Multilayer Perceptrons: Expressiveness, overfitting, regularization

Marc Lelarge   Kevin Scaman   Jill-Jênn Vie

Oct 21, 2022

Multilayer Perceptrons (MLP)

$$\begin{aligned} \boldsymbol{x}^{(0)} &= \boldsymbol{x} \\ \boldsymbol{x}^{(\ell+1)} &= \sigma(\boldsymbol{W}^{(\ell)}\boldsymbol{x}^{(\ell)} + \boldsymbol{b}^{(\ell)}) \quad \ell = 0, \ldots, L-2 \\ y = \boldsymbol{x}^{(L)} &= \boldsymbol{W}^{(L-1)}\boldsymbol{x}^{(L-1)} + \boldsymbol{b}^{(L-1)} \end{aligned}$$

Multilayer Perceptrons (MLP)

$$\begin{aligned}
\boldsymbol{x}^{(0)} &= \boldsymbol{x} \in \mathbf{R}^{d_0} \\
\boldsymbol{x}^{(\ell+1)} &= \sigma\big(\boldsymbol{W}^{(\ell)}\boldsymbol{x}^{(\ell)} + \boldsymbol{b}^{(\ell)}\big) \in \mathbf{R}^{d_\ell} \quad \ell = 0, \ldots, L-2 \\
y = \boldsymbol{x}^{(L)} &= \boldsymbol{W}^{(L-1)}\boldsymbol{x}^{(L-1)} + \boldsymbol{b}^{(L-1)} \in \mathbf{R}^{d_L}
\end{aligned}$$

The ℓth layer has d_ℓ neurons. Input layer $\ell = 0$, output layer $\ell = L$.
σ is the link function. Usually, $\sigma = \mathrm{ReLU} = \max(\mathbf{0}, \boldsymbol{x})$. #params?

Multilayer Perceptrons (MLP)

$$
\begin{aligned}
\boldsymbol{x}^{(0)} &= \boldsymbol{x} \in \mathbf{R}^{d_0} \\
\boldsymbol{x}^{(\ell+1)} &= \sigma(\boldsymbol{W}^{(\ell)}\boldsymbol{x}^{(\ell)} + \boldsymbol{b}^{(\ell)}) \in \mathbf{R}^{d_\ell} \quad \ell = 0, \ldots, L-2 \\
y = \boldsymbol{x}^{(L)} &= \boldsymbol{W}^{(L-1)}\boldsymbol{x}^{(L-1)} + \boldsymbol{b}^{(L-1)} \in \mathbf{R}^{d_L}
\end{aligned}
$$

The ℓth layer has d_ℓ neurons. Input layer $\ell = 0$, output layer $\ell = L$.
σ is the link function. Usually, $\sigma = \mathrm{ReLU} = \max(\mathbf{0}, \boldsymbol{x})$. #params?

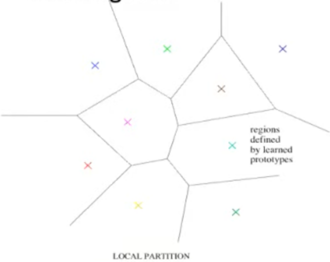
```
from torch import nn

mlp = nn.Sequential(
    nn.Linear(d_0, d_1),
    nn.ReLU(),
    nn.Linear(d_1, d_2),
    ...
    nn.Linear(d_{L-1}, d_L)
)
```

Example: logistic regression is a 1-layer perceptron

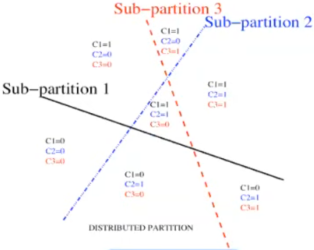
$y = \sigma(\boldsymbol{w}^T \boldsymbol{x} + b)$ where $\sigma = \text{sigmoid} = 1/(1 + \exp(-x))$

Distributed Representations: The Power of Compositionality – Part 1

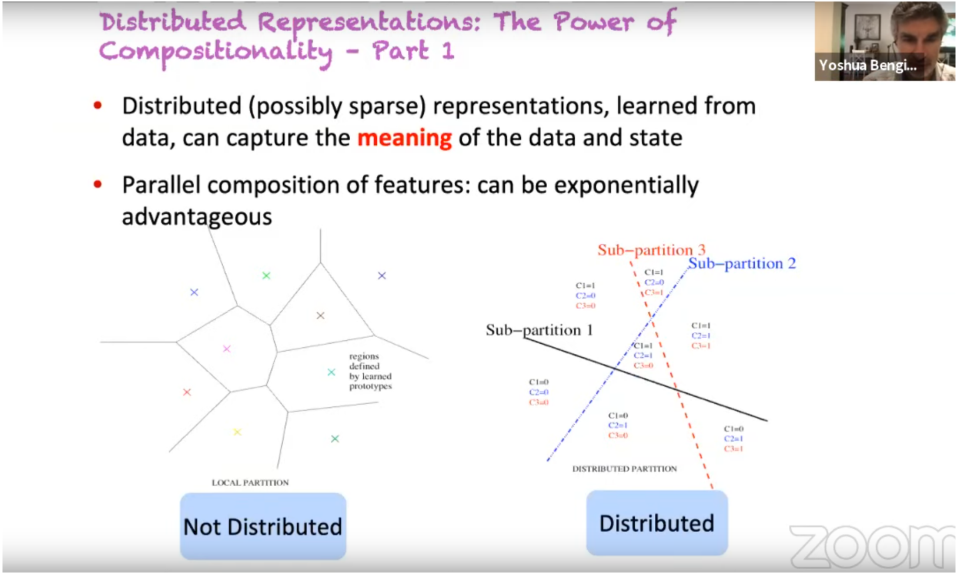
- Distributed (possibly sparse) representations, learned from data, can capture the **meaning** of the data and state
- Parallel composition of features: can be exponentially advantageous



Not Distributed



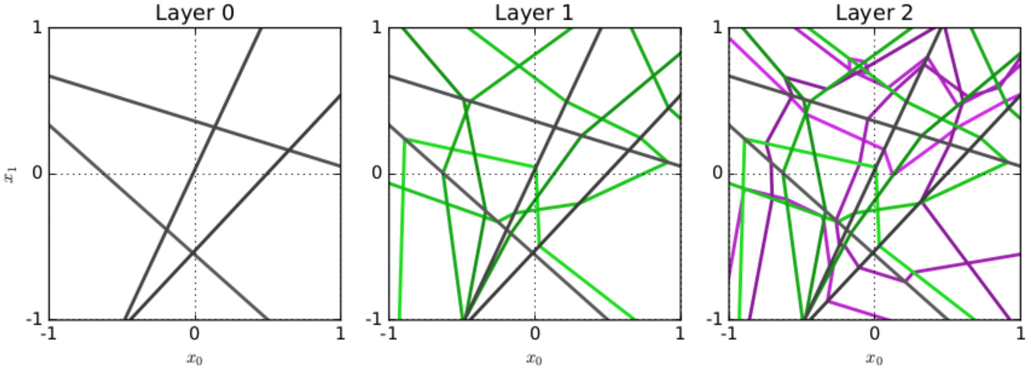
Distributed



A ReLU-based MLP with inputs in $\mathbf{R}^n$, L layers of width $k \geq n$, can compute functions that have $\Omega((k/n)^{n(L-1)} n^k)$ linear regions.

Expressiveness

The number of activation patterns ($\sim$ regions) of a ReLU-based MLP with L layers of width k, inputs in $\mathbf{R}^n$ is upper bounded (tightly) by $O(k^{nL})$ as $L \to \infty$.



Maithra Raghu et al. "On the expressive power of deep neural networks". In: *International Conference on Machine Learning*. PMLR. 2017, pp. 2847–2854

Universal approximation theorems

Fixed depth 2 arbitrary width k (Pinkus, 1999) (Cybenko, 1989)

Let $\sigma \in \mathcal{C}(\mathbf{R})$ a continuous function from $\mathbf{R}$ to $\mathbf{R}$.
Then: σ is not polynomial $\iff$
For all $\varepsilon > 0$, $n, m \in \mathbf{N}$, compact $K \subseteq \mathbf{R}^n$, function $f \in \mathcal{C}(K, \mathbf{R}^m)$,
there exist latent dimension k and weights $\boldsymbol{W}, \boldsymbol{b}, \boldsymbol{C}$ such that

$$\sup_{\boldsymbol{x} \in K} ||f(\boldsymbol{x}) - \mathrm{MLP}(\boldsymbol{x})|| < \varepsilon \quad \mathrm{MLP}(\boldsymbol{x}) = \boldsymbol{C}\sigma(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}).$$

Universal approximation theorems

Fixed depth 2 arbitrary width k (Pinkus, 1999) (Cybenko, 1989)

Let $\sigma \in \mathcal{C}(\mathbf{R})$ a continuous function from $\mathbf{R}$ to $\mathbf{R}$.
Then: σ is not polynomial $\iff$
For all $\varepsilon > 0$, $n, m \in \mathbf{N}$, compact $K \subseteq \mathbf{R}^n$, function $f \in \mathcal{C}(K, \mathbf{R}^m)$,
there exist latent dimension k and weights $\boldsymbol{W}, \boldsymbol{b}, \boldsymbol{C}$ such that

$$\sup_{\boldsymbol{x} \in K} ||f(\boldsymbol{x}) - \mathrm{MLP}(\boldsymbol{x})|| < \varepsilon \quad \mathrm{MLP}(\boldsymbol{x}) = \boldsymbol{C}\sigma(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}).$$

In other words, 2-layer MLPs are dense in $\mathcal{C}(K, \mathbf{R}^m)$.
They are universal approximators of continuous functions.

Universal approximation theorems

Arbitrary depth, minimal width (Park, 2020)

For any function $f \in L^p(\mathbf{R}^n, \mathbf{R}^m)$ and any $\varepsilon > 0$
there exists a MLP with ReLU of width $\max(n+1, m)$ such that

$$||f - \mathrm{MLP}||_p = \left(\int_{\mathbf{R}^n} ||f(x) - \mathrm{MLP}(x)||^p dx \right)^{1/p} < \varepsilon.$$

Universal approximation theorems

Arbitrary depth, minimal width (Park, 2020)

For any function $f \in L^p(\mathbf{R}^n, \mathbf{R}^m)$ and any $\varepsilon > 0$
there exists a MLP with ReLU of width $\max(n+1, m)$ such that

$$||f - \mathrm{MLP}||_p = \left(\int_{\mathbf{R}^n} ||f(x) - \mathrm{MLP}(x)||^p dx \right)^{1/p} < \varepsilon.$$

Moreover:

Arbitrary depth, constrained width (Kidger and Lyons, 2020)

Let $\mathcal{N}$ be the space of MLP : $\mathbf{R}^n \to \mathbf{R}^m$ with any layers having $n + m + 2$ neurons.
Then: $\mathcal{N}$ is dense in $\mathcal{C}(K, \mathbf{R}^m)$ where compact $K \subseteq \mathbf{R}^n$.

Some other theoretical results

Infinite-depth limit

- Untrained MLP with random weights (Karakida, Akaho & Amari, 2018)

The Fisher information matrix i.e. $\frac{\partial^2 \mathcal{L}}{\partial \theta^2}$ has eigenvalues having mean $O(1/M)$, variance $O(1)$ and max $O(M)$.

- Neural Tangent Kernels (Jacot, Gabriel & Hongler, 2018)

Turing-completeness

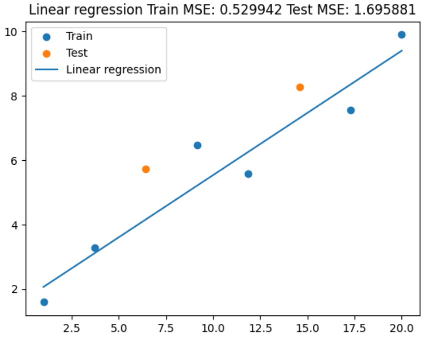
- RNNs are Turing-complete (Siegelmann & Sontag, 1995)
- LSTMs can perform unbounded counting while GRUs cannot (Weiss, Goldberg & Yahav, 2018)
- Neural Turing Machines with external memory (Graves, Wayne & Danihelka, 2014)

Training MLP with random labels?! (Maennel et al., 2020)

Overfitting

Fitting

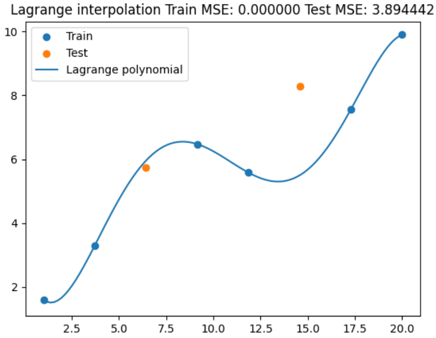
Polynomial degree 1 $Y = wX + b$



Linear regression
`scipy.stats.linreg`

Overfitting

Polynomial degree 6 $Y = \sum_{k=0}^{6} w_k X^k$



Lagrange interpolation
`scipy.interpolation.lagrange`

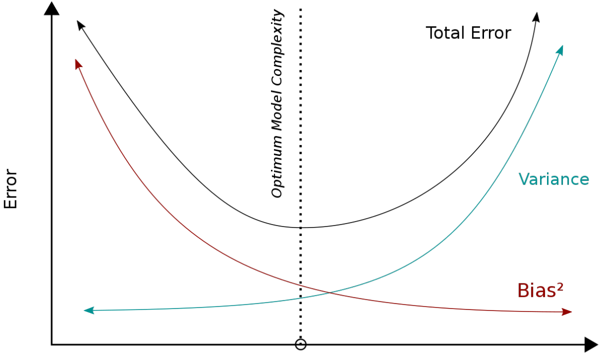
Bias-variance decomposition

Samples $x_i, y_i \in \mathbf{R}$. We train a model $g_{\boldsymbol{\theta}}$.

Let us assume that $y_i = f(x_i) + \varepsilon_i$ where $\varepsilon_i \in \mathcal{N}(0, \sigma^2)$.

Then: the generalization error for the squared loss verifies

$$\begin{aligned}\mathbb{E}[(Y - g_{\boldsymbol{\theta}}(X))^2] &= \mathrm{Bias}(g_{\boldsymbol{\theta}})^2 + \mathrm{Var}(g_{\boldsymbol{\theta}}) + \sigma^2 \\ &= \mathbb{E}[g_{\boldsymbol{\theta}} - f]^2 + \mathbb{E}[(g_{\boldsymbol{\theta}} - \mathbb{E}g_{\boldsymbol{\theta}})^2] + \sigma^2.\end{aligned}$$



Bias-variance decomposition

Samples $x_i, y_i \in \mathbf{R}$. We train a model $g_{\boldsymbol{\theta}}$.

Let us assume that $y_i = f(x_i) + \varepsilon_i$ where $\varepsilon_i \in \mathcal{N}(0, \sigma^2)$.

Then: the generalization error for the squared loss verifies

$$\begin{aligned}\mathbb{E}[(Y - g_{\boldsymbol{\theta}}(X))^2] &= \mathrm{Bias}(g_{\boldsymbol{\theta}})^2 + \mathrm{Var}(g_{\boldsymbol{\theta}}) + \sigma^2 \\ &= \mathbb{E}[g_{\boldsymbol{\theta}} - f]^2 + \mathbb{E}[(g_{\boldsymbol{\theta}} - \mathbb{E}g_{\boldsymbol{\theta}})^2] + \sigma^2.\end{aligned}$$

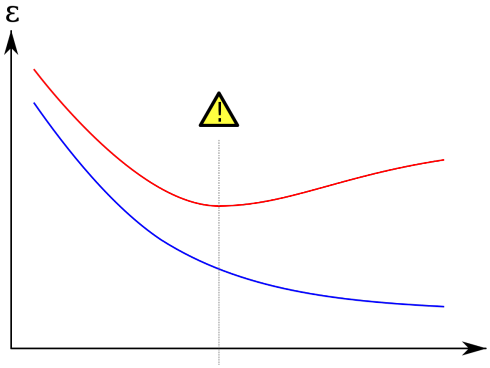
Proof.

$$\mathbb{E}f = f \quad \mathbb{E}Y = f \quad \mathrm{Var}(Y) = \sigma^2$$

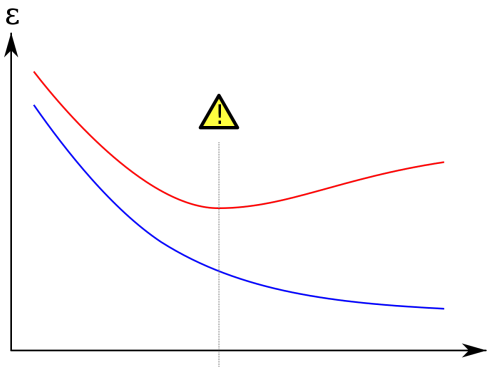
As ε and $g_{\boldsymbol{\theta}}$ are independent:

$$\begin{aligned}\mathbb{E}\left[(Y - g_{\boldsymbol{\theta}})^2\right] &= \mathbb{E}\left[Y^2 + {g_{\boldsymbol{\theta}}}^2 - 2Y g_{\boldsymbol{\theta}}\right] \\ &= \mathbb{E}\left[Y^2\right] + \mathbb{E}\left[{g_{\boldsymbol{\theta}}}^2\right] - \mathbb{E}[2Y g_{\boldsymbol{\theta}}] \\ &= \mathrm{Var}(Y) + \mathbb{E}[Y]^2 + \mathrm{Var}(g_{\boldsymbol{\theta}}) + \mathbb{E}[g_{\boldsymbol{\theta}}]^2 - 2f\mathbb{E}[g_{\boldsymbol{\theta}}] \\ &= \mathrm{Var}(Y) + \mathrm{Var}(g_{\boldsymbol{\theta}}) + (f - \mathbb{E}[g_{\boldsymbol{\theta}}])^2 \\ &= \mathrm{Var}(Y) + \mathrm{Var}(g_{\boldsymbol{\theta}}) + \mathbb{E}[f - g_{\boldsymbol{\theta}}]^2 \\ &= \sigma^2 + \mathrm{Var}(g_{\boldsymbol{\theta}}) + \mathrm{Bias}(g_{\boldsymbol{\theta}})^2.\end{aligned}$$

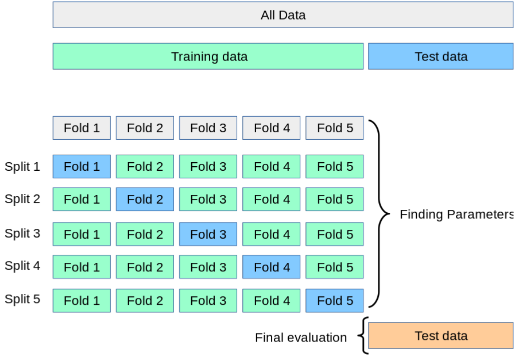
How to detect overfitting?



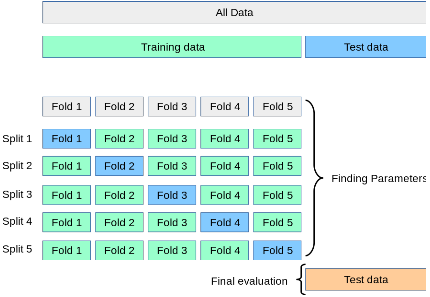
How to detect overfitting?



By keeping a validation set

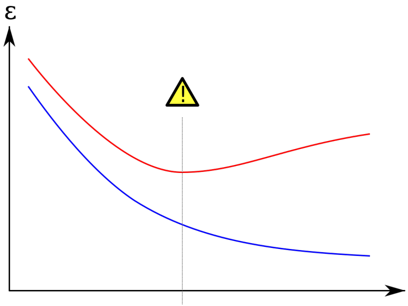


Hyperparameter selection by cross-validation




```
trainval, test = split data into 80:20
train, valid = split trainval into 80:20

for each hyperparameter λ:
    minimize error on train using λ
    valid_score_λ ← evaluate metric on valid
λ* ← λ achieving best valid_score_λ
minimize error on trainval using λ* (= refit)
```

Early stopping




```
def training_loop(train, valid):
    for each epoch:
        for x, y in train:
            do one step of gradient
            valid_score ← evaluate metric on valid
            if valid_score is worse than before:
                return
```

Example

Let us consider logistic regression i.e. 1-layer MLP:

$f(\boldsymbol{x}_i) = \sigma(\boldsymbol{W}\boldsymbol{x}_i + b)$

Logistic loss: $\mathcal{L} = \sum_i -(1 - y_i)\log(1 - f(\boldsymbol{x}_i)) - y_i \log f(\boldsymbol{x}_i)$

If all samples have same target $y_i = 1$ (or if there's only 1 sample), what will happen?

Example

Let us consider logistic regression i.e. 1-layer MLP:

$f(\boldsymbol{x}_i) = \sigma(\boldsymbol{W}\boldsymbol{x}_i + b)$

Logistic loss: $\mathcal{L} = \sum_i -(1 - y_i)\log(1 - f(\boldsymbol{x}_i)) - y_i \log f(\boldsymbol{x}_i)$

If all samples have same target $y_i = 1$ (or if there's only 1 sample), what will happen?

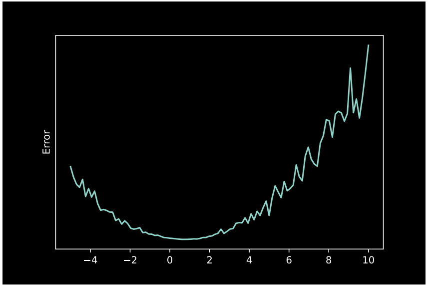
MLP believes everything is a cat.

- Minimize $-\log \sigma(\boldsymbol{W}\boldsymbol{x}_1 + b)$
- $\sigma(\boldsymbol{W}\boldsymbol{x}_1 + b) \to 1$
- $\boldsymbol{W}\boldsymbol{x}_1 + b \to +\infty$
- Parameters $|\boldsymbol{W}|$ and b diverge to $+\infty$

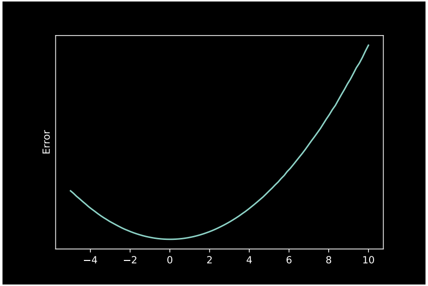
Add penalty to loss $||\boldsymbol{W}||_2^2 + ||b||_2^2$ (= assuming a Gaussian prior centered in $\mathbf{0}$), called L_2 regularization

Regularize to generalize

Minimizing loss:
May fall in local minima or diverge to ∞



Minimizing loss $+$ regularization:
Easier to optimize



We will see an example next week.

Take home message

Expressiveness

- 2-layer MLPs are universal approximators of continuous functions
- Try to overfit a single batch; otherwise your model cannot express the data.

Bias-variance trade-off

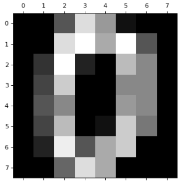
There is incompressible error due to inherent noise

Overfitting

- Don't look at test data it's forbidden
- Implement early stopping
- And L_2 regularization

Today's practical: datasets

Digits



$1797 \times 8 \times 8$ images representing numbers between 0 and 9.

Red Wine Quality[1] (Cortez et al., 2009)

1599 wines $\times$ 11 features[2], have to predict quality which is an integer between 0 and 10 (in practice between 3 and 8).

Also: Faces, Cats and dogs

[1] https://kaggle.com/datasets/uciml/red-wine-quality-cortez-et-al-2009

[2] fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density, pH, sulphates, alcohol

[1] Guido F Montufar et al. “On the number of linear regions of deep neural networks”. In: *Advances in Neural Information Processing Systems*. Vol. 27. 2014, pp. 2924–2932.

[2] Maithra Raghu et al. “On the expressive power of deep neural networks”. In: *International Conference on Machine Learning*. PMLR. 2017, pp. 2847–2854.