

Machine Learning

Textbook: Machine Learning Refined, by Watt, Borhani, and Katsaggelos, 2nd edition:
MLRefined.com

Previous Lecture

- ♦ A kernel computes an inner product $K(\mathbf{x}, \mathbf{z}) = \phi(\mathbf{x})^T \phi(\mathbf{z})$ without requiring the transformed coordinates explicitly.
- ♦ The kernel / Gram matrix contains all pairwise training-set inner products in feature space and must be symmetric positive semidefinite for a valid kernel.
- ♦ Polynomial kernels implicitly generate polynomial and interaction features; the RBF kernel corresponds to an infinite-dimensional feature representation.
- ♦ Kernel ridge regression solves $(G + P\lambda I)\alpha = \mathbf{y}$ and predicts through weighted kernel similarities to the training observations.
- ♦ Kernel SVMs remain linear separators in feature space while producing nonlinear boundaries in the original input space.
- ♦ Kernel hyperparameters and preprocessing must be selected with leakage-safe cross-validation, and kernel computation becomes expensive when the number of observations is very large.



§13.1 thru 13.3 - Neural-Network Architecture

Kernel methods used

- a nonlinear feature map
- chosen in advance.

Now:

- make the feature map itself trainable.

A neural network builds

- a sequence of parameterized transformations.

The hidden representation is learned

- together with the final predictor.

Today:

- build the architecture
- and compute its forward pass.

Next time:

- differentiate it.

From Fixed Features to Learned Features

Earlier:

$$\begin{aligned} f(\mathbf{x}) \\ = \mathbf{w}^T \phi(\mathbf{x}), \end{aligned}$$

where
 ϕ
was fixed before training.

Neural-network idea:

$$f(\mathbf{x}) = \mathbf{v}^T \phi_{\theta}(\mathbf{x}) + c,$$

where
 θ
contains parameters
inside the feature map itself.

Thus training learns:

the final predictor
and

the representation used by that predictor.

A Single Computational Unit

Start with input

$$\mathbf{x} \in \mathbb{R}^N.$$

Form an affine combination:

$$a = \mathbf{w}^T \mathbf{x} + b.$$

Then apply a nonlinear function

$$\sigma:$$
$$h = \sigma(a).$$

Terminology:

$\mathbf{w}$:
weights;

b :
bias;

a :
pre-activation;

h :
activation / hidden-unit output.

Why the Bias Matters

Without a bias:

$$a = \mathbf{w}^T \mathbf{x}.$$

The affine boundary

$$a = 0$$

must pass through the origin.

With a bias:

$$a = \mathbf{w}^T \mathbf{x} + b.$$

The boundary becomes

$$\mathbf{w}^T \mathbf{x} + b = 0.$$

Thus the bias allows the unit’s response
to shift through input space.

This is the same role
played by the intercept
in linear regression and classification.

Activation Functions

The activation function

$$\sigma(a)$$

creates nonlinearity.

Common choices:

ReLU

$$\sigma(a) = \max(0, a);$$

tanh

$$\sigma(a) = \tanh(a);$$

sigmoid

$$\sigma(a) = \frac{1}{1+e^{-a}}.$$

Different activations

create different nonlinear building blocks.

The ReLU Function

ReLU:

$$\text{ReLU}(a) = \max(0, a).$$

Piecewise:

$$\text{ReLU}(a)$$

$$= \begin{cases} 0, & a \leq 0, \\ a, & a > 0. \end{cases}$$

Thus a ReLU unit is:

inactive on one side
of an affine boundary,

and linear on the other.

Combinations of many ReLU units
can construct flexible piecewise-linear functions.

Two ReLUs Can Produce $|x|$

Recall:

$$\begin{aligned}\text{ReLU}(x) \\ &= \max(0, x).\end{aligned}$$

Also:

$$\begin{aligned}\text{ReLU}(-x) \\ &= \max(0, -x).\end{aligned}$$

Therefore:

$$|x| = \text{ReLU}(x) + \text{ReLU}(-x).$$

For

$$\begin{aligned}x \geq 0: \\ \text{result is} \\ x + 0 = x.\end{aligned}$$

For

$$\begin{aligned}x < 0: \\ \text{result is} \\ 0 - x = -x.\end{aligned}$$

Two hidden units already create
a nonlinear function of the original input.

A Shifted ReLU Creates a Hinge

Let

$$h(x) = \text{ReLU}(x - c).$$

Then:

$$h(x) = 0$$

for

$$x \leq c,$$

and

$$h(x) = x - c$$

for

$$x > c.$$

The slope changes at

$$x = c.$$

Thus models of the form

$$f(x) = w_0 + w_1x + \sum_{j=1}^M a_j \text{ReLU}(x - c_j)$$

are continuous piecewise-linear functions
whose slopes can change

at the hinge locations

c_j .

A Hidden Layer

Instead of one unit,

use

m

units at once.

Let

$$W \in \mathbb{R}^{m \times N}$$

and

$$\mathbf{b} \in \mathbb{R}^m.$$

Pre-activation vector:

$$\mathbf{a} = W\mathbf{x} + \mathbf{b}.$$

Apply the activation componentwise:

$$\mathbf{h} = \sigma(\mathbf{a}).$$

Then

$$\mathbf{h} \in \mathbb{R}^m$$

is a learned feature vector.

The hidden-layer width is

m .

What Does “Fully Connected” Mean?

Write

$$W = [w_{ij}].$$

Hidden unit

i

receives:

$$\begin{aligned} a_i \\ = \sum_{j=1}^N w_{ij}x_j + b_i. \end{aligned}$$

Every hidden unit

can use every input coordinate.

Therefore the layer is

fully connected

or

dense.

A fully connected network

stacks several dense layers.

One Hidden Layer for Regression

Hidden features:

$$\mathbf{h} = \sigma(W\mathbf{x} + \mathbf{b}).$$

Regression output:

$$\hat{y} = c + \mathbf{v}^T \mathbf{h}.$$

Combine:

$$\hat{y} = c + \mathbf{v}^T \sigma(W\mathbf{x} + \mathbf{b}).$$

This model is nonlinear
in the input
 $\mathbf{x}$.

It is also generally nonlinear
in the joint parameter collection
 $(W, \mathbf{b}, \mathbf{v}, c)$.

This is different from
fixed nonlinear features.

A Numerical Forward Pass

Take

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

and

$$W = \begin{bmatrix} 1 & -1 \\ 1/2 & 1 \end{bmatrix},$$

with

$$\mathbf{b} = \begin{bmatrix} 0 \\ -1/2 \end{bmatrix}.$$

Then:

$$\begin{aligned} \mathbf{a} &= W\mathbf{x} + \mathbf{b} \\ &= \begin{bmatrix} 1 \\ 3/2 \end{bmatrix}. \end{aligned}$$

Using ReLU:

$$\mathbf{h} = \begin{bmatrix} 1 \\ 3/2 \end{bmatrix}.$$

Both units are active.

Finish the Regression Forward Pass

Let

$$\mathbf{v} = \begin{bmatrix} 2 \\ -1 \end{bmatrix}$$

and

$$c = 1/4.$$

Then:

$$\begin{aligned} \hat{y} &= \frac{1}{4} + \begin{bmatrix} 2 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 3/2 \end{bmatrix} \\ &= \frac{1}{4} + 2 - \frac{3}{2} \\ &= \frac{3}{4}. \end{aligned}$$

The sequence

$$\mathbf{x} \rightarrow \mathbf{a} \rightarrow \mathbf{h} \rightarrow \hat{y}$$

is the

forward pass.

Why We Need a Nonlinear Activation

Suppose we remove

$$\sigma.$$

Then one hidden layer gives

$$\begin{aligned} \hat{y} &= c + \mathbf{v}^T(W\mathbf{x} + \mathbf{b}). \end{aligned}$$

Rearrange:

$$\begin{aligned} \hat{y} &= (c + \mathbf{v}^T\mathbf{b}) + (\mathbf{v}^TW)\mathbf{x}. \end{aligned}$$

This is just

$$\tilde{c} + \tilde{\mathbf{w}}^T\mathbf{x}.$$

A linear model.

Without nonlinear activations,

the hidden layer adds no new expressive power.

Several Affine Layers Still Collapse

Suppose:

$$\mathbf{h}_1 = W_1\mathbf{x} + \mathbf{b}_1,$$

and

$$\mathbf{h}_2 = W_2\mathbf{h}_1 + \mathbf{b}_2.$$

Then:

$$\begin{aligned}\mathbf{h}_2 \\ &= W_2 W_1 \mathbf{x} + W_2 \mathbf{b}_1 + \mathbf{b}_2.\end{aligned}$$

Define:

$$\widetilde{W} = W_2 W_1,$$

and

$$\widetilde{\mathbf{b}} = W_2 \mathbf{b}_1 + \mathbf{b}_2.$$

Therefore:

$$\mathbf{h}_2 = \widetilde{W} \mathbf{x} + \widetilde{\mathbf{b}}.$$

Still one affine transformation.

Nonlinearity between layers is essential.

Two Hidden Layers

First hidden layer:

$$\mathbf{h}_1 = \sigma_1(W_1 \mathbf{x} + \mathbf{b}_1).$$

Second hidden layer:

$$\mathbf{h}_2 = \sigma_2(W_2 \mathbf{h}_1 + \mathbf{b}_2).$$

Regression output:

$$\hat{y} = \mathbf{v}^T \mathbf{h}_2 + c.$$

Thus:

$$\hat{y}$$

is a composition

of affine maps

and nonlinear activations.

This repeated composition

is the basic fully connected neural-network architecture.

General L -Layer Notation

Let

$$\mathbf{h}_0 = \mathbf{x}.$$

For hidden layer

$$\ell = 1, \dots, L:$$

$$\mathbf{a}_\ell = W_\ell \mathbf{h}_{\ell-1} + \mathbf{b}_\ell,$$

and

$$\mathbf{h}_\ell = \sigma_\ell(\mathbf{a}_\ell).$$

Then an output layer

maps

$$\mathbf{h}_L$$

to the final score or prediction.

This notation will be essential
when we differentiate the network.

Width and Depth

Width
of layer

ℓ :
number of units
 n_ℓ .

Depth:
number of successive trainable layers
or hidden transformations,
depending on convention.

We will state our convention explicitly:

hidden depth

L
means
 L
hidden layers.

Architecture example:
 $2 \rightarrow 3 \rightarrow 2 \rightarrow 1$.

Two inputs,
hidden widths 3 and 2,
one output.

Layer Dimensions

Suppose:
 $n_{\ell-1}$
inputs enter layer
 ℓ ,

and
 n_ℓ
units leave it.

Then:
 $W_\ell \in \mathbb{R}^{n_\ell \times n_{\ell-1}}$,

and
 $\mathbf{b}_\ell \in \mathbb{R}^{n_\ell}$.

For a single observation:

$$\mathbf{h}_{\ell-1} \in \mathbb{R}^{n_{\ell-1}},$$

so

$$W_{\ell}\mathbf{h}_{\ell-1} + \mathbf{b}_{\ell} \in \mathbb{R}^{n_{\ell}}.$$

Dimension checks prevent many implementation errors.

Parameter Count for One Dense Layer

A dense layer

$$n_{\ell-1} \rightarrow n_{\ell}$$

contains:

weights:

$$n_{\ell}n_{\ell-1};$$

biases:

$$n_{\ell}.$$

Total:

$$n_{\ell}n_{\ell-1} + n_{\ell} = n_{\ell}(n_{\ell-1} + 1).$$

For a network:

add this quantity
over all trainable layers.

Parameter Count Example

Architecture:

$$2 \rightarrow 3 \rightarrow 2 \rightarrow 1.$$

First layer:

$$\begin{aligned} &3(2) + 3 \\ &= 9. \end{aligned}$$

Second layer:

$$\begin{aligned} &2(3) + 2 \\ &= 8. \end{aligned}$$

Output layer:

$$\begin{aligned} &1(2) + 1 \\ &= 3. \end{aligned}$$

Total:

$$\begin{aligned} &9 + 8 + 3 \\ &= 20 \end{aligned}$$

trainable parameters.

Architecture determines

the dimension of the optimization problem.

Regression Output Layer

For scalar regression:

last hidden representation:

$\mathbf{h}_L$.

Use a linear output:

$\hat{y} = \mathbf{v}^T \mathbf{h}_L + c$.

No sigmoid or softmax is required
if the response is unrestricted real-valued.

Then squared error may be used:

$g = \frac{1}{P} \sum_p (\hat{y}_p - y_p)^2$.

The network architecture determines
the prediction function.

The loss determines
how its parameters will be trained.

Binary Classification Output

For binary classification:

last hidden representation:

$\mathbf{h}_L$.

Compute score:

$f = \mathbf{v}^T \mathbf{h}_L + c$.

Probability:

$p = \sigma(f) = \frac{1}{1+e^{-f}}$.

Then use binary cross-entropy.

Thus:
hidden layers learn nonlinear features,

while the output layer
uses the same logistic machinery
from earlier lessons.

Multiclass Classification Output

For

K

classes:

output score vector:

$\mathbf{f} = \mathcal{W}_{\text{out}} \mathbf{h}_L + \mathbf{b}_{\text{out}} \in \mathbb{R}^K$.

Apply softmax:

$$p_k = \frac{e^{f_k}}{\sum_{j=1}^K e^{f_j}}.$$

Use categorical cross-entropy.

Again:
neural networks do not replace
our previous output models.

They learn the features
fed into those output models.

A Useful View: Feature Learner Plus Linear Head

A fully connected network can be written as:

$$\mathbf{x} \rightarrow \phi_{\theta}(\mathbf{x}) \rightarrow \text{linear output head}.$$

Hidden layers:

$$\phi_{\theta}(\mathbf{x}) = \mathbf{h}_L.$$

Output head:

regression:

$$\mathbf{v}^T \mathbf{h}_L + c;$$

classification:

linear scores
followed by sigmoid or softmax.

This connects neural networks
directly to our earlier feature-engineering framework.

Fixed Features Versus Neural Features

	Fixed Feature Model	Neural Network
Feature map	chosen in advance	trainable
Feature parameters	fixed	learned
Final predictor	linear in features	linear head common
Joint optimization	often convex	generally nonconvex
Representation	prescribed	data-adaptive

The new difficulty is not
merely more parameters.

The features themselves
now depend on trainable parameters.

This changes the optimization problem fundamentally.

Batch Matrix Form

Store

$$P$$

observations in rows:

$$X \in \mathbb{R}^{P \times N}.$$

For a hidden layer with

$$m$$

units:

$$W \in \mathbb{R}^{m \times N},$$

$$\mathbf{b} \in \mathbb{R}^m.$$

Then:

$$A = XW^T + \mathbf{1}\mathbf{b}^T.$$

Dimensions:

$$A \in \mathbb{R}^{P \times m}.$$

Apply activation entrywise:

$$H = \sigma(A).$$

Every row of

$$H$$

is one observation's hidden representation.

Batch Form for Two Hidden Layers

First layer:

$$A_1 = XW_1^T + \mathbf{1}\mathbf{b}_1^T,$$

$$H_1 = \sigma_1(A_1).$$

Second layer:

$$A_2 = H_1W_2^T + \mathbf{1}\mathbf{b}_2^T,$$

$$H_2 = \sigma_2(A_2).$$

Regression output:

$$\hat{\mathbf{y}} = H_2\mathbf{v} + c\mathbf{1}.$$

This is the same forward computation
performed simultaneously
for a batch of observations.

What Does Depth Add?

One hidden layer:

learns nonlinear combinations

of the original input.

A second hidden layer:
learns nonlinear combinations
of the first learned features.

Deeper layers therefore build
compositions of learned representations.

Symbolically:
$$\mathbf{h}_2 = \sigma_2(W_2\sigma_1(W_1\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2).$$

Composition is what distinguishes
deep architectures
from one large fixed basis expansion.

Universal Approximation, What It Does and Does Not Say

Under suitable conditions:
a sufficiently wide neural network
with a nonlinear activation
can approximate broad classes
of continuous functions
on compact domains.

This is a
representation result.

It does not say:

gradient descent will find the desired parameters;

finite data identify the correct function;

the model will generalize well;

a particular finite architecture is optimal.

Expressive power
and trainability
are different questions.

Architecture Choices Are Hyperparameters

Choices include:

number of hidden layers;

width of each layer;

activation functions;

output architecture.

These determine the model family.

Thus they are
hyperparameters.

Weights and biases are
parameters.

Architecture choices should ultimately be judged
using validation procedures
rather than training loss alone.

Parameter Symmetry

Suppose two hidden units
use the same activation.

If we swap:

their incoming weights and biases,

and simultaneously swap
their outgoing weights,

the network function does not change.

Therefore:
different parameter vectors
can represent the same prediction function.

Neural-network parameterization
is not unique.

This contributes to
the complicated geometry
of the training objective.

Neural Networks Are Generally Nonconvex in Their Parameters

For fixed nonlinear features:
regression was linear in
 $\mathbf{w}$.

For a neural network:
hidden features depend on
 $W_1, \mathbf{b}_1, \dots$.

Example:

$\hat{y}$

$$= \mathbf{v}^T \sigma(W\mathbf{x} + \mathbf{b}) + c.$$

Parameters occur:
inside a nonlinear function
and
outside it.

The loss is therefore generally
nonconvex.

The optimization methods from earlier lessons
still matter,

but global guarantees become much harder.

Forward Pass in Python

Define ReLU:

```
import numpy as np

def relu(a):
    return np.maximum(0, a)
```

One observation:

```
x = np.array([2., 1.])

W = np.array([[1., -1.],
              [0.5, 1.]])

b = np.array([0., -0.5])

v = np.array([2., -1.])

c = 0.25
```

Now compute the layer.

Complete the Forward Pass in Python

```
a = W @ x + b

h = relu(a)

y_hat = v @ h + c

print(a)
print(h)
print(y_hat)
```

Output:

```
[1.  1.5]

[1.  1.5]

0.75
```

This matches the hand calculation:

$$\hat{y} = 0.75.$$

No training has occurred yet.

We are only evaluating
the network function
for a given parameter choice.

Batch Forward Pass in Python

Let observations be rows:

```
X = np.array([[ 2.,  1.],  
              [ 1.,  3.],  
              [-1.,  2.]])
```

Compute:

```
A = X @ W.T + b  
  
H = relu(A)  
  
y_hat = H @ v + c
```

New Python idea:

adding
b
to the
 $P \times m$
matrix
 $X @ W.T$
uses NumPy broadcasting.

The length- m bias vector
is added to every row.

Batch Example Output

For the three observations:

pre-activations:

$$A = \begin{bmatrix} 1 & 1.5 \\ -2 & 3 \\ -3 & 1 \end{bmatrix}.$$

After ReLU:

$$H = \begin{bmatrix} 1 & 1.5 \\ 0 & 3 \\ 0 & 1 \end{bmatrix}.$$

Regression predictions:

$$\hat{\mathbf{y}} = \begin{bmatrix} 0.75 \\ -2.75 \\ -0.75 \end{bmatrix}.$$

Each row follows
the same network parameters.

Two Hidden Layers in Python

```
def forward_two_layer(
    X,
    W1, b1,
    W2, b2,
    v, c
):
    A1 = X @ W1.T + b1
    H1 = relu(A1)

    A2 = H1 @ W2.T + b2
    H2 = relu(A2)

    y_hat = H2 @ v + c

    return y_hat
```

This function performs only
forward propagation.

Training requires gradients
of the loss
with respect to every entry
of
 $W_1, \mathbf{b}_1, W_2, \mathbf{b}_2, \mathbf{v}, c$.

That is the next mathematical problem.

The Computational Graph View

A network is a composition of simple operations:

matrix multiplication;

vector addition;

activation;

another matrix multiplication;

loss calculation.

Each intermediate quantity depends
on earlier quantities.

This creates a
computational graph.

To train efficiently:
we will differentiate
backward through this graph
using the chain rule.

This is backpropagation.

What did we learn?

- ◆ A neural network learns a feature map by composing trainable affine transformations with nonlinear activation functions.
- ◆ A dense layer has the form $\mathbf{h} = \sigma(W\mathbf{x} + \mathbf{b})$, with one row of W for each hidden unit.
- ◆ Without nonlinear activations, stacked affine layers collapse to a single affine transformation and gain no expressive power.
- ◆ Width counts units in a layer; depth counts successive hidden transformations under our stated convention; layer sizes determine the number of trainable parameters.
- ◆ Regression, binary classification, and multiclass classification use familiar output heads applied to the learned final hidden representation.
- ◆ The forward pass evaluates the network; training requires differentiating the loss through all layers, which leads to the chain rule and backpropagation.



Next

- The forward pass is a composition of simple differentiable or piecewise-differentiable operations.
- Training requires derivatives of one loss with respect to every weight and bias in the network.
- Computing those derivatives independently would repeat large amounts of work.
- The multivariable chain rule lets us reuse intermediate derivatives efficiently.
- **1.33 - The Chain Rule and Backpropagation**