję się w Microsoft Research

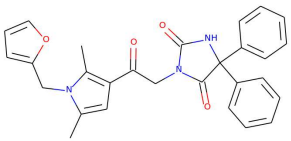
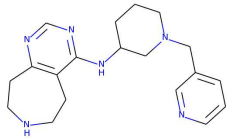
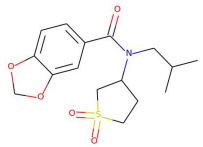
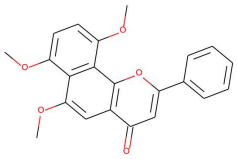
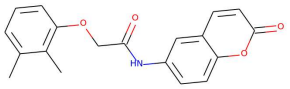
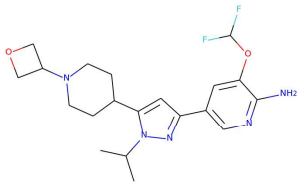
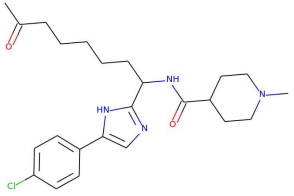
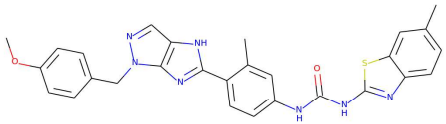
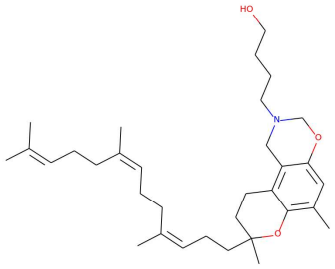
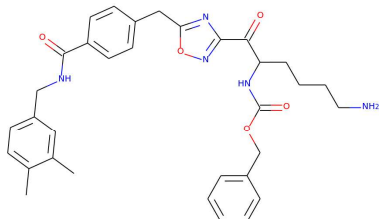
Plan

1. Co to jest machine learning?
2. Przykłady zastosowań
   - …głównie rzeczy, które jeszcze kilka lat temu nie były możliwe!
   - trochę o tym, czym sam zajmuję się w Microsoft Research
3. Przykład jak rozwiązać za pomocą ML (prosty) problem
   - wyjaśniony "do samego spodu" razem z kodem (C++)
   - bez wywoływania "magicznych bibliotek, które wszystko za nas robią"

Co to jest Machine Learning?

Machine Learning

Metody, które uczą się rozwiązywać problemy *na bazie przykładów*.

Machine Learning

Często używany gdy trudno jest nawet formalnie zdefiniować problem, a co dopiero opisać algorytm który miałby go rozwiązywać.

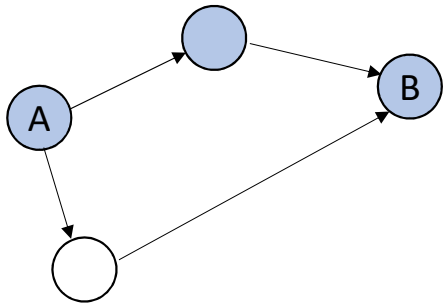
Często są to problemy które ludzie potrafią rozwiązywać całkiem dobrze, ale nie potrafią *formalnie* wyjaśnić jak to robią...

→ Człowiek → ? → Odpowiedź: muffin

Machine Learning

Klasyczna algorytmika

- Formalna definicja problemu i rozwiązania
- Odpowiedzi pewne i poprawne (a jeśli nie, to formalnie zdefiniowane jak bardzo pewne, i jak często poprawne)

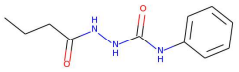
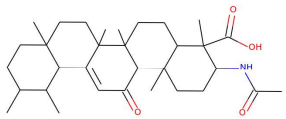
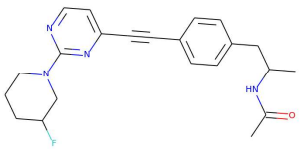
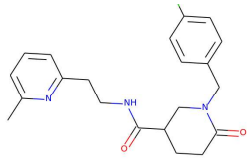
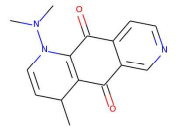
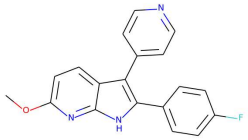
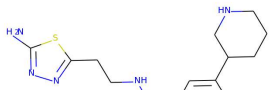
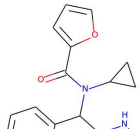
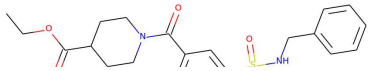
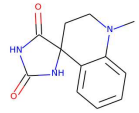


Machine Learning

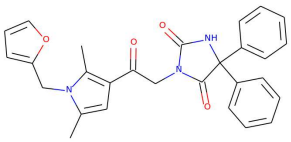
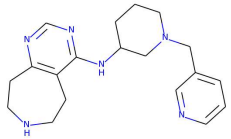
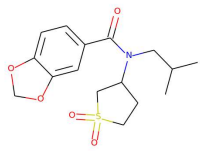
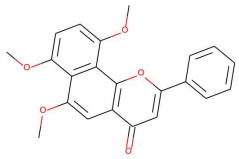
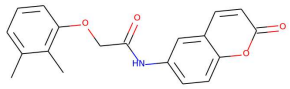
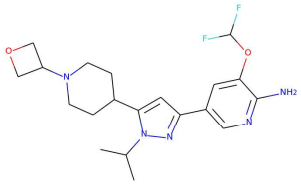
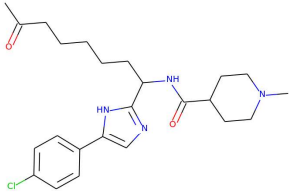
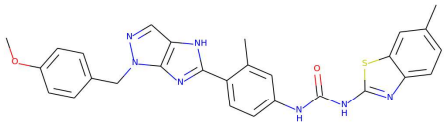
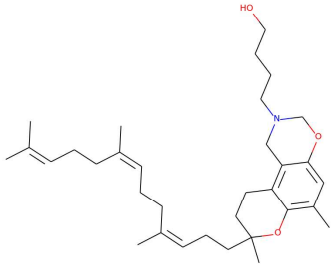
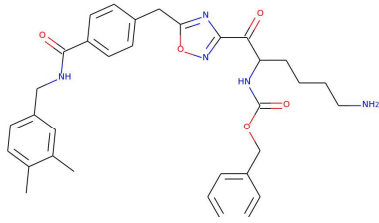
- Często brak ścisłej definicji, definicja przez przykłady
- Odpowiedzi często poprawne ale ciężko jest być którejkolwiek pewnym

Machine Learning: ogólny schemat

1. Mamy (luźno zdefiniowane) *dane*
2. Definiujemy *model*
   - schemat algorytmu, który wczytuje wejście (np. obrazek) i wypisuje odpowiedź (np. czy na obrazku jest chihuahua czy muffin)
   - w modelu są pewne "luki" (*parametry*): zmienne, których wartości nie znamy
3. Próbujemy dopasować wartości *parametrów*, tak żeby *model* dawał jak najlepsze odpowiedzi na *danych*

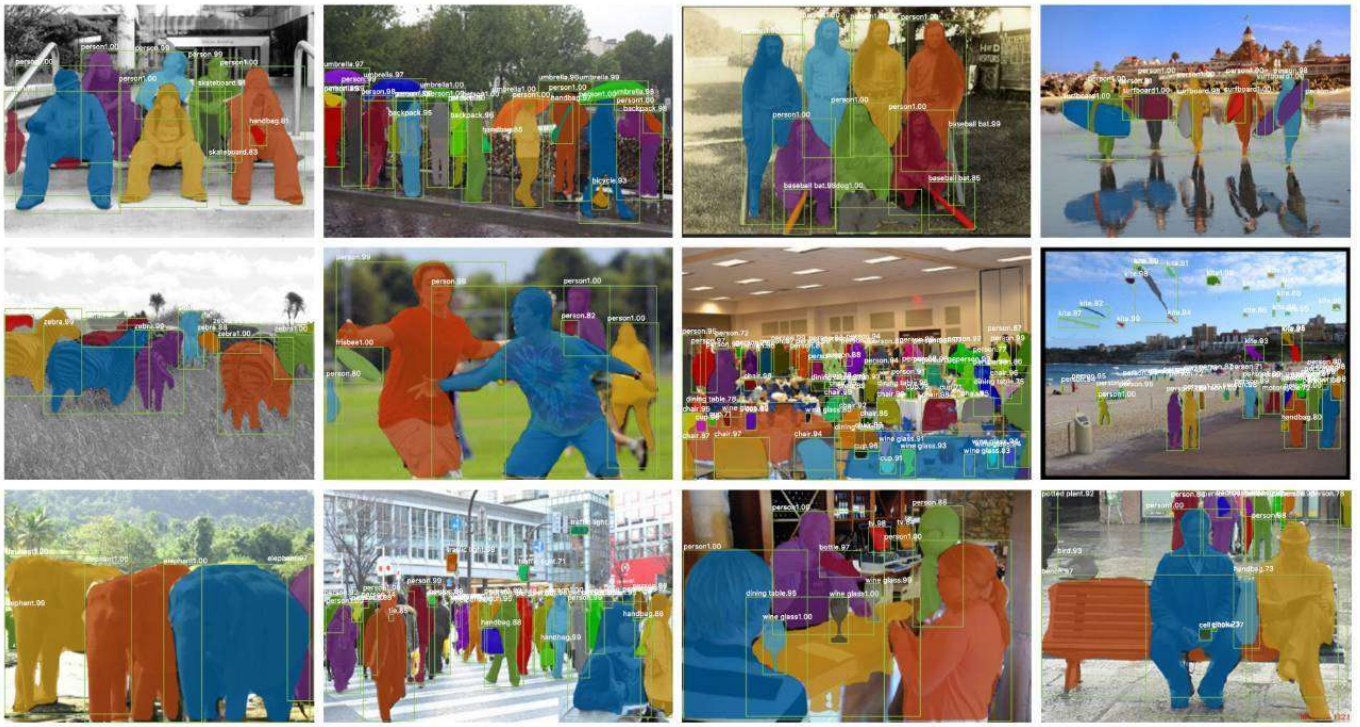











Przegląd zastosowań

Segmentacja obrazów (Facebook AI)

Klasyfikacja *każdego pixela* obrazka (rozróżniając pixele należące np. do różnych ludzi).



Generowanie tekstu (OpenAI)

Przewidywanie kolejnego słowa w tekście (uczone na 8 milionach stron internetowych).

When scientists finally managed to get into the ruins of Staszic, they were greeted with a peculiar sight. The unicorns were gone, replaced by monsters covered in ragged, bloody, bat-like wings.

Surprisingly, these monstrosities don't appear to be attacking anyone. Staszic is known to be home to many isolated societies. People come here to work, study, and flee the war-torn state of Croatia.

In early April, all of Staszic suddenly broke into a mass panic. Complete chaos ensued with hundreds of people fleeing to nearby villages for protection. Rumors circulated among the various clans and tribes. Some said that a deadly creature known as the Heptadrome Hydra had entered Stasz

The Staszic high school in Warsaw is marked by its incredible design. Built in 1976 by MPZ Intajan is easily one of the most beautiful buildings in Poland. http://www.bridgemedia.pl/wiki/A-Naive-Friend/Staszic-High-School-Warsaw-Pi... The beautiful tower contains many historical elements, including the two icons on the exterior: the front gate of the school, and the Collegium on the roof.
In the 1970s MPZ Intajan wanted to change the program at the high school, and add an outdoor theatre to the academic building. A problem at the time was that it was illegal to build

talktotransformer.com

Generowanie tekstu (OpenAI)

Model radzi sobie gorzej jeśli dany "rodzaj" tekstu był rzadki w danych treningowych.

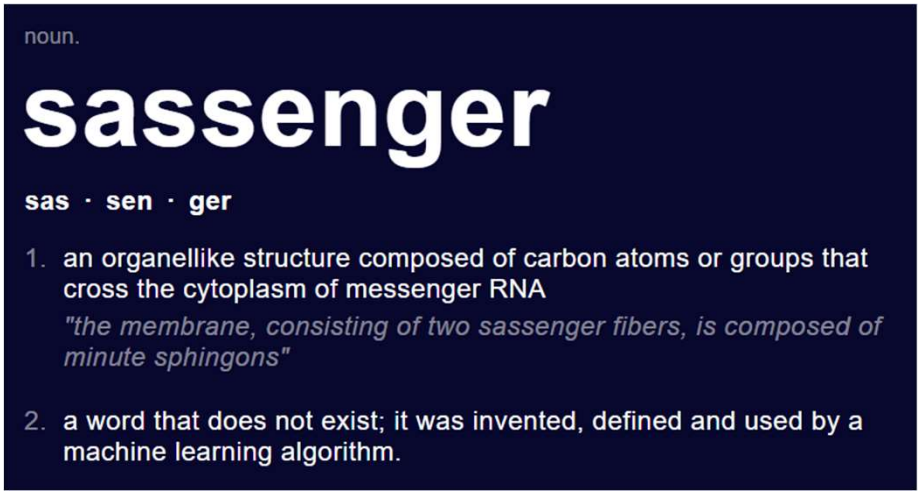
Ingredients:
- two dark chocolates
- like, a ton of peanut butter
- a pinch of sea salt
- 2 tablespoons unsalted butter
Directions:
Preheat the oven to 350 degrees. Line a 9x13 baking dish with parchment paper. Spread the chocolate chips out in an even layer on the baking dish.
Take the peanut butter out of the jar and shake it around to coat it. Then set the jar in the microwave for 20 seconds, until warm.
After microwaving, start melting the butter, mix it together, and pour it over the chocolate chips. Stir until it melts.
Stir it around until the peanut butter and chocolate are covered. Place the dish in the oven and bake for 30 minutes.

Przeglądając 8 milionów stron internetowych widziałem też parę po polsku, ale nie udało mi się nauczyć języka, gdyż hajcyc wartyło dwa, ale pozić kobiet nie karapyszeczyłym języki. Choszystając zostały moni jest krótkowe, wskorzysta na iżywienie wnieniał, sie zażeł przyję

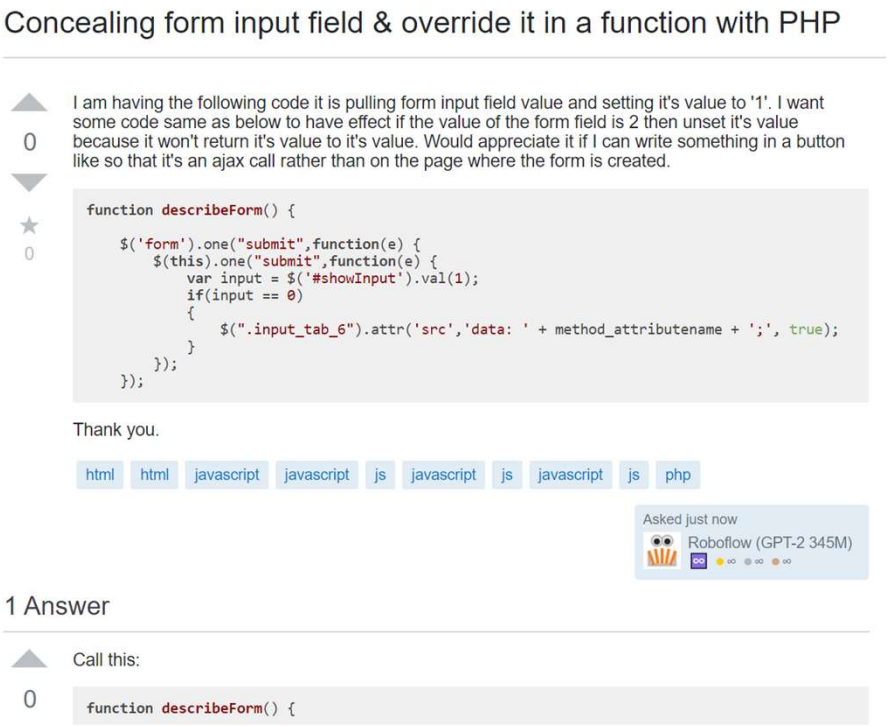
```
int compute_stuff(int x, int y, int z) {

if (z > 20) {

// This number is too big, as are all the coordinates above it.

return -1;

}
```

talktotransformer.com

Generowanie tekstu (OpenAI)



thisworddoesnotexist.com



stackroboflow.com

Generowanie obrazów (NVIDIA)

Inna z metod trenuje dwa modele uczenia maszynowego: *generator*, który generuje "z niczego" losowe obrazki, i *dyskryminator*, który uczy się odróżniać obrazki wygenerowane od tych w zbiorze danych.

Generator trenowany jest aby "oszukać" dyskryminator.

Dyskryminator trenowany jest aby odróżnić obrazki z generatora od prawdziwych.

Generowanie obrazów (NVIDIA)

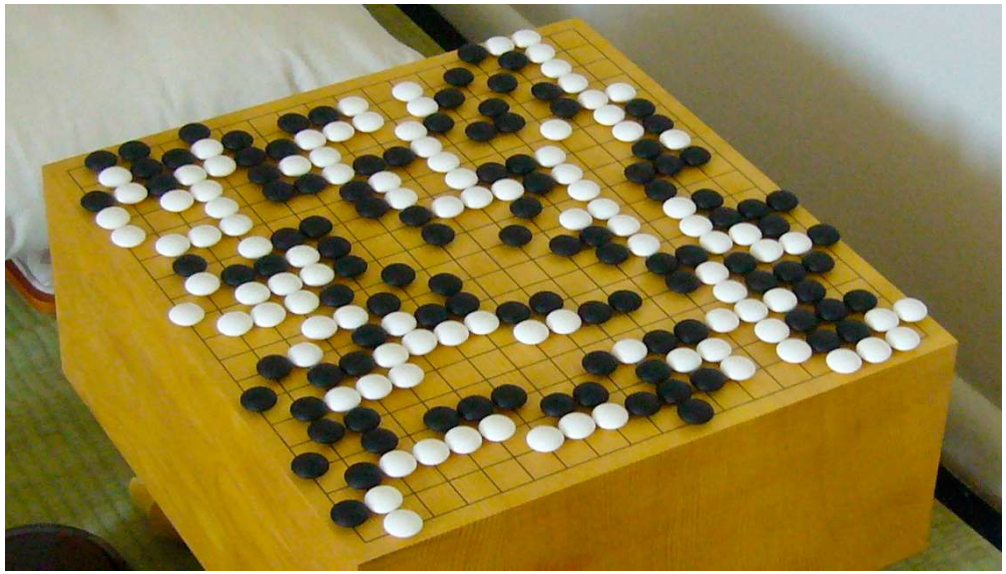
Z czasem generator zaczyna tworzyć obrazki, które ciężko odróżnić od prawdziwych...



thispersondoesnotexist.com

Gra w Go (Google DeepMind)

Go: gra strategiczna, uważana za trudniejszą od szachów - szczególnie dla komputerów (o *wiele* więcej możliwych ruchów...)



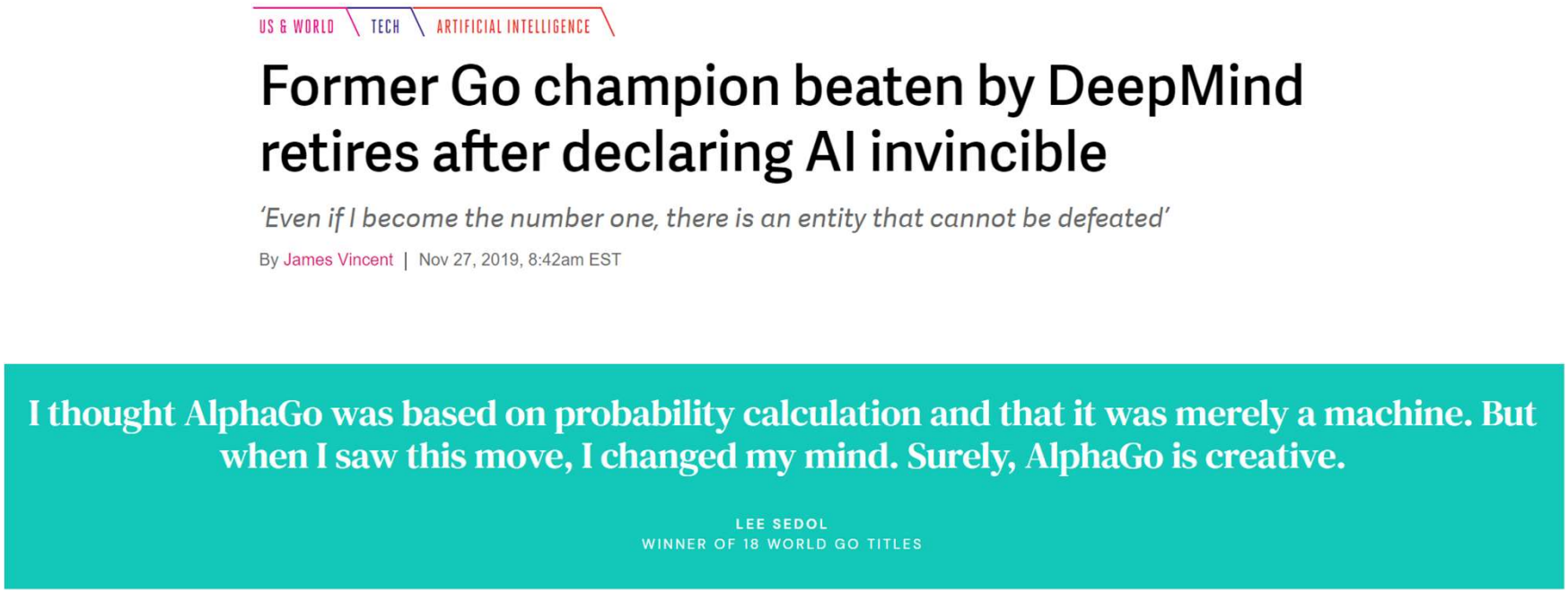
Gra w Go (Google DeepMind)

Można nauczyć model grać w grę, trenując go by przewidywał ruch jaki zrobiłby człowiek (mając bazę gier rozegranych przez profesjonalnych graczy).

...ale wtedy nie będzie w stanie być lepszym od *wszystkich* ludzi.

Gra w Go (Google DeepMind)

Lepszy pomysł: trenować model aby próbował wygrać z *poprzednią wersją* samego siebie.



US & WORLD \ TECH \ ARTIFICIAL INTELLIGENCE \

Former Go champion beaten by DeepMind retires after declaring AI invincible

'Even if I become the number one, there is an entity that cannot be defeated'

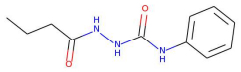
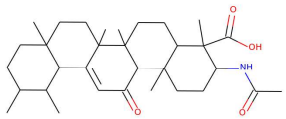
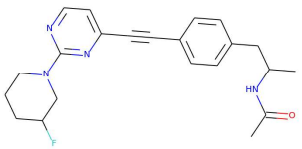
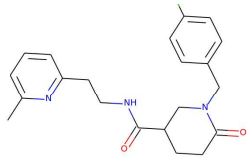
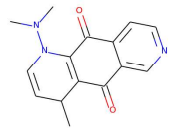
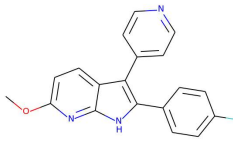
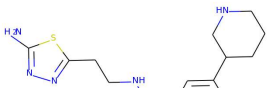
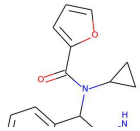
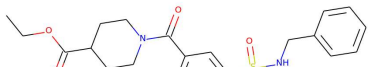
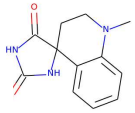
By James Vincent | Nov 27, 2019, 8:42am EST

I thought AlphaGo was based on probability calculation and that it was merely a machine. But when I saw this move, I changed my mind. Surely, AlphaGo is creative.

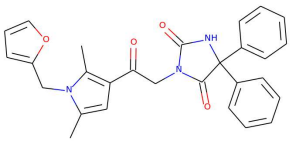
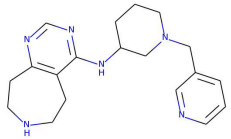
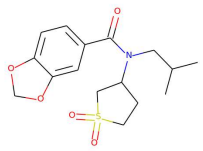
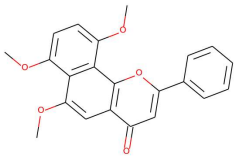
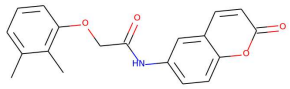
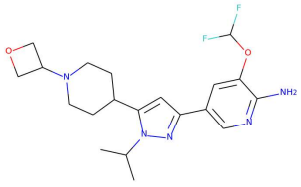
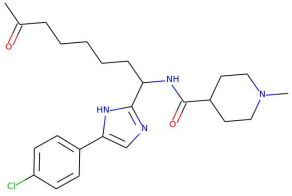
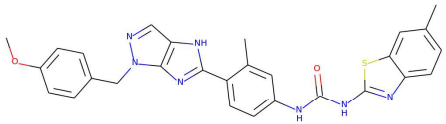
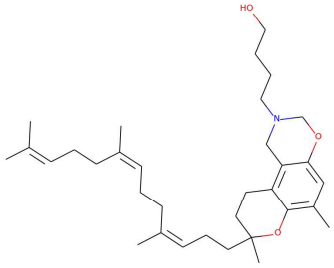
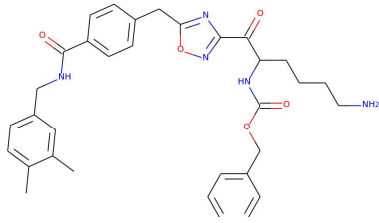
LEE SEDOL
WINNER OF 18 WORLD GO TITLES

Transfer stylu (deepart.io)

Mając model który potrafi patrzeć na obrazek zarówno wysokopoziomowo (co na nim jest) jak i niskopoziomowo (styl), można łączyć zawartość jednego zdjęcia ze stylem innego...

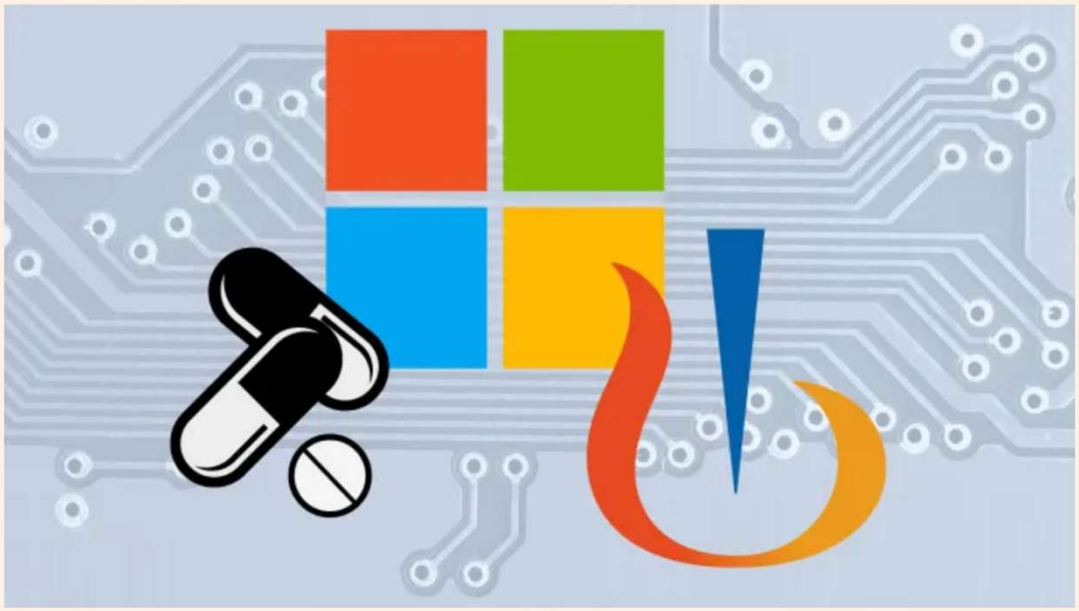
Generowanie cząsteczek

Generowanie cząsteczek (Microsoft Research)

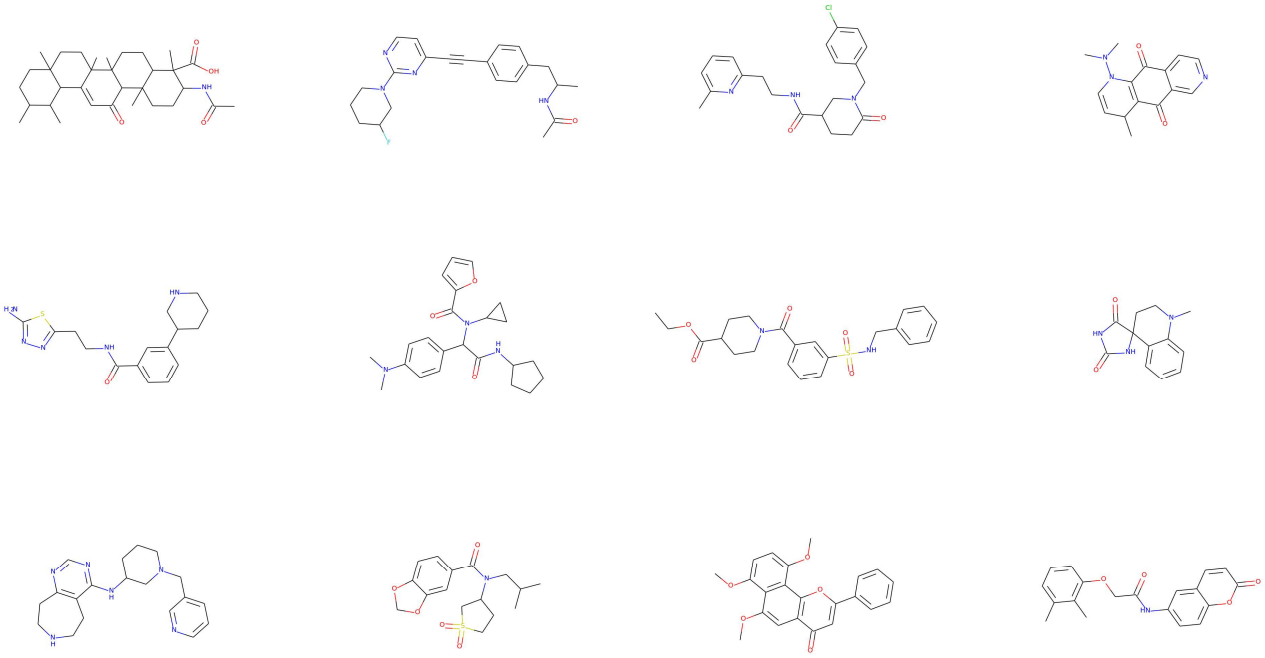
Novartis and Microsoft join forces to develop drugs using AI

Five-year agreement is one of the most expansive tie-ups between big pharma and big tech



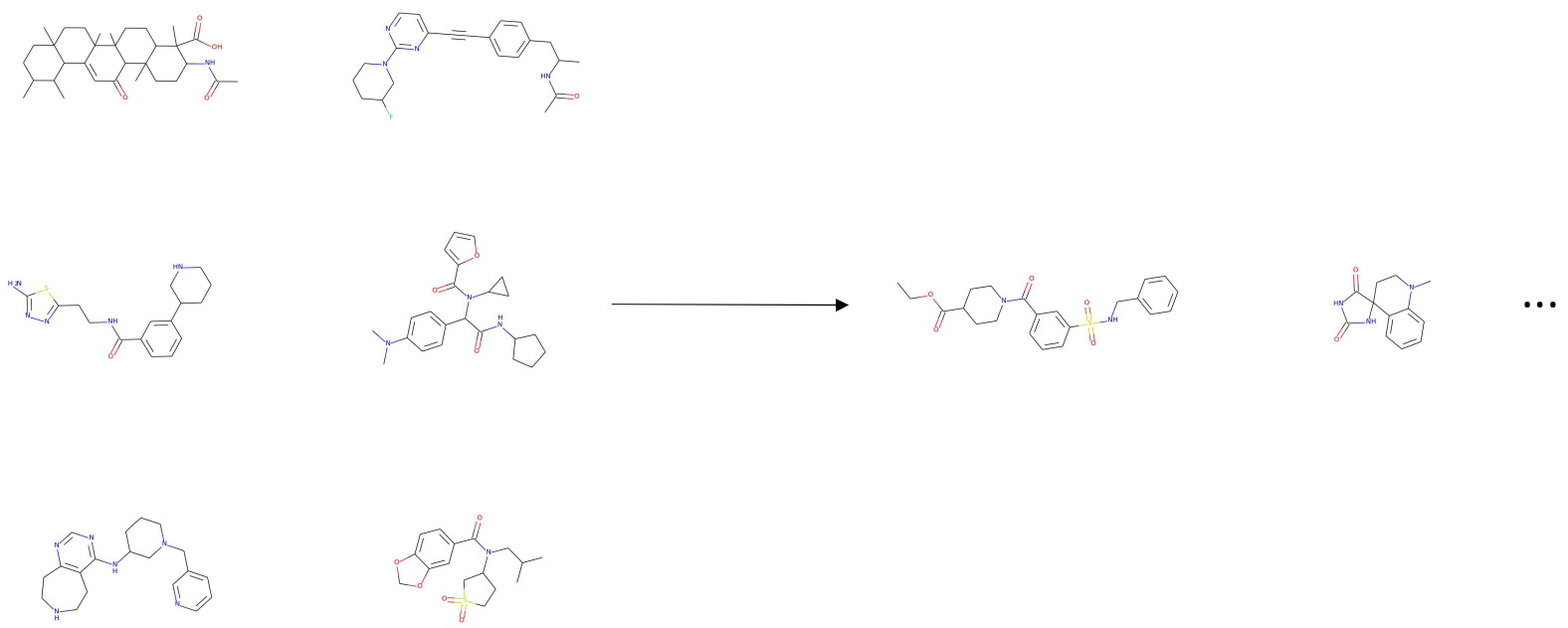
Generowanie cząsteczek (Microsoft Research)

Proces projektowania nowych leków zaczyna się od proponowania wielu cząsteczek, które nie były jeszcze zbadane, ale są w jakimś sensie *podobne* do istniejących leków.



Generowanie cząsteczek (Microsoft Research)

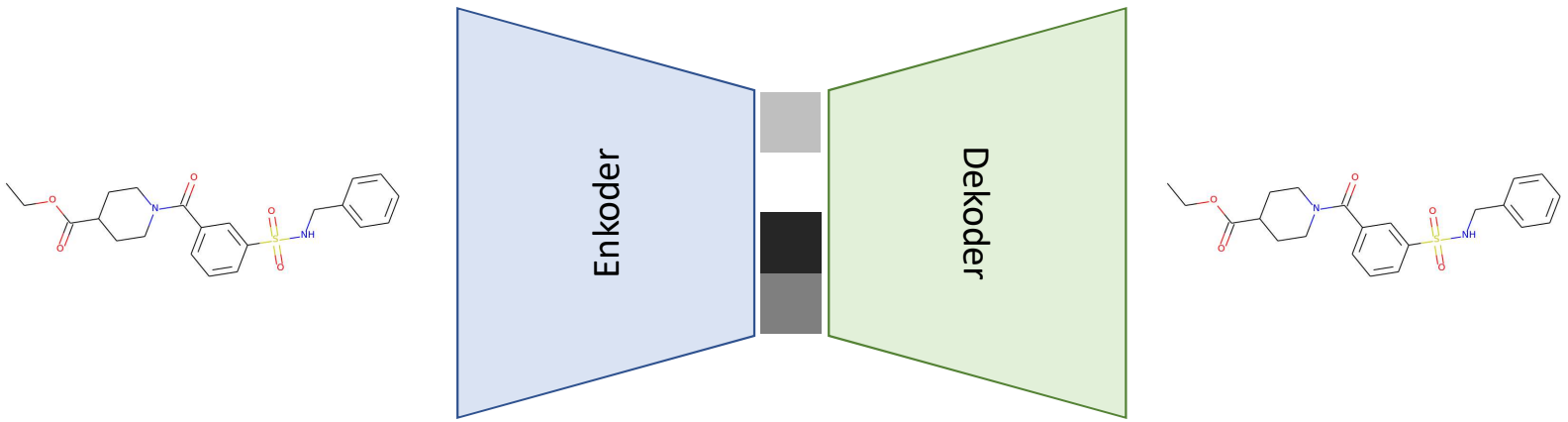
Mając dużą bazę cząsteczek (na przykład leków, lub po prostu "sensownych" substancji), chcemy generować *nowe, ale podobne*...



Generowanie cząsteczek (Microsoft Research)

Jedno z podejść opiera się o model nazywany *autoenkoderem.*

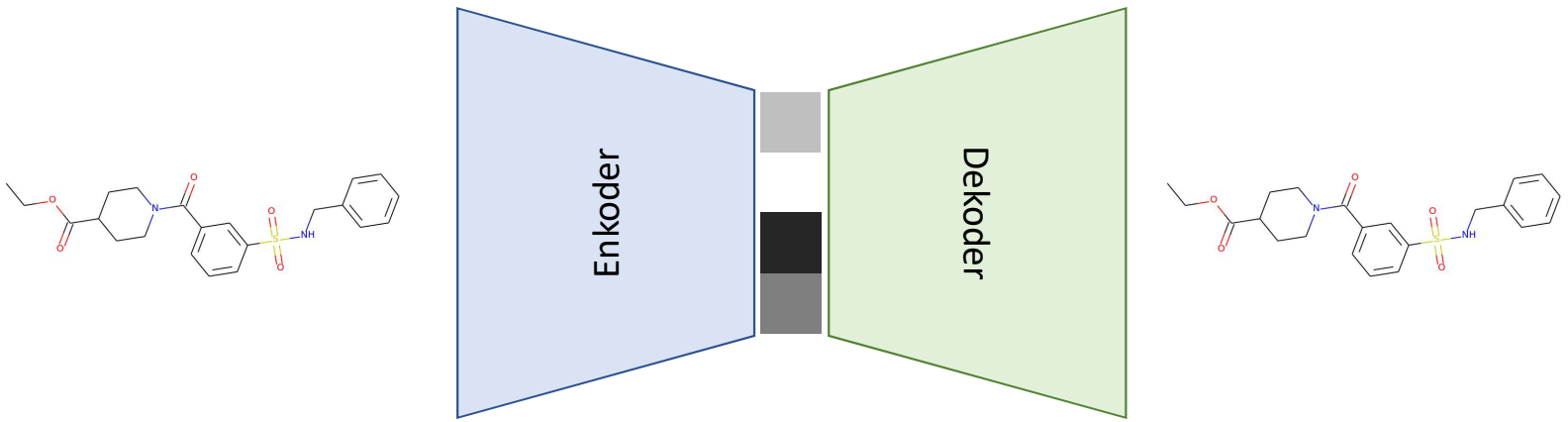
Jest to taki model, który podane wejście (tutaj cząsteczkę) musi skompresować w (dosyć kompaktowy) *kod* (przykładowo złożony z 64 liczb), a następnie rozkompresować (spowrotem do cząsteczki).



Generowanie cząsteczek (Microsoft Research)

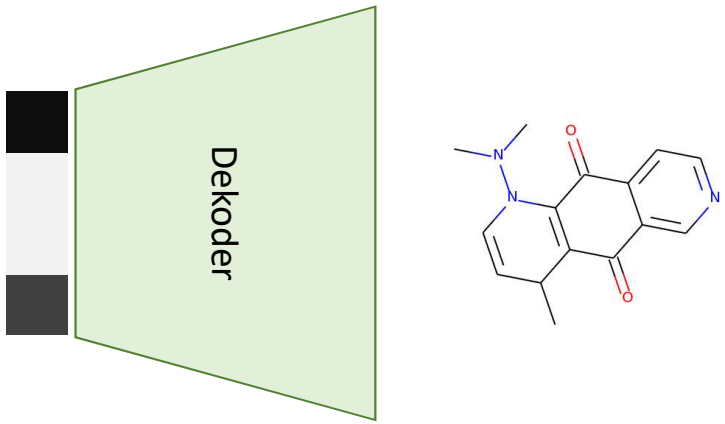
Model jest trenowany by wyjściowa (zdekodowana) cząsteczka była taka, jak wejściowa.

Gdy cząsteczek w danych jest dużo, model upakuje je w "przestrzeni możliwych kodów" bardzo gęsto...



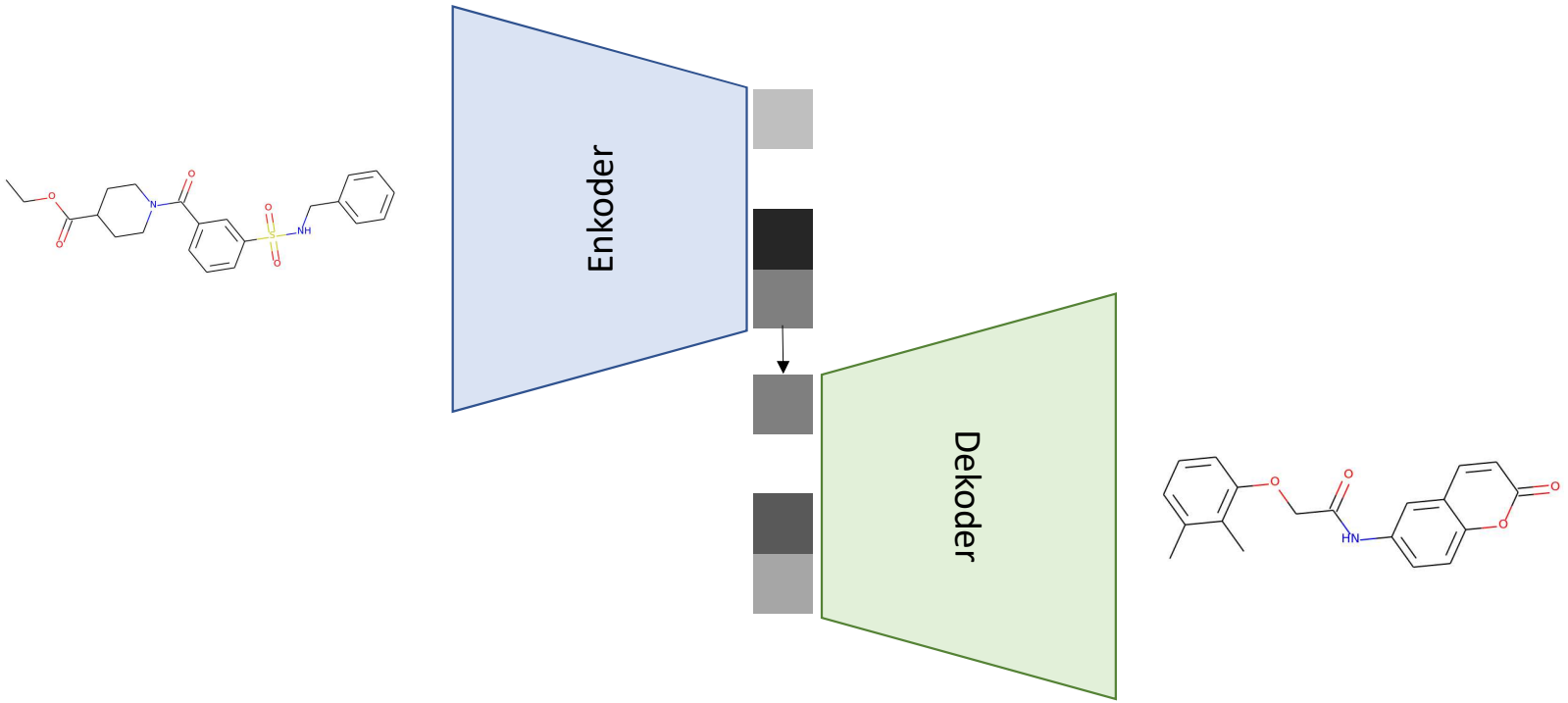
Generowanie cząsteczek (Microsoft Research)

Nowe cząsteczki możemy generować biorąc *losując kod*, i uruchamiając na nim dekoder:



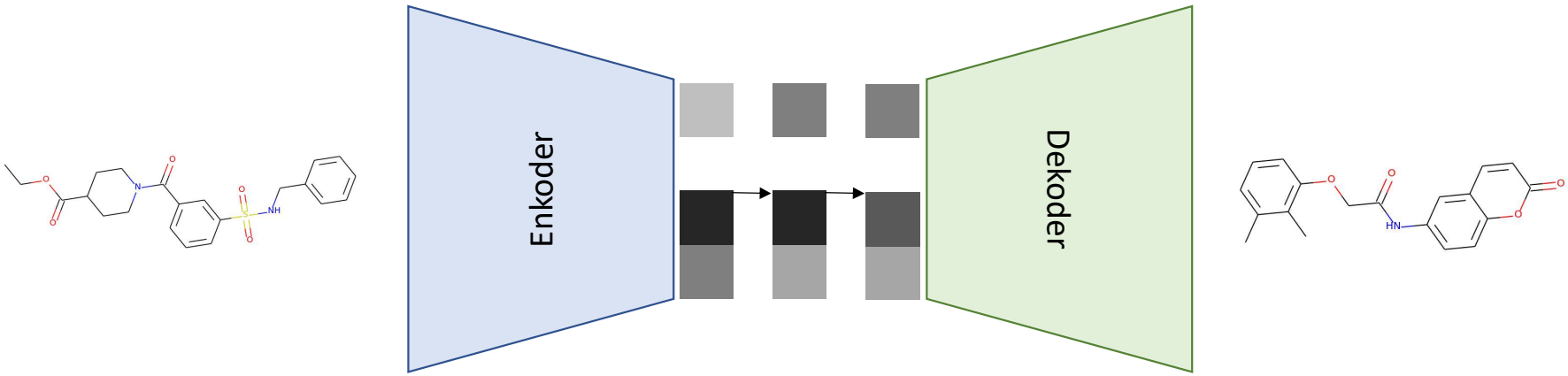
Generowanie cząsteczek (Microsoft Research)

Cząsteczki podobne do zadanej możemy generować zakodowując ją, i rozkodowując "niedalekie" kody (kody to ciągi liczb, więc łatwo jest mówić o niedalekich kodach):



Generowanie cząsteczek (Microsoft Research)

Jeśli szukamy cząsteczki podobnej do zadanej, ale "lepszej" pod względem jakichś własności (np. lepiej przyswajalnej przez człowieka), możemy zakodować startową cząsteczkę, potem próbować iteracyjnie poprawiać* jej kod, i na końcu zdekodować.

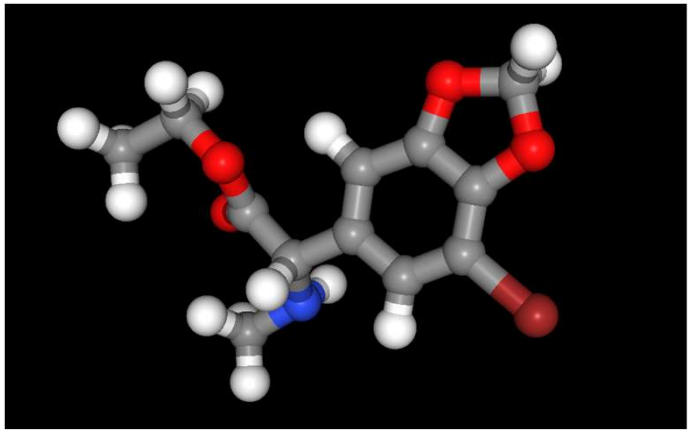


*potrzebujemy sposobu przewidywania przyswajalności na podstawie kodu – na przykład oddzielnego modelu...

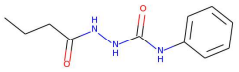
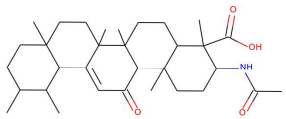
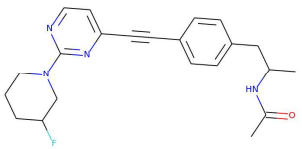
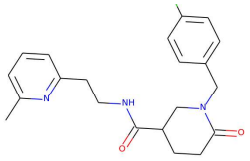
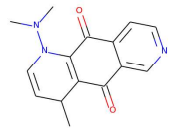
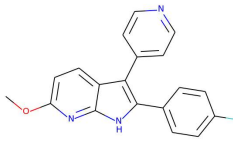
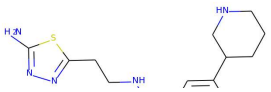
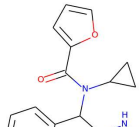
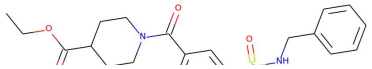
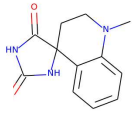
Generowanie cząsteczek (?)

Szybkie demo* generowania cząsteczek, w podobnym stylu do *thispersondoesnotexist*:

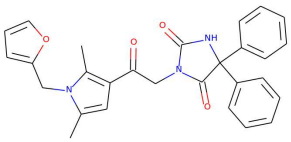
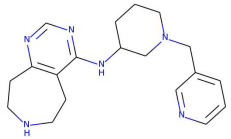
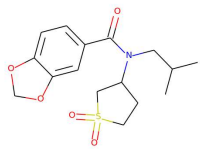
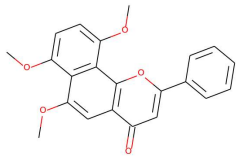
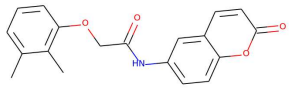
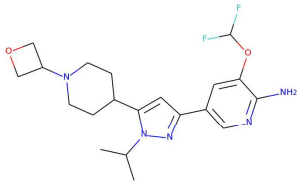
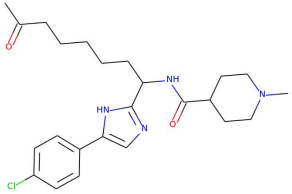
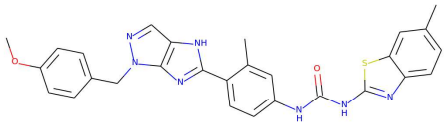
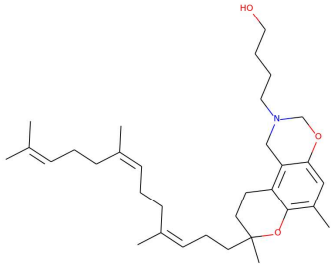
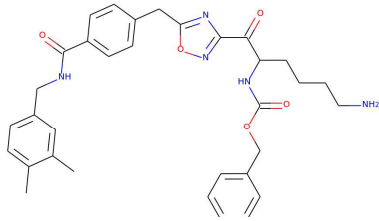
thischemicaldoesnotexist.com



* niestety brak tutaj kontekstu - nie wiem kto jest autorem i jaki model został użyty, ale jest wizualizacja w 3D

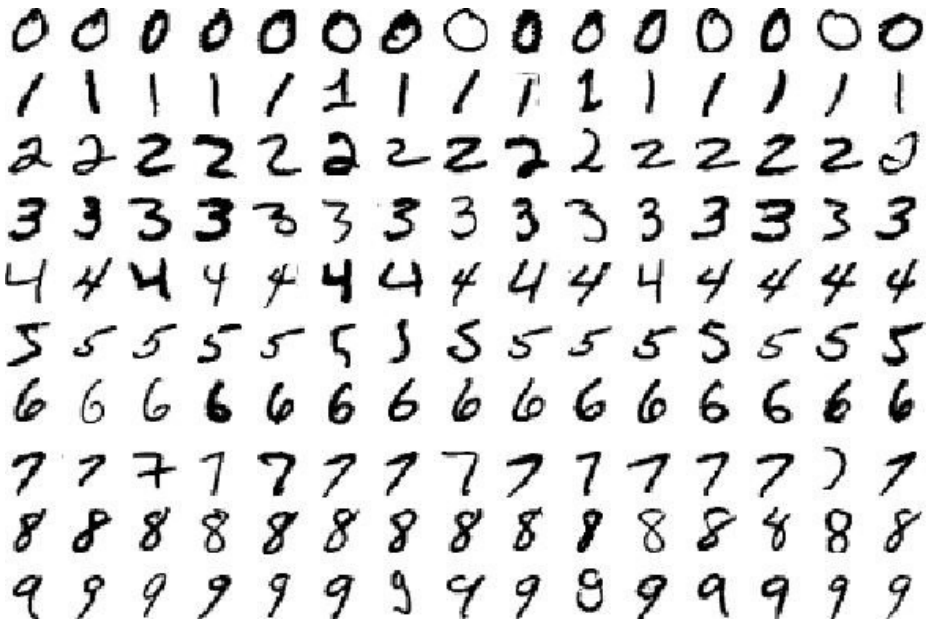











Prosty przykład z kodem

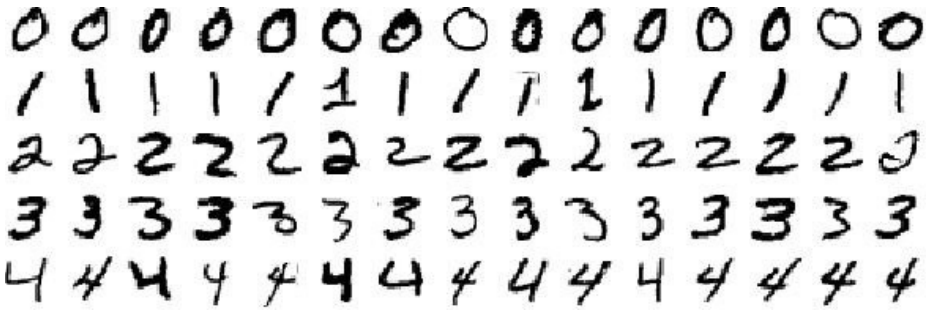
MNIST

"Hello World" uczenia maszynowego: 60 000 czarno-białych obrazków 28x28 przedstawiających ręcznie pisane cyfry.



MNIST

Dla uproszczenia, ograniczymy się do połowy cyfr (0-4):



Dane można pobrać z http://yann.lecun.com/exdb/mnist/ (pliki *train-images-idx3-ubyte.gz* oraz *train-labels-idx1-ubyte.gz*).

Pierwszy plik zawiera 60 000 obrazków, drugi 60 000 liczb określających cyfrę dla każdego obrazka.

MNIST

Każdy element w naszych danych to obrazek (28 * 28 = 784 pixele), i klasa (cyfra):

```
// Ograniczamy sie do cyfr [0, 1, ..., NUM_CLASSES-1].
const int NUM_CLASSES = 5;

// Obrazek (jako ciag 28 * 28 liczb) + klasa.
struct Datapoint {
    vector <int> image;
    int label;
};
```

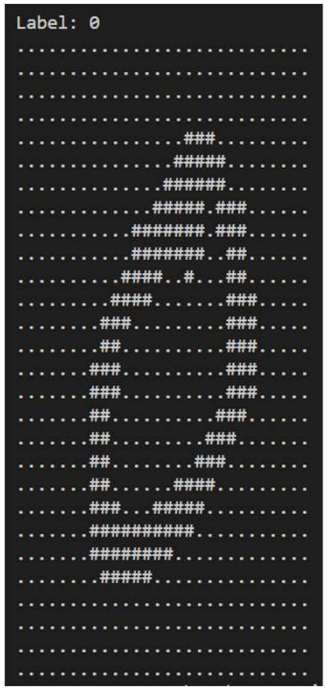
Stosunkowo łatwo możemy wczytać ściągnięte dane jako:

```
vector <Datapoint> read_mnist_data(string filename_images, string filename_labels)
```

MNIST

Jasność każdego pixela to liczba z przedziału [0, 255]. Żeby sprawdzić czy wszystko jest okej, możemy wyświetlić sobie kilka z nich jako ascii-arty :-)

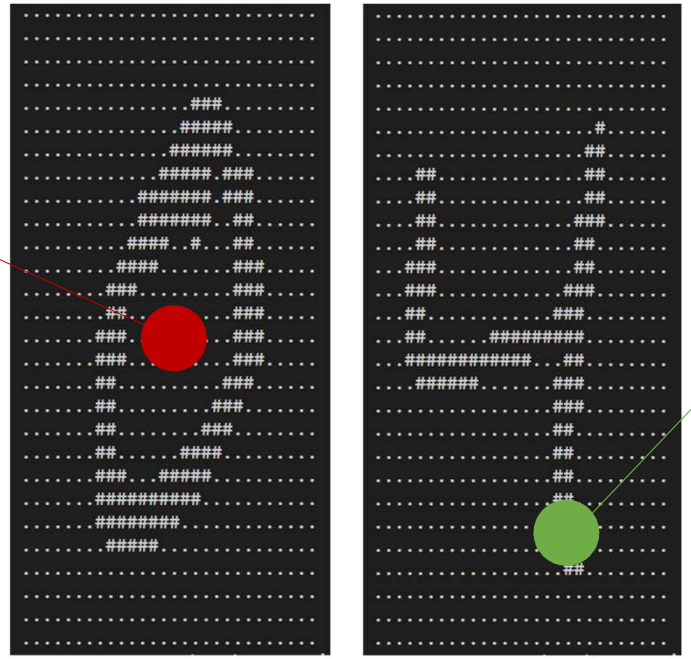
```
void show_datapoint(Datapoint &datapoint) {
    // Prymitywne wyswietlanie obrazka na standardowe wyjscie.

    cout << "Label: " << datapoint.label << endl;
    for (int i = 0; i < (int) datapoint.image.size(); i++) {
        int pixel_value = datapoint.image[i];

        if (pixel_value > 100) {
            cout << '#';
        } else {
            cout << '.';
        }

        if (i % 28 == 27) {
            // Koniec wiersza.
            cout << '\n';
        }
    }
}
```



MNIST

Musimy teraz stworzyć *model*. Nasz model będzie prosty: popatrzymy na każdy pixel z osobna, i założymy, że każdy pixel jest "związany" z jednymi cyframi bardziej, a z innymi mniej.

Jeśli te pixele są czarne, to *zmniejsza* to szansę, że odpowiedzią jest 0.



Jeśli te pixele są czarne, to *zwiększa* to szansę, że odpowiedzią jest 4.

MNIST

Dla każdej z klas (tj. 5 rodzajów cyfr), nasz model będzie miał 784 *parametry* (każdy parametr to po prostu liczba rzeczywista).

Parametry te będą odpowiadać poszczególnym pixelom. Parametr dla klasy *d* i pixela *i* będzie opisywał, czy gdy pixel *i* jest czarny, to czy przemawia to za tym, że cyfra na którą patrzymy to *d*, czy może przeciwko temu.

```
vector <vector <double>> &coefficients
```

Sumarycznie, nasz model jest opisywany przez 5 * 784 liczby.

MNIST

Musimy jeszcze zdefiniować, jak nasz model będzie obliczał wynik:

```
int predict(vector <int> &image, vector <vector <double>> &coefficients) {
    vector <double> sums(NUM_CLASSES, 0.0);

    // Liczymy odpowiednia sume dla kazdej mozliwej cyfry...
    for (int digit = 0; digit < NUM_CLASSES; digit++) {
        for (int i = 0; i < (int) image.size(); i++) {
            sums[digit] += image[i] * coefficients[digit][i];
        }
    }

    // ...wybieramy cyfre o najwiekszej sumie jako nasza predykcje.
    int max_digit = 0;
    for (int digit = 1; digit < NUM_CLASSES; digit++) {
        if (sums[digit] > sums[max_digit]) {
            max_digit = digit;
        }
    }

    return max_digit;
}
```

MNIST

Mając dane, oraz parametry modelu, możemy wyznaczyć procent obrazków na których nasz model daje poprawną odpowiedź:

```
double test(vector <Datapoint> &test_data, vector <vector <double>> &coefficients) {
    int correct = 0;
    for (auto &datapoint : test_data) {
        // Przewidujemy klase obrazka testowego...
        int prediction = predict(datapoint.image, coefficients);

        // ...i sprawdzamy, czy sie udalo.
        correct += prediction == datapoint.label;
    }

    return (double) correct / test_data.size();
}

void test_and_print(vector <Datapoint> &test_data, vector <vector <double>> &coefficients) {
    auto correct_fraction = test(test_data, coefficients);
    cout << 100.0 * correct_fraction << "%" << endl;
}
```

MNIST

Niestety to jeszcze nie koniec – nie wiemy jakie są "dobre" wartości dla naszych parametrów...

Pozostaje napisać funkcję, która dla danych treningowych zwróci parametry które do nich "pasują" - czyli funkcję, która wytrenuje nasz model:

```
vector <vector <double>> train(vector <Datapoint> &train_data, int num_steps)
```

(o parametrze *num_steps* za chwilę - z grubsza będzie mówił jak dużo mamy czasu...)

MNIST

Na dobry początek zainicjalizujmy jakoś nasze parametry – powiedzmy na 0, i odpalmy funkcję testującą:

```
int image_length = train_data[0].image.size(); // = 784

// Inicjalizujemy wspolczynniki na 0.
vector <vector <double>> coefficients(NUM_CLASSES, vector <double> (image_length, 0.0));

cout << "Wynik na danych treningowych bez treningu: ";
test_and_print(train_data, coefficients);
```

Wszystkie parametry są 0, więc wszystkie sumy w funkcji *predict* wyjdą 0, i nasz model zawsze będzie zwracał tą samą klasę (pierwszą z brzegu, czyli 0). Jako że w danych każdej z cyfr jest mniej więcej po równo, to taki model dostanie wynik około 20%:

```
Wynik na danych treningowych bez treningu: 18.93%
```

MNIST

Teraz parę razy spróbujemy poprawić parametry. W jednym kroku każdy parametr zmienamy o losową wartość; jeśli tak zmodyfikowane działają lepiej, to zostawiamy:

```
double score = test(train_data, coefficients);

// Wykonujemy num_steps krokow...
for (int step = 0; step < num_steps; step++) {
    auto new_coefficients = coefficients;

    // ...w kazdym kroku zmieniamy losowo wspolczynniki...
    for (int digit = 0; digit < NUM_CLASSES; digit++) {
        for (int i = 0; i < image_length; i++) {
            new_coefficients[digit][i] += random_double();
        }
    }

    // ...liczymy wynik po zmianie, jeśli jest lepszy, to zapisujemy.
    double new_score = test(train_data, new_coefficients);
    if (new_score > score) {
        coefficients = new_coefficients; score = new_score;
        cout << "Krok " << step << ": " << new_score << endl;
    }
}
```

MNIST

No to odpalamy:

```
auto coefficients = train(train_data, 5000);
```

```
Wynik na danych treningowych bez treningu: 18.93%
Krok 3: 0.3034
Krok 4: 0.3688
Krok 5: 0.3804
Krok 6: 0.4186
Krok 7: 0.4278
Krok 14: 0.4286
Krok 15: 0.4409
Krok 16: 0.4802
Krok 22: 0.5015
Krok 27: 0.5155
Krok 28: 0.5654
Krok 48: 0.5823
```

Działa :-)

MNIST

...kilka minut później:

```
Krok 3739: 0.9339
Krok 3745: 0.934
Krok 3749: 0.9349
Krok 3774: 0.9351
Krok 3797: 0.9356
Krok 3813: 0.9362
Krok 3899: 0.9369
Krok 4091: 0.937
Krok 4115: 0.9377
Krok 4134: 0.9393
Krok 4511: 0.9394
Krok 4623: 0.9403
Krok 4689: 0.9405
```

Nasz model daje poprawną odpowiedź w 94% przypadków - całkiem nieźle.

MNIST

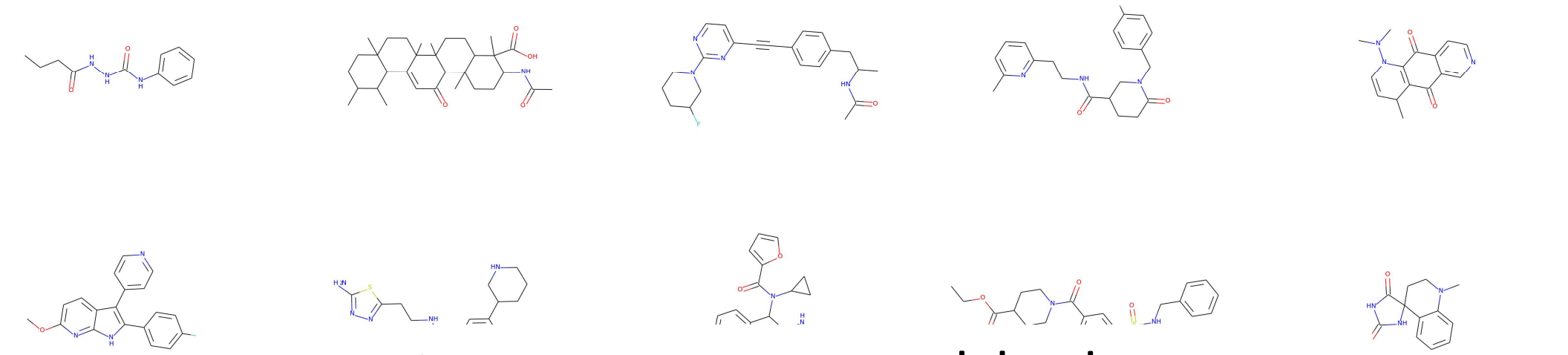
Typowo po wytrenowaniu modelu testuje się go na oddzielnych danych które nie były użyte do treningu, żeby sprawdzić, czy model nie "dopasował się nadmiernie" do danych treningowych. Innymi słowy, po prostu czy działa na "nowych" obrazkach.

Na stronie MNIST można pobrać oddzielny zbiór 10 000 obrazków, przeznaczonych właśnie do testowania.

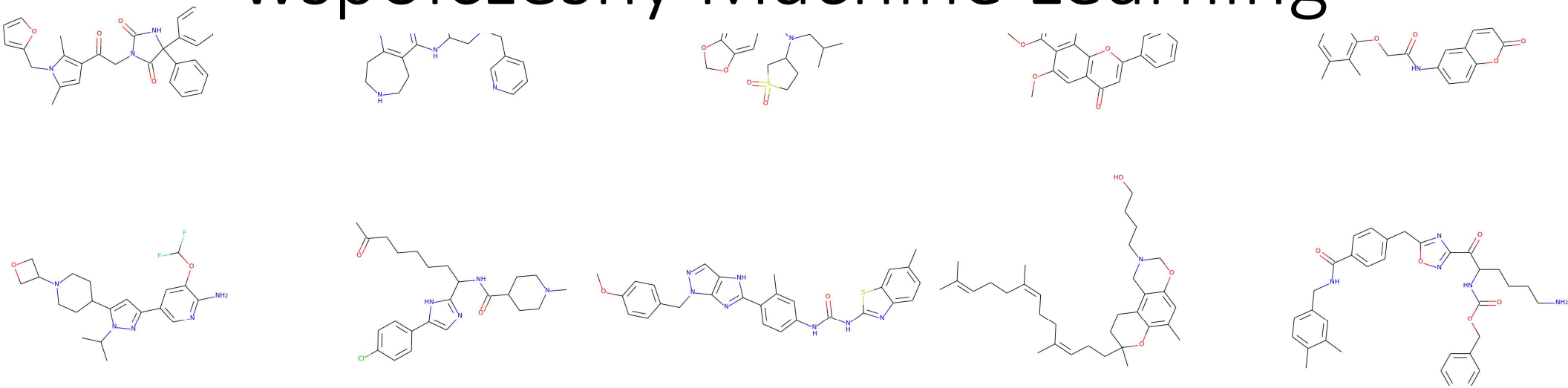
```
cout << "Wynik na danych testowych: ";
test_and_print(test_data, coefficients);
```

```
Wynik na danych testowych: 94.1428%
```

Ponownie ok. 94% - czyli wszystko w porządku.



Nasz prosty przykład vs współczesny Machine Learning



Nasz prosty przykład a ML w praktyce

Nasz prosty przykład jest *naprawdę nie tak daleko* od wszystkich imponujących zastosowań ML wspomnianych na tym wykładzie :-)

Wszystkie z zastosowań o których wspomniałem pasują do tego samego schematu co nasz prosty przykład na MNIST:

1. Definiujemy model o parametrach będących liczbami rzeczywistymi.
2. Inicjalizujemy parametry.
3. Wykonujemy wiele kroków, w których próbujemy trochę zmienić parametry, tak, żeby model był "lepszy".

Nasz prosty przykład a ML w praktyce: różnice

Nasza metoda zmieniania parametrów jest zbyt chaotyczna. Gdy model jest już całkiem dobry nie jest łatwo trafić na losowe zmiany, które go poprawią...

W praktyce używa się narzędzi opartych o *pochodną funkcji*, które potrafią wyliczyć *drobną zmianę parametrów która najbardziej poprawia wynik modelu*.

Zmiana którą wyliczamy to wciąż "lokalna, mała poprawka"; więc wciąż aplikuje się ją iteracyjnie (bardzo) wiele razy.

Nasz prosty przykład a ML w praktyce: różnice

My szukamy zmiany parametrów która poprawia model na całym zbiorze danych. W praktyce danych mamy zbyt dużo by mieć czas robić to w każdym kroku...

Typowo, bierze się mały losowy fragment danych (na przykład 64 obrazki), i szuka zmiany parametrów żeby poprawić wynik na tym fragmencie. Potem bierzemy inny losowy fragment, i tak dalej...*

*akurat to usprawnienie łatwo można zaaplikować w naszym kodzie MNIST - ćwiczenie dla czytelnika :-)

Nasz prosty przykład a ML w praktyce: różnice

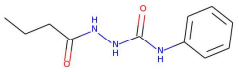
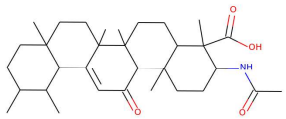
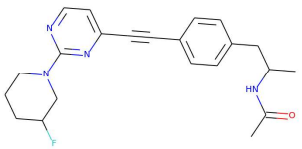
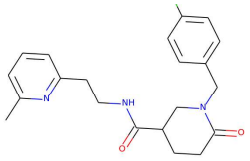
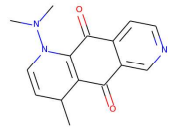
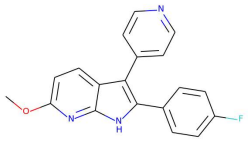
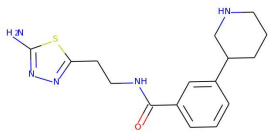
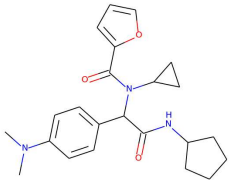
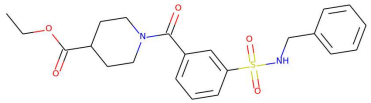
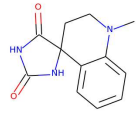
Nasz model jest bardzo prosty – na przykład, nie bierze on pod uwagę żadnych *korelacji* między pixelami. Jest też czuły na lekkie przesunięcia obrazka...

W praktyce używa się modeli bardziej skomplikowanych i dopasowanych do problemu.

Przykładowo, dla obrazków używa się modeli które mają wiele *warstw*: początkowe uwzględniające tylko interakcje między pixelami położonymi blisko siebie, i kolejne, coraz bardziej "wysokopoziomowe"…*

*W takich modelach (po wytrenowaniu) okazuje się, że parametry w "niskich" warstwach nauczyły się rozpoznawać krawędzie, w kolejnych kształty, a wreszcie koncepcje takie jak konkretne gatunki zwierząt... ale to już temat na osobny wykład.

Microsoft Research Cambridge

Dziękuję za uwagę!