Gradient descent (continued)
Gradient descent algorithm

- Set $\mathbf{w}$ to random values; call this initial choice $\mathbf{w}^{(0)}$.
- Compute the gradient: $\nabla_{\mathbf{w}} f(\mathbf{w}^{(0)})$
- Update $\mathbf{w}$ by moving opposite the gradient, multiplied by a step size $\varepsilon$.
  
  $\mathbf{w}^{(1)} \leftarrow \mathbf{w}^{(0)} - \varepsilon \nabla_{\mathbf{w}} f(\mathbf{w}^{(0)})$

Repeat...

$\mathbf{w}^{(2)} \leftarrow \mathbf{w}^{(1)} - \varepsilon \nabla_{\mathbf{w}} f(\mathbf{w}^{(1)})$

$\mathbf{w}^{(3)} \leftarrow \mathbf{w}^{(2)} - \varepsilon \nabla_{\mathbf{w}} f(\mathbf{w}^{(2)})$

... 

$\mathbf{w}^{(t)} \leftarrow \mathbf{w}^{(t-1)} - \varepsilon \nabla_{\mathbf{w}} f(\mathbf{w}^{(t-1)})$

- ...until convergence:

  $|f(\mathbf{w}^{(t-1)}) - f(\mathbf{w}^{(t)})| < \delta$

  $\delta$ is a chosen convergence tolerance.
Gradient descent demos

- 1-d
- 2-d
Convergence

- In general, gradient descent is useful for finding a local minimum of $f$:

- **Local minimum:** gradient is 0; second derivative is positive.
Convergence

- In general, gradient descent is useful for finding a local minimum of $f$:

- **Global minimum** is the smallest value of the function $f$.
Convergence

• For the special case of linear regression (and a few other ML models), gradient descent will (for appropriate $\varepsilon$) converge to the \textit{global} minimum of $f_{\text{MSE}}$. 

• What does “appropriate $\varepsilon$” mean (intuitively)?
  
  • Big enough to make progress (from random starting point) to local minimum.
  
  • Small enough not to “jump around” too much.

• Show demo.
Convergence

- For the special case of linear regression (and a few other ML models), gradient descent will (for appropriate $\varepsilon$) converge to the global minimum of $f_{\text{MSE}}$.

- What does “appropriate $\varepsilon$” mean (intuitively)?
  
  - Big enough to make progress (from random starting point) to local minimum.

  - Small enough not to “jump around” too much.

  - Show demo.
Convergence

• For the special case of linear regression (and a few other ML models), gradient descent will (for appropriate $\epsilon$) converge to the *global* minimum of $f_{MSE}$.

• In practice:
  
  • Choose some $\epsilon$ so that the cost $f_{MSE}$ declines *smoothly*.
  
  • If it’s too slow, try increasing.
  
  • If it jumps around, try decreasing.
Polynomial regression
Linear regression

- Linear regression is efficient to optimize and very useful, but is limited in its expressiveness.

- Unsurprisingly, it can only model linear (technically affine) relationships:

\[ \hat{y} = x^\top w + b \]

- In 1-d:

\[ \hat{y} = wx + b \]
Linear regression

• But sometimes the target values $y$ have a non-linear relationship with the input $x$.

• Linear regression may not do a good job then.

• Example: $y = 0.2x - 1.8x^2 + 0.8x^3 + \text{noise}$

Not a great fit.
Polynomial regression

• If the labels $y$ are a polynomial function of the inputs $x$, why not enable the model to express polynomial relationships?

• In 1-d, we can build a *cubic* regression model as follows:

$$\hat{y} = w_1 x^1 + w_2 x^2 + w_3 x^3 + b$$

$$= w_0 x^0 + w_1 x^1 + w_2 x^2 + w_3 x^3$$

Much better fit!
Polynomial regression

• How do we train the weights of the polynomial regression (for 1-d inputs)?

• Pretend that each power of $x$ is a separate feature.

• Form $\mathbf{x} = [x^0, x^1, x^2, x^3]^T$
Polynomial regression

• How do we train the weights of the polynomial regression (for 1-d inputs)?

• Pretend that each power of \( x \) is a separate feature.

• Form \( \mathbf{x} = [x^0, x^1, x^2, x^3]^T \)

• Example:

• Suppose the raw input \( x = -1.5 \).

• Then \( x_0 = 1, x_1 = -1.5, x^2 = 2.25, \) and \( x^3 = -3.375 \). Hence, \( \mathbf{x} = [1, -1.5, 2.25, -3.375]^T \).
Polynomial regression

• Now, notice that:

\[ \hat{y} = w_0 x^0 + w_1 x^1 + w_2 x^2 + w_3 x^3 \]
Polynomial regression

• Now, notice that:

\[ \hat{y} = w_0 x^0 + w_1 x^1 + w_2 x^2 + w_3 x^3 \]

\[ = \begin{bmatrix} x^0 & x^1 & x^2 & x^3 \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ w_3 \end{bmatrix} \]
Polynomial regression

• Now, notice that:

\[ \hat{y} = w_0 x^0 + w_1 x^1 + w_2 x^2 + w_3 x^3 \]

\[ = \begin{bmatrix} x^0 & x^1 & x^2 & x^3 \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ w_3 \end{bmatrix} \]

\[ = \mathbf{x}^\top \mathbf{w} \]

When we “pre-compute” each power of \( x \), we convert the polynomial regression back into a linear regression model.
Polynomial regression

- We can convert each input $x$ into a feature vector $\mathbf{x}$, and create a design matrix $\mathbf{X}$, and compute the optimal $\mathbf{w}$ as before...
Polynomial regression

• Suppose we have raw inputs -1.5, -1, and 3.25.

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$i=1$</td>
<td>$i=2$</td>
<td>$i=3$</td>
</tr>
<tr>
<td>-1.5</td>
<td>-1</td>
<td>3.25</td>
</tr>
</tbody>
</table>
Polynomial regression

- Suppose we have raw inputs -1.5, -1, and 3.25.

- Then for each (scalar) $x$ we build a vector $\mathbf{x}$ consisting of $[x^0, x^1, x^2, x^3]^T$:

<table>
<thead>
<tr>
<th></th>
<th>$i=1$</th>
<th>$i=2$</th>
<th>$i=3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d=0$</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$d=1$</td>
<td>-1.5</td>
<td>-1</td>
<td>3.25</td>
</tr>
<tr>
<td>$d=2$</td>
<td>2.25</td>
<td>1</td>
<td>10.5625</td>
</tr>
<tr>
<td>$d=3$</td>
<td>-3.375</td>
<td>-1</td>
<td>4.32812</td>
</tr>
</tbody>
</table>
Polynomial regression

- The matrix of all our examples (as column vectors) constitutes the design matrix $X$, as usual.

\[
X = \begin{bmatrix}
1 & 1 & 1 \\
-1.5 & -1 & 3.25 \\
2.25 & 1 & 10.5625 \\
-3.375 & -1 & 4.328125
\end{bmatrix}
\]
Polynomial regression

- The matrix of all our examples (as column vectors) constitutes the design matrix $X$, as usual.

- We can now find the optimal polynomial regression coefficients by computing:

$$w = (XX^\top)^{-1} Xy$$

- ...just like with linear regression.

$$X = \begin{bmatrix}
1 & 1 & 1 \\
-1.5 & -1 & 3.25 \\
2.25 & 1 & 10.5625 \\
-3.375 & -1 & 4.32812
\end{bmatrix}$$
Overfitting and regularization
Overfitting

- If polynomial regression with degree 3 worked well, why not increase the degree even higher?
- Let’s try with degree 25…
Overfitting

• If polynomial regression with degree 3 worked well, why not increase the degree even higher?

• Let’s try with degree 25…

We nailed almost every point exactly!… but maybe this is overkill?
Overfitting

• Why is this bad? Recall that overfitting means that training error is low, but testing error is high.

• Testing error represents how well we expect our machine to perform on data we have not seen before.
Overfitting

• When we collect a dataset, we are sampling from probability distribution \( p(x) \)
Overfitting

- When we collect a dataset, we are *sampling* from probability distribution $p(x)$ and conditional probability distribution $p(y \mid x)$.

$$y = 0.2x - 1.8x^2 + 0.8x^3 + \text{noise}$$
Overfitting

• When we sample multiple times, we will get different results. Here is a possible *training* sample:
Overfitting

• When we sample multiple times, we will get different results. Here is a possible *testing* sample:
Overfitting

- Here are the machine’s predictions using polynomial regression, with either degree 3 or degree 5:

For these data points, the predictions are very inaccurate, which makes $f_{\text{MSE}}$ large.
Preventing overfitting

- How to prevent this? Two strategies:
  - Keep the *degree* $d$ of the polynomial modest.
Preventing overfitting

- How to prevent this? Two strategies:
  - Keep the degree $d$ of the polynomial modest.
  - Keep the weight associated with each term modest.

\[
\hat{y} = w_0 x^0 + w_1 x^1 + w_2 x^2 + \ldots + w_d x^d
\]
Random polynomials

- Let’s generate some polynomials by randomly selecting each \( w_i \) in:

\[
\hat{y} = w_0 x^0 + w_1 x^1 + w_2 x^2 + \ldots + w_d x^d
\]

- Compute the average squared coefficient as:

\[
\mu = \frac{1}{d} \sum_{i=0}^{d} w_i^2
\]

- We will generate random polynomials for different degrees \( d \) and different coefficient magnitudes \( \mu \).
Random polynomials

\[ \mu = \frac{1}{d} \sum_{i=0}^{d} w_i^2 \]

- Examples:

  \( d = 2 \):  \( \hat{y} = 4 + 2x - 2x^2 \quad \mu = \quad ? \)
  
  \( d = 4 \):  \( \hat{y} = x^2 + 0.5x^3 - 2x^4 \quad \mu = \quad ? \)
Random polynomials

\[ \mu = \frac{1}{d} \sum_{i=0}^{d} w_i^2 \]

- Examples:
  - \( d = 2 \):
    \[ \hat{y} = 4 + 2x - 2x^2 \quad \mu = 24/3 = 8 \]
  - \( d = 4 \):
    \[ \hat{y} = x^2 + 0.5x^3 - 2x^4 \quad \mu = 5.25/5 = 1.05 \]
Random polynomials

$\mu = 7.25 \times 10^{-7}$
Random polynomials

$\mu = 0.00063$
Random polynomials

\( \mu = 0.058 \)
Random polynomials

$\mu = 3.31$
Random polynomials

$\mu=90.9$
Random polynomials

$\mu = 537.8$
Regularization

- The larger the coefficients (weights) $w$ are allowed to be, the more the polynomial regressor can overfit.

- If we “encourage” the weights to be small, we can reduce overfitting.

- This is a form of regularization — any practice designed to improve the machine’s ability to generalize to new data.
• One of the simplest and oldest regularization techniques is to *penalize* large weights in the cost function.

• The “unregularized” $f_{\text{MSE}}$ is:

$$f_{\text{MSE}}(w) = \frac{1}{2n} \sum_{i=1}^{n} (y^{(i)} - \hat{y}^{(i)})^2$$
Regularization

• One of the simplest and oldest regularization techniques is to penalize large weights in the cost function.

• The “unregularized” $f_{MSE}$ is:

$$f_{MSE}(w) = \frac{1}{2n} \sum_{i=1}^{n} (y^{(i)} - \hat{y}^{(i)})^2$$

• The $L_2$-regularized $f_{MSE}$ becomes:

$$f_{MSE}(w) = \frac{1}{2n} \sum_{i=1}^{n} (y^{(i)} - \hat{y}^{(i)})^2 + \frac{\alpha}{2} w^\top w$$