CPEN 455: Deep Learning
Linear Models

Companion notes to Lecture 2

Renjie Liao
With assistance from ChatGPT-6 Astra Ultra.
University of British Columbia Winter Term 1, 2026

1 Statistical Learning Setup

A supervised learning problem has the following components. Linear regression will instantiate each of them.

Component	Role
Problem specification	Identify the input, output, and prediction task.
Model design	Choose a family of prediction functions.
Loss design	Measure prediction errors.
Prediction/inference	Compute an output using a fitted model.
Learning/training	Fit model parameters using training data.
Validation/testing	Select hyperparameters and evaluate on new data.

Data, model, and loss

$$\mathcal{D} = \{(x_n, y_n)\}_{n=1}^{N}, \qquad (x_n, y_n) \overset{\text{i.i.d.}}{\sim} \mathbb{P}_{\text{data}}(x, y).$$

Here N is the number of examples and $x_n \in \mathbb{R}^{D\times 1}$ is a column vector with D input features. *Independent and identically distributed* (i.i.d.) means that examples are sampled independently from the same distribution. An input and its label may still be dependent. A model $f(\cdot, w)$ belongs to a hypothesis class $\mathcal{H}$, the chosen family of functions. It predicts $\hat{y}_n = f(x_n, w)$; $\ell(\hat{y}_n, y_n)$ measures its error. In this general setup, w denotes all learnable parameters; regression will separate weights w from the bias b. A *hyperparameter*, such as the learning rate, controls the model or training procedure and is selected using validation data.

Expected loss and training loss

$$R(w) = \mathbb{E}_{(x,y)\sim\mathbb{P}_{\text{data}}}[\ell(f(x, w), y)], \qquad L(w) = \frac{1}{N}\sum_{n=1}^{N} \ell(f(x_n, w), y_n).$$

We want small expected loss R. Since the distribution is unknown, empirical risk minimization (ERM) instead seeks to minimize the training loss L [1]. Low training loss alone does not guarantee good predictions on new data.

Learning and prediction

For an iterative learning algorithm, write

$$\widehat{w} = \mathcal{A}(\mathcal{D}, w_0, \lambda), \qquad \widehat{y} = f(x, \widehat{w}),$$

where w_0 is the initialization and λ denotes hyperparameters. Learning fits parameters; prediction uses them. The loss specifies what to optimize, and the algorithm specifies how. Gradient descent is one choice; for a general model, it need not find a global minimum.

2 Linear Regression

Problem, model, and loss

For scalar-output regression, $y_n \in \mathbb{R}$. The linear model is

$$\hat{y}_n = f(x_n, w, b) = w^\top x_n + b = \sum_{d=1}^{D} w[d]x_n[d] + b, \qquad w \in \mathbb{R}^{D\times 1}, \quad b \in \mathbb{R}.$$

The bias provides an intercept. The model is affine in x_n, conventionally called linear. Mean squared error (MSE) averages $(\hat{y}_n - y_n)^2$. For the gradient derivation, the slides use half the MSE:

$$l_n := \hat{y}_n - y_n, \qquad \ell(\hat{y}_n, y_n) = \frac{1}{2} l_n^2, \qquad L(w, b) = \frac{1}{2N} \sum_{n=1}^{N} l_n^2. \tag{1}$$

The Latin l_n is a signed *residual*; ℓ is the per-example loss. The factor $1/2$ preserves the minimizer and rescales the gradient.

3 Learning Algorithm: Gradient Descent

Loss gradients

For one example, the chain rule gives

$$\frac{\partial \ell}{\partial w[i]} = \frac{\partial(\frac{1}{2}l_n^2)}{\partial l_n} \frac{\partial l_n}{\partial w[i]} = l_n x_n[i], \qquad \frac{\partial \ell}{\partial b} = l_n.$$

Averaging over the training dataset yields

$$\nabla_w L = \frac{1}{N} \sum_{n=1}^{N} l_n x_n, \qquad \frac{\partial L}{\partial b} = \frac{1}{N} \sum_{n=1}^{N} l_n. \tag{2}$$

The weight gradient has the same shape as w. Each example contributes its residual, weighted by its input features. Appendix A gives the matrix form.

One update and repeated training

With learning rate $\eta > 0$, update both parameters using derivatives at the *same current parameters*:

$$w^{(t+1)} = w^{(t)} - \eta \nabla_w L(w^{(t)}, b^{(t)}), \qquad b^{(t+1)} = b^{(t)} - \eta \frac{\partial L}{\partial b}(w^{(t)}, b^{(t)}).$$

For the slide example, $x_1 = 2$, $y_1 = 3$, $w^{(0)} = 1$, $b^{(0)} = 0$, and $\eta = 0.1$. The residual is -1, the gradients are $(-2, -1)$, and the update gives $(w^{(1)}, b^{(1)}) = (1.2, 0.1)$. The half-MSE decreases from 0.5 to 0.125.

Repeat the update, recomputing gradients each time. A step that is too large can increase the loss. Full-batch gradient descent uses all N examples. Mini-batch SGD uses a random subset $\mathcal{B}$, replacing $N^{-1}\sum_{n=1}^{N}$ with $|\mathcal{B}|^{-1}\sum_{n\in\mathcal{B}}$. Each update is cheaper but noisy.

4 Generalization and Model Evaluation

Optimization reduces the training objective. Generalization concerns performance on new examples from the intended setting [2].

Split	Role
Training	Fit/learn model parameters and any preprocessing statistics.
Validation	Tune hyperparameters and select a model or checkpoint.
Test	Evaluate the selected model without using the results to tune it.

Choose a checkpoint by validation performance; the last iterate need not be best. With limited data, k-fold cross-validation fits on $k-1$ folds and validates on the remaining fold, rotating the held-out fold and averaging its scores. Keep test data outside this selection, and choose splits that prevent information leakage.

An *underfitted* model misses important structure, often showing high training and validation error. An *overfitted* model learns training-specific details that do not generalize, often showing a large training–validation gap. These are diagnostic patterns, not guarantees.

Bias, variance, and noise

For squared-error regression, let $f_{\mathcal{D}}(x)$ be the model fitted on a random training dataset of fixed size. At a fixed input x, define

$$\bar{y}(x) := \mathbb{E}[y \mid x], \qquad \bar{f}(x) := \mathbb{E}_{\mathcal{D}}[f_{\mathcal{D}}(x)],$$
$$\begin{aligned} \mathrm{Bias}(x) &:= \bar{f}(x) - \bar{y}(x), \\ \mathrm{Variance}(x) &:= \mathbb{E}_{\mathcal{D}}[(f_{\mathcal{D}}(x) - \bar{f}(x))^2], \\ \mathrm{Noise}(x) &:= \mathbb{E}[(y - \bar{y}(x))^2 \mid x]. \end{aligned}$$

Bias measures error in the average prediction. Variance measures changes in the prediction across training datasets at the *same input*. Noise is output variation unexplained by that input.

For a fresh test example independent of the training data, assuming finite second moments,

$$\boxed{\mathbb{E}_{\mathcal{D},y}[(y - f_{\mathcal{D}}(x))^2 \mid x] = \mathrm{Bias}(x)^2 + \mathrm{Variance}(x) + \mathrm{Noise}(x).} \tag{3}$$

Training randomness may be fixed or included in the expectation. Averaging both sides over x gives expected test MSE. For half-MSE, divide every term by two. Appendix B gives a short proof.

The bias–variance tradeoff

Increasing flexibility can reduce bias while increasing sensitivity to the training sample. Stronger regularization can reduce variance while increasing bias. Validation helps select a useful balance. The noise term is irreducible given the available input information; adding informative features can change it.

These trends are not universal laws. The decomposition does not require a U-shaped test-error curve, and parameter count alone does not determine generalization. Some interpolating linear models can generalize well under suitable conditions; see Appendix B for the connection to benign overfitting.

5 Binary Classification

Linear score, hard decision, and classification error

For $y_n \in \{0, 1\}$, use the score $f(x_n, w, b) = w^\top x_n + b$. A threshold classifier is

$$\hat{c}_n = \mathbf{1}[f(x_n, w, b) > 0], \qquad \text{error rate} = \frac{1}{N}\sum_{n=1}^{N} \mathbf{1}[\hat{c}_n \neq y_n].$$

Here $\mathbf{1}[\cdot]$ is 1 when its condition holds and 0 otherwise. The decision boundary is a hyperplane; a score of zero is assigned to class 0. The 0–1 loss is constant between decision changes and discontinuous at those changes, so it provides no useful gradient for learning.

Logistic regression and binary cross-entropy

Convert the score, called a *logit*, to a probability using the logistic sigmoid:

$$\hat{y}_n = \sigma(f(x_n, w, b)), \qquad \sigma(a) = \frac{1}{1 + \exp(-a)}.$$

$\hat{y}_n$ estimates the probability of class 1. The hard decision remains $\hat{c}_n = \mathbf{1}[\hat{y}_n > 1/2]$. A differentiable loss is binary cross-entropy (BCE):

$$\ell(\hat{y}_n, y_n) = -y_n \log \hat{y}_n - (1 - y_n) \log(1 - \hat{y}_n), \qquad L(w, b) = \frac{1}{N}\sum_{n=1}^{N} \ell(\hat{y}_n, y_n). \tag{4}$$

BCE is the negative log-likelihood of the observed label under a Bernoulli model [2]. Confidently wrong predictions receive a large loss. Correct predictions can have different BCE values even when their accuracy is the same. All logarithms here are natural. Appendix A derives the BCE gradient.

6 Multiclass Classification

Class scores and one-hot targets

For one of K mutually exclusive classes, $y_n \in \{1, \dots, K\}$, use

$$f(x_n, W, b) = Wx_n + b \in \mathbb{R}^{K\times 1}, \qquad W \in \mathbb{R}^{K\times D}, \quad b \in \mathbb{R}^{K\times 1}.$$

Each row of W provides one class's weights. The output contains K logits; these scores need not sum to 1. The one-hot target has entry $\mathbf{1}[y_n = k]$ at coordinate k, so exactly one entry is 1.

Softmax and cross-entropy

Softmax turns the logits into a valid probability distribution:

$$\hat{y}_n[k] = \frac{\exp(f(x_n, W, b)[k])}{\sum_{j=1}^{K} \exp(f(x_n, W, b)[j])}, \qquad \hat{c}_n = \arg\max_k \hat{y}_n[k].$$

The probabilities are positive and sum to 1. Independent sigmoids instead suit multilabel tasks, where several labels may be true at once. For one-hot targets,

$$\ell(\hat{y}_n, y_n) = -\sum_{k=1}^{K} \mathbf{1}[y_n = k] \log \hat{y}_n[k] = -\log \hat{y}_n[y_n], \qquad L(W, b) = \frac{1}{N}\sum_{n=1}^{N} \ell(\hat{y}_n, y_n).$$

Only the probability of the correct class contributes to this example's loss. Averaging over examples gives the training objective.

More about softmax: temperature and argmax

Let $a = f(x_n, W, b)$ be a fixed vector of finite logits. Following the slides, introduce a *temperature* $\beta > 0$:

$$\mathrm{softmax}_\beta(a)[i] = \frac{\exp(a[i]/\beta)}{\sum_{j=1}^{K} \exp(a[j]/\beta)}.$$

Ordinary softmax uses $\beta = 1$. Smaller temperatures concentrate probability on larger logits, while larger temperatures flatten the distribution. Every positive temperature preserves the ordering of logits and hence the argmax decision.

The same subtraction used for log-sum-exp in Appendix A gives

$$\mathrm{softmax}_\beta(a)[i] = \frac{\exp((a[i] - m)/\beta)}{\sum_{j=1}^{K} \exp((a[j] - m)/\beta)}, \qquad m = \max_j a[j].$$

If the maximum is unique at $k^* = \arg\max_j a[j]$, its exponential is 1 and all others tend to zero as $\beta \to 0^+$. As $\beta \to \infty$, every exponential tends to 1. Therefore,

$$\lim_{\beta\to 0^+} \mathrm{softmax}_\beta(a)[i] = \mathbf{1}[i = k^*], \qquad \lim_{\beta\to\infty} \mathrm{softmax}_\beta(a)[i] = \frac{1}{K}.$$

With q tied maxima, the low-temperature limit assigns $1/q$ to each maximizing entry and zero elsewhere. Thus softmax is a smooth approximation to a one-hot argmax output when the maximum is unique, motivating the slides' term *softargmax*. Its output is a probability vector, rather than the scalar maximum or a class index.

7 Implementation and Neural Networks

Compute cross-entropy from logits

Use the combined losses in `torch.nn.functional`, abbreviated as `F`, to handle sigmoid or log-softmax internally [3]:

- Binary: `F.binary_cross_entropy_with_logits(logits, targets)`. Targets are floating-point binary labels with the same shape as the logits.
- Multiclass: `F.cross_entropy(logits, class_indices)`. For a batch of N examples, logits have shape $N \times K$ and class indices have shape N, with integer labels $0, \ldots, K-1$ in code.

Pass raw logits to these functions. Appendix A gives the stable log-sum-exp identity used to compute multiclass cross-entropy.

What changes in an MLP?

A linear model uses the given features. A multilayer perceptron learns a nonlinear feature representation $h(x)$:

$$f(x) = Wx + b \quad \longrightarrow \quad f(x) = Wh(x) + b.$$

The shape of W changes to match the representation. Nonlinear hidden layers allow more expressive functions; composing affine layers without nonlinearities still gives an affine function. The prediction loss, gradient updates, and validation procedure remain familiar.

Self-check questions

Try these before reading the answers in Appendix C.

1. Does multiplying the regression loss by two change its minimizer? What changes in gradient descent if the numerical learning rate stays fixed?
2. A model has lower training loss but higher validation loss than another model. Which result should guide selection, and what is the test set for?
3. At a fixed input, suppose $\bar{y}(x) = 3$, $\bar{f}(x) = 4$, $\mathrm{Variance}(x) = 2$, and $\mathrm{Noise}(x) = 1$. What are the bias and expected squared prediction error?
4. For $y = 0$, which probability of class 1 gives smaller BCE: 0.1 or 0.4?
5. If $D = 5$ and $K = 3$, what are the shapes of W, b, and $f(x, W, b)$ in multiclass classification?
6. If the same constant is added to every multiclass logit, do the softmax probabilities change?

A Selected Derivations

These details supplement the main text in the same topic order.

Linear regression in matrix form

Let $X \in \mathbb{R}^{N\times D}$ have row n equal to $x_n^\top$, let $y = [y_1, \dots, y_N]^\top$, and let $\mathbf{1} \in \mathbb{R}^{N\times 1}$ contain ones. With residual vector $l = Xw + b\mathbf{1} - y$,

$$L = \frac{1}{2N} l^\top l, \qquad \nabla_w L = \frac{1}{N} X^\top l, \qquad \frac{\partial L}{\partial b} = \frac{1}{N} \mathbf{1}^\top l.$$

The D entries of $X^\top l$ are precisely the sums in (2).

Binary cross-entropy gradient

Write $a = f(x_n, w, b)$ and $\hat{y}_n = \sigma(a)$. Since $\sigma'(a) = \hat{y}_n(1 - \hat{y}_n)$, the chain rule gives

$$\frac{\partial \ell}{\partial a} = \left(-\frac{y_n}{\hat{y}_n} + \frac{1 - y_n}{1 - \hat{y}_n} \right) \hat{y}_n(1 - \hat{y}_n) = \hat{y}_n - y_n.$$

Therefore $\nabla_w \ell = (\hat{y}_n - y_n)x_n$ and $\partial \ell / \partial b = \hat{y}_n - y_n$. The gradients resemble regression, but the output is now a probability and the loss is BCE.

Stable multiclass cross-entropy

For logits $a = f(x_n, W, b)$ and true class $k = y_n$,

$$\ell = -a[k] + \log \sum_{j=1}^{K} \exp(a[j]).$$

Set $m = \max_j a[j]$. Factoring out $\exp(m)$ gives

$$\ell = (m - a[k]) + \log \sum_{j=1}^{K} \exp(a[j] - m).$$

All exponent arguments are nonpositive, avoiding overflow from large positive logits. Compute the loss in this form rather than first forming a tiny probability and then taking its logarithm.

B Further Details on Generalization

Deriving the bias–variance decomposition

Fix an input x. The training dataset $\mathcal{D}$ is random, but its size N and the learning procedure are fixed. A fresh test pair (x, y) is independent of $\mathcal{D}$. Assume finite second moments. As in Section 4,

$$\bar{y}(x) = \mathbb{E}[y \mid x], \qquad \bar{f}(x) = \mathbb{E}_{\mathcal{D}}[f_{\mathcal{D}}(x)].$$

Any training randomness can be included with $\mathcal{D}$. The label y may depend on its own input x; the independence assumption concerns the test pair and the training dataset.

1. Separate label noise from prediction error. Add and subtract $\bar{y}(x)$, expand the square, and take expectations:

$$\begin{aligned}\mathbb{E}_{\mathcal{D},y}[(y - f_{\mathcal{D}}(x))^2 \mid x] &= \mathbb{E}_{\mathcal{D},y}\Big[\big((y - \bar{y}(x)) + (\bar{y}(x) - f_{\mathcal{D}}(x))\big)^2 \mid x\Big] \\ &= \underbrace{\mathbb{E}[(y - \bar{y}(x))^2 \mid x]}_{\text{Noise}(x)} + \mathbb{E}_{\mathcal{D}}[(f_{\mathcal{D}}(x) - \bar{y}(x))^2] \\ &\quad + 2\mathbb{E}_{\mathcal{D},y}[(y - \bar{y}(x))(\bar{y}(x) - f_{\mathcal{D}}(x)) \mid x].\end{aligned}$$

The last term is zero. Conditional on x and $\mathcal{D}$, the fitted prediction is fixed and the fresh label still has mean $\bar{y}(x)$:

$$\mathbb{E}_{\mathcal{D},y}[(y - \bar{y})(\bar{y} - f_{\mathcal{D}}) \mid x] = \mathbb{E}_{\mathcal{D}}\left[(\bar{y} - f_{\mathcal{D}})\underbrace{\mathbb{E}[y - \bar{y} \mid x, \mathcal{D}]}_{0}\right] = 0.$$

Here and below, arguments (x) are omitted inside intermediate expressions when there is no ambiguity.

2. Separate squared bias from model variance. Now add and subtract the average fitted prediction $\bar{f}(x)$:

$$\begin{aligned}\mathbb{E}_{\mathcal{D}}[(f_{\mathcal{D}} - \bar{y})^2] &= \mathbb{E}_{\mathcal{D}}\Big[\big((f_{\mathcal{D}} - \bar{f}) + (\bar{f} - \bar{y})\big)^2\Big] \\ &= \underbrace{\mathbb{E}_{\mathcal{D}}[(f_{\mathcal{D}} - \bar{f})^2]}_{\text{Variance}(x)} + \underbrace{(\bar{f} - \bar{y})^2}_{\text{Bias}(x)^2} + 2(\bar{f} - \bar{y})\underbrace{\mathbb{E}_{\mathcal{D}}[f_{\mathcal{D}} - \bar{f}]}_{0}.\end{aligned}$$

The second cross term vanishes by the definition of $\bar{f}(x)$. Combining the two steps gives (3). Finally, average over test inputs:

$$\mathbb{E}_{\mathcal{D},(x,y)}[(y - f_{\mathcal{D}}(x))^2] = \mathbb{E}_x[\text{Bias}(x)^2] + \mathbb{E}_x[\text{Variance}(x)] + \mathbb{E}_x[\text{Noise}(x)].$$

For the half-MSE loss in (1), divide all four terms by two. This identity holds for any learning procedure under these assumptions. It does not specify how the terms change when the model becomes more flexible.

Benign overfitting in linear regression

Consider centered Gaussian inputs, $y = x^\top w^* + \varepsilon$, and independent zero-mean Gaussian noise with variance $\sigma^2 > 0$. Here w^* is the true parameter vector and ε is label noise. For $D > N$ and full row rank $X \in \mathbb{R}^{N\times D}$, the minimum-norm interpolator is

$$\widehat{w} = \operatorname*{arg\,min}_{w:\, Xw=y} \|w\|_2^2 = X^\top (XX^\top)^{-1} y, \qquad X\widehat{w} = y.$$

Interpolation gives zero training error, including an exact fit to the label noise. Bartlett et al. give nearly matching bounds on its excess test MSE [4]:

$$\mathcal{E}(\widehat{w}) := \mathbb{E}_{x,y}[(y - x^\top \widehat{w})^2 \mid \mathcal{D}] - \sigma^2 = (\widehat{w} - w^*)^\top \Sigma (\widehat{w} - w^*), \quad \Sigma = \mathbb{E}[xx^\top].$$

Covariance eigenvalues measure input variation in different directions. Under suitable spectra, the minimum-norm predictor can spread the fitting of training noise across many weak directions, keeping its effect on new predictions small.

For covariance eigenvalues $\mu_1 \geq \cdots \geq \mu_D > 0$, define tail effective ranks

$$r_k = \frac{\sum_{j>k} \mu_j}{\mu_{k+1}}, \qquad R_k = \frac{(\sum_{j>k} \mu_j)^2}{\sum_{j>k} \mu_j^2}, \qquad k^* = \min\{0 \leq k < D : r_k \geq cN\},$$

where $c > 1$ is a theorem constant ($k^* = \infty$ if the set is empty). As $N \to \infty$, allow D and Σ to vary with N. With bounded $\|w^*\|$, μ_1, and σ^2, sufficient conditions for $\mathcal{E}(\widehat{w}) \to 0$ in probability are

$$\frac{r_0}{N} \to 0, \qquad \frac{k^*}{N} \to 0, \qquad \frac{N}{R_{k^*}} \to 0.$$

The spectrum must contain many weak directions without excessive total variance. Up to constants and confidence factors, the noise contribution scales with $k^*/N + N/R_{k^*}$; parameter count alone misses this structure.

For example, take one eigenvalue equal to 1 and N^2 eigenvalues equal to N^{-2}. Then $D = N^2 + 1$, $r_0 = 2$, $k^* = 1$, and $R_{k^*} = N^2$ for large N: all three ratios vanish. This example follows directly from the displayed conditions.

Thus exact fitting can coexist with test MSE approaching the noise floor. The result concerns this estimator and data model, not arbitrary interpolation or a guarantee for deep networks. The bias–variance identity remains valid.

C Self-Check Answers

1. The minimizers are unchanged. Gradients double, so the update doubles at the same current parameters. Halving the learning rate restores the original update.
2. Use validation performance for selection. Use the held-out test set to evaluate the selected model without tuning it using those results.
3. $\mathrm{Bias}(x) = 4 - 3 = 1$. Expected squared error is $1^2 + 2 + 1 = 4$; expected half-squared error is 2.
4. 0.1: its loss is $-\log 0.9 \approx 0.105$, compared with $-\log 0.6 \approx 0.511$ for probability 0.4.
5. $W \in \mathbb{R}^{3\times 5}$, $b \in \mathbb{R}^{3\times 1}$, and $f(x, W, b) \in \mathbb{R}^{3\times 1}$.
6. No. For a shared constant c, the factor $\exp(c)$ cancels:

$$\frac{\exp(a[k]+c)}{\sum_j \exp(a[j]+c)} = \frac{\exp(c)\exp(a[k])}{\exp(c)\sum_j \exp(a[j])} = \frac{\exp(a[k])}{\sum_j \exp(a[j])}.$$

References

[1] V. N. Vapnik. *The Nature of Statistical Learning Theory*. Springer, second edition, 1999. doi:10.1007/978-1-4757-3264-1.

[2] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006. Chapters 3–4. Author's book page.

[3] PyTorch contributors. Official documentation: `binary_cross_entropy_with_logits` and `cross_entropy`.

[4] P. L. Bartlett, P. M. Long, G. Lugosi, and A. Tsigler. "Benign overfitting in linear regression." *PNAS*, 117(48):30063–30070, 2020. doi:10.1073/pnas.1907378117.